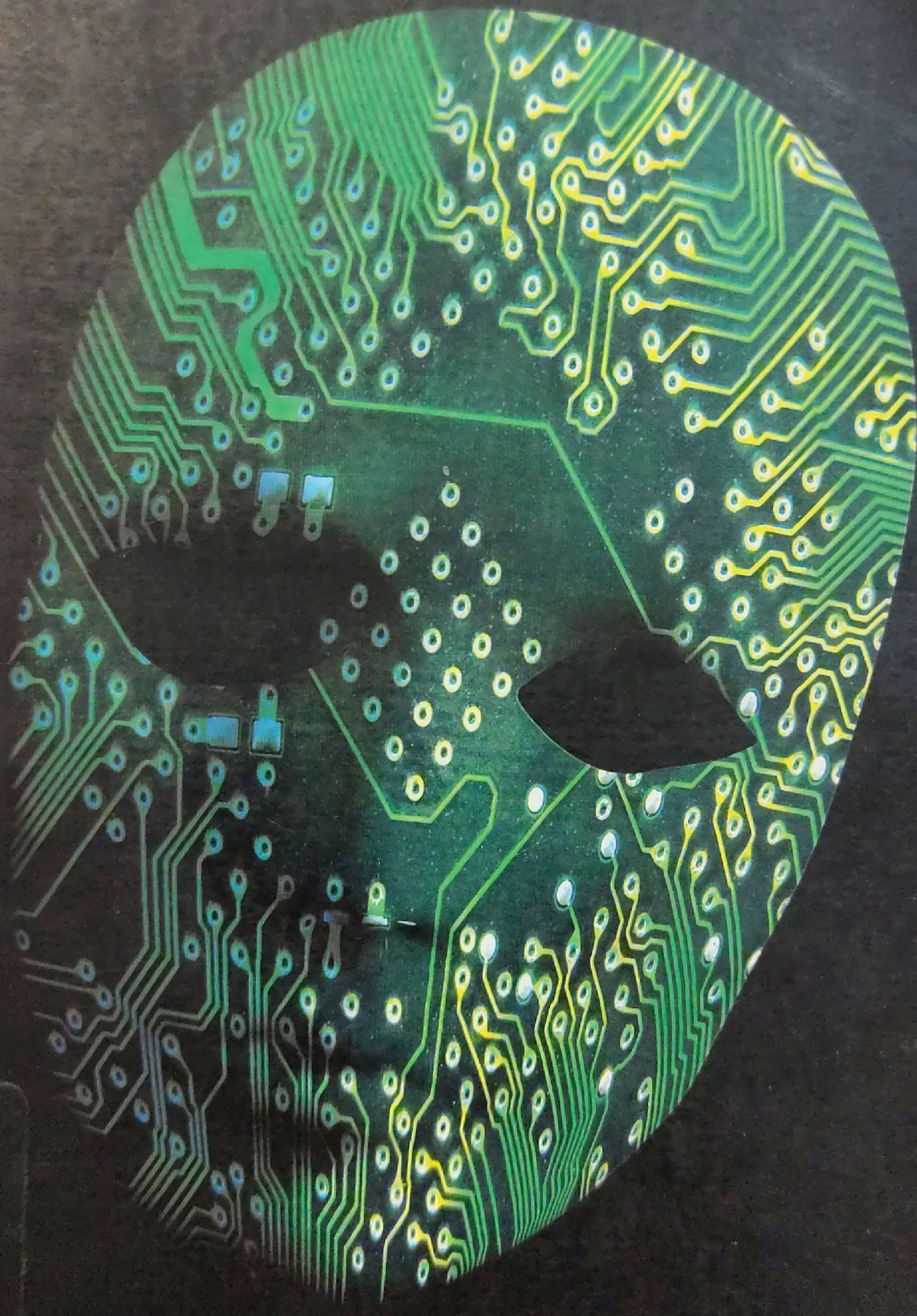
Yapay Öğrenme

Ethem Alpaydın



BOĞAZİÇİ ÜNİVERSİTESİ YAYINEVİ

YAPAY ÖĞRENME

YAPAY ÖĞRENME

Ethem Alpaydın

BOĞAZİÇİ
ÜNİVERSİTESİ
YAYINEVİ

Ethem Alpaydın
Introduction to Machine Learning
Copyright © The MIT Press, 2004, 2010. All rights reserved.

Yapay Öğrenme
© BÜTEK A.Ş. 2007. Tüm hakları saklıdır.

Boğaziçi Üniversitesi Yayınevi
Boğaziçi Üniversitesi Uçaksavar Kampüsü
Cengiz topel Caddesi, Garanti Kültür Merkezi, Arka Giriş
Etiler/İstanbul

bupress@boun.edu.tr
www.bupress.org, www.bupress.net
Telefon ve faks: (90) 212 257 87 27
Sertifika No: 10821

Çeviri ve yayına hazırlık: Ethem Alpaydın
Kapak tasarımı: Kerem Yeğin
Baskı: G.M. Matbaacılık ve Ticaret A.Ş.
100 Yıl Mah. MAS-SİT, 1. Cadde, No: 88
Bağcılar/İstanbul
Telefon: 0212 629 06 07
Sertifika No: 12358

Birinci Basım: Mart 2011

Boğaziçi University Library Cataloging in Publication Data
Alpaydın, Ethem, 1954-
Yapay Öğrenme / Ethem Alpaydın; çeviren Ethem Alpaydın.
496 p.; 23 cm.
Includes bibliographical references and index.
ISBN 978-605-4238-49-1
1. Machine learning. I. Title.
Q325.5

İçindekiler

Önsöz xv

İkinci Basım için Önsöz xvii

Türkçe Çeviri için Önsöz xix

Gösterim xxi

1 Giriş 1
- 1.1 Yapay Öğrenme Nedir? . . . 1
- 1.2 Yapay Öğrenme Uygulamalarına Örnekler . . . 3
  - 1.2.1 İlişkilendirme Kuralları . . . 3
  - 1.2.2 Sınıflandırma . . . 4
  - 1.2.3 Bağlanım . . . 7
  - 1.2.4 Gözetimsiz Öğrenme . . . 8
  - 1.2.5 Pekiştirmeli Öğrenme . . . 10
- 1.3 Kısa Kısa . . . 10
- 1.4 İlgili Kaynaklar . . . 12
- 1.5 Alıştırmalar . . . 13
- 1.6 Kaynaklar . . . 14

2 Gözetimli Öğrenme 15
- 2.1 Bir Sınıfı Örneklerinden Öğrenme . . . 15
- 2.2 Vapnik-Chervonenkis (VC) Boyutu . . . 20
- 2.3 Olası Yaklaşık Doğru Öğrenme . . . 21
- 2.4 Gürültü . . . 23
- 2.5 Çok Sınıflı Öğrenme . . . 24
- 2.6 Bağlanım . . . 26
- 2.7 Model Seçimi ve Genelleme . . . 28
- 2.8 Gözetimli Yapay Öğrenmenin Boyutları . . . 31
- 2.9 Kısa Kısa . . . 33
- 2.10 Alıştırmalar . . . 33

2.11 Kaynaklar 34

3 Bayesçi Karar Kuramı **37**
3.1 Giriş 37
3.2 Sınıflandırma 38
3.3 Yitimler ve Riskler 40
3.4 Ayırtaç İşlevleri 42
3.5 Fayda Kuramı 43
3.6 İlişkilendirme Kuralları 44
3.7 Kısa Kısa 46
3.8 Alıştırmalar 46
3.9 Kaynaklar 47

4 Dağılıma Bağlı Yöntemler **49**
4.1 Giriş 49
4.2 En Büyük Olabilirlik Kestirimi 50
4.2.1 Bernoulli Dağılımı 50
4.2.2 Katlıterimli Dağılım 51
4.2.3 Gauss (Normal) Dağılım 52
4.3 Kestirici Değerlendirme: Yanlılık ve Değişinti 52
4.4 Bayesçi Kestirici 54
4.5 Dağılıma Bağlı Sınıflandırma 56
4.6 Bağlanım 59
4.7 Model Karmaşıklığı Ayarlama: Yanlılık/Değişinti İkilemi 62
4.8 Model Seçim Yordamları 65
4.9 Kısa Kısa 69
4.10 Alıştırmalar 69
4.11 Kaynaklar 70

5 Çok Değişkenli Yöntemler **71**
5.1 Çok Değişkenli Veri 71
5.2 Parametre Kestirimi 72
5.3 Eksik Değerlerin Kestirilmesi 73
5.4 Çok Değişkenli Normal Dağılım 73
5.5 Çok Değişkenli Sınıflandırma 76
5.6 Karmaşıklığın Ayarlanması 82
5.7 Kesikli Öznitelikler 84
5.8 Çok Değişkenli Bağlanım 85
5.9 Kısa Kısa 87
5.10 Alıştırmalar 88
5.11 Kaynaklar 88

6 Boyut Azaltma **89**
6.1 Giriş 89
6.2 Altküme Seçimi 90
6.3 Temel Bileşen Çözümlemesi 92
6.4 Etken Çözümlemesi 97
6.5 Çok Boyutlu Ölçekleme 102
6.6 Doğrusal Ayırtaç Çözümlemesi 105
6.7 Eşölçümsel Öznitelik Eşleme 109
6.8 Yerel Doğrusal Gömme 111
6.9 Kısa Kısa 113
6.10 Alıştırmalar 114
6.11 Kaynaklar 115

7 Öbekleme **117**
7.1 Giriş 117
7.2 Karışım Dağılımı 118
7.3 k Merkezli Öbekleme 119
7.4 Beklenti Büyütme Algoritması 122
7.5 Saklı Değişken Modellerinin Karışımı 127
7.6 Öbekleme Ardından Gözetimli Öğrenme 128
7.7 Öbek Ağaçları 129
7.8 Öbek Sayısının Belirlenmesi 131
7.9 Kısa Kısa 131
7.10 Alıştırmalar 132
7.11 Kaynaklar 132

8 Dağılımdan Bağımsız Yöntemler **135**
8.1 Giriş 135
8.2 Dağılımdan Bağımsız Olasılık Kestirimi 136
8.2.1 Çubukçizit Kestirimi 137
8.2.2 Çekirdek Kestiricisi 138
8.2.3 En Yakın k Komşu Kestirimi 140
8.3 Çok değişkenli Veriye Genelleme 141
8.4 Dağılımdan Bağımsız Sınıflandırma 142
8.5 Seyrek En Yakın Komşu 143
8.6 Modelden Bağımsız Bağlanım: Düzleştiriciler 145
8.6.1 Akan Ortalama Düzleştiricisi 145
8.6.2 Çekirdek Düzleştiricisi 147
8.6.3 Akan Doğru Düzleştiricisi 147
8.7 Düzleştirme Parametresinin Seçimi 148
8.8 Kısa Kısa 149
8.9 Alıştırmalar 150
8.10 Kaynaklar 151

9 Karar Ağaçları **153**
9.1 Giriş . . . 153
9.2 Tek Değişkenli Ağaç . . . 155
9.2.1 Sınıflandırma Ağaçları . . . 155
9.2.2 Bağlanım Ağacı . . . 159
9.3 Budama . . . 160
9.4 Ağaçtan Kural Çıkarma . . . 161
9.5 Kural Öğrenme . . . 165
9.6 Çok Değişkenli Ağaç . . . 168
9.7 Kısa Kısa . . . 170
9.8 Alıştırmalar . . . 171
9.9 Kaynaklar . . . 172

10 Doğrusal Sınıflandırma **175**
10.1 Giriş . . . 175
10.2 Doğrusal Modelin Genelleştirilmesi . . . 176
10.3 Doğrusal Ayırtacın Geometrisi . . . 177
10.3.1 İki Sınıf . . . 177
10.3.2 $K > 2$ Sınıf . . . 179
10.4 İkili Sınıflandırma . . . 181
10.5 Dağılıma Bağlı Sınıflandırmaya Yeniden Bir Bakış . . . 182
10.6 Eğim İniş . . . 184
10.7 S Biçimli Sınıflandırma . . . 185
10.7.1 İki Sınıf . . . 185
10.7.2 $K > 2$ Sınıf . . . 188
10.8 Bağlanımla Sınıflandırma . . . 192
10.9 Kısa Kısa . . . 193
10.10 Alıştırmalar . . . 194
10.11 Kaynaklar . . . 194

11 Çok Katmanlı Algılayıcılar **197**
11.1 Giriş . . . 197
11.1.1 Beyni Anlamak . . . 198
11.1.2 Koşut İşlem İçin Bir Örnek Olarak Yapay Sinir Ağları . . . 199
11.2 Algılayıcı . . . 200
11.3 Algılayıcının Eğitilmesi . . . 203
11.4 Mantık İşlevlerinin Öğrenilmesi . . . 205
11.5 Çok Katmanlı Algılayıcılar . . . 207
11.6 Evrensel bir Yaklaşıklayıcı olarak Çok Katmanlı Algılayıcı . . . 209
11.7 Hata Geri Yayma Algoritması . . . 210
11.7.1 Doğrusal Olmayan Bağlanım . . . 210
11.7.2 İki Sınıf Sınıflandırma . . . 213
11.7.3 $K > 2$ Sınıf Sınıflandırma . . . 215
11.7.4 Birden Çok Saklı Katman . . . 216

11.8 Öğrenme Yordamları 216
11.8.1 Yakınsamayı İyileştirme 216
11.8.2 Fazla Öğrenme 217
11.8.3 Ağın Yapılandırılması 219
11.8.4 İpucu 220
11.9 Ağ Büyüklüğünün Uyarlanması 222
11.10Öğrenmeye Bayesçi Bakış 224
11.11Boyut Azaltma 225
11.12Zamanı Öğrenmek 228
11.12.1 Zaman Gecikmeli Sinir Ağı 228
11.12.2 Özyineli Ağlar 229
11.13Kısa Kısa 231
11.14Alıştırmalar 232
11.15Kaynaklar 232

12 Yerel Modeller **235**
12.1 Giriş 235
12.2 Yarışmacı Öğrenme 235
12.2.1 k Merkezli Çevrimiçi Öbekleme 236
12.2.2 Artımlı Çevrimiçi Öbekleme 240
12.2.3 Özörgütlemeli Haritalar 241
12.3 Dairesel Tabanlı İşlev Ağları 242
12.4 Kural Tabanlı Bilginin Tanımlanması 248
12.5 Normalleştirilmiş Taban İşlevleri 249
12.6 Yarışmacı Taban İşlevleri 251
12.7 Öğrenen Yöney Nicemleme 253
12.8 Uzmanların Karışımı 253
12.8.1 Yardımlaşan Uzmanlar 256
12.8.2 Yarışmacı Uzmanlar 256
12.9 Uzmanların Karışım Ağacı 257
12.10Kısa kısa 258
12.11Alıştırmalar 258
12.12Kaynaklar 259

13 Çekirdek Makineleri **261**
13.1 Giriş 261
13.2 En İyi Ayıran Üstündüzlem 262
13.3 Doğrusal Ayrılamayan Sınıflar: Eşiksiz Kenarlı Üstündüzlem 266
13.4 ν-DYM 269
13.5 Çekirdeklemek 270
13.6 Yöney Çekirdekleri 272
13.7 Çekirdek Tanımlama 275
13.8 Çekirdek Birleştirme 276

13.9 Çok Sınıflı Çekirdek Makinesi 277
13.10Bağlanım için Çekirdek Makinesi 278
13.11Tek Sınıflı Çekirdek Makinesi 281
13.12Çekirdekli Boyut Azaltma 285
13.13Kısa Kısa 287
13.14Alıştırmalar 288
13.15Kaynaklar 288

14 Bayesçi Kestirim **291**
14.1 Giriş 291
14.2 Bir Dağılımın Parametrelerinin Kestirimi 293
14.2.1 Kesikli Değişkenler 293
14.2.2 Sürekli Değişkenler 295
14.3 Bir İşlevin Parametrelerinin Bayesçi Kestirimi 297
14.3.1 Bağlanım 297
14.3.2 Taban ve Çekirdek İşlevlerinin Kullanımı 301
14.3.3 Bayesçi Sınıflandırma 303
14.4 Gauss Süreçleri 304
14.5 Kısa Kısa 308
14.6 Alıştırmalar 308
14.7 Kaynaklar 308

15 Saklı Markov Modelleri **311**
15.1 Giriş 311
15.2 Kesikli Markov Süreçleri 312
15.3 Saklı Markov Modelleri 314
15.4 Saklı Markov Modellerinde Üç Temel Soru 316
15.5 Değerlendirme 317
15.6 Durum Dizisinin Bulunması 320
15.7 Model Parametrelerinin Öğrenilmesi 322
15.8 Sürekli Gözlemler 325
15.9 Girdiye Bağlı Saklı Markov Modeli 326
15.10Saklı Markov Modellerinde Model Seçimi 326
15.11Kısa Kısa 328
15.12Alıştırmalar 329
15.13Kaynaklar 330

16 Çizgesel Modeller **333**
16.1 Giriş 333
16.2 Koşullu Bağımsızlığın Asal Örnekleri 335
16.3 Örnek Çizgesel Modeller 341
16.3.1 Saf Bayes Sınıflandırıcı 341
16.3.2 Saklı Markov Modeli 342
16.3.3 Doğrusal Bağlanım 345

16.4 d Ayrımı 346
16.5 İnanç Yayılımı 347
16.5.1 Zincir 347
16.5.2 Ağaç 348
16.5.3 Çoklu Ağaç 350
16.5.4 Kavşak Ağacı 352
16.6 Yönsüz Çizgeler: Markov Rastsal Alanları 353
16.7 Çizgesel Modelin Yapısının Öğrenilmesi 356
16.8 Etki Çizemleri 356
16.9 Kısa Kısa 356
16.10Alıştırmalar 358
16.11Kaynaklar 358

17 Modellerin Birleştirilmesi **361**
17.1 Gerekçe 361
17.2 Farklı Modellerin Oluşturulması 362
17.3 Model Birleştirme Yöntemleri 364
17.4 Oylama 365
17.5 Hata Düzelten Çıktı Gizyazıları 368
17.6 Örnekleme ve Oylama 370
17.7 Sırayla Eğiterek İteleme 371
17.8 Uzmanların Karışımına Farklı Bir Bakış 373
17.9 Yığılmış Genelleme 374
17.10Model Kümesinin Ayarlanması 375
17.11Ardışık Sınıflandırıcılar 376
17.12Kısa Kısa 378
17.13Alıştırmalar 380
17.14Kaynaklar 380

18 Pekiştirmeli Öğrenme **383**
18.1 Giriş 383
18.2 Tek Durum: K Kollu Kumar Makinesi 384
18.3 Pekiştirmeli Öğrenmenin Öğeleri 385
18.4 Model Tabanlı Öğrenme 387
18.4.1 Değer Yineleme 388
18.4.2 Politika Yineleme 388
18.5 Zamansal Fark Öğrenme 388
18.5.1 Arama Yordamları 389
18.5.2 Gerekirci Ödül ve Eylemler 390
18.5.3 Rastgele Ödül ve Eylemler 390
18.5.4 Uygunluk İzleri 392
18.6 Genelleme 395
18.7 Kısmen Gözlenebilen Durumlar 396
18.7.1 Tanımlar 396

18.7.2 Örnek: Saklı Kaplan Problemi 398
18.8 Kısa Kısa 402
18.9 Alıştırmalar 403
18.10Kaynaklar 404

19 Deney Tasarımı ve Sonuçların Çözümlenmesi **407**
19.1 Giriş 407
19.2 Etkenler, Tepki, ve Deney Yordamları 410
19.3 Tepki Yüzeyi Tasarımı 412
19.4 Rastsallaştırma, Çoğaltma, ve Bölükleme 412
19.5 Yapay Öğrenme Deneyleri için İlkeler 413
19.6 Çapraz Geçerleme ve Yeniden Örnekleme 416
19.6.1 K Kat Çapraz Geçerleme 416
19.6.2 5×2 Çapraz Geçerleme 417
19.6.3 Yerine Geri Koyarak Örnekleme 417
19.7 Sınıflandırma Başarımının Ölçülmesi 418
19.8 Aralık Kestirimi 421
19.9 Denence Sınaması 424
19.10Bir Sınıflandırma Algoritmasının Başarısının Belirlenmesi 426
19.10.1 Binom Sınaması 426
19.10.2 Yaklaşık Normal Sınama 427
19.10.3 t Sınaması 427
19.11İki Algoritmanın Karşılaştırılması 428
19.11.1 McNemar Sınaması 428
19.11.2 K Kat Çapraz Geçerlenmiş Eşli t Sınaması 428
19.11.3 5 × 2 Çapraz Geçerlenmiş Eşli t Sınaması 429
19.11.4 5 × 2 Çapraz Geçerlenmiş Eşli F Sınaması 430
19.12İkiden Çok Algoritmanın Karşılaştırılması:
Değişkenlik Çözümlemesi 431
19.13Birden Çok Veri Kümesinde Karşılaştırma 434
19.13.1 İki Algoritmanın Karşılaştırılması 435
19.13.2 İkiden Çok Algoritmanın Karşılaştırılması 437
19.14Kısa Kısa 438
19.15Alıştırmalar 439
19.16Kaynaklar 439

A Olasılık **441**
A.1 Olasılık Kuramı 441
A.1.1 Olasılığın Belitleri 441
A.1.2 Koşullu Olasılık 442
A.2 Rastsal Değişkenler 443
A.2.1 Olasılık Dağılım, Kütle, ve Yoğunluk İşlevleri 443
A.2.2 Birleşik Dağılım ve Yoğunluk İşlevleri 443
A.2.3 Koşullu Dağılımlar 444

A.2.4 Bayes Kuralı 444
A.2.5 Beklenen Değer 445
A.2.6 Değişinti 445
A.2.7 Büyük Sayıların Zayıf Yasası 446
A.3 Bazı Özel Rastsal Değişkenler 446
A.3.1 Bernoulli Dağılımı 446
A.3.2 Binom Dağılımı 447
A.3.3 Katlıterimli Dağılım 447
A.3.4 Tekdüze Dağılım 447
A.3.5 Normal (Gauss) Dağılım 448
A.3.6 Ki Kare Dağılımı 449
A.3.7 t Dağılımı 449
A.3.8 F Dağılımı 450
A.4 Kaynaklar 450

B İngilizce – Türkçe Sözlük **451**

C Türkçe –İngilizce Sözlük **465**

Dizin **478**

Önsöz

Yapay öğrenme, bilgisayarların örnek veri ya da geçmiş deneyimi kullanarak bir ölçüte göre başarımlarını artıracak biçimde programlanmasıdır. Algoritmasını bilmediğimiz bir uygulama için elimizde örnek veri ya da deneyim varsa bu kitapta göreceğimiz yapay öğrenme yöntemlerinden birini kullanabiliriz. Bir iş için uzman insan azsa, ya da insanlar uzmanlıklarını algoritmaya çevrilebilecek kadar kesin açıklayamıyorsa öğrenen bilgisayar programları faydalı olur. Örneğin konuşma tanıma, yani ses sinyalini bir belgeye dönüştürmek, hepimizin kolayca yapabildiği bir iş olmasına rağmen nasıl yaptığımızı bilmeyiz. Farklı insanlar, yaş, cinsiyet ya da aksan farkından dolayı aynı sözcüğü farklı söyleyebilir. Yapay öğrenmede yaklaşımımız çok sayıda insandan büyük bir veri kümesi toplamak ve seslerle sözcükler arasındaki ilişkiyi öğrenmektir.

Gerçeklenmesi istenen uygulama zaman içinde değişebilir, ya da farklı durumlar için özelleştirilmesi gerekebilir. Olabilecek her koşul için ayrı bir tane geliştirmemiz gerekmesin diye çevresine uyum gösterebilen genel amaçlı dizgeler üretebilmek isteriz. Bir bilgisayar ağı üzerinde veri iletişimini düşünelim. Kaynakla amaç arasında servis kalitesini en iyi yapan yol, zaman içinde ağ trafiğine bağlı olarak değişecektir. Öğrenen bir yönlendirme algoritması trafiği izleyerek kendini ayarlayabilir. Başka bir örnek, kullanıcısının sesine, aksanına, el yazısına, vs. kendini uyarlayabilen bir arayüzdür.

Yapay öğrenmenin başarılı uygulamaları günlük yaşamımıza girmiş durumda. Konuşma ve el yazısı tanımak için geliştirilmiş ticari yazılımlar var. Perakende satış yapan şirketler geçmiş verilerini inceleyip sonuçları müşteri ilişkileri yönetimini iyileştirmede kullanıyor. Finans kurumları geçmiş müşterilerinin davranışlarından öğrendikleriyle yeni müşteriler için kredi risk hesabı yapıyor. Biyoinformatikte her geçen gün hızla çoğalan veriyi ancak bilgisayarlarla inceleyip bilgiye dönüştürebiliyoruz. Bunlar bu kitapta birlikte tartışacağımız uygulamalardan yalnızca birkaçı. Gelecekte neler olabileceğini öngörmek zor. Şoförsüz, kendini kullanan otomobiller, gerçek zamanda bir dilden ötekine çeviri yapan telefonlar, yeni bir ortamda, örneğin farklı bir gezegenin yüzeyinde, dolaşabilen robotlar. Yapay öğrenmenin en zevkli araştırma konulardan biri olduğunu düşünüyorum.

Bu kitapta, istatistik, örüntü tanıma, yapay sinir ağları, yapay zekâ, sinyal işleme, robotik, ve veri madenciliği gibi farklı alanlarla ilişkili birçok yöntemi tartışacağız. Geçmişte bu alanların bazılarında araştırma öteki alanlardan habersiz ve kısmen

farklı vurgularla yapılmıştır. Bu kitaptaki amacımız tüm bunları birleştirmek ve böylece hem problemler, hem de önerilen çözümler için ortak bir yaklaşım getirmek olacak.

Elinizdeki giriş düzeyinde bir ders kitabıdır. Lisans son sınıf ya da yüksek lisans derslerinde okutulabilir. Bu yöntemlerin belli bir uygulamasıyla ilgilenen mühendisler için de yararlı olacağını umuyorum. Önşartları, temel programlama, olasılık, türev, integral, ve doğrusal cebir bilgisidir. Amaç, tüm öğrenme algoritmalarını, denklemlerden programa kolayca geçilebilecek kadar açık anlatmaktır. Bazı algoritmalar için örnek programlar da verilmiştir.

Kitap, bölümler içinden seçim yaparak tek dönemde okutulabileceği gibi iki dönemde de kullanılabilir; dersi veren öğretim üyesi kendi araştırma konularına bağlı olarak bazı bölümleri ek makalelerle daha ayrıntılı işleyebilir—bu durumda bölüm sonundaki kaynaklar yardımcı olacaktır.

Kitabın web sayfası http://www.cmpe.boun.edu.tr/~ethem/i2ml/ adresindedir. Buraya kitap basıldıktan sonra elime geçen ek bilgiyi, örneğin hata listesini, koyuyorum. Bana alpaydin@boun.edu.tr adresinden ulaşabilirsiniz.

Bu kitabı yazmaktan çok zevk aldım; umarım sizler de okumaktan zevk alırsınız.

Teşekkürler

Yetenekli ve eğlenceli insanlarla birlikte çalışmanın insanı isteklendiren bir etkisi var. Boğaziçi Üniversitesi Bilgisayar Mühendisliği Bölümü çalışmak için harika bir yer ve meslektaşlarım bu kitabın oluşma sürecinde bana büyük destek oldular. Bu kitabın içindeki malzemeyi üzerlerinde denediğim geçmiş öğrencilerime de teşekkür ederim.

Bu kitap üzerinde çalışırken Türkiye Bilimler Akademisi Gebip programı kapsamında destek aldım. (EA-TÜBA-GEBİP/2001-1-1).

Michael Jordan'a, hem yıllar içindeki, hem de özelde bu kitap kapsamındaki desteği için çok teşekkür ederim. Kitap hakkındaki genel ve birinci bölüm için ayrıntılı yorumları kitabı hem içerik, hem de biçim açısından iyileştirdi. Taner Bilgiç, Vladimir Cherkassky, Tom Dietterich, Fikret Gürgen, Olcay Taner Yıldız, ve MIT Press'in hakemleri kitabın bazı bölümlerini okuyup çok değerli yorumlarını benimle paylaştılar. Umarım onların yorumlarından doğru dürüst teşekkür etmeden aldığım alıntıları fark ettiklerinde kendilerine ne kadar borçlu hissettiğimi anlarlar. Tabii ki hata ve eksikliklerden yalnızca ben sorumluyum.

Bana hep inanan ve beni hep destekleyen anne ve babama teşekkür ederim. Sema Oktuğ'a hep süren arkadaşlığı ve Hakan Ünlü'ye yıllar içinde, yaşam, evren ve her şey hakkındaki tartışmalarımız için teşekkür ederim.

Kitabın İngilizce kopyası Chris Manning tarafından hazırlanan LaTeX makrolarınca hazırlanmıştır, kendisine teşekkür ederim. Ayrıca seri editörlerine, ve MIT Press'ten Bob Prior, Valerie Geary, Kathleen Caruso, Sharon Deacon Warne, Erica Schultz, ve Emily Gutheinz'a sürekli destek ve yardımları için teşekkür ederim.

İkinci Basım için Önsöz

Kitabın ilk baskısının yayımlandığı 2004 yılından bu yana Yapay Öğrenme alanında çok önemli gelişmeler oldu. Öncelikle uygulama alanları hızla büyüdü. İnternet teknolojileri, örneğin arama motorları, ürün tavsiye eden ya da istenmeyen epostaları süzen yazılımlar, vs. artık yapay öğrenme algoritmaları kullanıyor. Biyoinformatik ve hesaplamalı biyolojide veriden öğrenen yöntemler gittikçe daha yaygınlaşıyor. Doğal dil işleme uygulamalarında, örneğin yapay çeviride, her geçen gün daha sık olarak, programlanmış uzman dizgelerden büyük öğrenme verileri üzerinde eğitilmiş yöntemlere geçildiğini görüyoruz. Robotik, tıbbi tanı, konuşma ve görüntü tanıma, biyometri, finans gibi çok farklı alanlarda, bazen örüntü tanıma, bazen veri madenciliği adı altında, farklı kılıklarının birine bürünmüş olarak, bu kitapta bahsettiğimiz yapay öğrenme yöntemlerinden birine rastlıyoruz.

İkinci olarak, o yıldan bu yana yapay öğrenme kuramında da destekleyen ilerlemeler oldu. Özellikle çekirdek işlevleri ve onları kullanan çekirdek makineleri uygulamaları daha iyi tanımlamayı sağladı ve ilgili dışbükey programlama yöntemleri eğim inişle eğitilen çok katmanlı algılayıcılardan bir adım öteye gitmeyi sağladı. Uygun önsel dağılımları kullanan Bayesçi yöntemler veriye uzman bilgisini eklemeyi kolaylaştırdı. Çizgesel modeller ağ biçiminde bir gösterim ve ağı sorgulamak için etkin algoritmalar kazandırdı. Bu yüzden birinci baskıda bölümler içindeki kısımlarda işlenen bu üç konu, yani, çekirdek makineleri, Bayesçi yöntemler ve çizgesel modeller, bu yeni baskıda üç yeni bölüm olarak daha ayrıntılı işlenmiştir.

Öne çıkan bir başka konu da yapay öğrenme deneylerinin daha iyi tasarlanmasının gerektiğidir. Bu yüzden bu ikinci baskıda istatistiksel sınama üzerine olan bölümü, yapay öğrenme deneylerinin tasarlanması ve çözümlenmesini de içerecek biçimde yeniden yazdım. Önemli olan nokta, sınamanın tüm programlar çalışıp bittikten sonra yapılması yerine (her ne kadar bu bölüm de kitabın en sonunda olsa da) deneyin en baştan tasarlanması, önemli etkenlerin belirlenmesi, uygun yordamların kararlaştırılması ve ancak ondan sonra deneylerin yapılması ve sonuçların çözümlenmesinin gerektiğidir.

Uzun yıllar boyunca çoğu araştırmacı, özellikle de bilim camiasının kıdemli üyeleri, bilgisayarların bizim kadar zeki olabilmesi, yani yapay zekânın gerçeklenebilmesi için şu anki genel bilimsel yetkinliğimizin, özelde de bilgisayar mühendisliği bilgimi-

zin, yeterli olmadığını savundu. Birçok araştırmacı, yepyeni bir teknoloji, yeni bir malzeme, yeni bir hesaplama kuramı, ya de yeni bir programlama yaklaşımı gerektiğini, ve o elimize geçene dek zekânın bazı kısımlarının ancak "benzetiminin" yapılabileceğini ama tamamiyle gerçeklenemeyeceğini düşünüyordu.

Yakında onları haksız çıkaracağımızı sanıyorum. Bunu önce satrançta gördük, bu son yıllarda başka uygulamalarda da görüyoruz. Yeterince bellek ve hesaplama gücü olduğunda birçok zor görünen işin oldukça basit algoritmalarla gerçeklenebildiği ortaya çıkıyor. Burada can alıcı nokta öğrenme, ya örnek veriden öğrenme, ya da deneme yanılmayla pekiştirmeli öğrenme. Görünen o ki gözetimli, ve büyük oranda gözetimsiz, öğrenmeyle, örneğin, yapay çeviri, çok yakında çözülmüş olacak. Aynısı birçok alan, örneğin robotikte yol bulma için, de geçerli. Bunun yapay zekâdaki birçok alan için süreceğine inanıyorum. Burada anahtar, öğrenme yeteneği. Yeterince veri (ille de gözetimli olmayan) ve o veriyi işleyebilecek hesap gücünü sağlayabilirsek bilgisayarlar öğrenebilecek ve onlar için program yazmaya uğraşmamız gerekmeyecek.

Birinci baskıyı kullanan tüm öğretim üyesi ve öğrencilere teşekkür ederim. Dünyanın dört yanından, Hindistan'daki tıpkıbasımı ve Almanca çevirisi de dahil olmak üzere, bana yazan ve geribesleme ileten herkese minnettarım. Lütfen epostalarınız durmasın. Adresim `alpaydin@boun.edu.tr`.

İkinci baskı için de Web üzerinde destek var. Adres
`http://www.cmpe.boun.edu.tr/~ethem/i2ml`.

Geçmiş ve şu anki tez öğrencilerime, Mehmet Gönen, Esma Kılıç, Murat Semerci, M. Aydın Ulaş, ve Olcay Taner Yıldız'a, ayrıca son birkaç yılda benden CmpE 544, CmpE 545, CmpE 591, ve CmpE 58E alan öğrencilere teşekkür ederim. İnsanın bir konudaki bilgisini sınamasının en iyi yolu onu öğretmeye çalışmak.

Bu ikinci baskıda da MIT Press'le çalışmak bir zevkti. Bob Prior, Ada Brunstein, Erin K. Shoudy, Kathleen Caruso, ve Marcy Ross'a tüm yardım ve destekleri için teşekkür ederim.

Türkçe Çeviri için Önsöz

Kafamda bu kitabı yazma düşüncesi oluşalı beri Türkçesini de yayımlayabilmeyi ümit ediyordum. 2007 yılı başında Boğaziçi Üniversitesi Yayınevi kitabın Türkçe çevirisi için MIT Press'le anlaştı. Sonrasında MIT Press'ten kitabın ikinci baskısını yayımlama önerisi gelince çeviriyi ertelemek zorunda kaldım. Kitabın Almanca çevirisi 2008 yılında Oldenbourg Verlag, Çince çevirisi de 2009 yılında Huazhang Press tarafından basıldı. Bu elinizdeki kitap, 2010 yılında yayımlanan ikinci baskının Türkçe çevirisidir.

Kitabı Türkçeye çevirirken beni en çok zorlayan tabii ki karşılık geçen Türkçe sözcükleri ve kalıpları bulmak oldu. Kullanılan sözcükleri içeren Türkçe-İngilizce ve İngilizce-Türkçe sözlükler kitabın sonunda bulunmaktadır.

Kitabı Türkçeye Olcay Kurşun çevirecekti, hatta sözlüğün ve bazı bölümlerin ilk taslağını da hazırladı, ama sonrasında hem ailevi, hem de profesyonel zorunluluklar yüzünde vazgeçmek zorunda kaldı. Olcay'a bu ilk çabaları için teşekkür ederim.

Boğaziçi Üniversitesi Yayınevi Yayın Kurulu'na kitabın Türkçe çevirisini basmaya layık gördükleri için, her aşamada desteğinden dolayı Yayın Kurulu Başkanı Murat Gülsoy'a ve basım sırasındaki desteği için Yayınevi Müdürü Ergun Kocabıyık'a çok teşekkür ederim.

Gösterim

x	Sayıl değer
$\boldsymbol{x}$	Yöney
$\mathbf{X}$	Dizey
$\boldsymbol{x}^T$	Devrik
$\mathbf{X}^{-1}$	Ters
X	Rastsal değişken
$P(X)$	Olasılık kütle işlevi (X kesikli)
$p(X)$	Olasılık yoğunluk işlevi (X sürekli)
$P(X\|Y)$	Y biliniyorken X rastsal değişkeninin olasılığı
$E[X]$	X rastsal değişkeninin beklenen değeri
$\text{Var}(X)$	X rastsal değişkeninin değişintisi
$\text{Cov}(X,Y)$	X ve Y rastsal değişkenlerinin eşdeğişintisi
$\text{Corr}(X,Y)$	X ve Y rastsal değişkenlerinin ilintisi
μ	Ortalama
σ^2	Değişinti
$\boldsymbol{\Sigma}$	Değişinti dizeyi
m	Ortalama için kestirici
s^2	Değişinti için kestirici
$\mathbf{S}$	Değişinti dizeyi için kestirici
$\mathcal{N}(\mu,\sigma^2)$	Ortalaması μ ve değişintisi σ^2 olan normal (Gauss) dağılım
$\mathcal{Z}$	Birim normal dağılım: $\mathcal{N}(0,1)$
$\mathcal{N}_d(\boldsymbol{\mu},\boldsymbol{\Sigma})$	Ortalama yöneyi $\boldsymbol{\mu}$ ve eşdeğişinti dizeyi $\boldsymbol{\Sigma}$ olan d boyutlu normal
$\boldsymbol{x}$	Girdi
d	Girdi (boyut) sayısı
y	Çıktı
r	İstenen çıktı
K	Çıktı (sınıf) sayısı
N	Öğrenme örneği sayısı
z	Saklı değer/boyut/etken

k	Saklı değer/boyut/etken sayısı
C_i	Sınıf i
$\mathcal{X}$	Öğrenme örneklemi
$\{x^t\}_{t=1}^N$	t dizini 1 ile N arasında değişen x^t örnekleri
$\{x^t, r^t\}_t$	t dizin değişkenli sıralı girdi ve istenen çıktı çiftleri
$g(x\|\theta)$	Girdisi x, parametresi θ olan işlev
$\arg\max_\theta g(x\|\theta)$	g işlevinin x için değerini en küçük yapan θ değeri
$\arg\min_\theta g(x\|\theta)$	g işlevinin x için değerini en büyük yapan θ değeri
$E(\theta\|\mathcal{X})$	$\mathcal{X}$ örneklemi üzerinde θ parametresiyle yapılan hata
$l(\theta\|\mathcal{X})$	$\mathcal{X}$ örneklemi üzerinde θ parametresinin olabilirliği
$\mathcal{L}(\theta\|\mathcal{X})$	$\mathcal{X}$ örneklemi üzerinde θ parametresinin log olabilirliği
$1(c)$	c doğruysa 1, değilse 0
$\#\{c\}$	c bildirimini doğru yapan örnek sayısı
δ_{ij}	Kronecker delta: $i = j$ ise 1, değilse 0

Bölüm 1

Giriş

1.1 Yapay Öğrenme Nedir?

Bir işi bilgisayarda gerçekleyebilmek için algoritmasını bilmek gerekir. Algoritma, girdiyi çıktıya çevirmek için uygulanacak komutlar dizisidir. Örneğin, sıralama yapacak bir algoritmanın girdisi bir sayılar kümesi, çıktısı da bu sayıların artan biçimde sıralandığı bir dizi olacaktır. Bu iş için çeşitli algoritmalar arasından en iyisini, yani en az sayıda komut ve/ya da en az bellek kullananı bulmak isteyebiliriz.

Ancak bazı işler için bilinen bir algoritma yoktur. Örneğin, istenmeyen elektronik postaların ayıklanmasında en basit durumda bir yazı dosyası olan bir elektronik posta iletisi girdidir, çıktıysa bu postanın istenip istenmediğini belirten bir evet ya da hayır işaretidir. Amaç, girdiye bağlı olarak çıktının belirlenmesidir, ama bunun için bilinen bir algoritma yoktur. Üstelik neyin istenmeyen olduğu zaman içinde ve kişiden kişiye değişebilir.

Bilgimiz eksikse de verimiz bol olabilir. Kolayca, hem istenen hem de istenmeyen binlerce örnek elektronik postayı içeren bir veri kümesi derleyebiliriz. Amaç, bu verideki örnekleri inceleyerek bir elektronik postayı neyin istenmeyen yaptığını bunlardan "öğrenebilmektir." Başka bir deyişle, bilgisayarın bu iş için kendi başına bir algoritma oluşturabilmesini isteriz. Sayıları sıralamak için bir algoritma öğrenmeye gerek yoktur, onun algoritmasını zaten biliyoruz, ama algoritması olmayıp örnek verisi olan birçok uygulamada böylesi bir yapay öğrenme yaklaşımı faydalı olacaktır.

Bilişim teknolojilerindeki gelişmeler sayesinde büyük miktarlarda veriyi saklayabiliyor, işleyebiliyor ve bu veriye ağ üzerinden fiziksel olarak çok uzak yerlerden erişebiliyoruz. Artık birçok veri toplama aleti sayısal olduğundan verileri güvenilir şekilde kaydedip çevrimiçi olarak ulaşma olanağı sağlıyor. Örneğin, ülkede yüzlerce mağazasında her gün milyonlarca müşteriye binlerce ürün satan bir süpermarket zincirini ele alalım. Satış kasaları her işlemi, tarih, müşteri numarası, alınan ürünler ve miktarları, toplam harcama, vs. gibi tüm ayrıntılarıyla kaydeder. Bu her gün gigabaytlarca veri demektir. Süpermarket zinciri bu veriyi inceleyerek müşteri dav-

ranışlarıyla ilgili bilgi çıkarmak ister, örneğin, bir ürün için olası müşteri profilini bulmak isteyebilir. Ama bunun algoritması da bilinmez; zamana ve coğrafi konuma göre değişir. Saklanan bunca veri, ancak çözümlenip gelecekle ilgili öngörü yapmaya yarayacak bilgiye dönüştürüldüğünde işe yarar.

Kimlerin hangi dondurma çeşidini ya da belli bir yazarın bir sonraki kitabını alıp almayacağını, kimlerin hangi yeni filmi izleyeceğini, hangi şehri ziyaret edeceğini, ya da hangi linki tıklayacağını tam olarak bilemeyiz. Zaten bilseydik veriyi incelemeye gerek duymazdık, oturup algoritmasını yazardık. Ama bilmediğimiz için yapabileceğimiz tek şey veri toplamak ve bu tür soruların yanıtlarını bu veriden çıkartmaya çalışmaktır.

Gözlediğimiz verileri açıklayan bir süreç olduğuna inanırız. Bu süreci, örneğin müşterilerin alışveriş davranışını, tüm ayrıntılarıyla bilmesek de rastgele olmadığını biliriz. İnsanlar süpermarkette sepetlerini gelişigüzel doldurmazlar. Bira aldıklarında yanında cips alırlar; yazın dondurma yer, kışın salep içerler. Veride bir düzen vardır.

Süreci net olarak tanımlayamasak da *iyi ve faydalı, yaklaşık bir açıklamasını* oluşturabileceğimize inanırız. Bu yaklaşık açıklama, hepsi için olmasa da verinin önemli bir bölümü için yeterli olabilir. Tam olarak süreci tanımlamak olası olmasa da verideki önemli örüntü ve düzenlilikleri sezimleyebileceğimizi düşünürüz. İşte yapay öğrenmenin alanı budur. Böylesi bir düzen süreci anlamamıza yardımcı olabileceği gibi bunu kullanarak öngörü de yapabiliriz: Geleceğin, en azından yakın geleceğin, verinin toplandığı geçmişten çok farklı olmadığını varsayarsak bu veriden oluşturulmuş kurallarla yapacağımız öngörüler de doğru olacaktır.

Yapay öğrenmenin büyük veritabanlarına uygulanmasına *veri madenciliği* denir. Bir madenden çıkarılan çok miktardaki toprak ve ham maddenin işlenmesiyle az miktarda ama değerli maden elde edilmesi gibi, veri madenciliğinde de çok miktarda veri işlenerek işe yarayan, örneğin yüksek öngörü başarısı olan, basit bir model oluşturulur. Birçok uygulama alanı vardır: Perakende satıştan başka finansta, kredi başvurularında, sahtekarlık sezimlinde, borsada kullanılır. Üretim sektöründe, öğrenilen modeller eniyileme, denetim, ve arıza bulmada, tıptaysa tanı koymada kullanılır. Haberleşmede, bağlantı örüntüleri çözümlenerek ağ iyileştirmesi ve hizmet kalitesini artırma amacıyla çalışmalar yapılmaktadır. Bilimde, fizik, astronomi ve biyolojide gittikçe daha büyük veri tabanları oluşmaktadır ve bunlar ancak bilgisayarlarla hızla incelenebilip bilgiye dönüştürülebilir. İnternet sürekli büyümektedir ve istediğimiz bilgiyi bulmak zeki arama motorları olmadan yapılamaz.

Yapay öğrenme yalnızca bir veritabanı uygulaması değil, yapay zekânın da bir parçasıdır. Değişen bir ortamdaki etmenin zeki olarak nitelendirilebilmesi için deneyimlerinden öğrenebilmesi gerekir. Eğer etmen öğrenerek ortamdaki değişikliklere uyum gösterebiliyorsa, dizgenin tasarlayıcısının olası her durumu öngörüp bunlara göre etmeni programlamasına gerek kalmaz.

Yapay öğrenme, bilgisayarla görü, konuşma tanıma, ve robotik gibi alanlarda da çözümler bulmamıza yardımcı olur. Yüz tanıma örneğini alalım: Her gün aile bireylerini ve arkadaşlarımızı, bakış açısı, aydınlatma, ya da saç şekillerindeki farklara rağmen, ya da fotoğraflarından kolaylıkla tanırız. Ama bunu bilinçsizce yaparız ve nasıl yaptığımızı açıklayamayız. Açıklayamadığımız için algoritmasını yazamayız.

Bununla birlikte, bir yüz imgesinin rastgele renkli noktalardan oluşmadığını biliriz; yüzün bir düzeni, bir yapısı vardır. Örneğin, yüzün sağı ve solu bakışımlıdır. Gözler, burun, ve ağzın yüzdeki yerleri bellidir. Her insanın yüzü bunların birleşiminden oluşan bir örüntüdür. Öğrenen bir program, bir insanın yüz resimlerini inceleyerek o kişiye özgü örüntüyü yakalayabilir ve sonra verilen bir resmin bu örüntüye ne kadar uyduğuna bakarak kişiyi tanıyabilir. Buna *örüntü tanıma* denir.

Yapay öğrenme, bilgisayarların örnek veri ya da geçmiş deneyimi kullanarak başarımlarını artıracak biçimde programlanmasıdır. Elimizde bazı parametrelere bağlı olarak tanımlanmış bir model, ve veri ya da geçmiş deneyim üzerinde modelin başarımını ölçmek için bir ölçüt tanımlıdır. Amaç, modelin parametrelerini bu başarım ölçütüne göre en iyi yapan parametre değerlerini bulmaktır. Model, gelecekle ilgili öngörüler yapmak için kullanılabilen *öngörücü* ve/ya da veriden bilgi çıkarmaya yönelik *açıklayıcı* bir model olabilir.

Yapay öğrenme, model oluşturmada istatistik kuramını kullanır, çünkü temeli gözlenmiş bir örneklemden çıkarım yapmaktır. Bilgisayar bilimlerinin göreviyse iki türlüdür: Öncelikle, öğrenme altında yatan iyileme probleminin genelde çok büyük miktarda olan veri üzerinde etkin çözülebilmesi gerekir. İkincisi, öğrenilmiş bir modelin bellekteki gösterimi ve onu kullanarak çıkarım yapmak da etkin olmalıdır. Bazı uygulamalarda, öğrenme ya da çıkarım algoritmasının etkinliği, yani bellek ve zaman karmaşıklığı, öngörü başarısı kadar önemli olabilir.

Şimdi gelin, yapay öğrenmenin çeşitleri ve kullanım alanlarını daha iyi kavrayabilmek için bazı örnek uygulamalara bakalım.

1.2 Yapay Öğrenme Uygulamalarına Örnekler

1.2.1 İlişkilendirme Kuralları

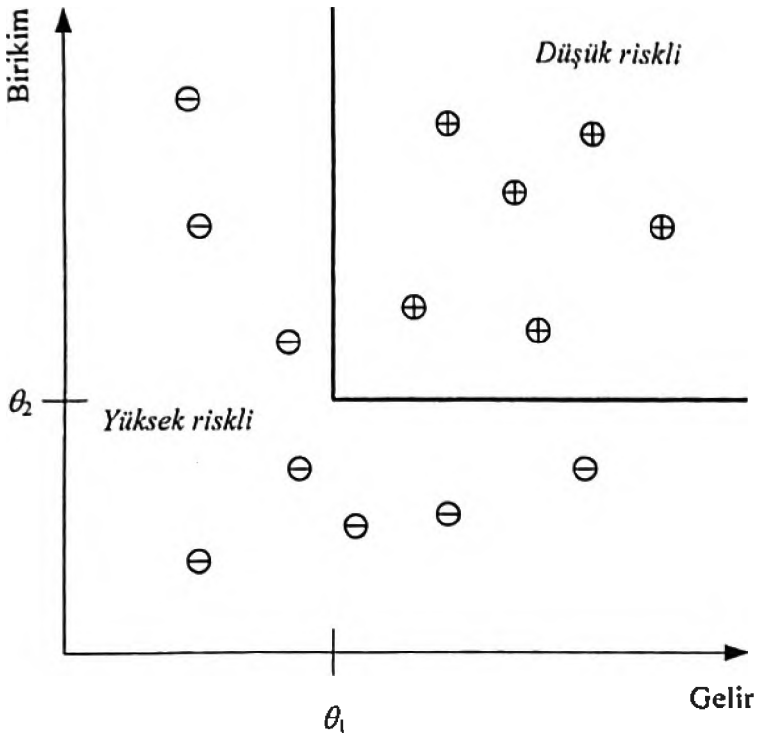
Perakende satışta, örneğin bir süpermarkette, yapay öğrenmenin bir uygulaması *sepet çözümlemesidir*: Eğer X malını alan müşteriler sıklıkla Y malını da alıyorsa ve bir müşteri X malını almışsa, o Y malını almaya aday bir müşteridir. Buna iş dünyasında *çapraz satış* denir.

İlişkilendirme kuralı bulurken müşterinin geçmişte satın almış olduğu ürün ya da ürünleri içeren X kümesine koşullu olarak Y malı için bir $P(Y|X)$ olasılığı öğrenmeyi amaçlıyoruz.

Örneğin, elimizdeki veriden $P(\text{cips}|\text{bira}) = 0.7$ bulduğumuzu düşünelim. Bu durumda,

Bira satın alan müşterilerin yüzde 70'i cips de alır,

biçiminde bir kural çıkarabiliriz. Müşteriler arasında, cinsiyet, yaş, medeni durum gibi özelliklere bağlı olarak ayrım yapmak ve D müşterinin demografik özelliklerini gösterirken, $P(Y|X, D)$ olasılığını kestirmek isteyebiliriz. Mallar, bir kitapçı için kitaplar ya da yazarlar olabileceği gibi bir web portalinde linkler olabilir.



Şekil 1.1: Örnek bir öğrenme kümesi. Her çember, eksenlerde karşılık gelen girdi değerlerine sahip bir veri örneğini, işareti de sınıfını gösterir. Basit olması için yalnızca iki müşteri özelliği, gelir ve birikim, girdi olarak alınmıştır. İki sınıf, düşük riskli ('+') ve yüksek riskli ('-') müşterilere karşılık gelir. Sınıfları ayıran örnek bir ayırtaç da gösterilmiştir.

1.2.2 Sınıflandırma

Kredi, bir finans kurumu, örneğin, bir banka, tarafından faiziyle birlikte, genelde taksitler halinde, geriye ödenmek üzere borç verilen paradır. Bu borçla ilintili riski, yani müşterinin borcunu geriye ödeyememe olasılığını öngörebilmek banka için önemlidir. Bu hem bankanın kâr etmesi, hem de müşteriyi mali gücünün üzerinde bir borçla sıkıntıya sokmamak için gereklidir.

Kredi derecelendirmesinde (Hand, 1998) banka, kredi miktarı ve müşterinin bilgilerini kullanarak bir risk hesaplar. Müşterinin gelir düzeyi, birikimleri, teminatları, mesleği, yaşı, ve kredi geçmişi gibi gözleyebildiğimiz ve müşterinin mali gücüyle ilgili bilgi verdiğine inandığımız veriler kullanılır. Banka geçmişte verdiği krediler için bu tür müşteri bilgilerini ve kredilerin nasıl sonuçlandığını bilir; amaç, bu kayıtlardaki verileri kullanarak müşteri özellikleriyle kredi riski arasındaki ilişkiyi açıklayan kurallar çıkarmaktır; buna *skor kartı* denir. Yani, yapay öğrenme yapan yazılım geçmiş verileri kullanarak bir model öğrenir ve ardında yeni bir başvuru bu modele göre kabul ya da reddedilir.

Bu iki sınıflı bir *sınıflandırma* problemidir; sınıflar, düşük riskli ve yüksek riskli müşterilerdir. Müşterinin bilgileri modelin *girdisi* olur ve buna bağlı olarak iki sınıftan biri seçilir.

Şekil 1.1'de örnek bir veri kümesi ve sınıfların nasıl ayrıldığını görüyoruz. Geçmiş verilerle eğitimden sonra öğrenilen bir sınıflandırma kuralı, uygun θ_1 ve θ_2 eşik değerleriyle,

EĞER gelir> θ_1 VE birikimler> θ_2 İSE düşük riskli DEĞİLSE yüksek riskli

biçiminde yazılabilir. Bu kural, örnek bir *ayırtaçtır*; farklı sınıfları birbirinden ayıran bir işlevdir.

Böyle bir kuralı bir kez oluşturunca artık *öngörü* yapabiliriz: Geçmiş verilere uyan bir kuralımız varsa ve geleceğin de geçmişe benzediğine biliyorsak, yeni örnekler için doğru öngörüler yapabiliriz. Gelirini ve birikimini bildiğimiz yeni bir müşterinin başvurusu için bu kuralla düşük ya da yüksek riskli kararını kolayca verebiliriz.

Bazı durumlarda, 0/1 (düşük/yüksek riskli) biçiminde bir karar yerine bir olasılık hesaplamak isteyebiliriz. X müşteri özelliklerini, Y de düşük riskli ve yüksek riskli sınıflarına karşılık gelen 0 ya da 1 değerlerini alıyor olsun, bu durumda istediğimiz $P(Y|X)$ koşullu olasılığını hesaplamaktır. Bu bakış açısıyla sınıflandırmayı X ile Y arasında bir birliktelik kuralı öğrenmek olarak da düşünebiliriz. Böylelikle, verilen bir $X = x$ için, örneğin $P(Y = 1|X = x) = 0.8$ ise müşterinin yüzde 80 olasılıkla yüksek riskli, ya da bir başka deyişle, yüzde 20 olasılıkla düşük riskli olduğunu söyleyebiliriz. Ardından, her iki durumdaki olası kâr ve zarar miktarlarına göre kredi başvurusu için kabul ya da ret kararı verebiliriz.

Yapay öğrenmenin *örüntü tanımada* birçok uygulaması vardır. Harfleri resimlerinden tanımayı hedef alan *görsel karakter tanımada*, tanımak istediğimiz karakter kadar sayıda sınıf vardır. Örneğin zarflardaki posta kodlarında ya da çeklerdeki miktarda olduğu gibi karakterler elle yazıldığında bu daha zor bir sınıflandırma problemi olur. İnsanların el yazıları farklı farklıdır; karakterler küçük ya da büyük, eğik, kurşun ya da tükenmez kalemle yazılabilirler, ve aynı karakter için birçok farklı görüntü olabilir. Yazı insan icadı olmasına rağmen, bir insan kadar hatasız okuyan bir yazılıma henüz sahip değiliz. Elimizde bütün 'A'ları kapsayan, ve bütün 'A' olmayanları dışarıda bırakan bir tanım yoktur. Olmadığı için de yapabileceğimiz tek şey, yazarlardan örnekler almak ve tanımı bu örneklerden öğrenmeye çalışmaktır. Her ne kadar bir görüntüyü 'A' yapanın ne olduğunu bilmesek de, bütün farklı 'A' görüntülerinde ortak bir A-lık olduğunu biliriz ve elimizdeki görüntülerden çıkarmayı istediğimiz de bu A-lık tanımdır. Bir karakter imgesi rastgele noktalardan oluşmaz; çizgilerden ve onların özel bir biçimde biraraya gelmesinden oluşur ve öğrenen bir program tarafından yakalanabileceğine inandığımız bir düzeni vardır.

Bir belge okurken dildeki bağlamdan da faydalanabiliriz. Bir sözcük, bir *dizi* harften oluşur ve ardışık harfler bağımsız değil, dildeki sözcüklerle kısıtlanmıştır. Bu bağımlılık yardımıyla bir harfini tanıyamasak da sö?cüğü hâlâ okuyabiliriz. Bu gibi bağlamsal bağımlılıklar daha üst düzeylerde, dilin sözdizimsel ya da anlambilimsel kurallarına göre, sözcükler ve cümleler arasında da geçerli olabilir. Dizileri öğrenen ve bu tür bağımlılıkları modellemede kullanılabilecek yapay öğrenme algoritmalarını bu kitabın ileriki bölümlerinde göreceğiz.

Yüz tanıma örneğinde girdi bir yüz imgesi, sınıflar ise tanınması istenen insanlardır, ve amaç, yüz imgelerini kimliklerine eşlemeyi öğrenmektir. Bu, karak-

ter tanımadan daha zordur, çünkü hem daha çok sınıf vardır, hem de imge daha karmaşıktır. Yüz imgesi daha büyüktür; yüz üç boyutludur ve duruş ya da ışıktaki küçük değişiklikler imgede büyük farklılıklar oluşturabilir. Yüzün bazı parçaları örtülebilir, örneğin, gözlükler göz ve kaşları, sakal çeneyi saklayabilir.

Tıbbi tanıda girdi, hasta hakkında bildiğimiz tüm ilgili bilgiler, sınıflar ise hastalıklardır. Hastanın yaşı, cinsiyeti, geçmiş tıbbi kayıtları, ve şimdiki bulguları girdidir. Bazı tahlillerin hastaya henüz uygulanmamış olması durumunda o girdiler eksik olacaktır. Tahliller zaman alır, pahalıdır, ve hastaya rahatsızlık verebilir, dolayısıyla faydalı bilgi vereceğine emin olmadan tahlil yapmak istemeyiz. Yanlış tanı, hatalı, eksik, ya da gereksiz tedaviye yol açar ve bu yüzden şüpheli durumlarda sınıflandırıcının bir sınıfı seçmeyi reddedip kararı insana bırakması yeğlenir.

Konuşma tanımada girdi ses, sınıflar ise söylenebilecek sözcüklerdir. Yaş, cinsiyet, ya da aksan yüzünden insanlar aynı sözcüğü farklı söyler. Başka bir özellik de girdinin zamansal olmasıdır; bir sözcük, zaman içinde sıralanmış temel ses birimlerinin dizisidir ve farklı sözcükler farklı uzunlukta olabilir.

Ses bilgisi bize bir noktaya kadar yardımcı olabilir, ama görsel karakter tanımada olduğu gibi bir "dil modeli"nin katılması çok faydalıdır. Bunu yapmanın en iyi yolu da böyle bir modeli yine büyük bir örnek veri kümesinden öğrenmektir. Yapay öğrenmenin *doğal dil işlemedeki* uygulamaları sürekli artmaktadır. *İstenmeyen elektronik postaların süzülmesi* bunlardan birisidir; bir yandan üreteçler, öbür yandan süzgeçler birbirini alt etmek için gitgide daha usta yollar bulmaktadırlar. Belki en etkileyici olan doğal dil işleme uygulamasıysa *bilgisayarla çeviridir*. Onlarca yıl elle tasarlanmış çeviri kuralları üzerinde çalıştıktan sonra son zamanlarda bu konuda öğrenen yazılımlar gittikçe daha yaygınlaşmıştır. Bir programın çevrilmiş çok sayıda belgeyi inceleyip bir dildeki bir cümleyi başka bir dildeki bir cümleye çeviren kuralları kendisinin örneklerden çıkarması en ümit veren yol görünmektedir.

Biyometri, insanların bedensel ve/ya da davranışsal özelliklerinden tanınması ya da kimliklerinin doğrulanmasıdır. Değişik kiplerdeki verilerin birleştirilmesi kararların doğruluğunu artırır. Yüz, parmak izi, iris, el ayası resimleri gibi bedensel özellikler yanında, imza, ses, yürüyüş, tuş vuruşları gibi davranışsal özellikler de kullanılabilir. Fotoğraf, imza, ya da şifre gibi alışılagelmiş kimlik doğrulama yordamlarının aksine, birçok farklı tür (ilintisiz) girdi birleştirildiğinden aldatma daha zor olacak, tanıma daha güvenilir yapılacaktır. Burada tabii ki bir ister, kullanıcıya ek güçlük çıkarmamaktır. Yapay öğrenme, farklı kipler için tasarlanmış ayrı ayrı tanıyıcılarda da, kiplerin güvenilirlik oranlarını dikkate alarak kararların birleştirilmesinde de kullanılmaktadır.

Veriden kural öğrenme *bilgi çıkarımı* sağlar. Kural, veriyi açıklayan basit bir modeldir ve bu modele bakarak verinin altında yatan süreç hakkında bir açıklama elde ederiz. Örneğin, düşük ve yüksek riskli müşterileri için bir ayırtaç öğrendiğimizde, düşük riskli müşterileri yüksek risklilerden ayıran özellikler hakkında bilgi sahibi oluruz. Sonra bu bilgiyi düşük riskli müşterileri hedefleyen kampanyalar ya da onlar için özel reklam kanalları tasarlamak için kullanabiliriz.

Veriden kural çıkardığımızda elimize daha basit bir açıklama geçer ve bu bağlamda, öğrenmenin bir tür *sıkıştırma* yaptığını söyleyebiliriz. Modeli saklamak ve işlemek,

tüm veriyi saklamak ve işlemeye göre daha ucuzdur. Toplamanın nasıl yapıldığını bir kez öğrenirseniz, sonra olası her sayı çiftinin toplamını akılda tutmanız gerekmez.

Yapay öğrenmenin bir başka kullanımı da kurala uymayan ve ender rastlanan *aykırılıkların* bulunmasıdır. Böyle bir uygulamada veriden bir kural öğrendikten sonra kuralla değil, kurala uymayan aykırı örneklerle ilgileniriz. Bunlar da sahtekârlık gibi olağandışı durumlara işaret edebilir.

1.2.3 Bağlanım

İkinci el araba fiyatlarını öngören bir yazılım tasarladığımızı düşünelim. Arabanın fiyatını etkileyebilecek özellikler, örneğin, markası, yılı, motor hacmi, kaç kilometre yaptığı, sahibinin mesleği, vs. yazılımın girdisidir. Çıktı da arabanın fiyatıdır. Bunun gibi çıktının bir sayı olduğu uygulamalara *bağlanım* uygulamaları denir.

Arabanın özelliklerini X, fiyatını Y ile gösterelim. Geçmişte alınıp satılmış arabalardan bir öğrenme veri kümesi oluşturup bu veriyi kullanarak Y değerini X değerlerinin bir işlevi olarak öğrenmek isteriz. Bir örnek işlev şekil 1.2'de verilmiştir. Bu doğrusal model, uygun w ve w_0 değerleri için

$$y = wx + w_0$$

biçiminde yazılır; bu örnekte tek girdi vardır,o da arabanın kaç kilometre yaptığıdır.

Hem bağlanım, hem de sınıflandırma, X değişkeninin girdi, Y değişkeninin çıktı olduğu ve amacın girdiden çıktıya bir eşleme öğrenmek olduğu *gözetimli öğrenme* uygulamalarıdır. Yapay öğrenmede çıktıyı, girdiyi kullanan ve bazı parametrelere göre tanımlı bir model ile yaklaşık olarak kestirmeyi amaçlarız:

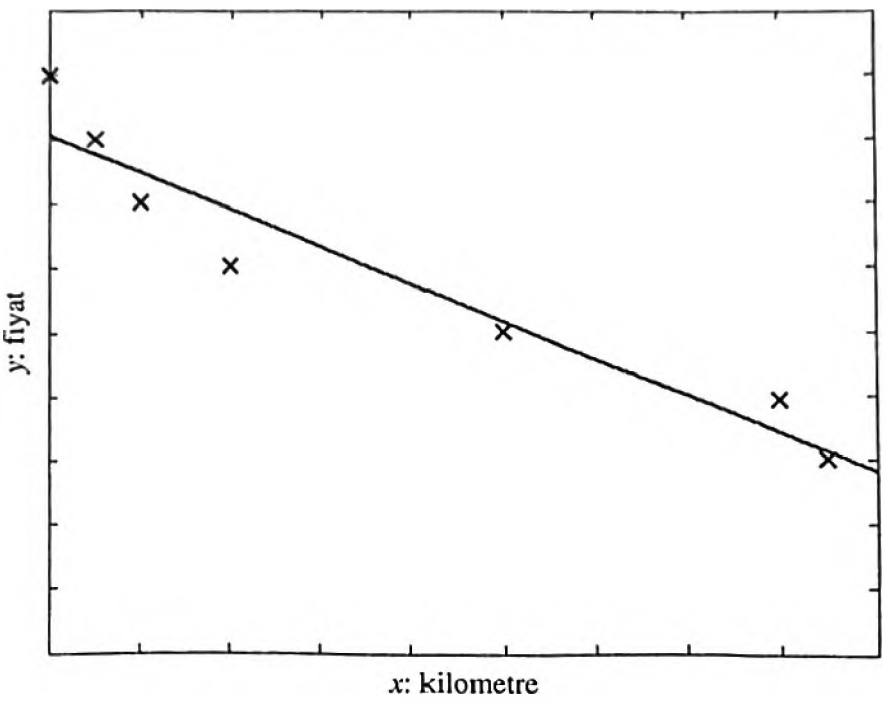
$$y = g(x|\theta)$$

Model $g(\cdot)$, parametreleri de θ ile gösterilir. Y ile gösterdiğimiz çıktı, bağlanım uygulamasında bir sayı, sınıflandırmadaysa 0 ya da 1 değerini alan bir sınıf etiketidir. $g(\cdot)$ ile gösterdiğimiz model, bağlanım işlevi, ya da sınıflandırmada farklı sınıfları ayıran ayırtaç işlevidir. Yapay öğrenme programı, kestirim hatasını, yani öğrenme kümesindeki girdilerde modelin öngörüsüyle gerçek çıktılar arasındaki farkı, en aza indirecek θ parametre değerlerini arar. Örneğin, şekil 1.2'de model doğrusaldır, ve w, w_0 parametreleri öğrenim verisine en yakın geçecek biçimde bulunur. Doğrusal modelin çok sınırlayıcı kaldığı durumlarda ikinci

$$y = w_2x^2 + w_1x + w_0$$

ya da daha yüksek dereceden bir çokterimli, ya da doğrusal olmayan başka bir işlev kullanılarak parametreleri veriye uydurulabilir.

Bağlanımın bir başka uygulaması, hareket edebilen bir robotun, örneğin özerk bir arabanın, engellere çarpmadan ve rotadan ayrılmadan yönetilebilmesidir. Burada bağlanım çıktısı, o anda direksiyonun hangi yöne çevrilmesi gerektiğidir. Bu tür bir problemde girdiler, araba üzerindeki, örneğin kamera, GPS, ve başka alıcılardan



Şekil 1.2: İkinci el arabaların fiyatları için örnek bir veri ve yaklaşık bir model. Basitlik için tek girdi olarak arabanın kaç kilometre yaptığı alınmış ve doğrusal bir model varsayılmıştır.

gelir. Öğrenim verisi insan sürücülerin hareketlerini gözleyip o anki girdilerle birlikte kaydederek oluşturulabilir.

Bir sürecin çıktısını en iyi yapan girdi değerlerini bulmak istediğimizi düşünelim; bu da bir bağlanım örneğidir[1]. Diyelim ki bir kahve makinesi yapmak istiyoruz. Kaliteyi etkileyen birçok girdi vardır: Sıcaklık, pişirme süresi, kahve çekirdek türü, vs. Bunların farklı değerleri için denemeler yapıp her biri için çıktının kalitesini, örneğin müşteri memnuniyetini ölçer, ardından, bu girdileri kahve kalitesine bağlayan bir bağlanım modeli oluşturur ve yeni değerleri bu modelin en yüksek değerleri çevresinde seçeriz. Bu değerleri uygulayıp kaliteyi ölçer, bu noktayı veriye ekler ve yeni bir model çatarız. Buna *tepki yüzeyi tasarımı* denir.

1.2.4 Gözetimsiz Öğrenme

Gözetimli öğrenmede amaç, girdiyle, doğru değeri bir gözetmen tarafından verilen çıktı arasında bir eşleme öğrenmektir. Gözetimsiz öğrenmede böyle bir gözetmen yoktur ve elimizde sadece girdi verisi vardır; amaç, girdideki düzenlilikleri bulmaktır. Girdi uzayının bir yapısı vardır, bazı örneklere daha sık rastlanır; istediğimiz neyin sık, neyin seyrek gerçekleştiğini görmektir. İstatistikte buna *dağılım kestirimi* denir.

Dağılım kestirimi için bir yöntem, *öbeklemedir*. Bir şirket, geçmiş müşterilerinin hareketleri yanında onların demografik bilgilerini saklayarak müşterilerinin profille-

[1]Bu örnek için Michael Jordan'a teşekkür ederim.

rini, yani ne tür müşterilerin hangi yoğunlukta olduğunu görmek isteyebilir. Bu durumda bir öbekleme algoritması, birbirine benzeyen müşterileri aynı öbeklere atar; buna *müşteri bölütleme* denir. Bu öbekler bulunduktan sonra şirket her öbek için farklı servis ve ürünlere karar verebilir; buna *müşteri ilişkileri yönetimi* denir. Böyle bir öbekleme, aykırı örnekleri, yani ötekilerden farklı müşterileri bulmak için de kullanılabilir; böyle müşteriler henüz yeterince değerlendirilmeyen yeni bir alanı gösterebilir.

Öbeklemenin ilginç bir uygulaması *görüntü sıkıştırmadır*. Girdi örnekleri, imge noktalarındaki renk değerleridir ve öbekleme yazılımı, benzer renkteki imge noktalarını aynı öbeklere toplar, ki bunlar imgede sıkça bulunan renkleri gösterir. Eğer bir resimde az sayıda rengin farklı tonları varsa ve aynı öbeğe düşen imge noktalarını tek bir renkle, örneğin o öbekteki tonların ortalamasıyla gösterirsek, renkler *nicelendirilmiş* olur. Diyelim ki 16 milyon rengi desteklemek için her nokta 24 bit ile gösterilsin. İmgede yalnızca 64 ana rengin tonları varsa, her nokta için 24 yerine 6 bit yeter. Örneğin, imgenin farklı yerlerinde mavinin çeşitli tonları varsa ve hepsini aynı ortalama maviyle gösterirsek imgedeki ayrıntıları kaybederiz ama imgeyi saklarken ya da ağ üzerinden yollarken işimiz kolaylaşır. Görüntüleri inceleyerek daha üst düzey düzenlilikleri, örneğin dokuları, kenarları, nesneleri, vs. bulabilirsek imgeyi daha basit ve kullanışlı betimleyebilir ve örneğin nokta düzeyine göre daha yüksek oranda sıkıştırabiliriz. Bir belge imgesinin rastgele 0/1 noktalarından değil harf imgelerinden oluştuğunu bildiğimizde verinin bu yapısını kullanabilirsek daha kısa bir gösterim oluşturabiliriz: 'A'nın 16×16 imgesi 32 bayt tutarken, ASCII kodu yalnızca 1 bayttır.

Belge öbeklemede amaç, benzer belgeleri öbeklemektir. Örneğin haber bültenleri, politika, spor, moda, gibi alt başlıklara bölünebilir. Çoğunlukla bir belge, bir *sözcük torbası* olarak betimlenir. Önce N sözcükle bir sözlük tanımlar ve her belgeyi, i. öğesi, eğer i. sözcük belgede geçiyorsa 1, geçmiyorsa 0 olan N boyutlu bir yöneyle gösteririz. "–ler," "–de" gibi sonekler atılıp yalnızca kökler kullanılır; "bir," "ve" gibi bilgi vermeyen sözcükler de sözlükte yer almaz. Sonrasında iki belgenin benzerliği kaç tane ortak sözcük içerdiklerine göre hesaplanır ve benzer belgeler öbeklenir. Burada tabii ki sözlüğün seçimi çok önemlidir.

Yapay öğrenme, *biyoinformatikte* de kullanılır. Genomumuzdaki DNA, A, G, C, ve T bazlarının dizilimlerinden oluşur ve bizi tanımlar. RNA, DNA'dan, proteinler de RNA'dan çevriyazılır. Proteinler yaşayan bedenin ne olduğunu ve ne yaptığını belirler. Nasıl ki DNA bazların dizisiyse, protein de (bazlar tarafından belirlenen) aminoasitlerin dizisidir. Bilgisayar biliminin moleküler biyolojideki bir uygulaması iki dizinin *hizalanmasıdır*. Bu zor bir dizgi eşleme problemidir, çünkü hem dizgiler oldukça uzun olabilir, hem de verilen dizginin eşleneceği birçok şablon dizgi vardır; üstelik, bazı öğeler silinmiş, eklenmiş ya da değiştirilmiş olabilir. Öbekleme, *motif öğrenmede*, yani, proteinlerde sıkça rastlanan aminoasit dizilerini bulmakta kullanılır. Motifler, içinde bulundukları dizilerdeki yapısal ya da işlevsel özelliklere karşılık gelebilecekleri için önemlidir. Bir benzetme olarak aminoasitleri harfler, proteinleri de cümleler olarak düşünürsek motifler, farklı cümlelerde sıkça rastlanan ve belirli bir anlam taşıyan harf dizilerine, yani, sözcüklere karşılık gelir.

1.2.5 Pekiştirmeli Öğrenme

Bazı uygulamalarda dizgenin çıktısı bir *eylem* dizisidir. Böyle bir durumda önemli olan tek bir eylem değil, bir *politika*, yani, hedefe ulaşmak için gerçekleştirilen doğru eylem dizisidir. Ara bir durumda en iyi eylem tanımlanamaz; bir eylem ancak iyi bir politikanın parçasıysa iyidir. Böyle bir uygulamada yapay öğrenme yazılımı, politikaların iyiliklerini değerlendirebilmeli ve geçmişteki iyi eylemlerden başarılı bir politika üretebilmelidir. Bu tür öğrenmeye, *pekiştirmeli öğrenme* denir.

Buna iyi bir örnek *oyun oynamaktır*. Oyunda tek bir hamle değil doğru hamleler dizisi önemlidir. Bir hamle, iyi bir politikanın parçasıysa iyidir. Oyunlar, hem yapay zekâ, hem de yapay öğrenmede önemli bir araştırma alanıdır, çünkü oyunların kuralları basittir, ama iyi oynanması oldukça zor olabilir. Satranç gibi bir oyunun az sayıda kuralı vardır, ama her durumda olası hamle sayısının çok olmasından ve oyunun çok sayıda hamle yapmayı gerektirmesinden dolayı çok karmaşıktır. İyi oyun oynamayı öğrenebilen algoritmalar geliştirebilirsek onları ekonomik faydası olan gerçek uygulamalarda da kullanabiliriz.

Örneğin, pekiştirmeli öğrenmenin bir uygulama alanı bir robotun ortamda hedefi bulabilmesidir. Robot bir anda belirli yönlerden birine doğru yer değiştirebilir. Deneme sürüşleri ardından başlangıçtan hedefe olabildiğince hızlı ve engellere çarpmadan ulaşabilmek için gerekli eylem dizisini öğrenebilmelidir.

Robotun çevreyi algısı gürültülü ya da eksikse pekiştirmeli öğrenme daha zor olur. Örneğin, kamera yalnızca bir açıyı görüyorsa, robot ortamı *kısmen gözleyebilir* ve eylem kararını belirsizliklere rağmen vermek zorundadır. Örneğin, robot odadaki yerini tam olarak bilmez, yalnızca solunda duvar olduğunu bilir. Bir görev, *birden çok etmenin* ortak bir amaç için birlikte çalışmasını gerekirebilir. Futbol oynayan bir robot takımı buna bir örnektir.

1.3 Kısa Kısa

Evrim, bedenimiz yanında içgüdü ve reflekslerimizi de belirler. Yaşam boyunca davranışlarımızı değiştirmeyi de öğreniriz. Bu da içinde yaşadığımız ortamda evrim tarafından öngörülemeyen değişikliklerle başedebilmemizi sağlar. Dar bir çevrede kısa bir hayat yaşayan canlıların tüm davranışları önceden tanımlanmış olabilir, ama karşılaşabileceğimiz her durumda neler yapmamız gerektiğini beynimize kazımak yerine evrim bize büyük bir beyin ve öğrenebilme yeteneği vermiştir. Öğrenme sayesinde deneyimlerimizle kendimizi geliştirir, yeni ortamlara uyum sağlayabiliriz. Belirli bir durum için en iyi davranışı öğrendiğimizde bu bilgi beynimizde saklanır ve benzer bir durumla karşılaştığımızda bu bilgiyi hatırlar, ona göre davranırız. Yine de öğrenmenin de sınırı vardır; nasıl ki ne kadar yararlı olursa olsun, üçüncü bir kol ya da başımızın arkasında bir göz çıkartamazsak beynimizin sınırlı yetenekleriyle hiçbir zaman öğrenemeyeceğimiz şeyler de vardır. Leahey ve Harris (1997) öğrenmeyi psikoloji açısından ele alır. Psikoloji, bilişsel bilim, ya da sinirbilimin aksine yapay öğrenmede amaç, insan ve hayvanların öğrenmelerinin altında yatan süreçleri anla-

mak değil, her mühendislik dalında olduğu gibi işe yarar araçlar ya da makineler geliştirmektir.

Bilimin neredeyse tamamı veriyi açıklayan modeller kurmak üzerinedir. Bilim insanları deney tasarlar, gözlem yapar, ve veri toplar. Sonra ellerindeki veriyi açıklayan basit modeller bularak bilgi çıkarmaya çalışırlar. Amaç hep örnek durumlardan genel kural çıkarabilmektir.

Günümüzde öyle bir noktaya geldik ki verinin çok büyük oluşu, gerekli bilgiye sahip insanların azlığı, ve elle işlemenin masrafı yüzünden veri işleme ve bilgi çıkarma artık insanlar tarafından yapılamıyor. İşte bu yüzden veriyi inceleyip bilgi çıkarabilecek, yani öğrenecek bilgisayar yazılımlarına her zamankinden daha çok gereksinim vardır.

Gelecek bölümlerde ele alacağımız yöntemlerin kaynağı farklı bilimsel dallara dayanır. Bazen aynı algoritma farklı gelişim izleyen birden çok alanda birbirinden habersiz olarak önerilmiştir.

İstatistikte, örnek gözlemlerden genel açıklamalara geçmeye *çıkarım* ve öğrenmeye *kestirim* denir. Sınıflandırma, istatistikte *ayırtaç çözümlemesi* olarak bilinir (McLachlan 1992; Hastie, Tibshirani, ve Friedman 2001). Bilgisayarların ucuzlaması ve yaygınlaşmasından önce istatistikçiler küçük örneklemlerle çalışırdı ve matematikçi olduklarından ancak matematiksel yöntemlerle inceleyebildikleri basit, dağılım tabanlı modeller kullanırlardı. Mühendislikte sınıflandırmaya *örüntü tanıma* denir ve dağılımdan bağımsız, sonuç odaklı bir yaklaşım kullanılır (Duda, Hart, ve Stork 2001; Webb 1999). Yapay öğrenme, *yapay zekâ* ile yakından ilgilidir (Russell ve Norvig 2002), çünkü zeki bir etmenin çevresindeki değişikliklere uyum gösterebilmesi gerekir. Görüntü tanıma, konuşma tanıma, robotik gibi uygulamalar en iyi örnek verilerden öğrenilir. *Sinyal işleme* alanındaki araştırmalar, uyarlanır görüntü tanıma ve konuşma tanıma yazılımlarının geliştirilebilmesini sağlamıştır. Bunlar arasında, konuşma tanıma için geliştirilen *saklı Markov modelleri* özellikle öne çıkar.

1980'lerin sonlarında, VLSI teknolojisindeki gelişmeler sayesinde binlerce işlemcili koşut devreler tasarlayabilecek duruma geldik ve bu bağlamda, hesaplamaları çok sayıda işlemciye paylaştırabilmek için yapay sinir ağları alanı yeniden öne çıktı (Bishop 1995). Zaman içinde çoğu yapay sinir ağı algoritmasının istatistikte bilinen yöntemlere dayandığı anlaşıldı ve "beyin benzeri hesaplama" iddiaları unutulmaya başlandı.

Son yıllarda destek yöney makineleri gibi çekirdek tabanlı algoritmalar yaygın olarak kullanılmaktadır. Bunlar, çekirdek işlevleri sayesinde biyoinformatik ve dil işleme gibi farklı uygulamalarda kullanılabilmektedir. İyi bir veri gösteriminin öğrenme için çok önemli olduğu bugünlerde herkesçe bilinmektedir ve çekirdek işlevleri böylesi uzman bilgisini gösterebilmek için çok iyi bir yoldur.

Son zamanlarda bellek ve veri iletişim bedellerindeki azalmalar çok büyük veri kümelerini internet üzerinden paylaşabilmeye olanak sağlamıştır. Hesaplamadaki hızlanmaya koşut olarak öğrenme algoritmalarını çok büyük veriler üzerinde uygulamak olanağı ortaya çıkmıştır. Geçtiğiniz onyıllarda, yapay zekânın bir gün gerçek olabilmesi için yeni bir paradigmaya, yeni bir düşünce tarzına, yepyeni bir hesaplama modeline, ya da tamamen farklı algoritmalara gerek olduğuna inanılırdı.

Yapay öğrenmenin son yıllarda çeşitli uygulamalardaki başarılarına bakınca, belki de gerekenin yeni algoritmalar değil, büyük miktarda örnek veri ve yeterli hesaplama gücü olduğu söylenebilir. Örneğin, destek yöney makinelerinin kökleri, 1950 ve 1960'larda önerilmiş potansiyel işlevler, doğrusal modeller, ve en yakın komşu tabanlı yöntemlere dayanır; ancak o günlerde bu algoritmaların güçlerini gösterebilmeleri için gereken hızlı bilgisayarlar ve yeterli veri saklama yeteneği yoktu.

Büyük miktarda örnek veri üzerinde eğitilecek ya da çok sayıda deneme/yanılma yapabilecek bilgisayarların, çeviri, ve hatta planlama gibi görevleri oldukça basit öğrenme algoritmalarıyla gerçekleyebileceğini bugün öne sürebilriz. Zekâ, şimdiye kadar düşünülmemiş çok farklı bir formülden değil de, basit ve kolay bir algoritmanın büyük miktarda veri üstünde sabırla kullanımından kaynaklanıyor gibi görünüyor.

Yapay öğrenme algoritmalarının iş dünyasındaki büyük veritabanlarına uygulanmasına *veri madenciliği* deniyor (Witten ve Frank 2005; Han ve Kamber 2006). Bilgisayar bilimlerinde buna eskiden *veri tabanlarında bilgi buluşu* denirdi.

İstatistik, örüntü tanıma, yapay sinir ağları, işaret işleme, kontrol, yapay zekâ, ve veri madenciliği gibi farklı alanlardaki araştırmalar geçmişte farklı noktalara vurgularla farklı yollar izlemiştir. Bu kitapta amaç, bu farklı vurguları birleştirerek uygulamaları ve onlara önerilen çözümleri bütünleştirici bir yaklaşımla tartışmaktır.

1.4 İlgili Kaynaklar

Yapay öğrenme üzerine güncel çalışmalar değişik dergi ve konferanslara yayılmıştır. *Machine Learning* ve *Journal of Machine Learning Research*, yapay öğrenmeye genel bakarken, *Neural Computation*, *Neural Networks*, ve *IEEE Transactions on Neural Networks*, yapay sinir ağı vurguludur. *Annals of Statistics* ve *Journal of the American Statistical Association* gibi istatistik dergilerinde yapay öğrenme açısından ilginç olan makaleler basılır. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, bir başka önemli kaynaktır.

Ayrıca, yapay zekâ, örüntü tanıma, bulanık mantık, ve işaret işleme üzerine dergiler de yapay öğrenme makaleleri içerir. *Data Mining and Knowledge Discovery*, *IEEE Transactions on Knowledge and Data Engineering*, ve *ACM Special Interest Group on Knowledge Discovery and Data Mining Explorations Journal*, veri madenciliği ağırlıklı dergilerdir.

Yapay öğrenmeyle ilgili önemli konferansları, *Neural Information Processing Systems* (NIPS), *Uncertainty in Artificial Intelligence* (UAI), *International Conference on Machine Learning* (ICML), *European Conference on Machine Learning* (ECML), ve *Computational Learning Theory* (COLT) olarak sayabiliriz. *International Joint Conference on Artificial Intelligence* (IJCAI), ve yapay sinir ağları, örüntü tanıma, bulanık mantık, ve genetik algoritmalar üzerinde düzenlenen, ya da görüntü tanıma, konuşma tanıma, sinyal işleme, robotik, ve veri madenciliği gibi uygulama alanlarındaki konferanslarda yapay öğrenme oturumları yer alır.

Yapay öğrenme alanında çalışan araştırmacılarının ortak kullanımı için internet üzerinde birçok veri tabanı bulunmaktadır:

- *UCI Arşivi* en çok bilinendir:
  http://www.ics.uci.edu/~mlearn/MLRepository.html
- *UCI KDD Arşivi*:
  http://kdd.ics.uci.edu/summary.data.application.html
- *Statlib*: http://lib.stat.cmu.edu
- *Delve*: http://www.cs.utoronto.ca/~delve/

Bunlara ek olarak, hesaplamalı biyoloji, yüz tanıma, konuşma tanıma, vs. gibi belirli uygulamalar için özel veri tabanları da vardır.

Bunlara her gün başka ve daha büyük veri kümeleri eklenmektedir. Yine de bazı araştırmacılar bu arşivlerin gerçek veri özelliklerinin tamamını yansıtmayıp sınırlı bir kapsama sahip olduğuna ve dolayısıyla bu arşivlerdeki başarı oranlarının bir gösterge olmadığına inanır. Hatta, aynı arşivden veri kümelerini daha iyi algoritmalar elde etmek için tekrar tekrar kullandığımızda bunlara uzmanlaşmış "UCI algoritmaları" ürettiğimiz iddia edilebilir.

Gelecek bölümlerde göreceğimiz gibi, farklı algoritmalar farklı uygulamalarda iyi çalışır ve bu yüzden tek bir uygulamayı ve onunla ilgili bir ya da daha çok veri kümesini ele alıp algoritmaları bu çerçevede karşılaştırmak mantıklıdır.

Yapay öğrenme üzerine çalışan araştırmacıların yakın zamanlı yayınlarının çoğuna internet üzerinden erişilebilir ve başlamak için iyi bir yer, NEC Araştırma Dizinidir: http://citeseer.ist.psu.edu. Çoğu yazar, algoritmalarının kaynak kodlarını Web üzerinde paylaşıma açmıştır. Çeşitli yapay öğrenme algoritmalarını gerçekleyen ücretsiz yazılım paketleri de vardır ve bunlar arasında Weka özellikle dikkate değer: http://www.cs.waikato.ac.nz/ml/weka/.

1.5 Alıştırmalar

1. Bir dökümanı faksla imge olarak, ya da görsel karakter okuyucuyla tarayıp harfleri bir yazı dosyası olarak gönderebiliriz. Bu iki yaklaşımın getiri ve götürülerini tartışın. Hangi durumlarda hangisi daha kullanışlıdır?
2. Görsel karakter okuyucu tasarladığımızı düşünelim. Bunun için bir yol, her karakterin bit haritasını bir şablon olarak tutmak ve okunan karakterle nokta nokta eşleştirmektir. Böyle bir sistemin hangi durumlarda başarısız olacağını açıklayınız. Barkod okuyucuları sizce hâlâ neden kullanılmaktadır?
3. İstenmeyen epostaları ayırt edebilecek bir yazılım tasarlamak istediğimizi varsayalım. Bir epostanın istenmeyen eposta olduğunu nasıl anlayabiliriz? Bir epostanın istenmeyen olduğuna karar verdiğinde yazılımın ne yapmasını istersiniz, silmesini mi, başka bir dizine atmasını mı, yoksa yalnızca ekranda farklı bir şekilde işaretlemesini mi?
4. Kendi kendini kullanan bir taksi tasarlamak isteseydiniz, kısıtlar neler olurdu? Girdi nedir? Çıktı nedir? Yolcu ile nasıl iletişim kurulabilir Öteki taksilerle iletişim gerekli olur mu, yani bir "dil"e gerek var mıdır?

5. Sepet çözümlemesinde X ve Y ürünleri arasında bir bağımlılık ararız. Bir müşteri işlem veri tabanı verildiğinde, bu bağımlılıkları nasıl bulabiliriz? Bunu ikiden çok ürüne nasıl genelleyebiliriz?
6. Kullanıcı tarafından girilecek bir sonraki komutu, ya da indirilecek bir sonraki sayfayı nasıl öngörebiliriz? Böyle bir öngörü ne zaman faydalı, ne zaman rahatsız edici olur?
7. Gazetenizde, politika, spor, ve sanat konusunda beşer tane haber bulun ve bu haberleri inceleyip her konu için sıkça kullanılan sözcükleri belirleyin. Bu sözcükler bize farklı konuları ayırmakta yardımcı olacaktır. Örneğin, politika üzerine bir haber bülteninde "hükümet", "parti", "meclis" gibi sözcükler daha olasıyken sanat üzerine bir bültende "konser", "sergi" gibi sözcüklere daha sık rastlanır. Ayrıca, "amaç" ya da "oyun" gibi muğlak sözcükler de vardır.
8. 100×100 çözünürlüğünde bir yüz görüntüsü, 10.000 boyutlu bir yöney oluşturur. Görüntüyü bir nokta sağa kaydırdığımızda bu, 10.000 boyutlu uzayda çok farklı bir yöney olur. Bu tür kaymalardan etkilenmeyen bir yüz tanıyıcıyı nasıl oluşturabiliriz?
9. Bir sözcük seçin, örneğin, "yapay." On kez siz, on kez de bir arkadaşınıza yazdırın. Bu 20 resmi inceleyip sizin elyazınızı arkadaşınızınkinden ayırt edebilecek çizgi türleri, eğrilikler, döngüler, gibi özellikler bulmaya çalışın.
10. İkinci el arabaların fiyatını kestirirken arabanın mutlak fiyatı yerine sıfır fiyatının yüzde kaçı ettiğini kestirmek daha akla yakındır. Neden?

1.6 Kaynaklar

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition.* Oxford: Oxford University Press.

Duda, R. O., P. E. Hart, D. G. Stork. 2001. *Pattern Classification,* 2nd ed. New York: Wiley.

Han, J., M. Kamber. 2006. *Data Mining: Concepts and Techniques,* 2nd ed. San Francisco: Morgan Kaufmann.

Hand, D. J. 1998. "Consumer Credit and Statistics." *Statistics in Finance,* ed. D. J. Hand, S. D. Jacka, 69–81. London: Arnold.

Hastie, T., R. Tibshirani, J. Friedman. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* New York: Springer.

Leahey, T. H., R. J. Harris. 1997. *Learning and Cognition,* 4th ed. New York: Prentice Hall.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition.* New York: Wiley.

Russell, S., P. Norvig. 2002. *Artificial Intelligence: A Modern Approach,* 2nd ed. New York: Prentice Hall.

Webb, A. 1999. *Statistical Pattern Recognition.* London: Arnold.

Witten, I. H., E. Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques,* 2nd ed. San Francisco: Morgan Kaufmann.

Bölüm 2

Gözetimli Öğrenme

Gözetimli öğrenmeyi en basit türünden, bir sınıfı artı ve eksi örneklerinden öğrenmekten başlayarak ele alacağız. Ardından, çok sınıflı durumu ve çıktının sürekli bir sayı olduğu bağlanımı tartışacağız.

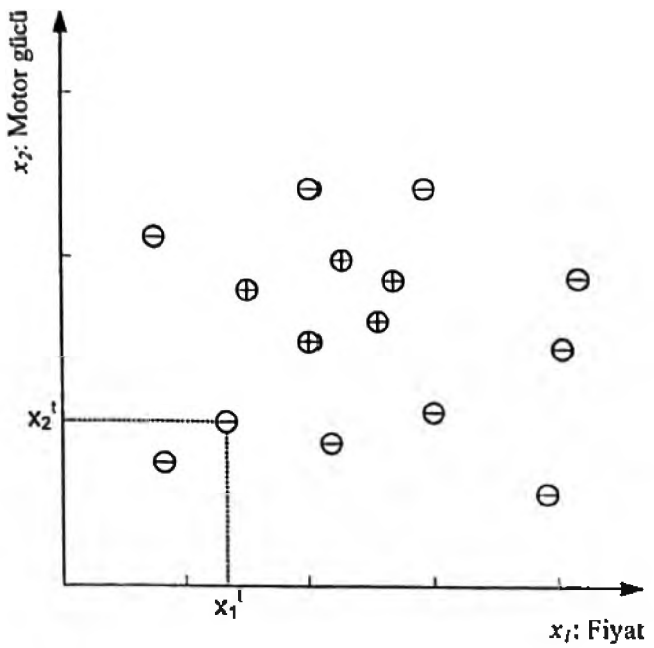
2.1 Bir Sınıfı Örneklerinden Öğrenme

"Aile arabası" kavramını ya da *sınıfını* (C) öğrenmek istediğimizi varsayalım. Elimizde bazı araba örnekleri ve anket yaptığımız insanların bunlarla ilgili yanıtları olsun. Ankette insanlar arabalara bakıp aile arabası olduğunu düşündüklerini *artı örnekler*, olmayanları *eksi örnekler* olarak etiketlesin. *Sınıf öğrenme*, tüm artı örnekleri kapsayan ama hiçbir eksi örneği kapsamayan bir tanım bulmaktır. Böyle bir tanımı kullanarak öngörü yapabiliriz: Ankette sorulmamış bir arabanın özelliklerini öğrenilen tanımla karşılaştırarak aile arabası olup olmadığını söyleyebiliriz. Ya da bilgi çıkarabiliriz: Böyle bir çalışma, insanların bir aile arabasından neler beklediğini anlamak için bir araba firması tarafından desteklenmiş olabilir.

Uzmanlarla görüştükten sonra, bir arabanın olası tüm özellikleri içinde bir aile arabasını ötekilerden ayırt etmede en önemli olanların fiyat ve motor gücü olduğuna karar verdiğimizi düşünelim. Bu iki öznitelik, sınıflandırıcının *girdisi* olur. Böyle bir *girdi gösterimine* karar verdiğimizde tüm öteki öznitelikleri önemsiz varsayıp gözardı etmiş oluruz. Her ne kadar koltuk sayısı, renk, vs. gibi başka öznitelikler de araba türlerini ayırt etmekte önemli olabilirse de, bu örneği basit tutmak için yalnızca fiyat ve motor gücünü dikkate alacağız.

Fiyatı ilk girdi özniteliği, x_1 (örneğin, TL), ve motor gücünü ikinci öznitelik, x_2 (örneğin, santimetre küp), olarak alalım. Böylece bir arabayı iki sayısal değerle

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{2.1}$$



Şekil 2.1: "Aile arabası" sınıfı için öğrenme kümesi. Her veri noktası örnek bir arabaya karşılık gelir ve noktanın koordinatları, fiyat ve motor gücünü belirtir. '+', sınıfın artı bir örneğini (aile arabası) ve '−', eksi bir örneğini (aile arabası değil) gösterir.

gösteririz ve etiketi sınıf değerini belirler:

$$r = \begin{cases} 1 & \text{eğer } \boldsymbol{x} \text{ artı bir örnekse} \\ 0 & \text{eğer } \boldsymbol{x} \text{ eksi bir örnekse} \end{cases} \tag{2.2}$$

Her araba böyle bir sıralı $(\boldsymbol{x}, r)$ çiftiyle gösterilir ve öğrenme kümesi böyle N örnekten oluşur:

$$\mathcal{X} = \{\boldsymbol{x}^t, r^t\}_{t=1}^N \tag{2.3}$$

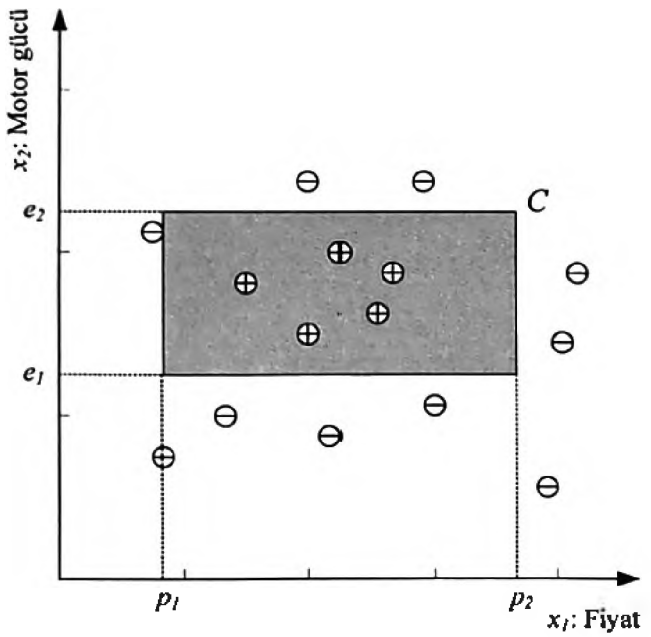
Burada t, kümedeki farklı örnekleri gösteren dizin değişkenidir; zaman ya da başka bir tür sıralamaya karşılık gelmez.

Şimdi öğrenme kümemizi iki boyutlu (x_1, x_2) uzayında çizebiliriz. Her t örneği, (x_1^t, x_2^t) koordinatlarında bir veri noktasıdır ve sınıfı, yani artı ya da eksi olması, r^t ile gösterilmiştir (bkz. şekil 2.1).

Uzmanlarla biraz daha görüşüp veriyi inceledikten sonra, bir arabanın aile arabası olabilmesi için fiyatının ve motor gücünün belirli aralıklarda olması gerektiği kanısına varabiliriz:

$$(f_1 \leq \text{fiyat} \leq f_2) \text{ VE } (m_1 \leq \text{motor gücü} \leq m_2) \tag{2.4}$$

Burada, f_1, f_2, m_1, ve m_2, aralık sınırları için uygun değerleri gösterir. Denklem 2.4, C sınıfının fiyat-motor gücü uzayında bir dikdörtgen olduğunu varsayar (bkz. şekil 2.2).



Şekil 2.2: Bir denence sınıfı örneği. Aile arabası sınıfı, fiyat-motor gücü uzayında bir dikdörtgendir.

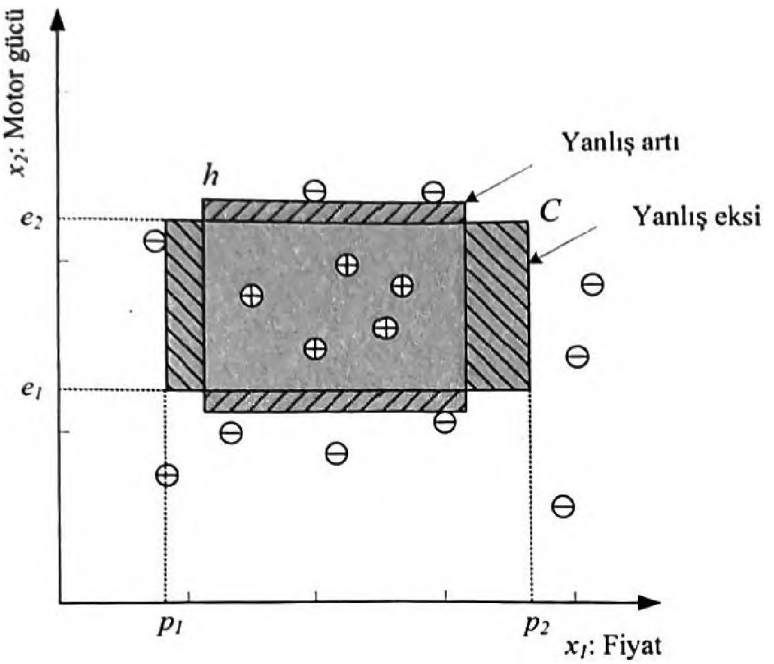
Denklem 2.4, sınıf tanımının (C) geldiğine inandığımız *denence sınıfını* ($\mathcal{H}$) bir dikdörtgenler kümesi olarak belirler. Öğrenme algoritması, $\mathcal{H}$ içinden C sınıfına en yakın $h \in \mathcal{H}$ *denencesini* bulur.

Her ne kadar denence sınıfını uzman belirlese de parametreleri bilinmez; yani $\mathcal{H}$ denence sınıfını bilsek de, hangi özel $h \in \mathcal{H}$ denencesinin C sınıfına eşit ya da en yakın olduğunu bilmeyiz. Ama ilgimizi bu denence sınıfına kısıtladığımızda sınıf öğrenme problemi, h denencesini tanımlayan dört parametreyi bulmaya indirgenmiş olur.

Amaç, C sınıfına en çok benzeyen $h \in \mathcal{H}$ denencesini bulmaktır. Denencenin girdiye bağlı olarak 0/1 öngörüsü yaptığını varsayalım:

$$h(\boldsymbol{x}) = \begin{cases} 1 & \text{eğer } h, \boldsymbol{x} \text{ girdisini artı olarak sınıflandırırsa} \\ 0 & \text{eğer } h, \boldsymbol{x} \text{ girdisini eksi olarak sınıflandırırsa} \end{cases} \tag{2.5}$$

Gerçek bir uygulamada tabii ki $C(\boldsymbol{x})$' sınıf tanımını bilmeyiz; bilsek h olarak onu seçerdik ve öğrenmeye gerek kalmazdı. Bu yüzden $h(\boldsymbol{x})$ denencesinin $C(\boldsymbol{x})$ ile ne kadar uyuştuğunu da ölçemeyiz. Elimizde yalnızca olası tüm $\boldsymbol{x}$ girdilerinin küçük bir altkümesi olan $\mathcal{X}$ öğrenme kümesi var. *Gözlemsel hata*, $\mathcal{X}$ içindeki örnekler üzerinde h denencesinin *öngörülerinin* verilmiş *istenen değerlerden* ne kadar farklı olduğudur. $\mathcal{X}$ öğrenme kümesi üzerinde h denencesinin hatası en basit olarak hatalı örneklerin



Şekil 2.3: C gerçek sınıfı, h öngörülen denenceyi gösterir. $C = 1$ ve $h = 0$ olan örnek yanlış eksi, $C = 0$ ve $h = 1$ olan örnek yanlış artıdır. Öteki türler, yani doğru artılar ve doğru eksiler, hata yapılmayan örneklerdir.

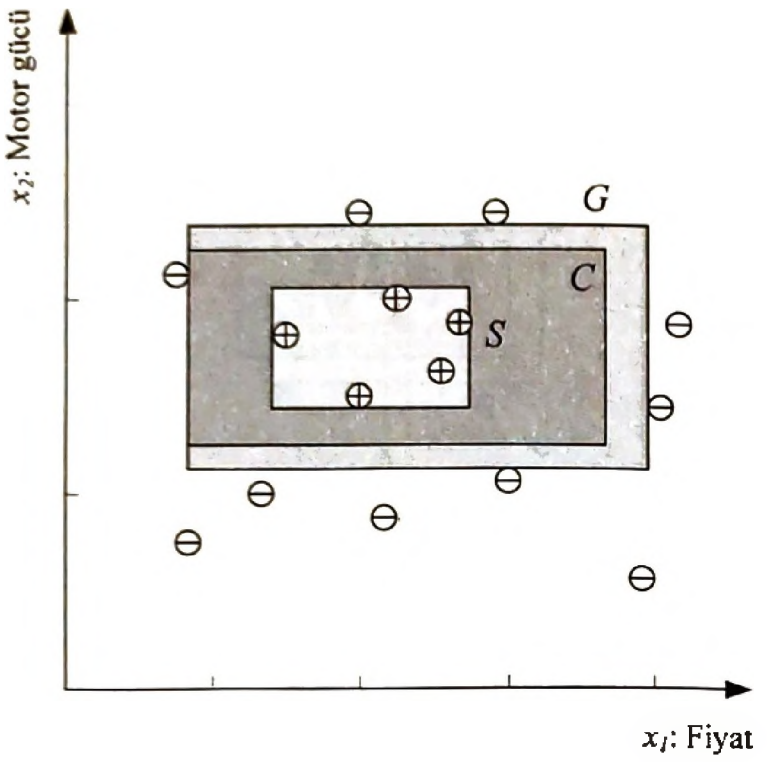
sayısıdır (bkz. şekil 2.3):

$$E(h|\mathcal{X}) = \sum_{t=1}^{N} 1(h(\boldsymbol{x}^t) \neq r^t) \tag{2.6}$$

Burada, $1(a \neq b)$, $a \neq b$ olduğunda 1, değilse 0 değerini alan 0/1 işlevidir.

Örneğimizde, denence sınıfı $\mathcal{H}$ olası tüm dikdörtgenlerin kümesidir. Her sınır dörtlüsü, $(f_1^h, f_2^h, m_1^h, m_2^h)$, $\mathcal{H}$ içinden bir denence h tanımlar ve olası bu denencelerin arasından en iyisini seçmemiz, bir başka deyişle, öğrenme kümesindeki tüm artı örnekleri içerecek ve hiçbir eksi örneği içermeyecek dört parametre değerini bulmamız gerekir. Eğer x_1 ve x_2 kesikli değilse, bunu sağlayan, yani hatayı (E) sıfırlayan, sonsuz sayıda h vardır. Bununla birlikte farklı denenceler, ileride sınıra yakın yerlerden gelebilecek yeni örneklerde farklı öngörüler yapacaktır. Buna, yani denencemizin öğrenme kümesinde yer almayan gelecek örnekleri ne kadar doğru sınıflandırdığına *genelleştirme* denir.

Bir olasılık, *en özgül denenceyi* (S), yani tüm artı örnekleri içeren ve hiçbir eksi örneği içermeyen en dar dikdörtgeni bulmaktır (bkz. şekil 2.4). Bu, öğrenilen sınıf olarak tek $h = S$ denencesini tanımlar. Gerçek sınıf C, S denencesinden daha geniş olabilir, ama daha dar değildir. *En genel denence* G, tüm artı örnekleri içeren ve hiçbir eksi örneği içermeyen en geniş dikdörtgendir (bkz. şekil 2.4). S ile G arasındaki her $h \in \mathcal{H}$, hatasız ve geçerli bir denencedir, öğrenme kümesi ile *tutarlıdır*, ve böyle



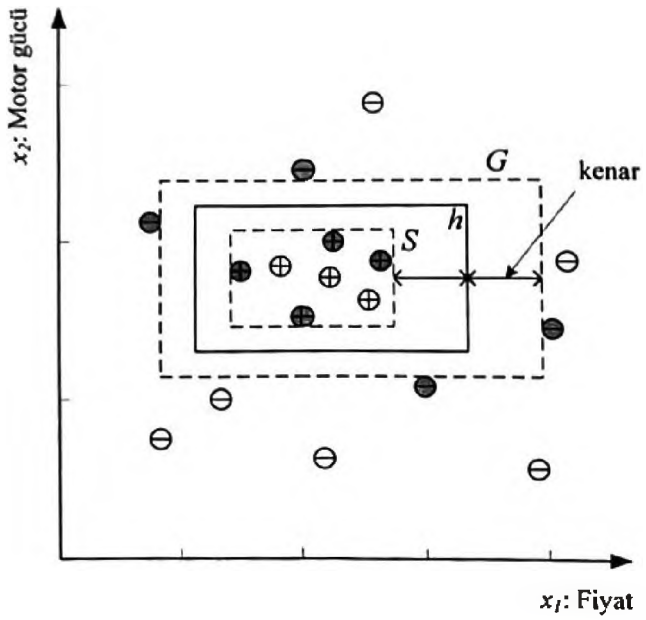
Şekil 2.4: S en özgül, G en genel denencedir.

h denenceleri *sürüm uzayını* oluşturur. Başka bir öğrenme kümesi için S, G, sürüm uzayı, parametreler, ve dolayısıyla öğrenilen h denencesi farklı olabilir.

Gerçekte, $\mathcal{X}$ ve $\mathcal{H}$ denence sınıfına bağlı olarak S-kümesi ve G-kümesini oluşturan bir çok S_i ve G_j olabilir. S-kümesinin her öğesi tüm örneklerle tutarlıdır ve daha özgül, tutarlı bir denence yoktur. Benzer biçimde, G-kümesinin her öğesi tüm örneklerle tutarlıdır ve daha genel, tutarlı bir denence yoktur. Bu ikisi sınır kümelerini oluşturur, aralarındaki tüm deneceler tutarlıdır, ve sürüm uzayının parçasıdır. *Aday eleme* algoritması (Mitchell 1997) öğrenme örneklerini birer birer gördükçe, S- ve G-kümelerini artımlı olarak günceller. Biz bu bölümde, $\mathcal{X}$'in, S ve G'nin tek olacakları kadar büyük olduğunu varsayıyoruz.

$\mathcal{X}$ verildiğinde, S, G, ya da sürüm uzayından herhangi bir h bulup denencemiz olarak kullanabiliriz. h denencesini S ile G denencelerinin ortasından geçecek biçimde seçmek akla yakındır; böylece, *kenar payı*, yani sınırla sınıra en yakın örnekler arasındaki uzaklık olabildiğince büyük olur (bkz. şekil 2.5). En büyük kenar payına sahip h denencesinin hatanın en küçük olduğu denence olabilmesi için, hata (yitim) işlevinin örneklerin yalnızca sınırın doğru tarafında olup olmadığını değil, sınırdan ne kadar uzakta olduğunu da ölçebilmesi gerekir. Yani, 0/1 döndüren bir $h(\boldsymbol{x})$ yerine sınıra olan uzaklığa bağlı bir denence kullanmalı ve bunun için eşitliğe bakıp 0/1 üreten $1(\cdot)$ işlevi yerine bu uzaklığı kullanan bir yitim işlevi kullanmalıyız.

Bazı uygulamalarda, yanlış bir kararın maliyeti çok yüksek olabilir, ve S ile G



Şekil 2.5: En iyi ayrım için en büyük kenar payına sahip olan denenceyi seçeriz. Boyanmış örnekler kenar payını tanımlayan (destekleyen) örneklerdir; öteki örneklerin atılması h'yi etkilemez.

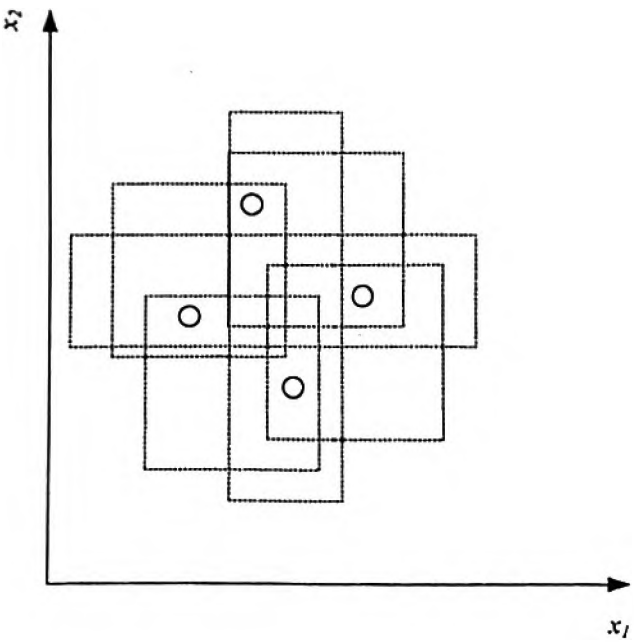
arasına düşen *kuşkulu* bir örnek için karar vermek yerine örneği *reddedip* kararı bir uzmana bırakmak isteyebiliriz.

Burada, $\mathcal{H}$ denence sınıfının gerçek C sınıfını içerdiğini, yani, $E(h|\mathcal{X}) = 0$ koşulunu sağlayan bir $h \in \mathcal{H}$ olduğunu varsaydık. Varsayılan $\mathcal{H}$ ile C öğrenilemeyebilir, yani hatayı sıfırlayan bir $h \in \mathcal{H}$ olmayabilir. Dolayısıyla herhangi bir uygulamada, $\mathcal{H}$ denence sınıfının yeterince esnek, ya da bir başka deyişle, C problemini öğrenebilecek "sığıma" sahip olduğuna emin olmamız gerekir.

2.2 Vapnik-Chervonenkis (VC) Boyutu

N nokta içeren bir veri kümemiz olsun. Bu N örneği 2^N farklı biçimde artı ve eksi olarak etiketleyebiliriz. Yani, N örnekle 2^N tane farklı öğrenme problemi tanımlanabilir. Eğer bu problemlerin hepsi için artı örnekleri eksilerden ayırabilen bir $h \in \mathcal{H}$ bulunabiliyorsa $\mathcal{H}$ denence sınıfının N noktayı *tuz buz ettiği* söylenir; yani, N tane örnekle tanımlanabilecek tüm öğrenme problemleri $\mathcal{H}$ içinden seçilen bir denence tarafından hatasız öğrenilebilir. $\mathcal{H}$ ile tuz buz edilebilen en çok nokta sayısına *Vapnik-Chervonenkis* (VC) *boyutu* denir, $VC(\mathcal{H})$ ile gösterilir ve $\mathcal{H}$ denence sınıfının *sığımını* ölçer.

Şekil 2.6'da dikdörtgenler denence sınıfının dört noktayı tuz buz edebildiğini görüyoruz. Öyleyse, $\mathcal{H}$ iki boyutlu dikdörtgenler denence sınıfı olduğunda $VC(\mathcal{H})$ dörttür. VC boyutunu hesaplarken tuz buz edilebilir dört nokta bulmamız yeterli-



Şekil 2.6: Dikdörtgenler denence sınıfı iki boyutta dört noktayı tuz buz edebilir. Sadece iki nokta içeren dikdörtgenler gösterilmiştir.

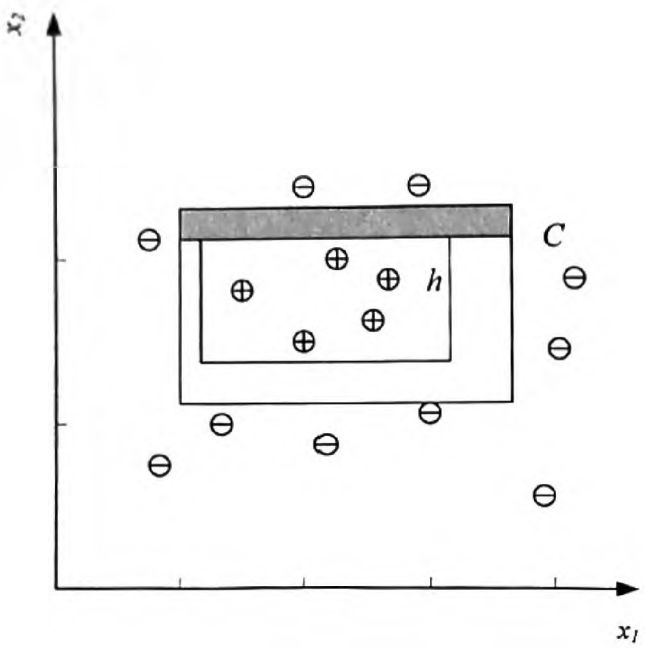
dir; iki boyuttaki *herhangi* dört noktayı tuz buz edebilmemiz gerekmez. Örneğin, bir doğru üzerindeki dört nokta dikdörtgenlerle tuz buz edilemez. Bununla birlikte, iki boyutta beş noktayı, tüm etiketlemeler için dikdörtgenlerle tuz buz edilebilecek biçimde, *hiçbir yere* koyamayız.

VC boyutu kötümser görünebilir, çünkü denence sınıfı olarak dikdörtgenleri kullandığımızda en çok dört nokta içeren verileri öğrenebileceğimizi söylemektedir. Dört noktalı veri kümelerini öğrenebilen bir öğrenme algoritması da çok işimize yaramaz. Ancak bu, VC boyutunun örneklerin geldiği olasılık dağılımından bağımsız olmasındandır. Gerçek yaşamda, değişimler yumuşak olur, birbirine yakın örnekler çoğunlukla aynı sınıftandır, ve *olası tüm etiketlemeleri* düşünmemiz gerekmez. Dörtten oldukça daha fazla örneği olan ve denence sınıfımız tarafından öğrenilebilir birçok veri kümesi vardır (örneğin şekil 2.1). Bu sayede, küçük VC boyutu olan denence sınıfları bile gerçek yaşamda uygulanabilir ve hatta büyük, örneğin sonsuz, VC boyutu olanlara yeğlenebilir.

2.3 Olası Yaklaşık Doğru Öğrenme

En dar S dikdörtgeni denencemiz olduğunda kaç örnek gerektiğine bakalım. Denencemizin yaklaşık olarak doğru, yani hata olasılığının bir değerle sınırlı olmasını isteriz. Ayrıca denencemizin, her zaman olmasa bile, çoğunlukla doğru karar verdiğine inanmak isteriz; yani belirleyebileceğimiz bir olasılıkla doğru olmak isteriz.

Olası yaklaşık doğru öğrenmede, bir C sınıfı ve bilinmeyen ama belirli bir $p(x)$ olasılık dağılımından çekilmiş örnekler verildiğinde, öyle bir örnek sayısı, N bulmak



Şekil 2.7: h ve C arasındaki fark, birisi gölgeli gösterilmiş dört tane dikdörtgen şeritin toplamına eşittir.

isteriz ki, istenen $\delta \leq 1/2$ ve $\epsilon > 0$ değerleri için h denencesinin hatası, en az $1 - \delta$ olasılıkla, en çok ϵ olsun:

$$P\{C\Delta h \leq \epsilon\} \geq 1 - \delta$$

Burada, $C\Delta h$, C ve h arasındaki alanı gösterir.

Bizim örneğimizde, S en dar dikdörtgen olduğundan C ve $h = S$ arasındaki hata bölgesi dört tane dikdörtgen biçimindeki şeritin toplamına eşittir (bkz. şekil 2.7). Artı bir örneğin bu alana düşme (ve hataya yol açma) olasılığının en çok ϵ olmasını isteriz. Bu şeritlerin her biri için, eğer olasılığın üst sınırının $\epsilon/4$ olmasını sağlayabilirsek, hata en çok $4(\epsilon/4) = \epsilon$ olur. Köşelerdeki çakışmaları ikişer kez saydığımızdan aslında hata bu durumda $4(\epsilon/4)$ değerinden de daha az olacaktır. Rastgele seçilen bir örneğin şeridi kaçırma olasılığı $1 - \epsilon/4$, N bağımsız örneğin hepsinin şeridi kaçırma olasılığı $(1 - \epsilon/4)^N$ ve hepsinin bütün şeritleri kaçırma olasılığı $4(1 - \epsilon/4)^N$ olacaktır, ki biz bunun en çok δ olmasını istiyoruz.

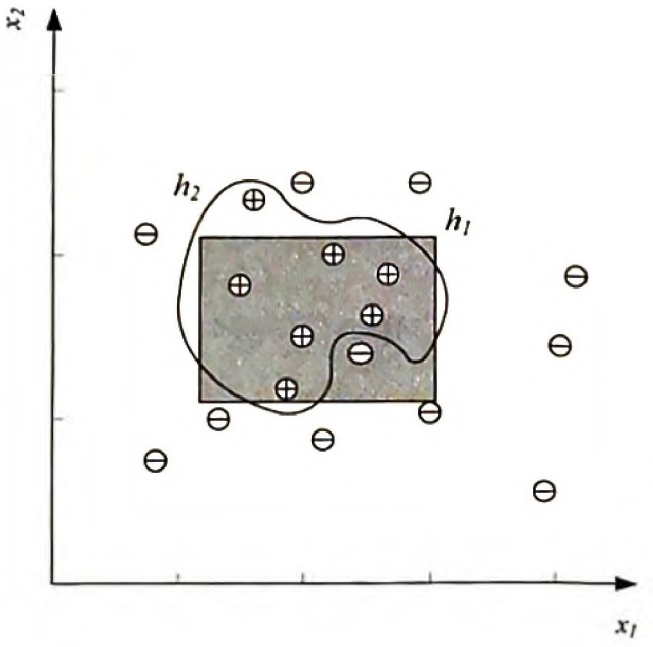
$$(1 - x) \leq \exp[-x]$$

eşitsizliğini biliyoruz. Dolayısıyla, eğer N ve δ değerlerini

$$4\exp[-\epsilon N/4] \leq \delta$$

eşitsizliğini sağlayacak biçimde seçersek, $4(1 - \epsilon/4)^N \leq \delta$ olur. Her iki yanı 4'e bölüp (doğal) logaritma alıp terimleri yeniden düzenleyelim:

$$N \geq (4/\epsilon)\log(4/\delta) \tag{2.7}$$



Şekil 2.8: Gürültü olduğunda artı ve eksi örnekler arasındaki sınır basit değildir ve basit bir denenceyle sıfır hata elde edilemez. Dikdörtgen, dört köşesinin dört parametreyle belirlendiği basit bir denencedir. Daha çok sayıda denetim noktası kullanan parçalı işlevlerle istenilen herhangi bir kapalı şekil çizilebilir.

Bu da gösterir ki, C sınıfından en az $(4/\epsilon)\log(4/\delta)$ bağımsız örnek alıp en dar dikdörtgeni denence olarak kullandığımızda, verilen bir noktanın *yanlış sınıflandırılma olasılığı*, en az $1-\delta$ *güven olasılığıyla*, en çok ϵ olacaktır. δ değerini azaltarak güven miktarını dilediğimiz ölçüde artırabilir, ya da ϵ değerini azaltarak dilediğimiz ölçüde az hata oluşmasını sağlayabiliriz. Denklem 2.7'de gördüğümüz gibi, gereken örnek sayısı $1/\epsilon$ ve $1/\delta$'nın yavaş artan, sırasıyla doğrusal ve logaritmik, bir işlevidir.

2.4 Gürültü

Gürültü, verideki istenmeyen aykırılıklardır. Gürültü yüzünden bir sınıfı öğrenmek daha zor ve basit bir denence sınıfıyla sıfır hata olanaksız olabilir (bkz. şekil 2.8). Gürültünün neden olduğu hakkında çeşitli açıklamalar vardır:

- Girdi özelliklerinin kaydı sırasındaki hatalar yüzünden veri noktaları kayabilir.
- Veri noktaları etiketlenirken bazı artı örnekler eksi ya da bazı eksiler artı olarak etiketlenmiş olabilir. Buna bazen *öğretmen gürültüsü* denir.
- Etiketi etkileyen ancak bizim dikkate almadığımız başka gözlenemeyen, *saklı* öznitelikler olabilir. Gözardı edilen bu özniteliklerin etkisi rastgele bir bileşen olarak modellenip "gürültü" olarak nitelenir.

Şekil 2.8'de görüldüğü gibi, gürültü olduğunda artı ve eksi örnekler arasındaki sınır basit olmayabilir ve onları ayırmak için daha yüksek sığımlı bir denence sınıfından daha karmaşık bir denence kullanmak gerekir. Bir dikdörtgen yalnızca dört sayıyla tanımlanabilse de daha karmaşık ayırıcıları tanımlamak için çok daha fazla parametreli daha karmaşık modeller gerekir. Şekil 2.8'deki kıvrımlı ayırıcı gibi daha karmaşık bir modelle veriyi sıfır hata ile modelleyebilir, ya da modeli dikdörtgen olarak basit tutup biraz hatayı hoş görebiliriz.

Basit dikdörtgeni kullanmak (öğrenme kümesi üzerinde hatası çok fazla değilse) daha anlamlıdır çünkü:

1. Kullanımı basittir. Bir noktanın dikdörtgenin içinde olup olmadığını denetlemek kolaydır ve böylece, gelecek veri örneklerinin artı mı, eksi mi olduğuna kolaylıkla karar verebiliriz.

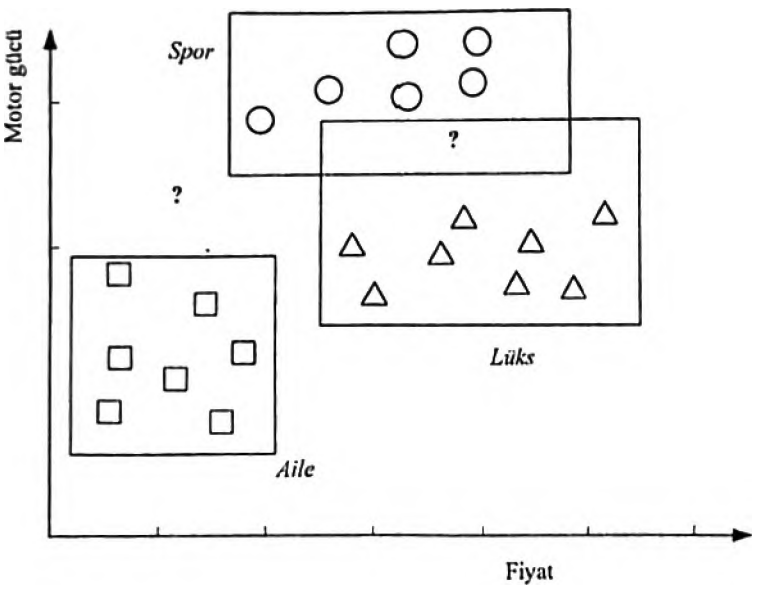
2. Eğitimi basittir ve az sayıda parametresi vardır. Bir dikdörtgenin köşe noktalarını bulmak karmaşık bir modelin denetleme noktalarını bulmaktan daha kolaydır. Öğrenme örneklerinin biraz değişebileceği küçük veri kümelerinde eğitim yapılacaksa, basit modellerin karmaşık modellere göre daha az oynamasını bekleriz: Basit modellerin bu yüzden *değişintisi* azdır. Öbür yandan, basit bir model daha çok varsayım yapar, daha az esnektir, ve öğrenilecek sınıf gerçekte basit değilse yetersiz kalabilir: Basit modelin bu yüzden *yanlılığı* daha fazladır. En uygun modeli bulmak, yanlılık ve değişintinin birlikte olabildiğince küçültülmesini gerektirir.

3. Açıklaması, anlaşılması kolay bir modeldir. Bir dikdörtgen, iki öznitelik için aralıklar tanımlamaya karşılık gelir. Basit bir modelle öğrenme kümesindeki ham veriden bilgi çıkartabiliriz.

4. Girdilerde gerçekten de yanlış etiketleme ya da gürültü varsa ve gerçek sınıf zaten dikdörtgen gibi basit bir modelse, karmaşık modele göre öğrenme kümesinde biraz daha çok hata yapsa da, daha az değişintisi olan ve gürültülü örneklerden daha az etkilenecek basit bir dikdörtgen daha iyi bir ayırtaç olacaktır. Gözlemsel hatalarının aşağı yukarı aynı olduğu durumlarda, (aşırı basit olmayan) basit bir modelin karmaşık bir modelden daha iyi *genelleme* yaptığını söyleyebiliriz. *Occam'ın usturası*, *Basit açıklamaların daha inandırıcı olduğunu* savunur ve gereksiz karmaşıklığın kesilip atılması gerektiğini belirtir.

2.5 Çok Sınıflı Öğrenme

Aile arabası sınıfını öğrenirken elimizde artı örnekler ve başka tip arabalara ait eksi örnekler vardı ve bunu *iki sınıflı* bir uygulama olarak tanımladık. Genel durumda, $C_i, i = 1, \ldots, K$, olarak gösterilen ve bir girdinin bunlardan sadece birine ait olduğu K farklı sınıfımız olabilir. Bu durumda öğrenme kümesi, r etiketinin K boyutlu



Şekil 2.9: Bu örnekte, aile arabası, spor araba, ve lüks araba olarak üç sınıf, ve bunlar için, herbiri bir sınıfın örneklerini kapsayan ve öteki iki sınıfın örneklerini dışarıda bırakan üç denence vardır. '?' ile gösterilen bölgeler, hiçbir sınıfın seçilmediği ya da birden çok sınıfın seçilebilir olduğu kuşkulu bölgelerdir.

olduğu ve

$$r_i^t = \begin{cases} 1 & \text{eğer } \boldsymbol{x}^t \in C_i \\ 0 & \text{eğer } \boldsymbol{x}^t \in C_j, j \neq i \end{cases} \tag{2.8}$$

olarak tanımlı olduğu örneklerden oluşur:

$$\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{r}^t\}_{t=1}^N$$

Şekil 2.9'da, aile arabası, spor araba, ve lüks araba olarak üç sınıftan örneklerin bulunduğu bir örnek verilmiştir.

Sınıflandırma yaparken bir sınıfı öteki tüm sınıflardan ayırt edecek bir sınır öğrenmek isteriz. Bu yüzden, K sınıfı ayırmayı K tane iki sınıf problemi olarak düşünebiliriz. Bu durumda, C_i sınıfından öğrenme örnekleri h_i denencesinin artı örnekleri ve öteki tüm sınıfların örnekleri de h_i denencesinin eksi örnekleri olur, yani bir K sınıf probleminde, bu biçimde K tane denence öğrenmemiz gerekir:

$$h_i(\boldsymbol{x}^t) = \begin{cases} 1 & \text{eğer } \boldsymbol{x}^t \in C_i \\ 0 & \text{eğer } \boldsymbol{x}^t \in C_j, j \neq i \end{cases} \tag{2.9}$$

Toplam gözlemsel hata, tüm sınıf ve örnekler üzerinden yapılan öngörü hatalarının toplamı olarak hesaplanır:

$$E(\{h_i\}_{i=1}^K | \mathcal{X}) = \sum_{t=1}^N \sum_{i=1}^K 1(h_i(\boldsymbol{x}^t) \neq r_i^t) \tag{2.10}$$

Verilen bir x için istenen yalnızca bir $h_i(x), i = 1, \ldots, K$ değerinin 1, ötekilerin 0 olmasıdır. Ancak hiçbirinin ya da birden çoğunun 1 olması durumunda seçim yapmak yerine bu *kuşkulu* durumda sınıflandırma yapmayı reddedebiliriz.

Aile arabası sınıfını öğrenme örneğinde tek bir denence kullanmış ve yalnızca artı örnekleri modellemiştik; dışarıda kalan eksi örnekler de aile arabası değildi. Bir başka olasılık olarak bazen biri artı, öteki eksi örnekler için olmak üzere iki ayrı denence öğrenmeyi yeğleyebiliriz. Bu, eksi örneklerin de kendine özgü bir yapısı olduğunu varsayar. Aile arabalarını spor arabalardan ayırmak buna bir örnektir; her sınıfın kendi yapısı vardır. Bunu yaptığımızda, örneğin girdi lüks bir arabaysa, her iki denence de olumsuz karar verir ve böylece girdiyi iki sınıftan birine atmak yerine reddedebiliriz.

Bir veri kümesinde, tüm sınıfların benzer dağılımları, yani girdi uzayında benzer biçimleri olmasını bekliyorsak, hepsi için aynı denence sınıfını kullanabiliriz. Örneğin, elle yazılmış rakamların tanınması uygulamasında tüm rakamların benzer dağılımlara sahip olmasını bekleriz. Ancak, bir tıbbi tanı veri kümesinde hastalar ve sağlıklılar olarak tanımlanmış iki sınıfın dağılımları çok farklı olabilir: Tüm sağlıklı insanlar birbirine benzer; hasta insanlar türlü türlüdür.

2.6 Bağlanım

Sınıflandırmada girdiye bağlı olarak üretilen çıktı mantıksal bir değişkendir; evet ya da hayırdır. Çıktı sayısal bir değer olduğunda öğrenmek istediğimiz bir $C(x) \in \{0, 1\}$ sınıfı değil, sayısal bir işlevdir. Yapay öğrenmede, işlevin ne olduğunu bilmesek de ondan seçilmiş örneklerden oluşan bir öğrenme kümemiz vardır:

$$\mathcal{X} = \{x^t, r^t\}_{t=1}^N$$

Sınıflandırmadan farklı olarak $r^t \in \Re$ olabilir. Gürültü yoksa yaptığımız *aradeğerlemedir*; öğrenme noktalarından geçen bir $f(x)$ işlevi bulmak isteriz:

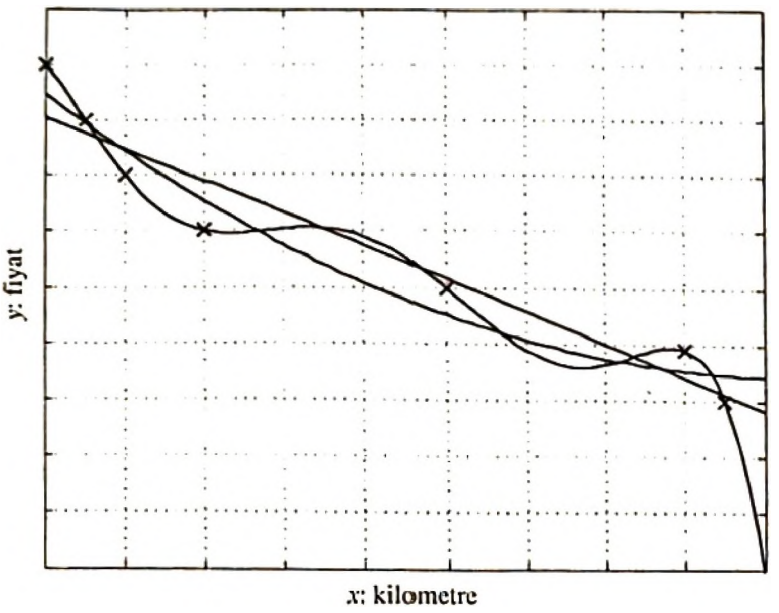
$$r^t = f(x^t)$$

Çokterimli aradeğerlemede, N nokta verildiğinde, bunlardan geçen $(N-1)$'. dereceden bir çokterimliyi bulup ardından başka bir x için çıktıyı öngörmede kullanabiliriz. Eğer x girdisi veri kümesindeki x^t'lerin değer aralığının dışındaysa, buna *dışdeğerleme* denir. Örneğin, zaman dizisi öngörüsünde şimdiye kadar gözlenen verileri öğrenme için kullanıp gelecekteki değerleri öngörebilmek isteriz. *Bağlanımda*, bilinmeyen $f(x) \in \Re$ işlevinin çıktısına rastgele ϵ gürültüsü eklenmiştir:

$$r^t = f(x^t) + \epsilon \tag{2.11}$$

Gürültü, gözlemleyemediğimiz z^t *saklı* değişkenlerinin etkisi olarak açıklanabilir:

$$r^t = f^*(x^t, z^t) \tag{2.12}$$



Şekil 2.10: Aynı nokta kümesini kullanarak eğitilmiş doğrusal, ikinci, ve altıncı dereceden çokterimliler. En yüksek dereceden olan en hatasız olandır ama bu kadar az örneğe dayanarak işlevin bu kadar karmaşık olduğunu varsaymak doğru değildir. İkinci derece çokterimli, doğrusal olana göre öğrenme kümesindeki eğilimi daha iyi yakalıyor gibi görünmektedir.

Modelimizi $g(\boldsymbol{x})$ ile gösterelim; amacımız öğrenme kümesindeki verileri yaklaşık kestirebilmektir. Öğrenme kümesi $\mathcal{X}$ üzerindeki *gözlemsel yaklaşıklama hatası*, örnekler üzerindeki hataların toplamıdır:

$$E(g|\mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}[r^t - g(\boldsymbol{x}^t)]^2 \tag{2.13}$$

Burada, r ve $g(\boldsymbol{x})$ sayısal değerler, örneğin, $\in \Re$, olduğundan bu değerler arasında fark ya da farkın karesi gibi bir uzaklık tanımlayabiliriz; bu bize sınıflandırmada kullanılan eşit ya da değil—0 ya da 1—türü karşılaştırmadan daha çok bilgi verir. Farkın karesi, kullanılabilecek hata (yitim) işlevlerinden biridir, bir diğeri farkın mutlak değeridir. Gelecek bölümlerde başka hata işlev örnekleri de göreceğiz.

Amacımız, gözlemsel hatayı en küçük yapan $g(\cdot)$ modelini bulmak ve yaklaşımımız yine aynıdır. Modelimiz $g(\cdot)$ için az sayıda parametresi olan bir denence sınıfı varsayacağız. Örneğin, $g(\boldsymbol{x})$ doğrusal olabilir:

$$g(\boldsymbol{x}) = w_1 x_1 + \cdots + w_d x_d + w_0 = \sum_{j=1}^{d} w_j x_j + w_0 \tag{2.14}$$

Şimdi, bölüm 1.2.3'te gördüğümüz ikinci el araba fiyatı kestirimi örneğimize geri dönelim. Orada tek girdili, w_1 ve w_0 parametrelerinin veriden öğrenildiği doğrusal

bir model kullanmıştık:

$$g(x) = w_1 x + w_0 \tag{2.15}$$

w_1 ve w_0 parametreleri toplam yaklaşıklama hatasını en küçük yapacak biçimde hesaplanır:

$$E(w_1, w_0|\mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}[r^t - (w_1 x^t + w_0)]^2 \tag{2.16}$$

Hata işlevinin parametrelere göre türevlerini sıfıra eşitleyip bu iki bilinmeyenli, iki denklemi çözerek w_1 ve w_0 değerlerini bulabiliriz:

$$\begin{aligned} w_1 &= \frac{\sum_t x^t r^t - \overline{xr}N}{\sum_t (x^t)^2 - N\overline{x}^2} \\ w_0 &= \overline{r} - w_1 \overline{x} \end{aligned} \tag{2.17}$$

öyle ki $\overline{x} = \sum_t x^t/N$ ve $\overline{r} = \sum_t r^t/N$ olarak tanımlıdır. Bulunan doğru, şekil 1.2'de gösterilmiştir.

Doğrusal model çok basit kalırsa yaklaşıklama hatası yüksek olabilir; böyle bir durumda çıktıyı girdinin daha yüksek dereceden, örneğin karesel bir işlevi olarak yazabiliriz:

$$g(x) = w_2 x^2 + w_1 x + w_0 \tag{2.18}$$

Yine w_2, w_1, ve w_0 benzer biçimde hesaplanabilir. Çokterimlinin derecesi arttıkça elimizdeki model daha esnek olacak, öğrenme noktalarına yakın geçebilmek için eğilip bükülebilecek ve dolayısıyla öğrenme kümesi üzerindeki yaklaşıklama hatası azalacaktır. Ancak, yüksek dereceden bir çokterimli, verideki genel eğilimi yakalamak yerine, örneğin şekil 2.10'daki altıncı dereceden çokterimli gibi örneklerin çok yakınından geçebilir. Bu, Occam'ın usturasının bağlanımda da uygulanması gerektiğini ve model karmaşıklığını verinin altında yatan işlevin karmaşıklığına uyacak şekilde ayarlarken dikkatli olmamız gerektiğini gösterir.

2.7 Model Seçimi ve Genelleme

Gelin şimdi, mantıksal bir işlevi örneklerden öğrenmeyle başlayalım. Mantıksal bir işlevde tüm girdi ve çıktılar iki değer alır. d tane ikili değeri yazmanın 2^d farklı yolu olduğu için d girdi varsa bir öğrenme kümesinde en fazla 2^d örnek bulunabilir. Tablo 2.1'de görüldüğü gibi, bunların her biri 0 ya da 1 olarak etiketlenebileceğinden, d girdi için 2^{2^d} farklı mantıksal işlev (ya da öğrenme problemi) tanımlanabilir.

Her öğrenme örneğinde denencelerin yarısı yanlış öngörü yapar ve yanlış yapanlar elenebilir. Örneğin, $x_1 = 0$, $x_2 = 1$ ve çıktı 0 olduğunda h_5, h_6, h_7, h_8, h_{13}, h_{14}, h_{15}, h_{16}, bu koşulu sağlayamadığından budanır. Öğrenmeyi bu biçimde de açıklayabiliriz: olası tüm denencelerle başlayıp öğrenme örneklerini gördükçe bunlarla tutarlı olmayan denenceleri atarız. Mantıksal işlev örneğinde, tek bir denence kalması için tüm 2^d öğrenme örneklerini görmemiz gerekir. Eğer öğrenme kümesi olası tüm örnekler

Tablo 2.1: İki girdi için olası dört durum ve olası onaltı mantıksal işlev vardır.

x_1	x_2	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}	h_{12}	h_{13}	h_{14}	h_{15}	h_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

yerine bir altkümesini içeriyorsa, ki genelde böyle olur, yani eğer çıktıların ne olması gerektiğini olası durumların yalnızca küçük bir kısmı için biliyorsak tek çözüm elde edemeyiz. N örnek gördükten sonra geriye hâlâ 2^{2^d-N} olası işlev kalır. Bu, *kötü konumlanmış bir problemdir*, yani yalnızca veri, tek bir çözüm elde etmek için yeterli değildir.

Aynı sorun başka öğrenme uygulamalarında, sınıflandırma ve bağlanımda da kendini gösterebilir. Daha çok öğrenme örneği gördükçe verinin altında yatan işlev hakkında daha çok bilgi sahibi olur ve daha çok sayıda denenceyi tutarsız bulup eleyebiliriz, ama yine de geriye birçok tutarlı denence kalabilir.

Öğrenme kötü konumlanmış bir problem ve veri kendi başına çözüme ulaşmak için yeterli olmadığından, elimizdeki veriyle tek bir çözüm elde edebilmek için bazı ek varsayımlar yapmamız gerekir. Öğrenmeyi gerçeklenebilir kılmak için yaptığımız bu tür varsayımlara öğrenme algoritmasının *model varsayımları* denir. Model varsayımlarının bir türü, bir $\mathcal{H}$ denence sınıfı seçerek olası denenceleri kısıtlamaktır. Aile arabası sınıfını öğrenirken artı örnekleri eksilerden ayırmanın sonsuz sayıda yolu vardır. Dikdörtgen denence sınıfını ve ardından örneğin, en dar dikdörtgeni seçmek çözümü teke indirmek için yaptığımız varsayımlardır. Bağlanımda, doğrusal bir işlev kullanmak bir model varsayımı olduğu gibi olası tüm doğrular içinden kare hatayı en aza indireni seçmek de başka bir model varsayımıdır.

Ama her denence sınıfının bir sığımı vardır ve ancak belirli işlevleri öğrenebilir. Öğrenilebilecek işlevler sınıfı, sığımı daha çok ve daha karmaşık denenceler içeren bir denence sınıfı kullanarak genişletilebilir. Örneğin, iki dikdörtgenin birleşiminden oluşan denence sınıfının sığımı daha çoktur ama denenceleri de daha karmaşıktır. Bağlanımda da aynı biçimde çokterimlinin derecesini artırdıkça sığım ve karmaşıklık artar. Şimdi soru nerede duracağımızı bilebilmektir.

Model varsayımları olmadan öğrenme yapılamayacağına göre önemli olan bu varsayımların nasıl doğru seçileceğidir. Buna *model seçimi* denir ve olası $\mathcal{H}$ denence sınıfları içinden doğru olanı seçmektir. Bu seçimi yaparken, yapay öğrenmenin amacının öğrenme kümesini kopya etmek olmadığını ve istenenin yeni durumlar için doğru öngörü yapmak olduğunu aklımızda tutmamız gerekir. Yani, öğrenme kümemizde olmayan girdi örnekleri için de doğru çıktılar üretebilmek isteriz. Öğrenme kümesinde eğitilen bir modelin yeni örneklerin çıktılarını ne kadar iyi öngördüğüne *genelleme* denir.

En iyi genelleme için veri altında yatan işlevin karmaşıklığının denence sınıfımızın karmaşıklığına eş olması gerekir. Eğer $\mathcal{H}$ gerçek işlevden daha basit olursa, örneğin, üçüncü dereceden bir çokterimliden örneklenmiş bir veriye doğru oturtmaya çalışırsak, buna *eksik öğrenme* denir. Böylesi bir durumda karmaşıklığı artırdıkça öğrenme hatası düşer. Ama $\mathcal{H}$ gerekenden karmaşık olursa veri denenceyi doğru biçimde kısıtlamak için yeterli olmayabilir ve kötü bir $h \in \mathcal{H}$ elde edebiliriz; bir dikdörtgenden örneklenmiş veriye iki dikdörtgen oturtmak buna örnek olarak verilebilir. Ya da gürültü olduğunda, aşırı karmaşık bir denence verinin altında yatan işlev yanında içindeki gürültüyü de öğrenir, ve örneğin, üçüncü dereceden bir çokterimliden örneklenmiş gürültülü veriye altıncı dereceden bir çokterimli oturtunca olacağı gibi, kötü bir öğrenme yapar. Buna *aşırı öğrenme* denir. Böyle bir durumda, daha çok öğrenme verisi denenceyi istediğimiz gibi kısıtlamaya bir noktaya dek yardımcı olur. Bir öğrenme kümesi ve $\mathcal{H}$ verildiğinde, en az öğrenme hatasına sahip olan $h \in \mathcal{H}$ denencesini bulabiliriz ama $\mathcal{H}$ iyi seçilmemişse, hangi $h \in \mathcal{H}$ seçilirse seçilsin genelleme iyi olmaz.

Bu konuyu üçlü ödünleşimden (Dietterich, 2003) bahsederek özetleyelim. Örnek veriyle eğitilen tüm öğrenme algoritmalarında şu üç etmen arasında ödünleşim vardır:

- veriye oturtulan denencenin karmaşıklığı, yani denence sınıfının sığımı,
- öğrenme verisi miktarı, ve
- yeni örneklerdeki genelleme hatası.

Öğrenme kümesi büyüdükçe genelleme hatası azalır. Denence sınıfının karmaşıklığı arttıkça genelleme hatası önce düşer ve sonra artmaya başlar. Aşırı karmaşık bir $\mathcal{H}$ kullanırken öğrenme kümesini büyütmek genelleme hatasını bir noktaya kadar denetim altında tutma olanağı sağlar. Eğer veri, bir doğrudan örneklendiyse ama denence olarak daha üst dereceden bir çokterimli kullanırsak model, öğrenme örneklerinin bulunduğu yerlerde doğruya yakın kalacak biçimde kısıtlanacak, ama veri olmayan yerlerde düzensiz davranabilecektir.

Öğrenme kümesi dışında başka veriye erişimimiz varsa bir denencenin genelleme başarısını, yani model varsayımlarının iyiliğini, bu veri üzerindeki başarıma göre ölçebiliriz. Bunu öğrenme kümesini iki bölüme ayırarak da yapabiliriz. Birinci bölümü eğitim için kullanır ve denence sınıfı içinden en düşük hatalı denenceyi buluruz. *geçerleme kümesi* denilen öteki bölümü de genelleme bu farklı denencelerin başarısını ölçmek için kullanırız. Yani, öğrenme kümesi üzerinde olası farklı $\mathcal{H}_i$ denence sınıflarının herbirisi içinden en iyi $h_i \in \mathcal{H}_i$ denencesini bulur ve bunların hepsinin geçerleme kümesi üzerindeki hatalarını hesaplarız. Öğrenme ve geçerleme kümelerinin yeterince büyük olduklarını varsayarsak geçerleme kümesinde en başarılı olan denence, varsayımları en iyi olan denence sınıfındandır. Bu sürece *çapraz geçerleme* denir. Örneğin, çokterimli bağlanımda, verilen farklı dereceden çokterimliler farklı $\mathcal{H}_i$ denence sınıflarına karşılık gelir; doğru dereceyi bulmak için öğrenme kümesini kullanarak her dereceden en iyi çokterimliyi hesaplar, ardından geçerleme kümesi üzerindeki hatalarına bakar, ve en az geçerleme hatası yapanı en iyi çokterimli derecesi olarak belirleriz. Ardından tüm veriyi kullanarak bu dereceli bir çokterimliyi en son modelimiz olarak hesaplarız.

Burada şu noktaya dikkat etmemiz gerekir: Eğer bu biçimde bulduğumuz en iyi modelin beklenen hatasını belirtmemiz gerekirse bunun için geçerleme başarısını kullanamayız. Geçerleme kümesini en iyi modeli seçmekte kullandığımız için o veri artık öğrenme kümesinin bir parçası olmuştur. Sonuç başarımı kestirmek için eğitim ve geçerlemede kullanılmayan örnekler içeren, bazen *yayımlama kümesi* diye de adlandırılan üçüncü, ayrı bir sınama kümesine gerek duyarız. Bu okulda bir ders almaya benzer: Öğretmenin ders anlatırken sınıfta çözdüğü örnek sorular öğrenme kümesini, sınav soruları geçerleme kümesini, ve daha sonra meslek hayatımızda karşılaştığımız sorular da sınama kümesini oluşturur.

Sürekli aynı öğrenme ve geçerleme kümelerini kullamayız, çünkü bir kere kullanıldığında geçerleme kümesi öğrenme kümesinin parçası olur. Bu da sınavlarında her yıl aynı soruları soran öğretmene benzer; kurnaz bir öğrenci derslerle uğraşmaya gerek olmadığını anlayıp bu soruların yanıtlarını ezberleyecektir.

Kullandığımız öğrenme kümesinin rastgele bir örneklem olduğunu hep aklımızda tutmalıyız. Aynı uygulamadan bir kez daha veri topladığımızda biraz farklı bir veri kümesi elde eder, biraz farklı bir denence h öğrenir ve biraz farklı bir geçerleme hatası bulabiliriz. Sabit bir veri kümemiz olsa bile, bunu öğrenme, geçerleme, ve sınama kümelerine nasıl böldüğümüze göre farklı hata değerleri oluşabilir. Hatalardaki bu ufak değişiklikler, ne kadar farklılığın anlamlı ve önemli, ne kadarınınsa rastlantısal olduğunu kestirebilmemizi sağlar. Dolayısıyla iki denence sınıfı, $\mathcal{H}_i$ ve $\mathcal{H}_j$, arasından birini seçerken, her ikisini de birden çok öğrenme ve geçerleme kümesinde kullanıp h_i ve h_j'lerin ortalama hataları arasındaki farkın h_i'lerin kendi içindeki ortalama farkından büyük olup olmadığına bakmamız gerekir. 19. bölümde, yapay öğrenme deneylerinin tasarımını, bu deneylerin sonuçlarını nasıl çözümleyip istatistiksel anlam taşıyan ve rastsallıktan en az düzeyde etkilenen yargılara varabileceğimizi ve bu bağlamda, örneğin, hangi denence sınıfının en iyisi olduğu gibi soruları nasıl yanıtlayabileceğimizi tartışacağız.

2.8 Gözetimli Yapay Öğrenmenin Boyutları

Şimdi önemli noktaları bir kez daha gözden geçirip genelleyelim. Her şey bir örnek kümesiyle başlar:

$$\mathcal{X} = \{x^t, r^t\}_{t=1}^N \tag{2.19}$$

Örneklem, *bağımsız ve özdeşçe dağılmıştır*; yani, örneklerin küme içindeki sırası önemli değildir ve tüm örnekler aynı $p(x, r)$ birleşik dağılımından örneklenmiştir. Dizin değişkeni t, N örnekten birini gösterir. x^t girdi ve r^t ilgili istenen çıktı değeridir. r^t, iki sınıf öğrenmede 0/1, $K > 2$ sınıflı sınıflandırmada K boyutlu (yalnızca bir boyutun 1 ötekilerin 0 olduğu) ikili bir yöney, ve bağlanımda gerçel bir değerdir.

Amaç, $g(x^t|\theta)$ modelini kullanarak r^t değerine yakın ve kullanışlı bir yaklaşım yapmaktır. Bunu yaparken üç boyuta karar vermemiz gerekir:

1. Öğrenmede kullanacağımız *model*

$$g(x|\theta)$$

ki burada, $g(\cdot)$ ile modeli, x ile girdiyi, ve θ ile de model parametrelerini gösteriyoruz.

Bir $g(\cdot)$ modeli seçtiğimizde $\mathcal{H}$ denence sınıfını tanımlarız ve bu sınıf içinde her θ değeri farklı bir $h \in \mathcal{H}$ denencesine karşılık gelir. Sınıf öğrenmede, örneğin, bir dikdörtgeni model olarak aldığımızda θ, dört köşenin koordinatlarından oluşur. Bağlanımda model, örneğin, eğim ve kesme noktası veriden öğrenilen parametreler olan doğrusal bir işlev olarak seçilebilir. Modeli (ve varsayımlarını), bir başka deyişle $\mathcal{H}$ denence sınıfını, yapay öğrenme tasarımcısı, uygulama hakkında bilgilerini de gözönüne alarak belirler. Öğrenme algoritması verilen bir $\mathcal{H}$ içinden $p(x, r)$'den örneklenen öğrenme kümesi üzerinde en uygun h denencesini seçer, bir başka deyişle en iyi θ parametrelerini bulur.

2. *Yitim işlevi*, $L(\cdot)$, istenen r^t çıktısıyla $g(x^t|\theta)$ öngörümüz arasındaki farkı hesaplar. *Yaklaşıklama hatası*, her örnekteki yitimlerin toplamıdır:

$$E(\theta|\mathcal{X}) = \sum_t L(r^t, g(x^t|\theta)) \tag{2.20}$$

Sınıflandırmada çıktılar 0/1 olduğu için $L(\cdot)$ bu iki değerin eşit olup olmadığına bakar; bağlanımda çıktı sayısal bir değerdir ve uzaklıklar hakkında sıralama bilgisi olduğundan fark(ın karesi) kullanılabilir.

3. *İyileme yordamı*, toplam hatayı en aza indirecek θ^* parametrelerini bulmayı amaçlar:

$$\theta^* = \arg\min_\theta E(\theta|\mathcal{X}) \tag{2.21}$$

arg min en aza indiren değişken değerini döndürür. Çokterimli bağlanımda en iyi değeri çözümsel olarak bulabiliriz. Daha karmaşık modeller ve hata işlevleri için eğim tabanlı yöntemler, benzetimli tavlama, ya da genetik algoritmalar gibi daha karmaşık iyileme yöntemleri kullanmamız gerekebilir. Örnekleri gelecek bölümlerde göreceğiz.

Öğrenmenin başarılı olabilmesi için şu koşulların sağlanması gerekir: İlk olarak, $g(\cdot)$ modelinin çekildiği denence sınıfı, verinin altında yatan ve r^t'de gürültü eklenmiş olarak kendini gösteren işlevi içerecek kadar geniş, yani yeterli sığıma sahip olmalıdır. İkinci olarak, denence sınıfı içinden doğru (ya da yeterince doğru) bir denenceyi saptamaya yetecek kadar öğrenme verisi olmalıdır. Üçüncü ve son olarak, öğrenme verisi verildiğinde doğru denenceyi bulabilecek başarılı bir iyileme yordamımız olmalıdır.

Gelecek bölümlerde birçok örneğini göreceğimiz farklı yapay öğrenme algoritmaları, ya varsaydıkları modellerde (denence sınıfları ya da model varsayımlarında), ya kullandıkları yitim ölçütlerinde, ya da kullandıkları iyileme yordamlarında birbirlerinden farklılık gösterir.

2.9 Kısa Kısa

Mitchell, sürüm uzayı kavramını ve örnekler birer birer görüldüğünde S ve G'nin artımlı olarak oluşturulduğu aday eleme algoritmasını önermiştir (Mitchell 1997). Dikdörtgen öğrenme, (Mitchell 1997) alıştırma 2.4'te verilmiştir. Hirsh (1990), örnekler gürültüyle değiştiğinde sürüm uzayının nasıl kullanılacağını ele alır.

Yapay öğrenme üzerine en eski çalışmalardan birinde, Winston (1975) "kılpayı örnek" kavramını kullanır. Kılpayı örnek, artıya çok benzeyen bir eksi örnektir. Bizim terimlerimizle ifade edecek olursak kılpayı örnek, S ile G arasındaki gri bölgeye düşen ve kenar payını daraltan bir örnektir, ve bu yüzden öğrenme sırasında sıradan artı ve eksi örneklere göre daha faydalıdır. Sınıra yakın örnekler, sınırı tanımlayan (ya da destekleyen) örneklerdir; aynı etiketli birçok örnek tarafından kuşatılmış örneklerin eklenip çıkartılması sınırı etkilemez.

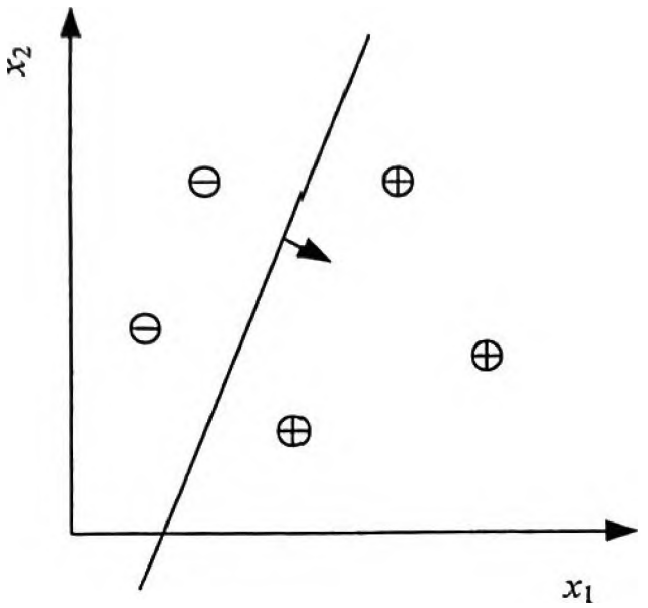
Etkin öğrenme de benzer bir düşünceye dayanır. Öğrenme algoritması örnekleri edilgen bir biçimde beklemek yerine kendisi örnekler oluşturup gözetmenden onları etiketlenmesini ister (Angluin 1988) (bkz. alıştırma 4).

VC boyutu, Vapnik ve Chervonenkis tarafından 1970'lerde ileri sürülmüştür. Konu üzerine güncel bir kaynak olan (Vapnik, 1995; s. x)'da Vapnik, "Hiçbir şey pratikte iyi bir kuramdan daha yararlı değildir," diye yazmıştır ve bu, bilimin her dalında olduğu gibi yapay öğrenmede de doğrudur. Hemen bilgisayar karşısına geçip çalakalem program yazmaya başlamamak gerekir; biraz düşünme, bir kalem, bir defter, ve belki bir silgi, sizi saatlerce faydasız programlama yapmaktan kurtaracaktır.

Olası yaklaşık doğru modelini Valiant (1984) önermiştir. Dikdörtgen öğrenmenin bu biçimde çözümlemesi (Blumer ve ötekiler 1989)'da verilmiştir. Bu konuyu ve VC boyutunu içeren hesaplamalı öğrenme kuramı üzerine iyi bir ders kitabı için bkz. (Kearns ve Vazirani 1994).

2.10 Alıştırmalar

1. Denence sınıfı bir dikdörtgen yerine bir çember olsa parametreler ne olur? Çember denencesinin parametreleri nasıl hesaplanabilir? Ya elips olursa? Neden elips, çemberden daha uygundur? Çözümünüzü $K > 2$ sınıfa nasıl genelleyebilirsiniz?
2. Denencemizin bir dikdörtgen yerine iki (ya da $m > 1$) dikdörtgenin birleşimi olduğunu düşünün. Böyle bir denence sınıfının getirileri nedir? Böyle bir denence sınıfı ve yeterince büyük bir m değeriyle herhangi bir sınıfın öğrenilebileceğini gösterin.
3. Çoğu öğrenme algoritmasının karmaşıklığı öğrenme kümesinin büyüklüğüne bağlıdır. Gereksiz öğrenme örneklerini bulup atacak bir süzme algoritması önerebilir misiniz?
4. Bize herhangi bir x girdisinin etiketini verebilecek bir gözetmenimiz olsa, en az sayıda sorgulamayla öğrenmek için x girdilerini nasıl seçmemiz gerekir?
5. Denklem 2.13'te istenenle kestirilen değerler arasındaki farkların karelerini topladık. Bu hata işlevi, olası farklı hata işlevleri içinden en yaygın kullanılandır, ama farkların karelerini topladığı için aykırı gözlemlere karşı gürbüz değildir. *Gürbüz bağlanım* için nasıl bir hata işlevi daha uygun olur?



Şekil 2.11: Artı ve eksi örnekleri ayıran bir doğru.

6. Denklem 2.17'yi türetin.
7. Artı örnekleri çevreleyip eksi örnekleri dışarıda bırakan dikdörtgenler yerine, denence sınıfımızın doğrulardan oluştuğunu ve artı örnekleri eksilerden ayırmak için bir doğru kullandığımızı (bkz. Şekil 2.11) varsayalım. İki boyutta bir doğrunun VC boyutunun 3 olduğunu gösterin.
8. İki boyutta üçgen denence sınıfının VC boyutunun 7 olduğunu gösterin. (İpucu: En iyi ayrım için noktaları bir çember üzerine eşit uzaklıklarla yerleştirin.)
9. Alıştırma 7'deki gibi denence sınıfımızın bir doğrular kümesi olduğunu varsayalım. Hatalı sınıflandırmaları en aza indirmekle kalmayıp sınır payını da olabildiğince büyütecek bir hata işlevi tanımlayın.
10. Gürültünün bir kaynağı da etiketlerdir. Yanlış etiketlenmiş veri noktalarını kestirmek için bir yöntem önerebilir misiniz?

2.11 Kaynaklar

Angluin, D. 1988. "Queries and Concept Learning." *Machine Learning* 2: 319–342.

Blumer, A., A. Ehrenfeucht, D. Haussler, M. K. Warmuth. 1989. "Learnability and the Vapnik-Chervonenkis Dimension." *Journal of the ACM* 36: 929–965.

Dietterich, T. G. 2003. "Machine Learning." *Nature Encyclopedia of Cognitive Science.* London: Macmillan.

Hirsh, H. 1990. *Incremental Version Space Merging: A General Framework for Concept Learning.* Boston: Kluwer.

Kearns, M. J., U. V. Vazirani. 1994. *An Introduction to Computational Learning Theory.* Cambridge, MA: MIT Press.

Mitchell, T. 1997. *Machine Learning*. New York: McGraw-Hill.

Valiant, L. 1984. "A Theory of the Learnable." *Communications of the ACM* 27: 1134–1142.

Vapnik, V. N. 1995. *The Nature of Statistical Learning Theory*. New York: Springer.

Winston, P. H. 1975. "Learning Structural Descriptions from Examples." *The Psychology of Computer Vision*, ed. P. H. Winston, 157–209. New York: McGraw-Hill.

Bölüm 3

Bayesçi Karar Kuramı

Belirsizlik durumunda olasılık kuramını göre karar veririz. Sınıflandırmada Bayes kuralını sınıfların olasılıklarını hesaplamakta kullanıyoruz. Bunu genelleyip birçok eylem arasından beklenen riski en aza indirecek biçimde nasıl seçim yapabileceğimizi konuşacağız. Tartışacağımız bir başka konu da ilişkilendirme kurallarının veriden nasıl öğrenilebileceği olacak.

3.1 Giriş

Bilgisayarların veriden çıkarım yapacak biçimde programlanması istatistik ve bilgisayar bilimlerinin kesişimindedir; istatistikçiler veri çözümlemenin matematiksel çatısını ortaya koyarken bilgisayarcılar bu yöntemlerin etkin ve verimli gerçeklemesi üzerinde çalışır.

Veri tam olarak bilinmeyen bir süreçten gelmektedir. Bu bilgi eksikliği, modellemenin rastsal bir süreçle yapılmasıyla gösterilir. Süreç gerçekte belirlenimci olabilir, ama hakkında tam bilgi sahibi olmadığımız için rastsal olarak modeller ve olasılık kuramıyla inceleriz. Bu bölümü okumaya devam etmeden önce kitabın sonundaki eke atlayıp temel olasılık kuramını gözden geçirmek isteyebilirsiniz.

Yazı tura atmak rastsal bir süreçtir; atmadan önce sonucun yazı mı tura mı geleceğini bilemeyiz. Zaten bu yüzden yazı tura atarız, piyango bileti alırız, ya da arabamızı sigortalarız. Sonucu bilmeyiz ama yazı ya da tura gelme olasılığından bahsedebiliriz. Belki paranın başlangıç konumunu, atış kuvvetini ve uygulama yönünü, nerede ve nasıl yakalandığını ve tüm öteki etkili etkenleri biliyor olsak sonucun ne olacağını öngörebileceğimiz söylenebilir, ama bilmediğimiz için sonuç bizce rastsaldır.

Erişimimiz olmayan böylesi ek bilgilere *gözlenemeyen değişkenler* denir. Yazı tura örneğinde tek *gözlenebilir değişken* paranın ne geldiğidir. Gözlenemeyenleri $\boldsymbol{z}$, gözlenebileni x ile gösterecek olursak $f(\cdot)$, gözlenemeyenlerden çıktıyı tanımlayan belirlenimci bir işlevdir:

$$x = f(\boldsymbol{z})$$

Süreci bu biçimde modelleyemeyeceğimiz için X çıktı değerini, süreci tanımlayan bir $P(X = x)$ olasılık dağılımından seçilmiş rastsal bir değişken olarak gösteririz.

Para atmanın sonucu olan yazı ya da turayı iki farklı değer alan rastsal bir değişkenle gösteririz: Yazı geldiğinde $X = 0$, tura geldiğinde $X = 1$ olsun. Böylesi X değişkenleri, p_o parametreli Bernoulli bir dağılımından çekilir; p_o, örneğin, paranın tura gelme olasılığıdır:

$$P(X = 1) = p_o \text{ ve } P(X = 0) = 1 - P(X = 1) = 1 - p_o$$

Bir sonraki atışı öngörmek istiyor olalım. Eğer p_o değerini biliyorsak öngörümüz, $p_o > 0.5$ ise tura, değilse yazı olacaktır. Çünkü daha olası durumu seçtiğimizde hata, yani 1 eksi seçimimizin olasılığı, en düşük olur. $p_o = 0.5$ ise ve para hilesizse hep yazı seçmek ya da sonucu belirlemek için yine yazı tura atmaktan daha iyi bir yol yoktur!

Eğer $P(X)$ dağılımını bilmiyor ve verilen bir örneklemden kestirmek istiyorsak istatistiğin alanına girmiş oluruz. *Örneklem* $\mathcal{X}$, bilinmeyen $p(x)$ olasılık dağılımından örneklenmiş x^t gözlemlerini içerir. Amacımız, bu örneklemi kullanarak $\hat{p}(x)$ ile gösterilen yaklaşık bir dağılım bulmaktır.

Para atma örneğinde örneklem, geçmiş N atışın sonucunu içerir. Bu örneklemi kullanarak, dağılımın tek parametresi olan p_o değerini kestirebiliriz:

$$\hat{p}_o = \frac{\#\{\text{sonucu tura olan atışlar}\}}{\#\{\text{atışlar}\}}$$

Rastsal değişkenleri kullanıp sonuçları sayılarla gösterelim: Eğer t'inci atış yazı geldiyse $x^t = 0$, tura geldiyse $x^t = 1$ olsun. Bu durumda örneğin, {tura, tura, tura, yazı, tura, yazı, yazı, tura, tura} örneklemi, $\mathcal{X} = \{1, 1, 1, 0, 1, 0, 0, 1, 1\}$ olarak gösterilir ve kestirimimizi bunu kullanarak hesaplayabiliriz:

$$\hat{p}_o = \frac{\sum_{t=1}^{N} x^t}{N} = \frac{6}{9}$$

3.2 Sınıflandırma

Kısım 1.2.2'de kredi derecelendirme konusundan bahsetmiştik. Bir bankanın geçmiş kayıtlarına bakıp borcunu ödemiş müşterileri düşük riskli, ödeyememiş müşterileri de yüksek riskli olarak etiketleyebiliriz. Ardından bu veriyi inceleyerek "yüksek riskli müşteri" sınıfını öğrenebilirsek, gelecekte yeni bir kredi başvurusu yapıldığında başvurunun bu tanıma uyup uymadığına bakarak kabul ya da red kararı verebiliriz. Müşterinin kredi derecesi hakkında bilgi verdiğine inandığımız bilgilerden ikisini, örneğin, müşterinin yıllık gelirini ve birikimlerinin toplamını girdi olarak kullandığımızı varsayalım ve bunları sırasıyla X_1 ve X_2 rastsal değişkenleriyle gösterelim.

Burada yine, eğer müşteriyi ve ülke ekonomisini tüm ayrıntılarıyla bilebilsek başvurunun düşük ya da yüksek riskli olduğunu belirlenimci olarak hesaplayabileceğimiz öne sürülebilir. Ancak bunlar gözlenemediği için başvuruyu ancak gözleyebildiğimiz bilgilere göre değerlendirebiliriz. Kredi riskini, $\boldsymbol{X} = [X_1, X_2]^T$ gözlenebilirlerine koşullu rastsal bir Bernoulli değişkeni C olarak gösterelim. Örneğin, $C = 1$ yüksek riskli, ve $C = 0$ düşük riskli müşteriyi belirtsin. Böylelikle $P(C|X_1, X_2)$' koşullu olasılığını bilirsek, $X_1 = x_1$ ve $X_2 = x_2$ olan yeni bir başvuru geldiğinde hangi sınıfın olasığının daha büyük olduğuna göre karar verebiliriz:

$$\text{Seç} \begin{cases} C = 1 & \text{eğer } P(C = 1|x_1, x_2) > 0.5 \\ C = 0 & \text{değilse} \end{cases}$$

ya da

$$\text{seç} \begin{cases} C = 1 & \text{eğer } P(C = 1|x_1, x_2) > P(C = 0|x_1, x_2) \\ C = 0 & \text{değilse} \end{cases} \tag{3.1}$$

Hata olasılığı, $1 - \max(P(C = 1|x_1, x_2), P(C = 0|x_1, x_2))$ olur. Bunun yazı tura örneğinden tek farkı C rastsal Bernoulli değişkeninin başka iki gözlenebilir değişkene koşullu olmasıdır. Gözlenebilir değişkenleri $\boldsymbol{x} = [x_1, x_2]^T$ yöneyiyle gösterdiğimizde yapmamız gereken $P(C|\boldsymbol{x})$ olasılığını hesaplayabilmektir. Bunu *Bayes kuralıyla* yapabiliriz:

$$P(C|\boldsymbol{x}) = \frac{P(C)p(\boldsymbol{x}|C)}{p(\boldsymbol{x})} \tag{3.2}$$

$P(C = 1)$, *önsel olasılık* olarak adlandırılır ve örneğimizde $C = 1$, yani müşterinin yüksek riskli olması olasılığına karşılık gelir. Önsel olasılık denmesinin nedeni de $\boldsymbol{x}$ değişkenlerini gözlemeden *önce* C hakkındaki bilgimizi göstermesidir. Öncel olasılıklar toplamı birdir:

$$P(C = 0) + P(C = 1) = 1$$

$p(\boldsymbol{x}|C)$, *sınıf olabilirliğidir* ve C bilindiğinde gözlenebilirlerin olasılığını belirtir. Örneğimizde $p(x_1, x_2|C = 1)$, yüksek riskli bir müşterinin $X_1 = x_1$ ve $X_2 = x_2$ değeri alması olasılığıdır. Sınıf olabilirlik değerleri, verinin bize sınıf hakkında verdiği bilgidir.

$p(\boldsymbol{x})$, *kanıt*, $\boldsymbol{x}$ değerinin gözlemlesinin artı ve eksi örnekler üzerinden tümleştirilmiş olasılığıdır:

$$p(\boldsymbol{x}) = \sum_C p(\boldsymbol{x}, C) = p(\boldsymbol{x}|C = 1)P(C = 1) + p(\boldsymbol{x}|C = 0)P(C = 0) \tag{3.3}$$

Bayes kuralını kullanarak önsel olasılıkla veriden gelen bilgiyi birleştirir ve $\boldsymbol{x}$ değişkeninin gözlenmesi sonrasında oluşan $P(C|\boldsymbol{x})$ *sonsal olasılıklarını* hesaplarız:

$$\text{sonsal} = \frac{\text{öncel} \times \text{olabilirlik}}{\text{kanıt}}$$

Kanıta bölünerek normalleştirildiği için sonsal olasılıkların toplamı 1 olur:

$$P(C = 0|\boldsymbol{x}) + P(C = 1|\boldsymbol{x}) = 1$$

Elimizde sonsal olasılıklar olduğunda denklem 3.1'e göre karar verir, yani olasılığı en büyük sınıfı seçeriz. Şimdilik önsel olasılıkları ve sınıf olabilirliklerini bildiğimizi varsayıyoruz; $P(C)$ ve $p(\boldsymbol{x}|C)$ dağılımlarını verilen bir öğrenme örnekleminden nasıl kestireceğimizi ileriki bölümlerde ele alacağız.

Genel durumda, K tane, $C_i, i = 1, \ldots, K$ ile gösterdiğimiz ayrık ve kapsayan sınıfımız vardır. Örneğin, görsel rakam tanımada, girdi ikili bir görüntüdür ve on sınıf vardır. Önsel olasıkların toplamı her zaman birdir:

$$P(C_i) \geq 0 \text{ ve } \sum_{i=1}^{K} P(C_i) = 1 \tag{3.4}$$

$p(\boldsymbol{x}|C_i)$, C_i sınıfından örnekler içinde girdisi $\boldsymbol{x}$ olan bir örnek görme olasılığıdır. Sonsal olasılıklar yine Bayes kuralıyla hesaplanır:

$$P(C_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_i)P(C_i)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|C_i)P(C_i)}{\sum_{k=1}^{K} p(\boldsymbol{x}|C_k)P(C_k)} \tag{3.5}$$

ve *Bayes sınıflandırıcısı*, en az hata için sonsal olasılığı en yüksek sınıfı seçer:

$$\text{seç } C_i \text{ eğer } P(C_i|\boldsymbol{x}) = \max_k P(C_k|\boldsymbol{x}) \tag{3.6}$$

3.3 Yitimler ve Riskler

Tüm doğru kararlar aynı iyilikte, tüm yanlışlar aynı kötülükte olmayabilir. Bir finans kurumu bir kredi başvurusunu değerlendirirken olası kazanım ve yitimleri de göz önüne almalıdır. Başvurusu kabul edilen düşük riskli bir müşteri kârı artırırken, yüksek riskli bir müşterinin reddedilen başvurusu yitimi azaltır. Yüksek riskli bir başvurunun yanlışlıkla kabul edilmesinden doğacak zarar (yitim), düşük riskli bir başvurunun yanlışlıkla reddedilmesiyle kaybedilen kârdan farklı olabilir. Bu durum, tıbbi tanı ya da deprem öngörüsü gibi alanlarda çok daha önemli ve bakışımlılıktan uzaktır.

Girdinin C_i sınıfına atanması eylemini α_i, girdi C_k sınıfındanken α_i eyleminin seçilmesiyle oluşan *yitimi* λ_{ik} ile gösterelim. Bu durumda, α_i eyleminin seçilmesinden kaynaklanacak *beklenen risk*, olası tüm sınıflar üzerinden yitimleri toplayarak bulunur:

$$R(\alpha_i|\boldsymbol{x}) = \sum_{k=1}^{K} \lambda_{ik} P(C_k|\boldsymbol{x}) \tag{3.7}$$

ve ardından beklenen riski en düşük eylem seçilir:

$$\text{seç } \alpha_i \text{ eğer } R(\alpha_i|\boldsymbol{x}) = \min_k R(\alpha_k|\boldsymbol{x}) \tag{3.8}$$

K tane $\alpha_i, i = 1, \ldots, K$ eylemi tanımlayalım; öyle ki α_i, $\boldsymbol{x}$ girdisinin C_i sınıfına atanması olsun. *0/1 yitiminde*

$$\lambda_{ik} = \begin{cases} 0 & \text{eğer } i = k \\ 1 & \text{eğer } i \neq k \end{cases} \tag{3.9}$$

tüm doğru kararlar yitimsizdir ve tüm hataların bedeli aynıdır. Bu durumda α_i eyleminin beklenen riski

$$\begin{aligned} R(\alpha_i|\boldsymbol{x}) &= \sum_{k=1}^{K} \lambda_{ik} P(C_k|\boldsymbol{x}) \\ &= \sum_{k \neq i} P(C_k|\boldsymbol{x}) \\ &= 1 - P(C_i|\boldsymbol{x}) \end{aligned}$$

olur ($\sum_k P(C_k|\boldsymbol{x}) = 1$ olduğunu hatırlayalım). Yani riski en aza indiren eylem, olasılığı en yüksek sınıfı seçmektir. Gelecek bölümlerde basitlik için hep bu durumu varsayacak ve en yüksek sonsal olasılığa sahip sınıfı seçeceğiz, ama gerçekte bunun özel bir durum olduğunu ve 0/1 yitimine uygulamalarda çok seyrek rastlanıldığını aklımızda tutmamız gerekir. Genel durumda belli bir yitim dizeyi verildiğinde sonsal olasılıklardan risklere geçerek en düşük riskli eylemi seçmek basit bir sonişlemeyle gerçeklenebilir.

Bazı uygulamalarda yanlış kararların, yani yanlış sınıflandırmaların bedeli çok yüksek olabilir ve eğer çok emin değilsek kararın daha karmaşık bir karar vericiye, örneğin bir insana (kullanıcıya) bırakılması istenir. Örneğin, zarflar üzerindeki posta kodlarını okumak için kullandığımız bir görsel rakam tanıyıcı kodu yanlış okursa posta yanlış bir yöne gönderilecek ve bu gecikmelere ve ek masrafa yol açacaktır.

Böyle bir durumda, $C_i, i = 1, \ldots, K$ sınıfları için olağan $\alpha_i, i = 1, \ldots, K$ eylemleri yanında *ret* ya da *kuşku* diye adlandırdığımız ek bir α_{K+1} eylemi tanımlarız (Duda, Hart, ve Stork 2001).

Reddetmenin yitimi, $0 < \lambda < 1$, yanlış sınıflandırmanınkinden düşüktür:

$$\lambda_{ik} = \begin{cases} 0 & \text{eğer } i = k \\ \lambda & \text{eğer } i = K+1 \\ 1 & \text{değilse} \end{cases} \tag{3.10}$$

Bu durumda, ret eyleminin beklenen riski

$$R(\alpha_{K+1}|\boldsymbol{x}) = \sum_{k=1}^{K} \lambda P(C_k|\boldsymbol{x}) = \lambda \tag{3.11}$$

ve C_i sınıfını seçmenin riski

$$R(\alpha_i|\boldsymbol{x}) = \sum_{k \neq i} P(C_k|\boldsymbol{x}) = 1 - P(C_i|\boldsymbol{x}) \tag{3.12}$$

hesaplanıp en düşük riskli eylem, olası $K+1$ eylem arasından seçilir

$$\begin{array}{ll} \text{seç } C_i & \text{eğer } R(\alpha_i|\boldsymbol{x}) < R(\alpha_k|\boldsymbol{x}) \text{ tüm } k \neq i \text{ için ve} \\ & R(\alpha_i|\boldsymbol{x}) < R(\alpha_{K+1}|\boldsymbol{x}) \\ \text{reddet} & \text{eğer } R(\alpha_{K+1}|\boldsymbol{x}) < R(\alpha_i|\boldsymbol{x}), i = 1, \ldots, K \end{array} \tag{3.13}$$

Denklem 3.10'daki yitim dizeyi kullanılırsa:

$$\begin{array}{ll} \text{seç } C_i & \text{eğer } P(C_i|\boldsymbol{x}) > P(C_k|\boldsymbol{x}) \text{ tüm } k \neq i \text{ için ve} \\ & P(C_i|\boldsymbol{x}) > 1 - \lambda \\ \text{reddet} & \text{değilse} \end{array} \tag{3.14}$$

Tüm bu yaklaşım ancak $0 < \lambda < 1$ olduğunda anlamlıdır: $\lambda = 0$ olduğunda reddetmek doğru karar vermek gibi bedelsiz olduğundan hep reddederiz; $\lambda \geq 1$ olduğunda reddetmek yanlış karar vermek kadar ya da daha fazla bedelli olduğundan hiç reddetmeyiz.

3.4 Ayırtaç İşlevleri

Sınıflandırma, $g_i(\boldsymbol{x}), i = 1, \ldots, K$, *ayırtaç işlevinin* belirlenmesi olarak da görülebilir; öyle ki

$$\text{seç } C_i \text{ eğer } g_i(\boldsymbol{x}) = \max_k g_k(\boldsymbol{x}) \tag{3.15}$$

Bayes sınıflandırıcısını bu biçimde yazabiliriz

$$g_i(\boldsymbol{x}) = -R(\alpha_i|\boldsymbol{x})$$

ve en büyük ayırtaç, en düşük riske karşılık gelir. 0/1 yitimi kullandığımızda ayırtaç değeri sonsal olasılık olur:

$$g_i(\boldsymbol{x}) = P(C_i|\boldsymbol{x})$$

Tüm sınıflar için ortak olan normalleştirme paydasını göz ardı ettiğimizde ayırtaç basitleşir:

$$g_i(\boldsymbol{x}) = p(\boldsymbol{x}|C_i)P(C_i)$$

Ayırtaç işlevleri, girdi uzayını K tane, $\mathcal{R}_1, \ldots, \mathcal{R}_K$ ile gösterdiğimiz *karar bölgesine* ayırır, öyle ki her sınıfın karar bölgesinde kendi ayırtaç değeri en büyüktür: $\mathcal{R}_i = \{\boldsymbol{x}|g_i(\boldsymbol{x}) = \max_k g_k(\boldsymbol{x})\}$. Bu bölgeler, komşu sınıfların ayırtaç değerlerinin birbirine eşit olduğu *karar sınırlarıyla* birbirinden ayrılır (bkz. şekil 3.1).

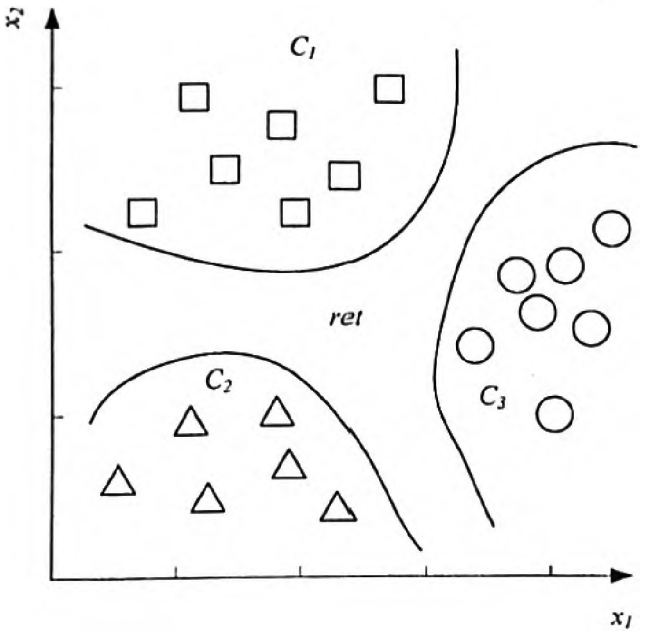
İki sınıf olduğunda tek bir ayırtaç işlevi yeter:

$$g(\boldsymbol{x}) = g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$$

Büyük ya da küçük yerine, sonucun artı ya da eksi olmasına göre karar veririz:

$$\text{seç} \begin{cases} C_1 & \text{eğer } g(\boldsymbol{x}) > 0 \\ C_2 & \text{değilse} \end{cases}$$

Örneğin, artı örnekler C_1, eksi örnekler C_2 olarak tanımlanıp bu yaklaşım kullanılabilir.



Şekil 3.1: Karar bölgeleri ve sınırları için bir örnek.

3.5 Fayda Kuramı

Denklem 3.7'de beklenen riski tanımlayıp beklenen riski en aza indirecek eylemi seçtik. Şimdi bunu, durum belirsiz olduğunda akılcı kararlar vermeyi amaçlayan *fayda kuramına* genelleyeceğiz. x kanıtı verildiğinde S_k durumunun olasılığını $P(S_k|x)$ ile gösterelim. Durum S_k iken α_i eylemini seçmenin ne kadar iyi olduğunu ölçen bir U_{ik} *fayda işlevi* tanımlayıp her eylemin beklenen faydasını hesaplarız:

$$EU(\alpha_i|x) = \sum_k U_{ik} P(S_k|x) \tag{3.16}$$

Akılcı bir karar verici, beklenen faydası en yüksek eylemi seçer:

$$\text{Seç } \alpha_i \text{ eğer } EU(\alpha_i|x) = \max_j EU(\alpha_j|x) \tag{3.17}$$

Sınıflandırma bağlamında karar, sınıflardan birini seçmeye ve beklenen faydayı artırmak riski azaltmaya karşılık gelir. U_{ik} değeri genelde mali terimlerle, parasal olarak belirlenir ve bu bize yitim matrisi oluşturmanın bir yolunu gösterir. Örneğin, ret seçeneği tanımladığımızda (denklem 3.10), eğer doğru bir karar sonucu ne kadar para kazanacağımızı, yanlış bir kararla ne kadar para kaybedeceğimizi, ve kararı uzman bir insana bırakmanın ne kadar masraflı olduğunu biliyorsak, söz konusu uygulamaya bağlı olarak, U_{ik} için, 0, λ, ve 1 yerine, doğru değerleri girip beklenen kazancı en yüksek yapacak biçimde karar verebiliriz.

Beklenen (ortalama) faydayı artırmak olasılıklardan yalnızca biridir; olası en yüksek kaybı azaltmak gibi başka tür akılcı davranışlar da tanımlanabilir.

Ret kararı verdiğimizde bilgisayar yazılımıyla, daha masraflı ama doğru karar verme olasılığı daha yüksek insan arasında seçim yapıyoruz. Aynı mantıkla, masrafı gittikçe artan ama daha başarılı sınıflandırıcılardan oluşan ardışık bir sistem de düşünebiliriz; böylesi modelleri, 17. bölümde birden çok modeli birleştirmeyi tartışırken göreceğiz.

3.6 İlişkilendirme Kuralları

İlişkilendirme kuralı, $X \rightarrow Y$ biçiminde bir gerektirmedir ve burada X, *öncel*, ve Y, *ardıl* olarak anılır. İki ürün, X ve Y, arasındaki ilişkiyi bulmak istediğimiz *sepet çözümlemesi* ilişkilendirme kuralına bir örnektir. Bölüm 1.2.1'de gördüğümüz gibi bunun yaygın bir uygulaması perakende sektöründedir.

İlişkilendirme kuralı öğrenmede yoğun olarak hesaplanan üç ölçüt vardır:

- $X \rightarrow Y$ ilişkilendirme kuralının *destek* değeri:

$$\text{Destek}(X,Y) \equiv P(X,Y) = \frac{\#\{X \text{ ve } Y \text{ almış müşteriler}\}}{\#\{\text{müşteriler}\}} \tag{3.18}$$

- $X \rightarrow Y$ ilişkilendirme kuralının *güven* değeri:

$$\begin{aligned} \text{Güven}(X \rightarrow Y) \equiv P(Y|X) &= \frac{P(X,Y)}{P(X)} \\ &= \frac{\#\{X \text{ ve } Y \text{ almış müşteriler}\}}{\#\{X \text{ almış müşteriler}\}} \end{aligned} \tag{3.19}$$

- $X \rightarrow Y$ ilişkilendirme kuralının *etki* değeri:

$$\text{Etki}(X \rightarrow Y) = \frac{P(X,Y)}{P(X)P(Y)} = \frac{P(Y|X)}{P(Y)} \tag{3.20}$$

Başka ölçütler de vardır (Omiecinski 2003) ama bu üçü, özellikle de ilk ikisi, en yaygın bilinen ve kullanılanlardır. *Güven*, bildiğimiz $P(Y|X)$ koşullu olasılığıdır. Kural yeteri kadar güvene sahiptir, diyebilmek için bu değerin bire yakın olması ve insanların Y malını satın alma olasılığı olan $P(Y)$ değerinden önemli derecede büyük olması gerekir. Kuralın *desteğinin* de yüksek olmasını isteriz, çünkü güven yüksek olsa da eğer böyle müşterilerin sayısı azsa kural önemsizdir. Destek, kuralın istatistiksel önemini, güven ise gücünü gösterir. En düşük destek ve güven değerleri kullanıcı tarafından belirlenir ve bu değerlerden daha yüksek destek ve güvene sahip kurallar veritabanında aranır.

X ve Y bağımsızsa, *etkinin* bire yakın olmasını bekleriz; eğer oran bundan farklıysa, yani $P(Y|X)$ ve $P(Y)$ farklılarsa, bu iki öğe arasında ilişki olmasını bekleriz: Etki birden büyükse, X malının Y malını daha çok, ve etki birden küçükse daha az olası yaptığını söyleyebiliriz.

Bu formüller kolaylıkla ikiden çok öğeye genellenebilirler. Örneğin, $\{X, Y, Z\}$ üç öğeli bir kümedir ve biz $X, Z \rightarrow Y$ şeklinde bir kural, yani $P(Y|X, Z))$, arayabiliriz. Böylesi kurallardan yeterince destek ve güvene sahip olanları bulmayı ve, genelde satış veritabanı çok büyük olduğu için, bunu veri üzerinden olabildiğince az kez geçerek yapmayı isteriz. Bunun için, *Apriori* (Agrawal ve ötekiler 1996) diye adlandırılan etkin bir algoritma vardır ve iki basamaktan oluşur: (1) *sık öğe kümelerini*, yani yeterince desteği olanları bulma, ve (2) bu kümeleri öncel ve ardıl nesneler olarak ikiye bölerek yeteri kadar güveni olan kurallara çevirme:

1. Sık öğe kümelerini hıza (olası tüm altkümelere bakmadan) bulmak için Apriori algoritması şu gerçeği kullanır: $\{X, Y, Z\}$'nin sık (yeterince destekli) olması için tüm altkümelerinin de sık olmaları gerekir; bir öğe eklemekle destek artmaz. Yani üç öğeli altkümeler içinden tüm ikili altkümeleri sık olanlara bakmamız yeterli olacaktır; başka bir deyişle, eğer ikili bir öğe kümesi sık değilse onun bütün üstkümeleri budanabilir ve bakılmalarına gerek yoktur.

   Sık, tek öğeli kümeleri bularak başlar ve her aşamada, tümevarımsal olarak, sık k öğeli kümelerden, aday $k+1$ öğeli kümeler oluşturur ve verinin üzerinden geçip yeterince destekli olup olmadıklarına bakarız. Apriori algoritması, kolay ulaşım için, sık nesne kümelerini anahtarlı bir çizelgede saklar. k arttıkça aday nesne kümelerinin sayısı hızla azalacaktır. Eğer en büyük öğe kümesi n öğeden oluşuyorsa, veri üzerinden $n + 1$ kez geçmemiz gerekecektir.

2. k öğeli kümeleri bulduktan sonra, onları öncel ve ardıl olmak üzere ikiye bölüp kurallara çevirmemiz gerekir. Kümeleri oluştururken yaptığımız gibi tek bir ardıl ile başlar ve öteki $k - 1$ nesneyi öncel olarak alır ve olası her tekli ardıl için kuralın yeterince güvene sahip olup olmadığına bakar, değilse kuralı atarız.

   Aynı öğe kümesi için farklı öncel ve ardıl altkümeleri olan birçok kural yazılabilir. Sonra tümevarımsal olarak, öncelden ardıla bir öğe daha taşıyabilir miyiz diye bakarız. Ardıl kısmında daha çok öğe bulunan kurallar daha özgül ve daha kullanışlıdır. Burada da, öğe kümelerinin oluşturulmasında olduğu gibi, iki öğenin ardıl olduğu bir kuralın yeterince güvene sahip olabilmesi için tek ardıllı her iki kuralın da yeterli güveni olması gerektiği gerçeğini kullanırız; yani, tek ardıllı kurallardan iki ardıllı kurallara geçerken olası iki nesneli ardılların tümüne bakmamız gerekmez (alıştırma 7).

$X \rightarrow Y$ kuralı bir ilişki tanımlar, ama nedensellik gerektirmez. Bir uygulamada, değerlerini asla bilemeyeceğimiz *saklı değişkenler* olabilir ve bunlarla ilişki yapısı daha kolay tanımlanabilir. Örneğin, satılan ürünler arasındaki ilişkileri bulmaya çalıştığımız sepet çözümlemesinde, diyelim ki "bebek maması," "bebek bezi" ve "süt" arasında, birini alan bir müşterinin öbür ikisini de almasının çok olası olduğu şeklinde bir ilişki olsun. Bu üçü arasındaki bağımlılıkları ikişer ikişer göstermek yerine, üçünün de tüketimindeki saklı neden olan "evde bebek" değişkenini tanımlayabiliriz. 16. bölümde ele alacağımız çizgesel modeller bu tip saklı değişkenleri gösterebilmemizi sağlar. Saklı düğümler olduğunda değerleri gözlenebilir olanları kullanarak kestirilir.

3.7 Kısa Kısa

Belirsizliğe rağmen karar vermenin uzun bir tarihçesi vardır ve zaman içerisinde insanlık belirsizlikleri ortadan kaldırmak için yıldızlar, kristal küreler, ve kahve fincanları gibi çeşitli, çok tuhaf yerlere bakmıştır. Olasılık kuramını kullanarak anlamlı kanıtlardan akılyürütmenin yalnızca birkaç yüzyıllık bir geçmişi vardır. Olasılık ve istatistiğin tarihçesi ve kuramın temellerini atan Laplace, Bernoulli, ve ötekilerin ilk makaleleri için (Newman 1988)'e bakınız.

Russell ve Norvig (1995), fayda kuramını ve bilgi değerini, faydaların mali cinsten hesaplanması konusunu da içererek çok aydınlatıcı bir biçimde ele almıştır. (Shafer ve Pearl 1990), belirsizlik altında karar verme konusundaki makalelerin erken bir derlemesidir.

İlişkilendirme kuralları birçok veri madenciliği uygulamasında başarıyla kullanılıyor ve böyle kuralları kullanarak kitap, yazar, film, ya da müzik öneren birçok web sayfası görüyoruz. Bu çok basit algoritmanın büyük veritabanlarında etkin gerçeklenmesi çok önemlidir (Zhang ve Zhang 2002; Li 2006). Çizgesel modelleri tartıştığımız 16. bölümde, ilişkilendirme kurallarından, farklı türde, ikili olması gerekmeyen ve saklı olabilen değişkenler içeren ilişkilere nasıl genelleme yapabileceğimizi göreceğiz.

3.8 Alıştırmalar

1. İki sınıflı bir problemde, *olabilirlik oranı*

$$\frac{p(x|C_1)}{p(x|C_2)}$$

cinsinden bir ayırtaç işlevi tanımlayın.

2. İki sınıflı bir problemde, *log olasılık oranı*

$$\log \frac{P(C_1|x)}{P(C_2|x)}$$

cinsinden bir ayırtaç işlevi tanımlayın.

3. İki sınıflı ve iki eylemli bir problemde, yitim işlevi, $\lambda_{11} = \lambda_{22} = 0$, $\lambda_{12} = 10$, ve $\lambda_{21} = 1$ ise nasıl karar vermek gerekir?

4. Bir düzey reddedince ardından gelenin 3.10. denklemdeki gibi seçildiği, üç düzeyden oluşan ardışık bir sistem önerin. Her düzeydeki λ değeri nasıl belirlenir?

5. Biri hilesiz bir parayla yazı tura atıyor; yazı gelirse size 5 TL verecek ama tura gelirse hiçbir şey vermeyecek. Bu oyunu oynamak için ne kadar para verirdiniz? Peki kazanacağınız 5 yerine 500 TL olsaydı?

6. Sepet çözümlemesi için gördüğümüz güven ve destek formüllerini k-ilişkileri, yani $P(Y|X_1, \ldots, X_k)$ koşullu olasılıklarını hesaplayabilecek biçimde genelleyiniz.

7. Bir öğeyi ardıldan öncele taşıdığımızda, güvenin artamayacağını gösteriniz: confidence($ABC \rightarrow D$) $\geq$ confidence($AB \rightarrow CD$).

8. Sepet çözümlemesinde, satılan her ürün için elimizde 0 ile 10 arasında bir müşteri memnuniyeti bilgisi de olsa, bu ek bilgiyi müşterilere öneri yapmakta nasıl kullanabiliriz?
9. $X \rightarrow Y$ kuralı için aşağıdaki koşulları sağlayan örnek işlem verisi nasıl olur:
   (a) Desteği ve güveni yüksek,
   (b) Desteği yüksek ama güveni düşük,
   (c) Desteği düşük ama güveni yüksek,
   (d) Desteği ve güveni düşük.

3.9 Kaynaklar

Agrawal, R., H. Mannila, R. Srikant, H. Toivonen, A. Verkamo. 1996. "Fast Discovery of Association Rules." *Advances in Knowledge Discovery and Data Mining*, ed. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy, 307–328. Cambridge, MA: MIT Press.

Duda, R. O., P. E. Hart, D. G. Stork. 2001. *Pattern Classification*, 2nd ed. New York: Wiley.

Li, J. 2006. "On Optimal Rule Discovery." *IEEE Transactions on Knowledge and Data Discovery* 18: 460–471.

Newman, J. R., ed. 1988. *The World of Mathematics.* Redmond, WA: Tempus.

Omiecinski, E. R. 2003. "Alternative Interest Measures for Mining Associations in Databases." *IEEE Transactions on Knowledge and Data Discovery* 15: 57–69.

Russell, S., P. Norvig. 1995. *Artificial Intelligence: A Modern Approach.* New York: Prentice Hall.

Shafer, G., J. Pearl, eds. 1990. *Readings in Uncertain Reasoning.* San Mateo, CA: Morgan Kaufmann.

Zhang, C., S. Zhang. 2002. *Association Rule Mining: Models and Algorithms.* New York: Springer.

Bölüm 4

Dağılıma Bağlı Yöntemler

Belirsizlikleri olasılıklarla modelleyip karar vermeyi önceki bölümde gördük; bu bölümdeyse bu olasılıkları bir öğrenme kümesinden kestirmeyi göreceğiz. Sınıflandırma ve bağlanım için dağılıma bağlı yaklaşımla başlayacağız; dağılıma yarı bağlı ve dağılımdan bağımsız yaklaşımlarıysa ileriki bölümlerde göreceğiz. Ayrıca model karmaşıklığı ve gözlemsel hata arasındaki ödünleşim için yanlılık/değişinti ikilemini ve model seçimi yöntemlerini tartışacağız.

4.1 Giriş

Bir örneklemden hesaplanan herhangi bir değere *istatistik* denir. İstatistiksel çıkarım yaparken örneklemin içindeki bilgiyi kullanarak karar veririz. İlk yaklaşımımız, örneklemin geldiği dağılımın, örneğin normal dağılım gibi bilinen bir model olduğunu varsaydığımız dağılım tabanlı yaklaşımdır. Bu yaklaşımın getirisi, modelin, *yeterli istatistikler* denilen az sayıda parametreyle, örneğin, normal dağılım için ortalama ve değişintiyle, tanımlanmasıdır. Bir kez bu parametreler örneklemden kestirildiğinde dağılımı tamamen biliyor oluruz. Verilen örneklerden bu parametreleri kestirir, bu kestirimleri varsayılan modele girer ve böylece karar vermede kullanacağımız dağılımı buluruz. Bir dağılımın parametrelerini kestirmede kullandığımız yöntem en büyük olabilirlik kestirimidir. Bu bölümde ayrıca bölüm 14'te işlemeye devam edeceğimiz Bayesci kestirime de kısa bir giriş yapacağız.

Herhangi bir $p(x)$ dağılımının kestirildiği genel durum olan *dağılım kestirimiyle* başlayacağız. Bunu, kestirilmesi gereken dağılımların $p(x|C_i)$ sınıf dağılımları ve $P(C_i)$ önsel olasılıkları olduğu *sınıflandırmada*, sonsal $P(C_i|x)$ olasılıklarını hesaplayıp karar vermekte kullanacağız. Sonra kestirilmesi gereken dağılımın $p(y|x)$ olduğu *bağlanımı* ele alacağız. Bu bölümde x tek boyutlu ve dolayısıyla dağılımlar tek değişkenlidir. Çok değişkenli duruma 5. bölümde genelleme yapacağız.

4.2 En Büyük Olabilirlik Kestirimi

Bağımsız ve özdeşçe dağılmış $\mathcal{X} = \{x^t\}_{t=1}^N$ örneklemimiz olsun. x^t ile gösterdiğimiz örneklerin θ parametresiyle tanımlanan bilinen bir $p(x|\theta)$ olasılık dağılımı ailesinden eldiğini varsayalım:

$$x^t \sim p(x|\theta)$$

Bu örneklerini örneklemenin olabilirliğini en fazla yapan θ değerini bulmak isteriz. x^t'ler birbirinden bağımsız olduğundan verilen $\mathcal{X}$ örneklemi için θ parametresinin *olabilirliği* tüm örnek noktalarının olabilirliklerinin çarpımıdır:

$$l(\theta|\mathcal{X}) \equiv p(\mathcal{X}|\theta) = \prod_{t=1}^{N} p(x^t|\theta) \tag{4.1}$$

En büyük olabilirlik kestiriminde(EBO), örneklemin görülmesini en olabilir yapan θ değerini bulmaya çalışırız. Yani $l(\theta|\mathcal{X})$ olarak gösterdiğimiz olabilirliği en büyük yapacak θ değerini arıyoruz. En büyük değeri alacağı noktayı değiştirmeyeceğinden olabilirliğin logaritmasını da kullanabiliriz. $\log(\cdot)$, çarpımları toplamlara çevirir ve üstel terimler içeren bazı dağılımlarda *Log olabilirlik* kullanmak hesabı basitleştirir:

$$\mathcal{L}(\theta|\mathcal{X}) \equiv \log l(\theta|\mathcal{X}) = \sum_{t=1}^{N} \log p(x^t|\theta) \tag{4.2}$$

Gelin şimdi, ilgilendiğimiz uygulamalarda sıkça karşılaşacağımız bazı dağılımları görelim. İki sınıflı bir değer varsa *Bernoulli* dağılımını kullanırız. $K > 2$ sınıf olduğunda genellenmiş hali *katlıterimli* dağılım olarak adlandırılır. *Gauss* ya da *normal* dağılım, sayısal girdi için sınıf koşullu girdi dağılımlarının modellenmesinde yaygın olarak kullanılır. Bu üç dağılımın parametrelerinin en büyük olabilirlik en büyük olabilirlik kestiricilerini sıkça kullanacağız.

4.2.1 Bernoulli Dağılımı

Bir Bernoulli dağılımında iki sonuç vardır: Bir olay gerçekleşir ya da gerçekleşmez; örneğin, bir örnek, sınıfın ya artı ya da eksi örneğidir. p olasılıkla olay gerçekleşir ve Bernoulli rastsal değişkeni, X, 1 değerini alır; olayın gerçekleşmeme olasılığı $1-p$'dir ve bu X'in 0 değerini almasıyla gösterilir. Bu kısaca yazılabilir:

$$P(x) = p^x(1-p)^{1-x}, x \in \{0,1\} \tag{4.3}$$

Bu dağılım için beklenen değer ve değişintiyi hesaplayalım:

$$\begin{aligned} E[X] &= \sum_x xp(x) = 1 \cdot p + 0 \cdot (1-p) = p \\ \text{Var}(X) &= \sum_x (x - E[X])^2 p(x) = p(1-p) \end{aligned}$$

p, dağılımın tek parametresidir. Bağımsız ve özdeş örneklem $\mathcal{X} = \{x^t\}_{t=1}^N$xt?0, 1 verildiğinde kestiricisi $\hat{p}$'yı hesaplamak isteriz. Log olabilirliği yazalım:

$$\begin{aligned}\mathcal{L}(p|\mathcal{X}) &= \log \prod_{t=1}^{N} p^{(x^t)}(1-p)^{(1-x^t)} \\ &= \sum_t x^t \log p + \left(N - \sum_t x^t\right) \log(1-p)\end{aligned}$$

Buna en büyük değeri veren $\hat{p}$ kestiricisini $d\mathcal{L}/dp = 0$ denklemini çözerek elde ederiz. Şapka, bulunan değerin bir kestirim olduğunu gösterir:

$$\hat{p} = \frac{\sum_t x^t}{N} \tag{4.4}$$

p için kestirim, olayın gerçekleşme sayısının toplam deney sayısına oranıdır. Bernoulli dağılmış X için $E[X] = p$ olduğundan, ve bekleneceği gibi ortalamanın en büyük olabilirlik kestiricisi örneklem ortalamasıdır.

Kestirim de örneklemin bir işlevidir ve başka bir rastsal değişkendir; aynı $p(x)$ dağılımından örneklenmiş farklı $\mathcal{X}_i$ örneklemlerinden hesaplanan $\hat{p}_i$ değerlerinin bir dağılımı vardır. Örneğin, $\hat{p}_i$ değerlerinin dağılımının değişintisinin N arttıkça azalması beklenir; büyüdükçe örneklemler (ve dolayısıyla ortalamaları) birbirlerine daha çok benzerler.

4.2.2 Katlıterimli Dağılım

Rastgele bir olayın sonucu iki durum yerine, $p_i, i = 1, \ldots, K$ olasılıkla $K > 2$ ayrık ve kapsayan durumdan, örneğin K farklı sınıftan biri olabilir. Sonuç olasılıklarının toplamı birdir: $\sum_{i=1}^K p_i = 1$. Bernoulli dağılımının genellemesi olarak rastsal değişkenler tanımlayalım: $x_1, x_2, \ldots, x_K$ gösterge değişkenler olsun, öyle ki, çıktı i. durumdayken $x_i = 1$, değilse $x_i = 0$ olsun.

$$P(x_1, x_2, \ldots, x_K) = \prod_{i=1}^{K} p_i^{x_i} \tag{4.5}$$

N bağımsız deneyin sonucunu $\mathcal{X} = \{x^t\}_{t=1}^N$ örneklemine kaydetmiş olalım:

$$x_i^t = \begin{cases} 1 & \text{deney } t\text{'nin sonucu durum } i \text{ ise} \\ 0 & \text{değilse} \end{cases}$$

öyle ki $\sum_i x_i^t = 1$. p_i parametresinin en büyük olabilirlik kestiricisini Bernoulli'ye benzer biçimde türetebiliriz:

$$\hat{p}_i = \frac{\sum_t x_i^t}{N} \tag{4.6}$$

Çıktının i. durum olma olasılığının kestirimi, çıktısı i. durum olan deneylerin sayısının toplam deney sayısına oranıdır. Bunu bulmanın iki yolu vardır: x_i değerleri

0/1 olduğundan bunları K ayrı Bernoulli deneyi olarak düşünebiliriz. Ya da, log olabilirliği açıkça yazıp onu en büyük yapan p_i değerini ($\sum_i p_i = 1$ ek koşuluyla) elde edebiliriz.

4.2.3 Gauss (Normal) Dağılım

Ortalaması $E[X] \equiv \mu$ ve değişintisi $\text{Var}(X) \equiv \sigma^2$ olan Gauss dağılımı, ya da normal dağılım, kısaca $\mathcal{N}(\mu, \sigma^2)$ olarak gösterilir:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right], -\infty < x < \infty \tag{4.7}$$

$x^t \sim \mathcal{N}(\mu, \sigma^2)$ örneklerini içeren bir örneklem $\mathcal{X} = \{x^t\}_{t=1}^N$ verildiğinde log olabilirlik

$$\mathcal{L}(\mu, \sigma|\mathcal{X}) = -\frac{N}{2}\log(2\pi) - N\log\sigma - \frac{\sum_t (x^t - \mu)^2}{2\sigma^2}$$

işlevini en büyük yapan parametre değerleri kısmi türevleri sıfıra eşitleyerek bulunur:

$$\begin{aligned} m &= \frac{\sum_t x^t}{N} \\ s^2 &= \frac{\sum_t (x^t - m)^2}{N} \end{aligned} \tag{4.8}$$

İstatistik yazınında alışılagelmiş düzeni izliyor ve kitle parametrelerini Yunan, onların kestirimlerini Latin harfleriyle gösteriyoruz. Bazen, kestirilen parametreyi vurgulamak için şapkayı da kullanabiliriz; örneğin, $\hat{\mu}$.

4.3 Kestirici Değerlendirme: Yanlılık ve Değişinti

$\mathcal{X}$, bir θ parametresiyle belirlenmiş belli bir kitleden alınan bir örneklem ve $d = d(\mathcal{X})$, θ için bir kestirici olsun. Bu kestiricinin iyiliğini değerlendirmek için kestirdiği değerin gerçek θ değerinden ne kadar farklı olduğunu, yani $(d(\mathcal{X}) - \theta)^2$ değerini hesaplarız. Ancak bu (örnekleme bağlı) bir rastsal değişken olduğundan, olası $\mathcal{X}$ örneklemleri üzerinden ortalama almalı ve d kestiricisinin *ortalama kare hatasına* bakmalıyız:

$$r(d, \theta) = E[(d(\mathcal{X}) - \theta)^2] \tag{4.9}$$

Bir kestiricinin *yanlılığı*, beklenen değerinin (ortalamasının) gerçek değerden ne kadar farklı olduğudur:

$$b_\theta(d) = E[d(\mathcal{X})] - \theta \tag{4.10}$$

Eğer θ parametresinin tüm değerleri için $b_\theta(d) = 0$ ise d, θ'nın *yansız bir kestiricidir*. Örneğin, μ ortalamalı bir dağılımdan çekilen x^t örneklerinin örneklem ortalaması olan m, kitle ortalaması μ'nün yansız bir kestiricisidir, çünkü

$$E[m] = E\left[\frac{\sum_t x^t}{N}\right] = \frac{1}{N}\sum_t E[x^t] = \frac{N\mu}{N} = \mu$$

Bunun anlamı şudur: Herhangi bir örneklem için m değeri, μ değerinden farklı olabilir, ama birçok $\mathcal{X}_i$ örneklemi alıp her biri için $m_i = m(\mathcal{X}_i)$ hesapladığımızda, örneklem sayısı arttıkça, m_i değerlerinin ortalaması, gerçek μ değerine yakınsayacaktır. Ayrıca, m *tutarlı* bir kestiricidir; $N \to \infty$ iken $\text{Var}(m) \to 0$ olur.

$$\text{Var}(m) = \text{Var}\left(\frac{\sum_t x^t}{N}\right) = \frac{1}{N^2}\sum_t \text{Var}(x^t) = \frac{N\sigma^2}{N^2} = \frac{\sigma^2}{N}$$

Örneklemdeki nokta sayısı olan N arttıkça m, μ değerinden daha az sapar. σ^2'nın en büyük olabilirlik kestiricisini s^2 ile göstermiştik; bir de ona bakalım:

$$\begin{aligned} s^2 &= \frac{\sum_t (x^t - m)^2}{N} = \frac{\sum_t (x^t)^2 - Nm^2}{N} \\ E[s^2] &= \frac{\sum_t E[(x^t)^2] - N \cdot E[m^2]}{N} \end{aligned}$$

$\text{Var}(X) = E[X^2] - E[X]^2$ olduğu için $E[X^2] = \text{Var}(X) + E[X]^2$ olur. Dolayısıyla

$$E[(x^t)^2] = \sigma^2 + \mu^2 \text{ ve } E[m^2] = \sigma^2/N + \mu^2$$

Bunları yerine yerleştirelim:

$$E[s^2] = \frac{N(\sigma^2 + \mu^2) - N(\sigma^2/N + \mu^2)}{N} = \left(\frac{N-1}{N}\right)\sigma^2 \neq \sigma^2$$

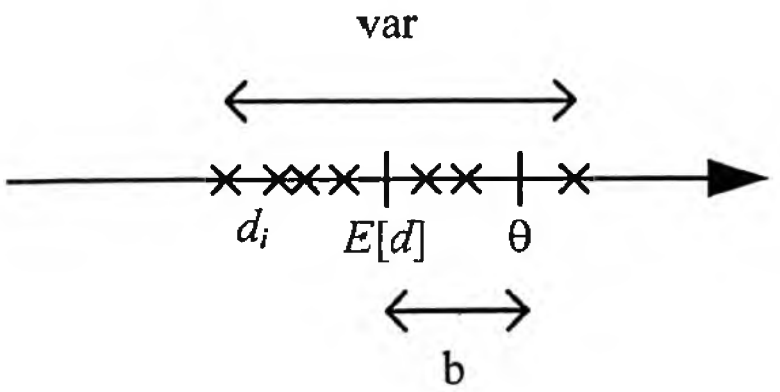
s^2, σ^2 için yanlı bir kestiricidir; $(N/(N-1))s^2$, yansız bir kestiricidir. N büyük olduğunda fark göz ardı edilebilir; N sonsuza giderken yanlılık sıfıra gider.

Ortalama kare hatayı açalım ($d(\mathcal{X})$ yerine kısaca d yazıyoruz):

$$\begin{aligned} r(d,\theta) &= E\left[(d-\theta)^2\right] \\ &= E\left[(d - E[d] + E[d] - \theta)^2\right] \\ &= E\left[(d-E[d])^2 + (E[d]-\theta)^2 + 2(E[d]-\theta)(d-E[d])\right] \\ &= E\left[(d-E[d])^2\right] + E\left[(E[d]-\theta)^2\right] + 2E\left[(E[d]-\theta)(d-E[d])\right] \\ &= E\left[(d-E[d])^2\right] + (E[d]-\theta)^2 + 2(E[d]-\theta)E[d-E[d]] \\ &= \underbrace{E\left[(d-E[d])^2\right]}_{\text{değişinti}} + \underbrace{(E[d]-\theta)^2}_{\text{yanlılık}^2} \end{aligned} \tag{4.11}$$

Burada, $E[d]$ ve $E[d] - \theta$ sabittir ve $E[d - E[d]] = E[d] - E[d] = 0$ olur. Denklem 4.11'de ilk terim, d_i'nin, beklenen değeri çevresinde, ortalama olarak (bir veri kümesinden öbürüne) ne kadar değiştiğini ölçen *değişintidir*; ikinci terimse beklenen değerin gerçek θ değerinden ne kadar saptığını ölçen *yanlılıktır* (şekil 4.1). Hatayı bu iki terimin, yani değişinti ve yanlılığın karesinin toplamı olarak yazabiliriz:

$$r(d,\theta) = \text{Var}(d) + (b_\theta(d))^2 \tag{4.12}$$



Şekil 4.1: θ, kestirilecek parametre, '×' ile gösterilen d_i değerleri, farklı $\mathcal{X}_i$'lerden hesaplanan kestirimlerdir. Yanlılık (Var), d değerlerinin ortalamasıyla θ arasındaki farktır. Değişinti (b) ise d_i değerlerinin kendi ortalamaları çevresinde ne kadar oynadığıdır. İkisinin de küçük olmasını isteriz.

4.4 Bayesçi Kestirici

Bazen, örnekleme bakmadan önce, bizim (ya da uygulama alanındaki uzmanların) θ parametresinin alabileceği olası değer aralığı hakkında *önsel* bilgimiz olabilir. Bu bilgi oldukça kullanışlıdır ve kullanılmalıdır, özellikle de örneklem küçükse. Önsel bilgi bize parametre değerinin kaç olması gerektiğini söylemez (söylese örneklemeye ve kestirime gerek kalmazdı), ama biz θ parametresini rastsal bir değişken olarak kabul edip onun için bir $p(\theta)$ önsel dağılımı tanımlayarak bu belirsizliği modelleyebiliriz. Örneğin, diyelim ki yaklaşık normal ve yüzde 90 güvenle 7 çevresinde, 5 ile 9 arasında, bakışımlı dağıldığını biliyoruz. Bu durumda önseli, 7 ortalamalı bir normal olarak yazabiliriz ve

$$\begin{aligned} P\{-1.64 < \frac{\theta - \mu}{\sigma} < 1.64\} &= 0.9 \\ P\{\mu - 1.64\sigma < \theta < \mu + 1.64\sigma\} &= 0.9 \end{aligned}$$

olduğu için $1.64\sigma = 2, \Rightarrow \sigma = 2/1.64$ olarak $p(\theta) \sim \mathcal{N}(7, (2/1.64)^2)$ varsayabiliriz.

$p(\theta)$ *önsel dağılımı* bize, örnekleme bakmadan *önce*, θ parametresinin alabileceği olası değerleri söyler. Bunu verinin bize söylediklerileyle, yani $p(\mathcal{X}|\theta)$ olabilirlik dağılımıyla Bayes kuralını kullanarak birleştirir ve θ parametresinin örnekleme baktıktan *sonraki* olası değerlerini belirten *sonsal olasılık dağılımını* elde ederiz:

$$p(\theta|\mathcal{X}) = \frac{p(\mathcal{X}|\theta)p(\theta)}{p(\mathcal{X})} = \frac{p(\mathcal{X}|\theta)p(\theta)}{\int p(\mathcal{X}|\theta')p(\theta')d\theta'} \tag{4.13}$$

Bu sonsal dağılımı kullanarak belirli bir x noktasındaki dağılımı kestirebiliriz:

$$p(x|\mathcal{X}) = \int p(x, \theta|\mathcal{X})d\theta$$

$$= \int p(x|\theta, \mathcal{X})p(\theta|\mathcal{X})d\theta$$
$$= \int p(x|\theta)p(\theta|\mathcal{X})d\theta$$

$p(x|\theta, \mathcal{X}) = p(x|\theta)$, çünkü θ yeterli istatistiğini bildiğimizde dağılım hakkında her şeyi biliyoruz demektir. Bu yüzden olasılıkları ağırlıklar olarak alıp olası tüm θ değerleri üzerinden öngörülerin ortalamasını alırız. Bağlanımda olduğu gibi, $y = g(x|\theta)$ biçiminde bir öngörü yapıyorsak öngörüler üzerinden olasılıklara bağlı ağırlıklı ortalama alırız:

$$y = \int g(x|\theta)p(\theta|\mathcal{X})d\theta$$

Sonsal dağılımın uygun olduğu bazı durumlar dışında bu integralleri almak çok zor olabilir ve tüm integral alınamadığında dağılımı tek bir noktaya indirgeyebiliriz. $p(\theta|\mathcal{X})$ sonsalının tepesi çevresinde dar dağıldığını varsayarsak *en büyük sonsal kestirimde* (EBS)

$$\theta_{EBS} = \arg\max_{\theta} p(\theta|\mathcal{X}) \tag{4.14}$$

tüm dağılım yerine tek bir nokta kullanır ve böylece integralden kurtuluruz:

$$p(x|\mathcal{X}) = p(x|\theta_{EBS})$$
$$y_{EBS} = g(x|\theta_{EBS})$$

Bazı θ değerlerini ötekilere yeğlememizi gerektiren önsel bir neden yoksa önsel dağılım tekdüze dağılmıştır ve sonsal ile olabilirlik benzer dağılır. Bu durumda, en büyük sonsal kestirimle en büyük olabilirlik kestirimi (bölüm 4.2) aynı sonucu verir:

$$\theta_{EBO} = \arg\max_{\theta} p(\mathcal{X}|\theta) \tag{4.15}$$

Bir başka olası yöntem, sonsal dağılımın beklenen değeri olarak tanımlanan *Bayesçi kestiricidir*:

$$\theta_{Bayes} = E[\theta|\mathcal{X}] = \int \theta p(\theta|\mathcal{X})d\theta \tag{4.16}$$

Beklenen değeri almamızın nedeni rastsal bir değişkenin en iyi kestiriminin onun ortalaması olmasıdır. Diyelim ki, θ değişkenini öngörmek istiyoruz, $E[\theta] = \mu$, ve θ için kestirimimiz sabit bir c olacak. Bu durumda beklenen kare hata

$$E[(\theta - c)^2] = E[(\theta - \mu + \mu - c)^2]$$
$$= E[(\theta - \mu)^2] + (\mu - c)^2 \tag{4.17}$$

$c = \mu$ olduğunda en küçük değeri alır. Normal dağılımın tepesiyle beklenen değeri aynı noktadadır ve $p(\theta|\mathcal{X})$ normalse, $\theta_{Bayes} = \theta_{EBS}$ olur.

Örneğin, $x^t \sim \mathcal{N}(\theta, \sigma^2)$ ve $\theta \sim \mathcal{N}(\mu_0, \sigma_0^2)$ olsun ve μ_0, σ_0^2, σ^2 biliniyor olsun:

$$p(\mathcal{X}|\theta) = \frac{1}{(2\pi)^{N/2}\sigma^N} \exp\left[-\frac{\sum_t (x^t - \theta)^2}{2\sigma^2}\right]$$

$$p(\theta) = \frac{1}{\sqrt{2\pi}\sigma_0} \exp\left[-\frac{(\theta - \mu_0)^2}{2\sigma_0^2}\right]$$

Bu durumda, $p(\theta|\mathcal{X})$ sonsal dağılımının normal olduğu ve beklenen değerinin (Bayesçi kestirici) μ_0 önsel değeriyle m örneklem ortalamasının ağırlıklı ortalaması olduğu gösterilebilir:

$$E[\theta|\mathcal{X}] = \frac{N/\sigma^2}{N/\sigma^2 + 1/\sigma_0^2} m + \frac{1/\sigma_0^2}{N/\sigma^2 + 1/\sigma_0^2} \mu_0 \tag{4.18}$$

Ağırlıkları değişintileriyle ters orantılıdır. Örneklem büyüklüğü N arttıkça Bayes kestiricisi örneklemin verdiği bilgiyi daha çok kullanır ve örneklem ortalamasına yaklaşır. σ_0^2 küçükken, yani θ hakkındaki önsel belirsizliğimiz azken, ya da N küçükse, μ_0 önsel öngörüsünün ağırlığı daha fazladır.

EBS ve Bayes.i kestiricilerin ikisi de tüm sonsal dağılımı tek bir noktaya indirgediğinden sonsalın tek ve dar tepeli olduğu durumlar dışında bilgi yitimine yol açarlar. Son yıllarda hesaplamanın ucuzlamasıyla, Monte Carlo yaklaşımını kullanarak sonsal dağılımdan örnekleyerek yaklaşık sonuç üreten yaklaşımlar daha yaygın kullanılır olmuştur (Andrieu ve ötekiler 2003). Tüm integrali hesaplamak için olası başka yaklaşıklama yöntemleri de vardır. Bayesçi kestirimi 14. bölümde daha ayrıntılı olarak ele alacağız.

4.5 Dağılıma Bağlı Sınıflandırma

3. bölümde Bayes kuralını kullanarak C_i sınıfının sonsal olasılığını nasıl hesaplayabileceğimizi gördük:

$$P(C_i|x) = \frac{p(x|C_i)P(C_i)}{p(x)} = \frac{p(x|C_i)P(C_i)}{\sum_{k=1}^{K} p(x|C_k)P(C_k)} \tag{4.19}$$

Bunu kullanarak ayırtaç işlevi

$$g_i(x) = p(x|C_i)P(C_i)$$

ya da özdeş bir başka ayırtacı

$$g_i(x) = \log p(x|C_i) + \log P(C_i) \tag{4.20}$$

tanımlayabileceğimizi görmüştük. $p(x|C_i)$ dağılımlarının normal olduğunu

$$p(x|C_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left[-\frac{(x - \mu_i)^2}{2\sigma_i^2}\right] \tag{4.21}$$

varsayabilirsek denklem 4.20 basitçe yazılabilir:

$$g_i(x) = -\frac{1}{2}\log 2\pi - \log \sigma_i - \frac{(x-\mu_i)^2}{2\sigma_i^2} + \log P(C_i) \tag{4.22}$$

Bir örnek görelim: K farklı otomobil satan bir acentemiz olsun ve basitlik için bir müşterinin otomobil seçiminde etkili olan tek etmenin x ile gösterdiğimiz yıllık geliri olduğunu varsayalım. Bu durumda, $P(C_i)$, i. tür otomobili satın alan müşterilerin oranı olur. Eğer belirli bir tür otomobil alan müşterilerin gelirlerinin normal dağıldığını varsayabilirsek $p(x|C_i) \sim \mathcal{N}(\mu_i, \sigma_i^2)$ olarak yazabiliriz, öyle ki, μ_i ve σ_i^2, i. tür otomobili satın alan müşterilerin ortalama geliri ve gelirlerinin değişintisini gösterir.

$P(C_i)$ ve $p(x|C_i)$ dağılımlarını bilmediğimizde örneklemden kestirir ve kestirdiğimiz değerleri ayırtaçtaki yerlerine koyarız. Örneklemimiz, girdi ve sınıf etiketlerinden oluşur:

$$\mathcal{X} = \{x^t, r^t\}_{t=1}^N \tag{4.23}$$

burada $x \in \Re$ tek boyutludur. Sınıf etiketleri, $r \in \{0,1\}^K$, K boyutlu ikili bir yöney olarak verilmiştir, öyle ki

$$r_i^t = \begin{cases} 1 & \text{eğer } x^t \in C_i \\ 0 & \text{eğer } x^t \in C_k, k \neq i \end{cases} \tag{4.24}$$

Her sınıf için ayrı ayrı ortalama ve değişinti değerlerini (denklem 4.8'i kullanarak)

$$m_i = \frac{\sum_t x^t r_i^t}{\sum_t r_i^t} \tag{4.25}$$

$$s_i^2 = \frac{\sum_t (x^t - m_i)^2 r_i^t}{\sum_t r_i^t} \tag{4.26}$$

ve öncel olasılıkları (denklem 4.6'yı kullanarak)

$$\hat{P}(C_i) = \frac{\sum_t r_i^t}{N} \tag{4.27}$$

kestirir, denklem 4.22'ye yerleştirip ayırtaç işlevini yazarız:

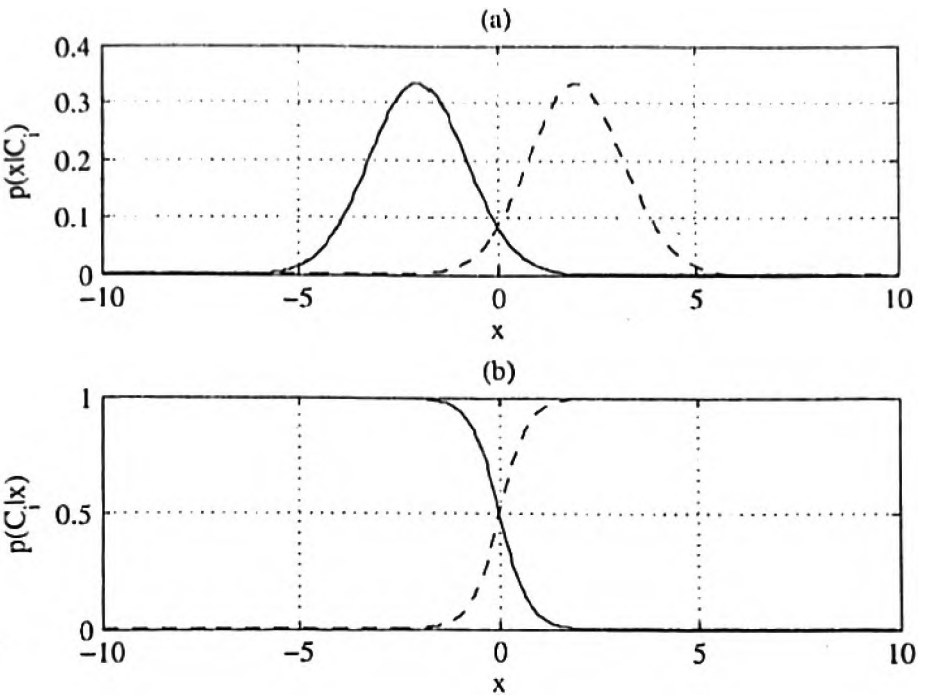
$$g_i(x) = -\frac{1}{2}\log 2\pi - \log s_i - \frac{(x-m_i)^2}{2s_i^2} + \log \hat{P}(C_i) \tag{4.28}$$

İlk terim sabit olduğundan tüm $g_i(x)$ işlevlerinde ortaktır ve atılabilir. Eğer önseller eşitse son terim de atılabilir. Değişintilerin de eşit olduğunu varsayarsak ayırtaç iyice basitleşir:

$$g_i(x) = -(x-m_i)^2 \tag{4.29}$$

Bu durumda x girdisi, ortalaması en yakın sınıfa atanır:

$$\text{Seç } C_i \text{ eğer } |x - m_i| = \min_k |x - m_k|$$



Şekil 4.2: Tek boyutlu, iki sınıflı bir girdi için (a) olabilirlik işlevleri ve (b) önsel dağılımların eşit olduğu durum için sonsal dağılımlar. Değişintiler eşit olduğundan sonsal dağılımlar tek bir noktada kesişir, bu da karar eşiğidir.

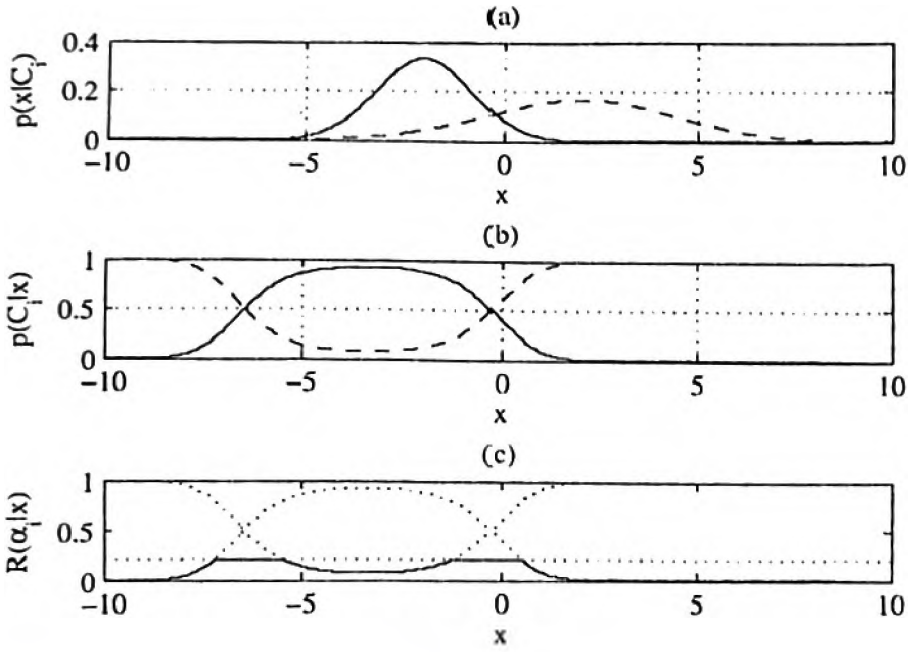
İki komşu sınıf için karar eşiği, iki sınıfın ortalamalarının ortasından geçer (bkz. şekil 4.2).

$$\begin{aligned} g_1(x) &= g_2(x) \\ (x-m_1)^2 &= (x-m_2)^2 \\ x &= \frac{m_1+m_2}{2} \end{aligned}$$

Değişintiler farklı olduğunda iki eşik vardır (bkz. şekil 4.3) ve bunlar kolayca hesaplanabilir (alıştırma 4). Eğer önseller farklıysa karar eşiği olasılığı daha düşük sınıfın ortalamasına doğru kayar.

Yukarıda parametreler için en büyük olabilirlik kestiricisini kullandık, ama parametreler hakkında (ortalamalarının ne olduğu gibi) önsel bilgilerimiz varsa, μ_i üzerine bir önsel dağılım tanımlayarak $p(x|C_i)$ dağılımını Bayesçi bir biçimde de kestirebiliriz.

Burada dikkat edilmesi gereken bir nokta var: x sürekliyse $p(x|C_i)$ için hemen normal dağılım varsaymamalıyız. Eğer dağılımlar normal değilse sınıflandırma algoritması, yani eşik noktaları, yanlış olacaktır. İstatistik yazınında verilen bir örneklemin normal olup olmadığını sınamak için denence sınamaları önerilmiştir ve normalliği varsaymadan önce bunlar kullanılmalıdır. Tek boyutlu veriler için en kolay sınama yolu verinin çubukçizitini çıkartarak dağılımın çan eğrisine benzer, yani tek tepeli ve tepe çevresinde bakışımlı olup olmadığına görsel olarak bakmaktır.



Şekil 4.3: Tek boyutlu, iki sınıflı bir girdi için (a) olabilirlik işlevleri ve (b) önsel dağılımların eşit olduğu durum için sonsal dağılımlar. Değişintiler farklı olduğundan sonsal dağılımlar iki noktada kesişir. (c) Ret eyleminin yitimi $\lambda = 0.2$ olduğunda beklenen risk işlevleri (kesikli çizgiler) ve en düşük risk (sürekli çizgi) (bölüm 3.3).

Burada, sınıflandırma için yaklaşımımız *olabilirlik tabanlıdır*; sınıf dağılımlarını ayrı ayrı kestirir, Bayes kuralıyla sonsal dağılımları hesaplar ve ayırtaçları buluruz. İleriki bölümlerde, dağılımların kestirimini atlayıp doğrudan ayırtaçları kestireceğimiz *ayırtaç tabanlı* yaklaşımı da ele alacağız.

4.6 Bağlanım

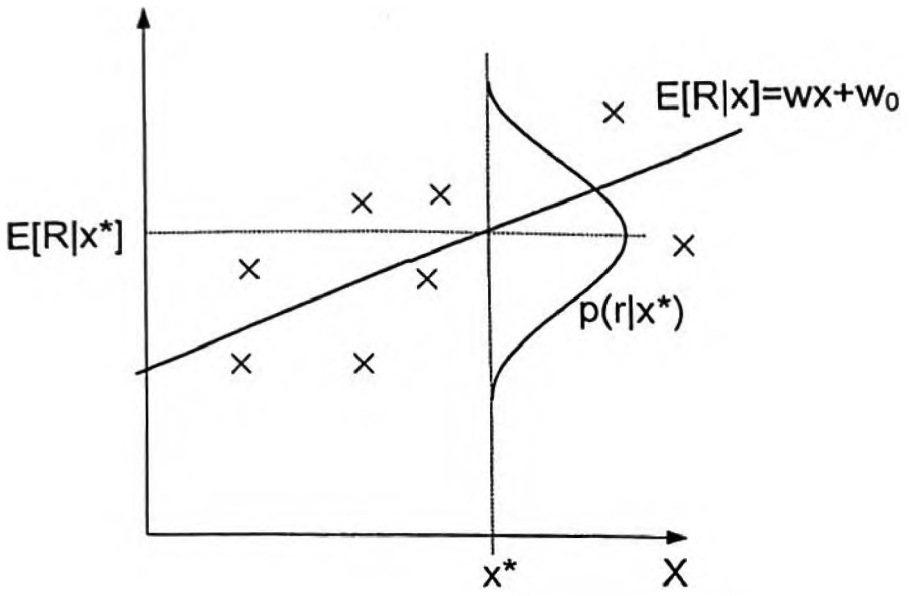
Bağlanımda, *bağımlı değişken* denilen sayısal çıktıyı *bağımsız değişken* denilen girdinin bir işlevi olarak yazmak isteriz. Sayısal çıktının, girdinin belirlenimci bir işleviyle rastgele gürültünün toplamı olduğunu varsayarız:

$$r = f(x) + \epsilon$$

Burada $f(x)$, bilinmeyen işlevi gösterir ve amacımız, θ parametreleriyle tanımlanan $g(x|\theta)$ kestiricisiyle onu yaklaşıklamaktır. ϵ ile gösterdiğimiz gürültünün sıfır ortalama ve sabit σ^2 değişintili normal olduğunu varsayarız: $\epsilon \sim \mathcal{N}(0, \sigma^2)$. $g(\cdot)$ kestiricimizi bilinmeyen $f(\cdot)$ işlevinin yerine koyduğumuzda çıktı için girdiye koşullu normal bir olasılık dağılımı elde ederiz (şekil 4.4):

$$p(r|x) \sim \mathcal{N}(g(x|\theta), \sigma^2) \tag{4.30}$$

θ değerlerini öğrenmek için yine en büyük olabilirlik kestirimini kullanırız. Öğrenme



Şekil 4.4: Bağlanım, 0 ortalamalı normal dağılmış gürültünün modele eklendiğini varsayar; buradaki model doğrusaldır.

kümesindeki (x^t, r^t) ikilileri bilinmeyen bir $p(x, r)$ birleşik olasılık dağılımından gelmektedir. Birleşik olasılığı koşullu olasılık cinsinden yazabiliriz:

$$p(x, r) = p(r|x)p(x)$$

$p(r|x)$, girdi verildiğinde çıktının olasılığı, $p(x)$ ise girdi olasılık dağılımıdır. Örneklem $\mathcal{X} = \{x^t, r^t\}_{t=1}^N$ için log olabilirliği yazalım:

$$\begin{aligned}\mathcal{L}(\theta|\mathcal{X}) &= \log \prod_{t=1}^{N} p(x^t, r^t) \\ &= \log \prod_{t=1}^{N} p(r^t|x^t) + \log \prod_{t=1}^{N} p(x^t)\end{aligned}$$

İkinci terim, kestiricimize bağlı olmadığı için göz ardı edilebilir.

$$\begin{aligned}\mathcal{L}(\theta|\mathcal{X}) &= \log \prod_{t=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{[r^t - g(x^t|\theta)]^2}{2\sigma^2}\right] \\ &= \log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)^N \exp\left[-\frac{1}{2\sigma^2}\sum_{t=1}^{N}[r^t - g(x^t|\theta)]^2\right] \\ &= -N\log(\sqrt{2\pi}\sigma) - \frac{1}{2\sigma^2}\sum_{t=1}^{N}[r^t - g(x^t|\theta)]^2\end{aligned} \tag{4.31}$$

İlk terim ve $1/\sigma^2$ çarpanı θ parametrelerinden bağımsız olduklarından atılabilir. Bunu büyütmekle, hata işlevini azaltmak özdeştir:

$$E(\theta|\mathcal{X}) = \frac{1}{2}\sum_{t=1}^{N}[r^t - g(x^t|\theta)]^2 \tag{4.32}$$

Bu hatayı en küçük yapan θ değerine *en küçük kare kestiricisi* denir. Bu, istatistikte sık sık yapılan bir dönüşümdür: l olabilirliği üstel terimler içerdiğinde, olabilirliği büyütmek yerine, eksi log olabilirliği bir hata işlevi olarak, $E = -\log l$, tanımlar ve onu küçültürüz.

Doğrusal bağlanımda modelimiz doğrusaldır:

$$g(x^t|w_1, w_0) = w_1 x^t + w_0$$

ve toplam kare hatanın (denklem 4.32) w_1 ve w_0 parametrelerine göre türevini alıp sıfıra eşitlediğimizde iki bilinmeyenli iki denklem elde ederiz:

$$\begin{aligned}\sum_t r^t &= Nw_0 + w_1\sum_t x^t\\ \sum_t r^t x^t &= w_0\sum_t x_t + w_1\sum_t (x^t)^2\end{aligned}$$

Bunu yöney-dizey biçiminde $\mathbf{A}\boldsymbol{w} = \boldsymbol{y}$ olarak yazabiliriz, öyle ki

$$\mathbf{A} = \begin{bmatrix} N & \sum_t x^t \\ \sum_t x^t & \sum_t (x^t)^2 \end{bmatrix},\ \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix},\ \boldsymbol{y} = \begin{bmatrix} \sum_t r^t \\ \sum_t r^t x^t \end{bmatrix}$$

$\boldsymbol{w} = \mathbf{A}^{-1}\boldsymbol{y}$ çözümüdür.

Genel durumda, *çokterimli bağlanımda*, modeli x girdisinin k. dereceden çokterimli bir işlevi olarak yazarız:

$$g(x^t|w_k, \ldots, w_2, w_1, w_0) = w_k(x^t)^k + \cdots + w_2(x^t)^2 + w_1 x^t + w_0$$

Model, parametrelere yine doğrusal bağlıdır ve türevleri aldığımızda $k+1$ bilinmeyenli $k+1$ denklem oluşur. Bunu yöney-dizey biçiminde $\mathbf{A}\boldsymbol{w} = \boldsymbol{y}$ olarak yazarız, öyle ki

$$\mathbf{A} = \begin{bmatrix} N & \sum_t x^t & \sum_t (x^t)^2 & \cdots & \sum_t (x^t)^k \\ \sum_t x^t & \sum_t (x^t)^2 & \sum_t (x^t)^3 & \cdots & \sum_t (x^t)^{k+1} \\ \vdots & & & & \\ \sum_t (x^t)^k & \sum_t (x^t)^{k+1} & \sum_t (x^t)^{k+2} & \cdots & \sum_t (x^t)^{2k} \end{bmatrix}$$

$$\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix},\ \boldsymbol{y} = \begin{bmatrix} \sum_t r^t \\ \sum_t r^t x^t \\ \sum_t r^t (x^t)^2 \\ \vdots \\ \sum_t r^t (x^t)^k \end{bmatrix}$$

$\mathbf{A} = \mathbf{D}^T\mathbf{D}$ ve $\boldsymbol{y} = \mathbf{D}^T\boldsymbol{r}$ olarak yazıp veri ve

$$\mathbf{D} = \begin{bmatrix} 1 & x^1 & (x^1)^2 & \cdots & (x^1)^k \\ 1 & x^2 & (x^2)^2 & \cdots & (x^2)^k \\ \vdots & & & & \\ 1 & x^N & (x^N)^2 & \cdots & (x^N)^k \end{bmatrix}, \boldsymbol{r} = \begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^N \end{bmatrix}$$

veri dizeylerini kullanarak daha basit bir gösterim elde edebiliriz:

$$\boldsymbol{w} = (\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T\boldsymbol{r} \tag{4.33}$$

Hatanın normal dağıldığını kabul edip olabilirliği büyütmek toplam kare hatayı küçültmeye karşılık gelir. Bir başka ölçüt, *görece kare hatadır* (GKH):

$$E_{GKH} = \frac{\sum_t [r^t - g(x^t|\theta)]^2}{\sum_t (r^t - \overline{r})^2} \tag{4.34}$$

E_{GKH}, bire yakınsa, öngörümüz ancak ortala kadar iyidir; sıfıra yaklaştıkça daha iyiyiz demektir. E_{GKH}, bire yakınsa, x girdisini kullanan model, elimizde hiç girdi olmasa yapacağımız öngörü olan ortalamaya göre daha iyi değil, yani girdiye bağlı bağlanım işe yaramıyor demektir; E_{GKH}, sıfıra yakınsa x girdisini kullanmak işe yarıyordur.

Bağlanımın ne kadar iyi uyduğunu ölçmekte *bağlanım belirleme katsayısı* kullanılır:

$$R^2 = 1 - E_{RSE}$$

Bağlanımın işe yarar olduğunu söyleyebilmek için R^2'nin bire yakın olması beklenir.

En iyi genelleme için öğrenen modelin karmaşıklığını verinin karmaşıklığına uydurmamız gerektiğini hatırlayalım. Çokterimli bağlanımda karmaşıklık parametresi oturtulan çok terimlinin derecesidir ve genelleme hatasını en aza indirecek en iyi dereceyi seçmek, yani verinin altında yatan bilinmeyen işlevin karmaşıklığına uyacak bir model karmaşıklığı belirlemek için bir yol bulmamız gerekmektedir.

4.7 Model Karmaşıklığı Ayarlama: Yanlılık/Değişinti İkilemi

Elimizde bilinmeyen $p(x, r)$ birleşik olasılık dağılımından çekilmiş $\mathcal{X} = \{x^t, r^t\}$ örneklemi olsun. Bu örneklemi kullanıp $g(\cdot)$ ile gösterdiğimiz kestiricimizi oluştururuz. x noktasındaki beklenen kare hata (birleşik dağılım üzerinden) (denklem 4.17'yi kullanarak) iki terim olarak yazılabilir:

$$E[(r - g(x))^2|x] = \underbrace{E[(r - E[r|x])^2|x]}_{\text{gürültü}} + \underbrace{(E[r|x] - g(x))^2}_{\text{kare hata}} \tag{4.35}$$

İlk terim, x verildiğinde r değerinin değişintisidir; $g(\cdot)$ ya da örnekleme bağlı değildir. Eklenmiş gürültünün değişintisi σ^2 olarak verilmiştir. Bu değer, hatanın, hangi kestirici kullanılırsa kullanılsın ortadan kaldırılamazyacak parçasıdır. İkinci terim ise $g(x)$ modelinin bağlanım işlevi $E[r|x]$ değerinden ne kadar saptığını ölçer. Bu, kestirici ve öğrenme kümesine bağlıdır. Bir örneklem için $g(x)$ veriye çok iyi oturmuşken, bir başka örneklem için kötü bir kestirim yapabilir. Bir $g(\cdot)$ kestiricisinin ne kadar iyi olduğunu nicelemek için olası veri kümeleri üzerinden ortalama almak gerekir.

Beklenen değeri (hepsi N örnekli ve aynı $p(r,x)$ birleşik dağılımından çekilmiş $\mathcal{X}$ örneklemleri üzerinden) yanlılığın karesi ve değişintinin toplamı olarak yazabiliriz (denklem 4.11'i kullanarak):

$$E_{\mathcal{X}}[(E[r|x]-g(x))^2|x] = \underbrace{(E[r|x]-E_{\mathcal{X}}[g(x)])^2}_{\text{yanlılık}} + \underbrace{E_{\mathcal{X}}[(g(x)-E_{\mathcal{X}}[g(x)])^2]}_{\text{değişinti}} \quad (4.36)$$

Daha önce belirttiğimiz gibi, yanlılık, örneklemlerdeki değişmeleri dikkate almayarak $g(x)$ modelinin ne kadar yanlış olduğunu, değişinti ise $g(x)$ modelinin farklı örneklemlerde beklenen $E[g(x)]$ değeri çevresinde ne kadar dalgalandığını ölçer. İkisinin de küçük olmasını isteriz.

Öğretici bir örnek görelim: Yanlılık ve değişintiyi kestirmek için bildiğimiz bir $f(\cdot)$ işlevine gürültü ekleyerek M farklı veri kümesi, $\mathcal{X}_i = \{x_i^t, r_i^t\}, i = 1,\ldots,M$, üretelim, sonra her bir veri kümesini kullanıp bir $g_i(\cdot)$ modeli oluşturalım ve üzerlerinden yanlılık ve değişintiyi hesaplayalım. Gerçek yaşamda, ne $f(\cdot)$ işlevini, ne de gürültü parametrelerini bilemeyeceğimiz için böyle bir şey yapamayız tabii ki. $E[g(x)]$, $g_i(\cdot)$ modelleri üzerinden ortalama alarak kestirilir:

$$\bar{g}(x) = \frac{1}{M}\sum_{i=1}^{M} g_i(x)$$

Yanlılık ve değişintiyi de $g_i(\cdot)$ değerlerinden hesaplayabiliriz:

$$\text{Bias}^2(g) = \frac{1}{N}\sum_t [\bar{g}(x^t) - f(x^t)]^2$$

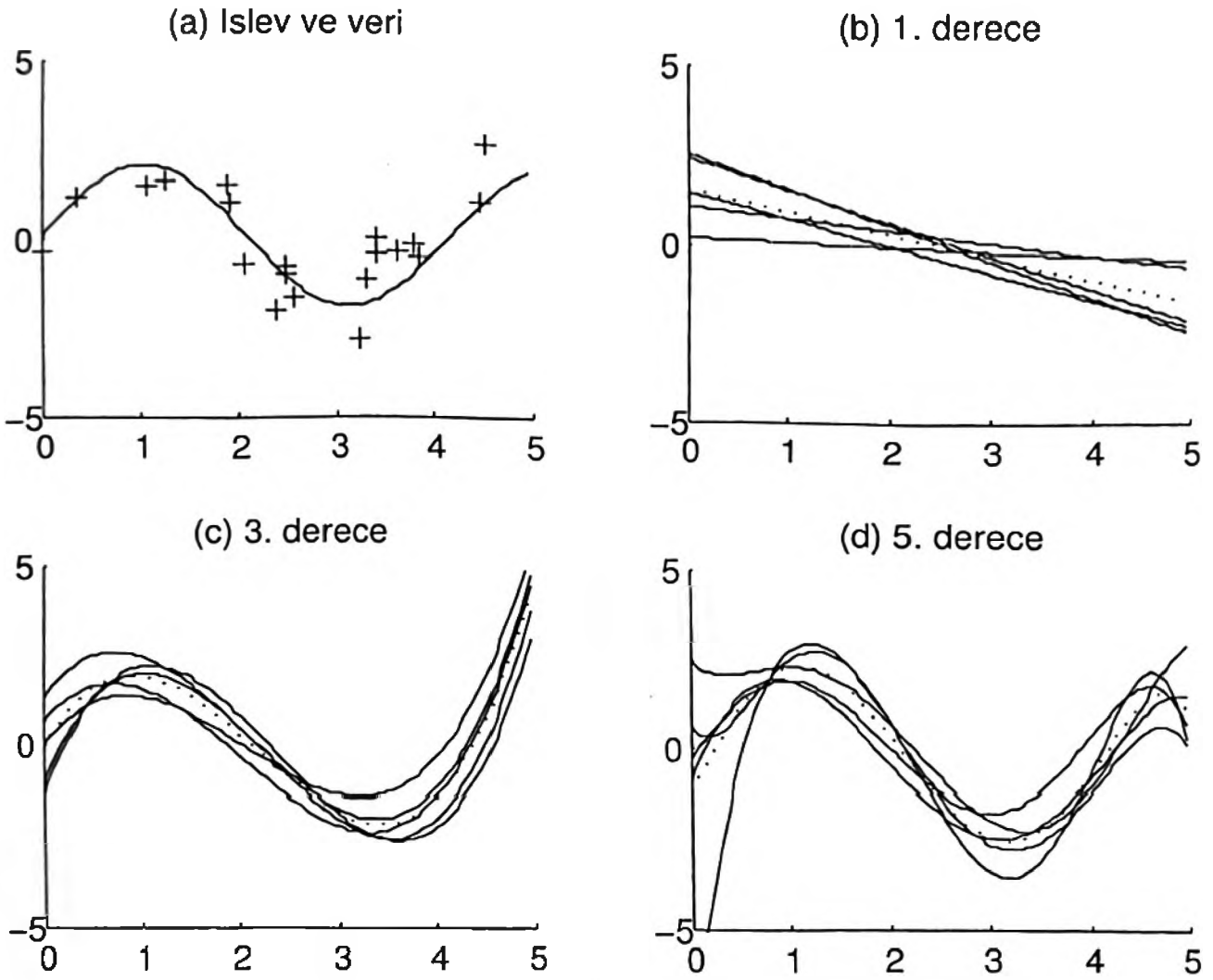
$$\text{Variance}(g) = \frac{1}{NM}\sum_t\sum_i [g_i(x^t) - \bar{g}(x^t)]^2$$

Şimdi, farklı karmaşıklığı olan bazı modelleri görelim: En basiti bir sabittir:

$$g_i(x) = 2$$

Bunun değişintisi yoktur çünkü veri kullanılmamaktadır ve tüm $g_i(x)$ modelleri eşittir. Ancak, $f(x)$, tüm x girdileri için 2 ya da yakın değilse, yanlılığı yüksetir. Sabit olarak, 2 yerine örneklemdeki r^t değerlerinin ortalamasını alırsak

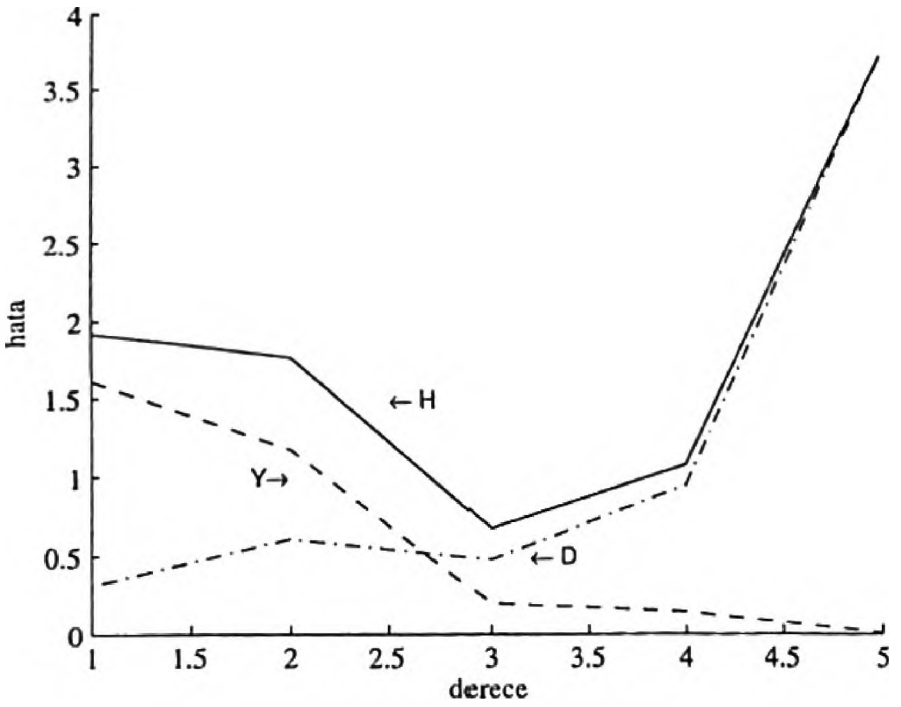
$$g_i(x) = \sum_t r_i^t/N$$



Şekil 4.5: (a) $f(x) = 2\sin(1.5x)$ işlevi ve gürültü ($\mathcal{N}(0,1)$) eklenmiş örnek veri kümesi. Her biri yirmi örnek içeren beş örneklem çekilmiştir. (b), (c), (d)'de beşer tane, sırasıyla 1, 3, and 5. dereceden oturtulmuş çokterimliler gösterilmiştir. Noktalı çizgi, beş kestirimin ortalamasını ($\bar{g}(\cdot)$) gösterir.

yanlılık azalır çünkü ortalamanın genelde daha iyi bir kestirim olmasını bekleriz. Ama değişinti artar, çünkü farklı $\mathcal{X}_i$ örneklemlerinin farklı ortalama değerleri olacaktır. Genelde böyle bir durumda, yanlılıktaki azalmanın değişintideki artıştan daha fazla olmasını ve hatanın azalmasını bekleriz.

Çokterimli bağlanım için bir örnek, şekil 4.5'te verilmiştir. Çokterimlinin derecesi arttıkça veri kümesindeki küçük değişiklikler oturtulan çokterimlilerde daha büyük değişikliklere neden olur, yani değişinti artar. Ama karmaşık bir model ortalama olarak altta yatan işleve daha iyi yaklaşır, yani yanlılık azalır (şekil 4.6). Buna *yanlılık/değişinti ikilemi* denir ve yalnız çokterimli bağlanım için değil, tüm yapay öğrenme model aileleri için geçerlidir (Geman, Bienenstock ve Doursat 1992). Yanlılığı azaltmak için, değişintiyi artırma pahasına, modelin esnekliğini artırmak gerekir. Değişinti küçük tutulursa veriye yeterince iyi yaklaşım yapılamayabilir ve



Şekil 4.6: Şekil 4.5'le aynı kurguda beş yerine yüz model kullanarak birden beşe dek dereceden çokterimliler için yanlılık (Y), değişinti (D) ve hata (H) çizilmiştir. 1. derecenin değişintisi, 5. derecenin yanlılığı en küçüktür. Derece arttıkça yanlılık azalırken değişinti artar. 3. derecenin hatası en küçüktür.

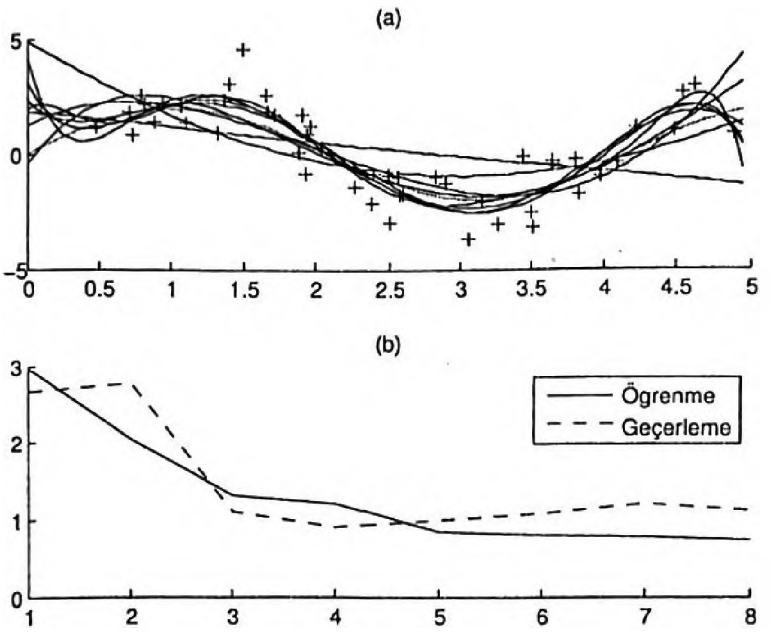
yüksek yanlılık oluşabilir. En iyi model, yanlılık ve değişinti arasında en iyi ödünleşime sahip olanıdır.

Yanlılık, model sınıfımızın çözümü içermediğini gösterir. Modelimizin zayıf kaldığı bu duruma *eksik öğrenme* denir. Eğer değişinti varsa, model sınıfımız fazla geneldir ve gürültüyü de öğrenmektedir; buna *aşırı öğrenme* denir. Eğer $g(\cdot)$, $f(\cdot)$ ile aynı denence sınıfındansa, örneğin aynı dereceden çokterimlilerse, kestiricimiz yansızdır ve model sayısı arttıkça yanlılık azalır. Bu da doğru modeli seçmenin hatayı azaltma etkisini gösterir; bundan 2. bölümde doğru model varsayımları belirlemek diye bahsetmiştik. Değişinti, öğrenme kümesinin büyüklüğüne bağlıdır; örneklem büyüklüğü arttıkça örnekleme bağlı değişebilirlik azalır. Özetle, hatanın küçük olabilmesi için, uygun model varsayımlarıyla yanlılığı küçük tutmalı, ve yeterince büyük bir veri kümesi kullanıp modelin değişebilirliğini kısıtlamalıyız.

Değişinti büyükken yanlılık küçük olur; bu da $\overline{g}(x)$ ortalama modelinin iyi bir kestirici olduğunu gösterir. Dolayısıyla hata miktarını küçültmek için çok sayıda yüksek değişintili model alıp ortalamalarını kestiricimiz olarak kullanabiliriz. Bu tür model birleştirme yaklaşımlarını 17. bölümde göreceğiz.

4.8 Model Seçim Yordamları

Model karmaşıklığını ayarlamak kullanabileceğimiz birkaç yordam vardır.



Şekil 4.7: Şekil 4.5'le aynı kurguda, her biri 50 örnekli öğrenme ve geçerleme kümeleri çekilmiştir. (a) Öğrenme verisi ve 1. dereceden 8. dereceye kadar oturtulan örnek çokterimliler. (b) Dereceye bağlı olarak öğrenme ve geçerleme hataları. "Dirsek" 3'tedir.

En uygun model karmaşıklığını bulmakta en yaygın kullanılan yöntem *çapraz geçerlemedir*. Bir model için yanlılık ve değişintiyi ayrı ayrı hesaplayamayız ama toplam hatayı hesaplayabiliriz. Verilen bir veri kümesini öğrenme ve geçerleme diye iki parçaya ayırır, farklı karmaşıklıklardaki aday modelleri öğrenme kümesi üzerinde eğitir ve geçerleme örnekleri üzerindeki hatalasını ölçeriz. Model karmaşıklığı artarken öğrenme hatası azalır ama geçerleme kümesinde üzerindeki hata belirli bir karmaşıklığa kadar azalır, sonra azalmaz ya da çok az azalır, hatta gürültü çoksa artabilir bile. Bu "dirsek" noktası en uygun karmaşıklığa karşılık gelir (bkz. şekil 4.7).

Gerçek hayatta, şekil 4.6'daki gibi yanlılığı ve dolayısıyla hatayı hesaplayamayız; şekil 4.7'deki geçerleme hatası bunun bir kestirimidir ama içinde gürültü de vardır: doğru (yansız ve değişintiyi yok sayabilecek ölçüde büyük veriyle eğitilmiş) bir modelin bile geçerleme hatası tam sıfır olmayabilir. Şekil 4.7'deki geçerleme hatası, şekil 4.6'daki gibi 'V' biçiminde değildir çünkü ilki daha fazla veri kullanır ve ek veri değişintiyi azaltır. Gerçekten de şekil 4.5(d)'de, veri olan yerlerde, beşinci dereceden çokterimlinin üçüncü derece gibi davrandığını, ama daha az veri olan uçlarda daha fazla hata yaptığını görüyoruz.

Yaygın kullanılan bir başka yaklaşım *düzenlileştirmedir* (Breiman 1998). Bu yaklaşımda, model karmaşıklığını da içeren genişletilmiş bir hata işlevi kullanılır:

$$E' = \text{veri üzerindeki hata} + \lambda \cdot \text{model karmaşıklığı} \tag{4.37}$$

İkinci terim, karmaşık (yüksek değişintili) modelleri cezalandırır ve λ bu cezanın

ağırlığını belirler. Sırf veri üzerindeki hata yerine genişletilmiş hatayı küçültmek karmaşık modelleri cezalandırır ve dolayısıyla değişintiyi azaltır. Eğer λ fazla büyük seçilirse, yalnızca çok basit modellere izin verilir ve yanlılık riski oluşur. λ parametresinin en uygun değeri çapraz geçerleme kullanılarak bulunur.

Denklem 4.37'de verilen E', yeni sınama verisindeki hata gibi de düşünülebilir. İlk terim öğrenme hatası, ikinci terim de öğrenme ve sınama hataları arasındaki farkı kestiren bir *iyimserlik* terimidir (Hastie, Tibshirani ve Friedman 2001). *Akaike bilgi ölçütü* ve *Bayesçi bilgi ölçütü* gibi yöntemler sınama hatasını kestirirken bu iyimserliği kestirip öğrenme hatasına eklerler ve geçerleme kümesi kullanmazlar. Bu iyimserlik teriminin büyüklüğü, girdi sayısı d ile (burada $k+1$) doğrusal olarak artar ve öğrenme kümesi büyüklüğü N arttıkça azalır, eklenen gürültünün değişintisi σ^2 (ki bunu düşük yanlılığı olan bir modelin hatasıyla kestirebiliriz) ile artar. Doğrusal olmayan modellerde d yerine "etkin" parametre sayısı kullanılmalıdır.

Yapısal risk küçültmesi (Vapnik 1995), karmaşıklıklarına göre sıralanmış bir dizi model kullanır. Artan dereceli çokterimliler buna bir örnektir. Karmaşıklık genelde serbest değişkenlerin sayısı olarak tanımlanır. VC boyutu, karmaşıklığın bir başka ölçütüdür. Denklem 4.37'de azalan sırada λ_i değerleri kullanarak artan karmaşıklıkta bir dizi model elde edebiliriz. Bu durumda yapısal risk küçültmesiyle model seçimi yapmak, en basit ve veri üzerinde gözlemsel hatası en düşük modeli belirlemeye karşılık gelir.

En küçük betimleme uzunluğu (Rissanen 1978; Grünwald 2007), bilişim kuramsal bir ölçüt kullanır. Bir veri kümesinin *Kolmogorov karmaşıklığı*, verinin en kısa betimlemesi olarak tanımlanmıştır. Veri basitse betimlemesi kısadır; örneğin, bir dizi 0 yerine, tek bir 0 yazıp yanına dizinin uzunluğunu koyabiliriz. Eğer veri tamamen rastgeleyse kendisinden daha kısa bir betimlemesi olamaz. Eğer bir model veri için uygunsa, veriye iyi oturmuştur ve veri yerine modelin betimlemesini gönderebilir ya da saklayabiliriz. Veriyi açıklayan modeller içinden en basit olanı, yani en kısa betimlemeye sahip olanı isteriz. Böylece, modelin ne kadar basit olduğu ile veriyi ne kadar açıkladığı arasında yine bir ödünleşme söz konusu olur.

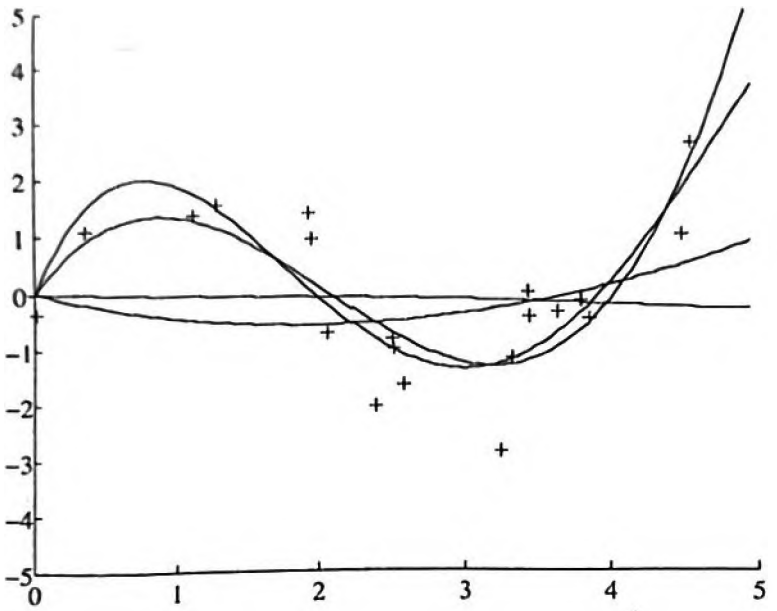
Bayesçi model seçimi, yaklaşıklama işlev sınıfı hakkında önsel bilgimiz olduğunda kullanılır. Bu önsel bilgi, modeller üzerinde bir $p(\text{model})$ önsel dağılımı tanımlar. Veri geldiğinde bir model varsayıp Bayes kuralıyla $p(\text{model}|\text{veri})$ sonsal olasılığı hesaplanabilir:

$$p(\text{model}|\text{veri}) = \frac{p(\text{veri}|\text{model})p(\text{model})}{p(\text{veri})} \tag{4.38}$$

$p(\text{model}|\text{veri})$, model hakkındaki öznel bilgimize bağlı önsel $p(\text{model})$ dağılımı ve veriden gelen $p(\text{data}|\text{model})$ nesnel desteği kullanılarak hesaplanan sonsal dağılımdır. En yüksek sonsal olasılığa sahip modeli seçebilir, ya da sonsal olasılıkları ağırlıklar olarak alıp tüm modellerin ağırlıklı ortalamasını alabiliriz. Denklem 4.38'in logaritmasını alırsak, denklem 4.37'yle aynı biçimde bir denklem elde ederiz:

$$\log p(\text{model}|\text{veri}) = \log p(\text{veri}|\text{model}) + \log p(\text{model}) - c \tag{4.39}$$

Verinin log olabilirliği öğrenme hatasına, önselin logaritması da bir ceza terimine karşılık gelir. Örneğin, bir bağlanım modelimiz olsun ve $p(\boldsymbol{w}) \sim \mathcal{N}(0, 1/\lambda)$ önselini



Şekil 4.8: Şekil 4.5'le aynı kurguda, birden dördüncü dereye dek çokterimlilerle bağlanım yapılmıştır. Çokterimlinin derecesi arttıkça katsayıların mutlak büyüklükleri de artar: 1 : $[-0.0769, 0.0016]^T$, 2 : $[0.1682, -0.6657, 0.0080]^T$, 3 : $[0.4238, -2.5778, 3.4675, -0.0002]^T$, 4 : $[-0.1093, 1.4356, -5.5007, 6.0454, -0.0019]^T$.

kullanıyor olalım. En büyük sonsal kestirime karşılık gelen hata işlevini yazalım:

$$E = \sum_t [r^t - g(x^t|w)]^2 + \lambda \sum_i w_i^2 \tag{4.40}$$

Yani, hem hatayı azaltan hem de sıfıra olabildiğince yakın w_i değerleri ararız. Sıfıra yakın istememizin nedeni, öğrenilen çokterimlinin daha düzgün olabilmesi içindir. Çok terimlinin derecesi arttıkça veriye daha iyi yaklaşmak için işlev yukarı aşağı değişecektir. Bu da katsayıların sıfırdan uzaklaşması anlamına gelir (bkz. şekil 4.8). Ceza terimi ekleyerek işlevi daha düz ve daha düzgün bir yaklaşım yapmaya zorluyoruz. Ne kadar cezalandırdığımız λ değerine bağlıdır, ki bu, önselin değişintisinin tersidir ve ağırlıkların sıfırdan ne kadar uzaklaşmasını beklediğimizdir. Yani böyle bir önselimizin olması parametreleri sıfıra yakın olmaya zorlamaya eşdeğerdir. Bu konuyu daha ayrıntılı olarak 14. bölümde ele alacağız.

Buradan görüyoruz ki önsel dağılım, (Occam'ın usturasına uygun olarak) basit modellere daha yüksek olasılıklar verecek biçimde seçildiğinde, Bayesçi yaklaşım, düzenlileştirme, yapısal risk küçültmesi ve en küçük betimleme uzunluğu benzeşir. Çapraz geçerleme, model hakkında hiçbir varsayım yapmadığı için öteki model seçim yordamlarından farklıdır ve yeterince büyük bir geçerleme kümesi olduğunda en iyi yaklaşımdır. Ötekiler küçük veri örneklemleri üzerinde kullanışlı olur.

4.9 Kısa Kısa

En büyük olabilirlik ve Bayesçi kestirimin temelleri üzerine iyi bir kaynak (Ross 1987)'dir. Çoğu örüntü tanıma ders kitabı dağılıma bağlı sınıflandırmayı ayrıntılı ele alır, örneğin, MacLachlan 1992; Devroye, Györfi, ve Lugosi 1996; Webb 1999; Duda, Hart, ve Stork 2001). Tek boyutlu normallik için sınamalar (Rencher 1995)'de bulunabilir.

Geman, Bienenstock, ve Doursat (1992), ileriki bölümlerde işleyeceğimiz birçok öğrenme modeli için yanlılık ve değişinti ayrıştırmasını ele alır. Yanlılık ve değişinti ayrıştırması, toplam kare hata ve bağlanım için geçerlidir; hatanın böyle güzel, eklemeli bir biçimde yanlılık ve değişintiye bölünmesi 0/1 yitimi için olası değildir, çünkü sınıflandırmada hata yalnız sınırın öteki tarafına geçersek oluşur. Örneğin, iki sınıflı bir problem için, doğru sonsal 0.7 ve bizim kestirimimiz 0.8 olsa bile hata yoktur; hata yalnız 0.5 değerinin altında kestirim yaparsak oluşur. Çeşitli araştırmacılar sınıflandırma için yanlılık ve değişintinin farklı tanımlarını öne sürmüşlerdir (Friedman 1997).

4.10 Alıştırmalar

1. Verilen bir p parametresine göre Bernoulli bir örneklemi üretip ardından ondan $\hat{p}$ değerini kestiren bir program yazınız.
2. Katlıterimli bir örneklemin log olabilirliğini yazıp denklem 4.6'yı gösterin.
3. Verilen μ ve σ için normal dağılmış bir örneklem üretip ardından m ve s değerlerini örneklemden kestiren bir program yazın. Aynı şeyi, μ için bir önsel dağılım varsayıp Bayes kestiricisi için de yapıp karşılaştırın.
4. İki normal dağılım $p(x|C_1) \sim \mathcal{N}(\mu_1, \sigma_1^2)$ ve $p(x|C_2) \sim \mathcal{N}(\mu_2, \sigma_2^2)$ ve önsel olasılıklar, $P(C_1)$ ve $P(C_2)$ için Bayesçi ayırtacın karar noktalarını çözümsel olarak hesaplayın.
5. Normal dağılım için olabilirlik oranı

$$\frac{p(x|C_1)}{p(x|C_2)}$$

nedir?

6. Farklı değişintisi olan iki normal sınıflı bir uygulama için örneklem üretin, karar noktalarını hem çözümsel, hem de veriden hesaplayıp karşılaştırın.
7. Doğrusal bir model varsayın ve 0 ortalamalı normal gürültü ekleyerek bir örneklem oluşturun. Örneklemi öğrenme ve geçerleme kümelerine bölün. Öğrenme kısmını kullanarak doğrusal bağlanım yapın. Geçerleme kümesi üzerindeki hatayı hesaplayın. Aynı şeyi ikinci ve üçüncü dereceden çokterimliler için de yapın.
8. Öğrenme kümesi küçük olduğunda değişintinin hataya katkısı yanlılıktan daha fazla olabilir, ve böyle bir durumda yapılması gereken iş için, fazla basit olduğunu bilsek de basit bir modeli yeğleyebiliriz. Bu duruma bir örnek verebilir misiniz?

9. $\mathcal{X}_i = \{x_i^t, r_i^t\}$ örneklemleri üzerinde $g_i(x) = r_i^1$ kestiricilerimizi şöyle olsun: Herhangi bir x için kestirdiğimiz değer (sıralanmamış) $\mathcal{X}_i$ içindeki ilk r değeri olsun. Bu kestiricinin yanlılık ve değişintisi için, $g_i(x) = 2$ ve $g_i(x) = \sum_t r_i^t/N$ ile karşılaştırdığımızda ne söyleyebiliriz ? Örneklem sıralanmışsa, $g_i(x) = \min_t r_i^t$, ne fark eder ?
10. Denklem 4.40'ta λ değerini değiştirmenin yanlılık ve değişinti üzerindeki etkisi nedir ?

4.11 Kaynaklar

Andrieu, C., N. de Freitas, A. Doucet, M. I. Jordan. 2003. "An Introduction to MCMC for Machine Learning." *Machine Learning* 50: 5–43.

Breiman, L. 1998. "Bias-Variance, Regularization, Instability and Stabilization." *Neural Networks and Machine Learning*, ed. C. M. Bishop, 27–56. Berlin: Springer.

Devroye, L., L. Györfi, G. Lugosi. 1996. *A Probabilistic Theory of Pattern Recognition.* New York: Springer.

Duda, R. O., P. E. Hart, D. G. Stork. 2001. *Pattern Classification*, 2nd ed. New York: Wiley.

Friedman, J. H. 1997. "On Bias, Variance, 0/1-Loss and the Curse of Dimensionality." *Data Mining and Knowledge Discovery* 1: 55–77.

Geman, S., E. Bienenstock, R. Doursat. 1992. "Neural Networks and the Bias/Variance Dilemma." *Neural Computation* 4: 1–58.

Grünwald, P. D. 2007. *The Minimum Description Length Principle.* Cambridge, MA: MIT Press.

Hastie, T., R. Tibshirani, J. Friedman. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* New York: Springer.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition.* New York: Wiley.

Rencher, A. C. 1995. *Methods of Multivariate Analysis.* New York: Wiley.

Rissanen, J. 1978. "Modeling by Shortest Data Description." *Automatica* 14: 465–471.

Ross, S. M. 1987. *Introduction to Probability and Statistics for Engineers and Scientists.* New York: Wiley.

Vapnik, V. 1995. *The Nature of Statistical Learning Theory.* New York: Springer.

Webb, A. 1999. *Statistical Pattern Recognition.* London: Arnold.

Bölüm 5

Çok Değişkenli Yöntemler

4. bölümde, sınıflandırma ve bağlanım için dağılıma bağlı yaklaşımları tartıştık. Şimdi bunu, çıktının çok sayıda girdinin bir işlevi olduğu çok değişkenli duruma genelleyeceğiz. Bu girdiler ayrık ya da sayısal olabilir. Çok değişkenli bir örneklemden böylesi sınıflandırma ya da bağlanım işlevlerinin nasıl öğrenilebileceğini ve model karmaşıklığının eldeki veriye nasıl uyarlanabileceğini göreceğiz.

5.1 Çok Değişkenli Veri

Çoğu uygulamada, her nesne, ya da olay üzerinde farklı ölçümler yapıp bir gözlem yöneyi oluşturulur. Böylesi bir örneklem, yapılan ölçümleri gösteren d *değişkene* karşılık gelen d sütunlu bir *veri dizeyi* olarak gösterilir:

$$\mathbf{X} = \begin{bmatrix} X_1^1 & X_2^1 & \cdots & X_d^1 \\ X_1^2 & X_2^2 & \cdots & X_d^2 \\ \vdots & & & \\ X_1^N & X_2^N & \cdots & X_d^N \end{bmatrix}$$

Bu değişkenlere, *girdi* ya da *öznitelik* denir. Dizeyin N satırı, N farklı nesne, ya da olaya karşılık gelir ve N tane bağımsız ve özdeş dağılmış *gözlem* ya da *örnektir*.

Örneğin, kredi başvurusu değerlendirirken gözlem yöneyi, yaş, medeni durum, yıllık gelir, vs. gibi müşteri bilgileridir ve böyle N eski müşterimiz vardır. Bu değerler farklı ölçeklerde olabilir; örneğin, yaş değeri yıl olarak onlar, gelir ise bir para birimi olarak binler ölçeğinde olabilir. Yaş, gelir gibi değişkenler sayısalken medeni durum, meslek gibi bazıları ayrık olabilir.

Çoğunlukla bu değişkenler ilintilidir. Değillerse çok değişkenli çözümlemeye gerek yoktur. Amacımız *basitleştirme*, yani bu karmaşık veriyi olabildiğince az parametreli bir modelle açıklamak olabilir. Ya da *bilgi çıkarmak*, yani veri hakkında

anlaşılabilir kurallar üretmek isteriz. Bazı uygulamalarda, bir değişkenin değerini ötekilerin değerlerini kullanarak öngörmeyi amaçlarız. Öngörülecek değişken ayrıksa buna çok değişkenli sınıflandırma, sayısalsa çok değişkenli bağlanım denir.

5.2 Parametre Kestirimi

μ ile gösterdiğimiz *ortalama yöneyinin* her boyutu, $\mathbf{X}$ veri dizeyinin karşılık gelen sütununun ortalamasıdır:

$$E[\boldsymbol{x}] = \boldsymbol{\mu} = [\mu_1, \ldots, \mu_d]^T \tag{5.1}$$

X_i boyutunun değişintisi, σ_i^2 ile gösterilir. X_i ve X_j boyutlarının eşdeğişintisi, ortalamalarına göre bu değişkenlerin nasıl değer aldıklarını ölçer:

$$\sigma_{ij} \equiv \text{Cov}(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)] = E[X_i X_j] - \mu_i \mu_j \tag{5.2}$$

$\sigma_{ij} = \sigma_{ji}$, ve $i = j$ ise $\sigma_{ii} = \sigma_i^2$. Eğer d değişkenimiz varsa, d tane değişinti ve $d(d-1)/2$ tane eşdeğişinti değeri vardır ve bunlar $d \times d$ büyüklüğünde, $\boldsymbol{\Sigma}$ adında ve (i, j). öğesi σ_{ij} olan bir *eşdeğişinti dizeyi* olarak gösterilir:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2d} \\ \vdots & & & \\ \sigma_{d1} & \sigma_{d2} & \cdots & \sigma_d^2 \end{bmatrix}$$

Köşegen üzerindeki terimler değişintiler, köşegen dışı terimler eşdeğişintilerdir. Bu dizey bakışımlıdır ($\sigma_{ij} = \sigma_{ji}$). Yöney-dizey gösterimi kullanalım:

$$\boldsymbol{\Sigma} \equiv \text{Cov}(\boldsymbol{X}) = E[(\boldsymbol{X} - \boldsymbol{\mu})(\boldsymbol{X} - \boldsymbol{\mu})^T] = E[\boldsymbol{X}\boldsymbol{X}^T] - \boldsymbol{\mu}\boldsymbol{\mu}^T \tag{5.3}$$

İki değişken doğrusal olarak ilişkiliyse, ilişkinin artı ya da eksi eğime sahip olmasına göre eşdeğişinti artı ya da eksi olacaktır. Ancak ilişkinin büyüklüğü, iki değişkenin ölçüldüğü birimlere bağlıdır ve yorumlanması daha zordur. X_i ve X_j arasındaki *ilinti*, -1 ve $+1$ arasında kalacak biçimde normalleştirilmiş bir istatistiktir:

$$\text{Corr}(X_i, X_j) \equiv \rho_{ij} = \frac{\sigma_{ij}}{\sigma_i \sigma_j} \tag{5.4}$$

İki değişken bağımsızsa, eşdeğişintileri ve dolayısıyla ilintileri sıfırdır. Ama karşıtı doğru değildir: değişkenler (doğrusal olmayan biçimde) bağımlıyken ilintileri 0 olabilir.

Çok değişkenli bir örneklem kullanarak bu parametreler kestirilebilir: *Örneklem ortalaması* için en büyük olabilirlik kestirimi m ile gösterilir ve i. boyutu $\mathbf{X}$ dizeyinin i. sütununun ortalamasıdır:

$$m = \frac{\sum_{t=1}^N x^t}{N} \text{ ve } m_i = \frac{\sum_{t=1}^N x_i^t}{N}, i = 1, \ldots, d \tag{5.5}$$

$\boldsymbol{\Sigma}$ için kestirici olan $\mathbf{S}$ *örneklem eşdeğişinti dizeyinin* öğeleri, örneklem değişinti ve eşdeğişinti değerlerinden oluşur:

$$s_i^2 = \frac{\sum_{t=1}^{N}(x_i^t - m_i)^2}{N} \tag{5.6}$$

$$s_{ij} = \frac{\sum_{t=1}^{N}(x_i^t - m_i)(x_j^t - m_j)}{N} \tag{5.7}$$

Bunlar yanlı kestirimlerdir, ama bir uygulamada N ya da $N-1$'e böldüğümüzde belirgin olarak farklı kestirimler elde ediyorsak zaten zor bir durumla karşı karşıyayız demektir.

Örneklem ilinti katsayıları

$$r_{ij} = \frac{s_{ij}}{s_i s_j} \tag{5.8}$$

olarak tanımlıdır; örneklem ilinti dizeyi $\mathbf{R}$ bu r_{ij} değerlerinden oluşur.

5.3 Eksik Değerlerin Kestirilmesi

Gözlemlerde bazı değişkenlerin değerleri eksik olabilir. En iyi yaklaşım bu gözlemleri tamamen atmaktır; genelde yeteri kadar büyük verimiz olmadığı için bunu yapmak istemeyiz çünkü eksik olmayan (değeri bilinen) girdiler bilgi taşırlar. Eksik değerleri veriden kestirip doldururuz. Buna *yükleme* denir.

Ortalama yüklemede, sayısal bir değişken için o değişken için örneklemde var olan verilerin ortalamasını kullanırız. Ayrık bir değişken için en olası değeri, yani veride en çok görülen değeri kullanırız.

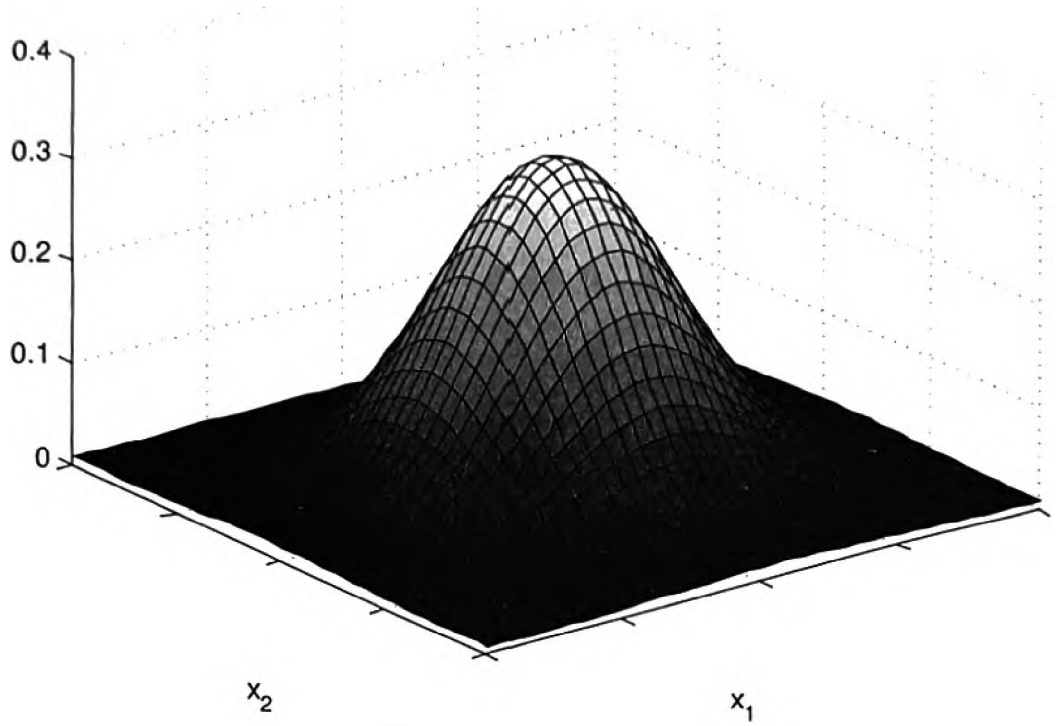
Bağlanımlı yüklemede, eksik bir değişkenin değerini o örnek için değeri bilinen öteki değişkenleri kullanarak kestiririz. Eksik değişkenin türüne göre bu, değerlerin bilindiği veri noktalarını kullanarak eğiteceğimiz ayrı bir bağlanım ya da sınıflandırma problemi tanımlar. Farklı birçok değişken eksikse, ortalamaları başlangıç kestirimleri olarak alır ve yükleme yordamını, değişkenler üzerinden sırayla, öngörülen değerler yakınsayana dek yineleriz. Değişkenler çok ilintili değilse, bağlanım yaklaşımıyla ortalama yükleme eşdeğerdir.

Ancak uygulamaya bağlı olarak bazen bir değişkenin değerinin eksik olması önemli olabilir. Örneğin, bir kredi kartı başvurusunda, başvuru sahibi telefon numarasını beyan etmemişse bu önemli bir bilgi olabilir. Bu gibi durumlarda, değişkenin eksik olduğunu belirten ayrı bir değer belirlenir ve o değer girdi olarak kullanılır.

5.4 Çok Değişkenli Normal Dağılım

Girdi yöneyi $\boldsymbol{x}$, d boyutlu ve normal dağıldığında, $\boldsymbol{x} \sim \mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, yeterli istatistikler ortalama yöneyi $\boldsymbol{\mu}$ ve eşdeğişinti dizeyi $\boldsymbol{\Sigma}$ olur (bkz. şekil 5.1):

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right] \tag{5.9}$$



Şekil 5.1: İki boyutlu normal dağılım.

Tek boyutta, x ve μ arasında standart sapmaya göre normalleştirilmiş uzaklık

$$\frac{(x-\mu)^2}{\sigma^2} = (x-\mu)(\sigma^2)^{-1}(x-\mu)$$

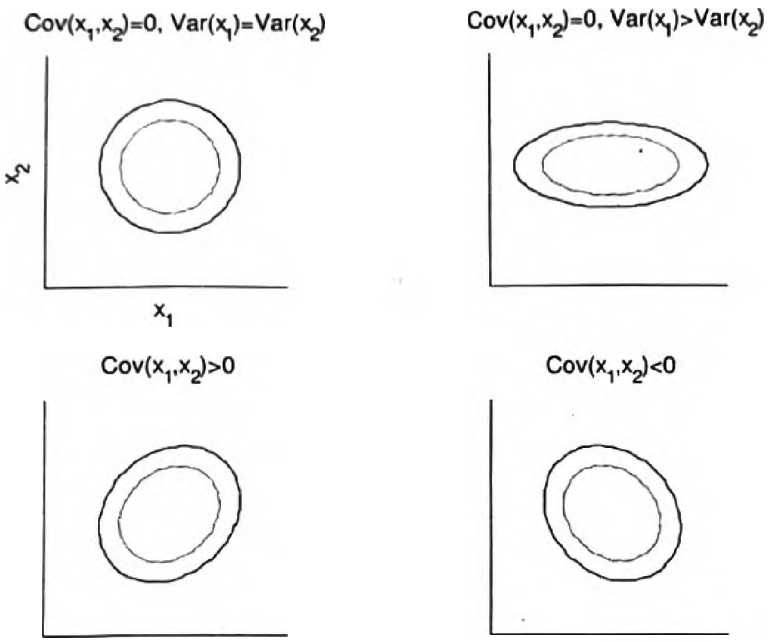
çok değişkenli durumda *Mahalanobis uzaklığı* olarak genelleştirilir:

$$(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) \tag{5.10}$$

$(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) = c^2$, $\boldsymbol{\mu}$ merkezli, biçimi ve yönelimi $\boldsymbol{\Sigma}$ tarafından belirlenen d boyutlu üstünelipstir. $\boldsymbol{\Sigma}$ dizeyinin tersinin kullanılmasından dolayı, eğer bir değişkenin değişintisi ötekine göre daha büyükse Mahalanobis uzaklığında ağırlığı daha düşüktür. Benzer biçimde, yüksek ilintili iki değişkenin katkısı az ilintili iki değişken kadar değildir. Eşdeğişinti dizeyinin tersi kullanılarak böylelikle tüm değişkenler birim değişintiye getirilir ve aralarındaki ilintiler giderilir.

$d = 2$ alıp iki değişkenli durumu görselleştirelim (bkz. şekil 5.2). Değişkenler bağımsızken dağılımın ana eksenleri girdi eksenlerine koşuttur. Değişintiler farklıysa dağılım bir elips biçimini alır ve bu elipsin ana ekseni, eşdeğişinti ya da ilintinin işaretine göre döner. $\boldsymbol{\mu}^T = [\mu_1, \mu_2]$ ortalama yöneyi iki boyutludur ve eşdeğişinti dizeyi, 2×2 bir dizeydir:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$$



Şekil 5.2: İki değişkenli normal dağılımın eşolasılık çevrit çizimi. Ortalama, merkez noktasını, eşdeğişinti dizeyi de biçimi ve yönelimini belirler.

Birleşik iki değişkenli dağılım, bu parametrelerle gösterilir (alıştırma 1):

$$p(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[-\frac{1}{2(1-\rho^2)}\left(z_1^2 - 2\rho z_1 z_2 + z_2^2\right)\right] \tag{5.11}$$

Burada, $z_i = (x_i - \mu_i)/\sigma_i, i = 1, 2$, standartlaştırılmış değişkenlerdir; bu işleme z *normalleştirme* denir. $|\rho| < 1$ iken

$$z_1^2 + 2\rho z_1 z_2 + z_2^2 = \text{sabit}$$

denkleminin bir elips tanımladığını hatırlayalım. $\rho > 0$ ise elipsin ana ekseni artı, $\rho < 0$ ise eksi eğime sahiptir.

Mahalanobis uzaklığının denklem 5.11'de yazılmış açık halinde her değişken birim değişintiye getirilmiştir ve aralarındaki ilintiyi hesaba katan bir çapraz terim vardır.

Dağılımın, iki ortalama, iki değişinti, ve ilinti olmak üzere beş parametresi vardır: Değişintiler sıfırdan farklı ve $|\rho| < 1$ olduğunda, Σ tekil değildir ve dolayısıyla kesin artıdır. ρ, +1 ya da -1 ise, iki değişken arasında doğrusal bir ilişki vardır ve birisi atılabilir. $\rho = 0$ ise, iki değişken bağımsızdır, çapraz terim kaybolur, ve iki tane tekdeğişkenli dağılımın çarpımını elde ederiz.

Çok değişkenli durumda, $|\Sigma|$ küçük değerler alırsa bu örneklerin μ merkezine yakın olduğunu belirtir, tıpkı tek değişkenli durumda küçük bir σ^2 değerinin örneklerin μ merkezine yakın olduğunu belirtmesi gibi. Küçük bir $|\Sigma|$ değeri, değişkenler arasında yüksek ilinti olduğunu gösteriyor da olabilir. Σ, bakışımlı ve kesin artı bir dizeydir; bu, $\text{Var}(X) > 0$ demenin çok değişkencesidir. Değilse, Σ tekildir ve determinantı sıfırdır. Bu, ya boyutlar arasındaki doğrusal bir ilişkiden ya da bir boyuttaki

değişintinin 0 olmasından kaynaklanır. Böyle bir durumda, kesin artı bir dizey elde etmek için boyut azaltılmalıdır; bunun nasıl yapılacağını 6. bölümde göreceğiz.

$\boldsymbol{x} \sim \mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ ise çok değişkenli $\boldsymbol{x}$ yöneyinin her boyutu tek değişkenli bir normaldir. (Karşıtı doğru değildir: Her X_i tekdeğişkenli normalken $\boldsymbol{X}$ çokdeğişkenli normal olmayabilir.) Aslında, değişkenlerin her $k < d$ öğeli alt kümesi k değişkenli normaldir.

Özel, *saf* durumda $\boldsymbol{x}$ yöneyinin boyutları bağımsız ve ilintisizdir, $\text{Cov}(X_i, X_j) = 0$, $i \neq j$, ve değişintileri aynıdır: $\text{Var}(X_i) = \sigma_i^2, \forall i$. Bu durumda, eşdeğişinti dizeyi köşegendir ve birleşik dağılım tek değişkenli dağılımların çarpımına indirgenir:

$$p(\boldsymbol{x}) = \prod_{i=1}^{d} p_i(x_i) = \frac{1}{(2\pi)^{d/2} \prod_{i=1}^{d} \sigma_i} \exp\left[-\frac{1}{2}\sum_{i=1}^{d}\left(\frac{x_i - \mu_i}{\sigma_i}\right)^2\right] \tag{5.12}$$

İleriki bölümlerde kullanacağımız başka bir özelliği de görelim. $\boldsymbol{x} \sim \mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ ve $\boldsymbol{w} \in \Re^d$ ise

$$\boldsymbol{w}^T\boldsymbol{x} = w_1x_1 + w_2x_2 + \cdots + w_dx_d \sim \mathcal{N}(\boldsymbol{w}^T\boldsymbol{\mu}, \boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w})$$

çünkü

$$\begin{aligned} E[\boldsymbol{w}^T\boldsymbol{x}] &= \boldsymbol{w}^T E[\boldsymbol{x}] = \boldsymbol{w}^T\boldsymbol{\mu} && (5.13)\\ \text{Var}(\boldsymbol{w}^T\boldsymbol{x}) &= E[(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})^2] = E[(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})] \\ &= E[\boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{w}] = \boldsymbol{w}^T E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T]\boldsymbol{w} \\ &= \boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w} && (5.14) \end{aligned}$$

Yani, d boyutlu normal dağılımın $\boldsymbol{w}$ yöneyi üzerine izdüşümü tek boyutlu normal bir dağılım tanımlar. Genel durumda, $\mathbf{W}$, kertesi $k < d$ olan $d \times k$ bir dizeyse, k boyutlu $\mathbf{W}^T\boldsymbol{x}$ k değişkenli bir normal dağılım oluşturur:

$$\mathbf{W}^T\boldsymbol{x} \sim \mathcal{N}_k(\mathbf{W}^T\boldsymbol{\mu}, \mathbf{W}^T\boldsymbol{\Sigma}\mathbf{W}) \tag{5.15}$$

Yani, d boyutlu normal dağılımın k boyutlu uzaya izdüşümü, k boyutlu bir normal dağılım tanımlar.

5.5 Çok Değişkenli Sınıflandırma

$\boldsymbol{x} \in \Re^d$ ise sınıf koşullu dağılımlar, $p(\boldsymbol{x}|C_i)$, çok değişkenli normal, $\mathcal{N}_d(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, alınabilir:

$$p(\boldsymbol{x}|C_i) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}_i|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i)\right] \tag{5.16}$$

Bunu yapmanın ana nedeni, çözümsel basitliktir (Duda, Hart, ve Stork 2001). Bunun yanında normal dağılım, doğal olarak gerçekleşen birçok olgu için iyi bir

modeldir, çünkü böyle olgularda bir sınıfın örneklerinin o sınıfın tek bir μ_i asıl örneğinin hafifçe değiştirilmiş (gürültülü) kopyaları olduğu ve $\mathbf{\Sigma}_i$ eşdeğişinti dizeyinin değişkenlerdeki gürültüyü ve bu gürültü kaynakları arasındaki ilintileri gösterdiği düşünülebilir. Gerçek veri tam olarak normal dağılmasa da bu, yine de faydalı bir yaklaşıklamadır. Matematiksel çözülebilirliğine ek olarak, birçok çalışmada ortaya konduğu gibi (örneğin, McLahlan 1992), bu varsayım normallikten kopmaya karşı gürbüzdür. Ancak, sınıf örneklerinin tek bir öbek oluşturması açık bir koşuldur, yoksa karışım modelleri (bölüm 7) kullanılmalıdır.

Bir müşterinin ilgilenebileceği araba türünü öngörmek istiyor olalım. Farklı türde arabalar sınıfları, $\boldsymbol{x}$ yöneyi de müşterilerin yaş ve gelirleri gibi gözlemlenebilir ve bilgi veren verileri içerir. μ_i, i. tür araba alan müşterilerin yaş ve gelirlerinin ortalama yöneyi, $\mathbf{\Sigma}_i$ ise onların eşdeğişinti dizeyini (σ_{i1}^2 ve σ_{i2}^2, yaş ve gelirlerin değişintilerini, σ_{i12} ise yaş ve gelirleri arasındaki eşdeğişintiyi) gösterir.

Ayırtaç işlevini

$$g_i(\boldsymbol{x}) = \log p(\boldsymbol{x}|C_i) + \log P(C_i)$$

olarak tanımladığımızda, $p(\boldsymbol{x}|C_i) \sim \mathcal{N}_d(\mu_i, \mathbf{\Sigma}_i)$ olduğunu varsayıp ayırtaç işlevini yazalım:

$$g_i(\boldsymbol{x}) = -\frac{d}{2}\log 2\pi - \frac{1}{2}\log|\mathbf{\Sigma}_i| - \frac{1}{2}(\boldsymbol{x}-\mu_i)^T\mathbf{\Sigma}_i^{-1}(\boldsymbol{x}-\mu_i) + \log P(C_i) \tag{5.17}$$

$K \geq 2$ sınıf için $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}$ örneklemi verilmiş olsun, öyle ki, $\boldsymbol{x}^t \in C_i$ ise $r_i^t = 1$, değilse $r_i^t = 0$. Ortalama ve eşdeğişinti kestirimleri, en büyük olabilirlik kestiricisi kullanılarak her sınıf için ayrı ayrı yapılır:

$$\begin{aligned}
\hat{P}(C_i) &= \frac{\sum_t r_i^t}{N} \\
\boldsymbol{m}_i &= \frac{\sum_t r_i^t \boldsymbol{x}^t}{\sum_t r_i^t} \\
\mathbf{S}_i &= \frac{\sum_t r_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T}{\sum_t r_i^t}
\end{aligned} \tag{5.18}$$

Ardından bunlar ayırtaç işlevinde yerlerine konulur. Sabit olan ilk terimi göz ardı edebiliriz:

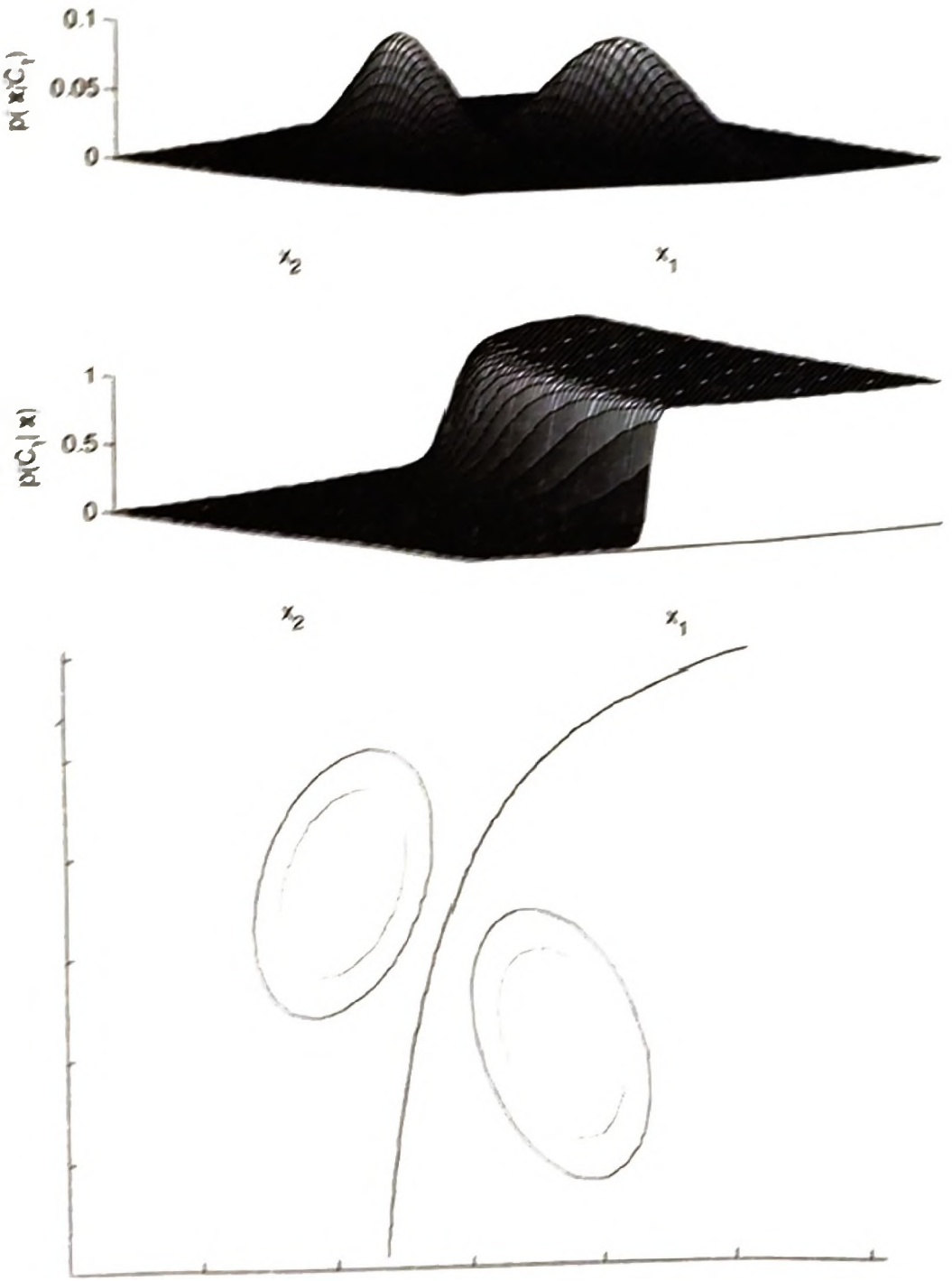
$$g_i(\boldsymbol{x}) = -\frac{1}{2}\log|\mathbf{S}_i| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_i)^T\mathbf{S}_i^{-1}(\boldsymbol{x}-\boldsymbol{m}_i) + \log\hat{P}(C_i) \tag{5.19}$$

Bunu açalım:

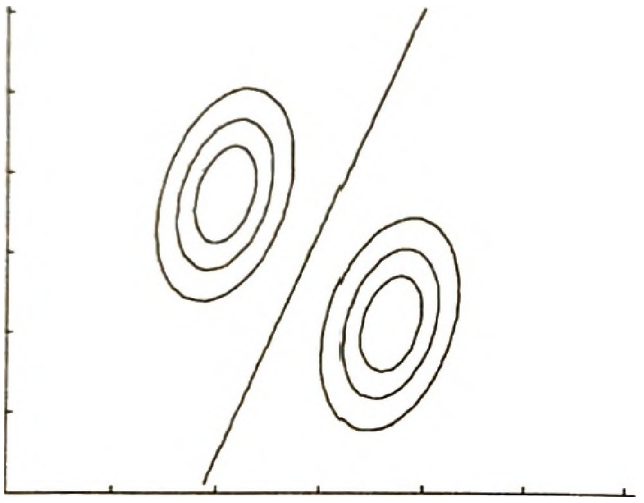
$$g_i(\boldsymbol{x}) = -\frac{1}{2}\log|\mathbf{S}_i| - \frac{1}{2}\left(\boldsymbol{x}^T\mathbf{S}_i^{-1}\boldsymbol{x} - 2\boldsymbol{x}^T\mathbf{S}_i^{-1}\boldsymbol{m}_i + \boldsymbol{m}_i^T\mathbf{S}_i^{-1}\boldsymbol{m}_i\right) + \log\hat{P}(C_i)$$

Çıkan *karesel ayırtaç* (bkz. şekil 5.3)

$$g_i(\boldsymbol{x}) = \boldsymbol{x}^T\mathbf{W}_i\boldsymbol{x} + \boldsymbol{w}_i^T\boldsymbol{x} + w_{i0} \tag{5.20}$$



Şekil 5.3: Sınıfların farklı eşdeğişinti dizeyleri olduğu duruma bir örnek gösterilmiştir. En üstte olabilirlik ve ortada sonsal dağılımlar çizilmiştir. En altta sınıf dağılımları eşolasılık çevritleriyle gösterilmiş ve sınıfları ayıran ayırtaç çizilmiştir.



Şekil 5.4: Eşdeğişinti dizeylerinin iki sınıf için ortak olduğu bir örnek.

olarak da yazılabilir, öyle ki

$$\begin{aligned}
\mathbf{W}_i &= -\frac{1}{2}\mathbf{S}_i^{-1} \\
\boldsymbol{w}_i &= \mathbf{S}_i^{-1}\boldsymbol{m}_i \\
w_{i0} &= -\frac{1}{2}\boldsymbol{m}_i^T\mathbf{S}_i^{-1}\boldsymbol{m}_i - \frac{1}{2}\log|\mathbf{S}_i| + \log\hat{P}(\mathcal{C}_i)
\end{aligned}$$

Kestirilecek parametreler, ortalamalar için $K \cdot d$ ve eşdeğişinti dizeyleri için $K \cdot d(d+1)/2$ tanedir. Girdi boyut sayısı d büyük ve örneklemler küçük olduğunda, $\mathbf{S}_i$ dizeyleri tekil olabilir ve tersleri alınamayabilir. Ya da, $|\mathbf{S}_i|$ değerleri sıfırdan farklı ama çok küçük olabilir ki bu durumda tersi kararsız olacak, $\mathbf{S}_i$ dizeyindeki küçük değişiklikler $\mathbf{S}_i^{-1}$ dizeyinde büyük değişikliklere yol açabilecektir. Kestirimlerin küçük örneklemlerde güvenilir olabilmesi için öznitelik çıkartıcıyı yeniden tasarlayarak d boyutu azaltıp özniteliklerin bir altkümesini seçmek ya da var olan öznitelikleri bir biçimde birleştirmek isteriz; böylesi yöntemleri 6. bölümde ele alacağız.

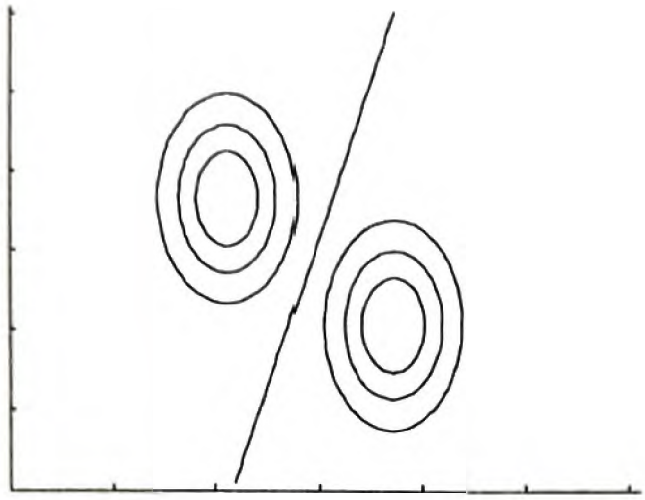
Başka bir olasılık ise veriyi birleştirerek tüm sınıflar için ortak bir eşdeğişinti dizeyi kullanmaktır:

$$\mathbf{S} = \sum_i \hat{P}(\mathcal{C}_i)\mathbf{S}_i \tag{5.21}$$

Ortak bir eşdeğişinti dizeyi kullanılan bu durumda denklem 5.19 sadeleşir:

$$g_i(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_i)^T\mathbf{S}^{-1}(\boldsymbol{x}-\boldsymbol{m}_i) + \log\hat{P}(\mathcal{C}_i) \tag{5.22}$$

Parametre sayısı, ortalamalar için $K \cdot d$ ve ortak eşdeğişinti dizeyi için $d(d+1)/2$ olur. Önseller eşitse, en iyi karar kuralı, girdiyi ortalamasına Mahalanobis uzaklığı en küçük olan sınıfa atamaktır. Daha önce gördüğümüz gibi, eşit olmayan önseller, sınırı önsel olasılığı daha az olan sınıfa doğru kaydırır. Eşdeğişinti dizeyleri ortak



Şekil 5.5: Eşdeğişinti dizeylerinin köşegen ve iki sınıf için ortak olduğu bir örnek.

olduğunda, $\boldsymbol{x}^T\mathbf{S}^{-1}\boldsymbol{x}$ karesel terimi tüm ayırtaçlarda ortak olduğu için atılır ve karar sınırı doğrusal olur. Çıkan *doğrusal ayırtaç* (bkz. şekil 5.4)

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T\boldsymbol{x} + w_{i0} \tag{5.23}$$

olarak yazılabilir, öyle ki

$$\begin{aligned} \boldsymbol{w}_i &= \mathbf{S}^{-1}\boldsymbol{m}_i \\ w_{i0} &= -\frac{1}{2}\boldsymbol{m}_i^T\mathbf{S}^{-1}\boldsymbol{m}_i + \log \hat{P}(C_i) \end{aligned}$$

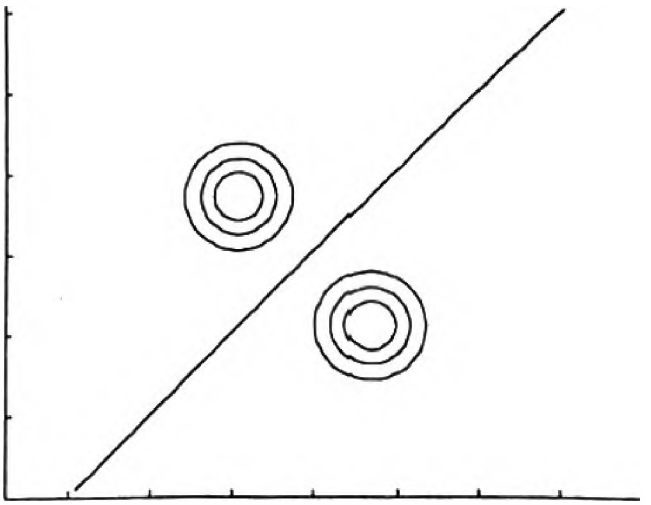
Böylesi doğrusal bir sınıflandırıcının karar bölgeleri dışbükeydir; yani, aynı karar bölgesinden herhangi iki nokta seçilip bir doğruyla birleştirildiğinde, doğru üzerindeki tüm noktalar bu alanda kalır.

Eşdeğişinti dizeyinin köşegen dışı terimlerini 0 alarak, yani değişkenleri bağımsız kabul ederek ayırtaç daha da basitleştirilebilir. Bu, her $p(x_j|C_i)$ dağılımının tek boyutlu bağımsız normal dağıldığı *saf Bayes sınıflandırıcısıdır*. $\mathbf{S}$ ve tersi köşegen olduğu için ayırtaç basitleşir:

$$g_i(\boldsymbol{x}) = -\frac{1}{2}\sum_{j=1}^{d}\left(\frac{x_j^t - m_{ij}}{s_j}\right)^2 + \log \hat{P}(C_i) \tag{5.24}$$

$(x_j^t - m_{ij})/s_j$ terimi her boyutu kendi ölçeğine göre normalleştirir ve uzaklığı standart sapma biriminden hesaplar. Geometrik olarak söyleyecek olursak eşdeğişintiler 0 olduğu için sınıflar eksen hizalı üstünelipslerdir (bkz. şekil 5.5). Parametre sayısı ortalamalar için $K \cdot d$ ve değişintiler için d, $\mathbf{S}$ dizeyinin karmaşıklığı $\mathcal{O}(d^2)$ yerine $\mathcal{O}(d)$ olur.

Daha da fazla basitleştirip tüm değişintileri eşit kabul edersek, Mahalanobis uzaklığı yerine *Euclid uzaklığı* kullanılır. Geometrik olarak dağılım, ortalama yöneyi



Şekil 5.6: Eşdeğişinti dizeylerinin köşegen, iki sınıf için ortak ve değişintilerin aynı olduğu bir örnek.

$\boldsymbol{m}_i$ çevresinde küreseldir (bkz. şekil 5.6). Bu durumda, $|\mathbf{S}| = s^{2d}$ ve $\mathbf{S}^{-1} = (1/s^2)\mathbf{I}$ olur ve parametre sayısı ortalamalar için $K \cdot d$ ve s^2 için birdir:

$$g_i(\boldsymbol{x}) = -\frac{\|\boldsymbol{x} - \boldsymbol{m}_i\|^2}{2s^2} + \log \hat{P}(C_i) = -\frac{1}{2s^2}\sum_{j=1}^{d}(x_j^t - m_{ij})^2 + \log \hat{P}(C_i) \qquad (5.25)$$

Önceller eşitse ayırtaç, $g_i(\boldsymbol{x}) = -\|\boldsymbol{x} - \boldsymbol{m}_i\|^2$ olarak yazılabilir. Buna, *en yakın ortalama sınıflandırıcısı* denir çünkü girdiyi ortalaması en yakın sınıfa atar. Her sınıfın ortalamasını, asıl örnek ya da şablon olarak düşünürsek bu bir *şablon eşleme* yordamıdır. Euclid uzaklığını açalım:

$$\begin{aligned} g_i(\boldsymbol{x}) &= -\|\boldsymbol{x} - \boldsymbol{m}_i\|^2 = -(\boldsymbol{x} - \boldsymbol{m}_i)^T(\boldsymbol{x} - \boldsymbol{m}_i) \\ &= -(\boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{m}_i^T\boldsymbol{x} + \boldsymbol{m}_i^T\boldsymbol{m}_i) \end{aligned} \qquad (5.26)$$

ilk terim, $\boldsymbol{x}^T\boldsymbol{x}$, tüm $g_i(\boldsymbol{x})$ ayırtaçlarında ortak olduğu için göz ardı edilebilir ve ayırtaç

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T\boldsymbol{x} + w_{i0} \qquad (5.27)$$

olarak yazılabilir; burada $\boldsymbol{w}_i = \boldsymbol{m}_i$ ve $w_{i0} = -(1/2)\|\boldsymbol{m}_i\|^2$ olarak tanımlıdır. Eğer tüm $\boldsymbol{m}_i$ yöneyleri yaklaşık aynı büyüklüğe sahipse bu terim de göz ardı edilebilir ve

$$g_i(\boldsymbol{x}) = \boldsymbol{m}_i^T\boldsymbol{x} \qquad (5.28)$$

olarak kullanılabilir. Yani, $\boldsymbol{m}_i$ yöneylerinin büyüklükleri benzer olduğunda, benzerlik ölçütü olarak (eksi) Euclid uzaklığı yerine iç çarpım da kullanılabilir.

Bir başka bakış açısından en iyi ayırtaç işlevini bulmayı en iyi uzaklık işlevini bulma olarak da düşünebiliriz. $g_i(\boldsymbol{x})$ ayırtaç işlevlerini öğrenmek yerine uygun bir

Tablo 5.1: Basitleştirici varsayımlarla değişintinin azaltılması.

Durum	Varsayım	Eşdeğişinti dizeyi	Parametre sayısı
1	Ortak, Üstünküre	$\mathbf{S}_i = \mathbf{S} = s^2\mathbf{I}$	1
2	Ortak, Eksen hizalı	$\mathbf{S}_i = \mathbf{S}$, ve $s_{ij} = 0$	d
3	Ortak, Üstünelips	$\mathbf{S}_i = \mathbf{S}$	$d(d+1)/2$
4	Farklı, Üstünelips	$\mathbf{S}_i$	$K \cdot (d(d+1)/2)$

$\mathcal{D}(\boldsymbol{x}_1, \boldsymbol{x}_2)$ uzaklık işlevi öğrenmek isteriz, öyle ki, $\boldsymbol{x}_1$ ve $\boldsymbol{x}_2$ aynı sınıfta, $\boldsymbol{x}_3$ bunlardan başka bir sınıfta olduğu her durum için

$$\mathcal{D}(\boldsymbol{x}_1, \boldsymbol{x}_2) < \mathcal{D}(\boldsymbol{x}_1, \boldsymbol{x}_3)$$

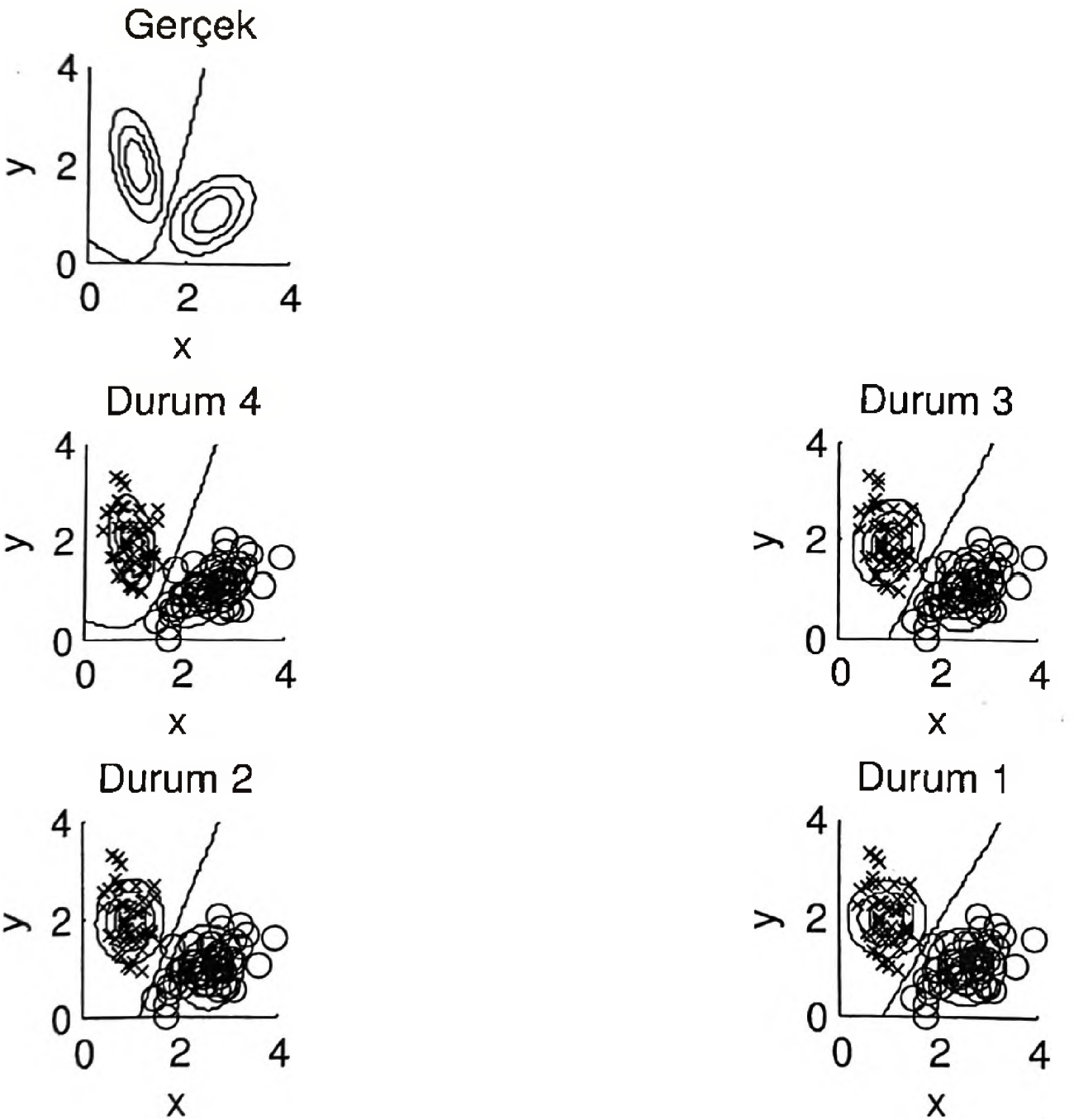
koşulu sağlansın.

5.6 Karmaşıklığın Ayarlanması

Tablo 5.1'de basit bir model kullanma rahatlığının modelin genelliğiyle ödünleşmesi sonucu eşdeğişinti dizeyinin parametre sayısının nasıl azaltılabildiğini görüyoruz. Bu yanlılık/değişinti ikileminin bir başka örneğidir. Eşdeğişinti dizeyi hakkında basitleştirici varsayımlar yaptığımız ve kestirilecek parametre sayısını azalttığımızda, yanlılık riskini artırmış oluruz (bkz. şekil 5.7). Öte yandan, böyle varsayımlar yapılmaz ve dizeylerde kısıt olmazsa, karesel ayırtacın değişintisi küçük verilerde yüksek olabilir. En iyi durum, eldeki verinin betimlediği problemin karmaşıklığına ve sahip olduğumuz verinin miktarına bağlıdır. Küçük bir veri kümemiz olduğunda, eşdeğişinti dizeyleri farklı bile olsa, ortak bir esdeğişinti dizeyi varsaymak daha iyi olabilir; tek bir eşdeğişinti dizeyinin daha az sayıda parametresi vardır ve tüm sınıfların örneklerini birleştirerek daha çok veriyle kestirilebilir. Bu, sınıflandırmada sıkça kullanılan ve 10. bölümde daha ayrıntılı işleyeceğimiz *doğrusal ayırtaç* kullanmaya karşılık gelir.

Benzerlikleri ölçmek için Euclid uzaklığını kullanmak, tüm değişkenlerin aynı değişintili ve birbirlerinden bağımsız olduğunu varsaymak demektir. Çoğu durum için bu varsayımlar geçerli değildir; örneğin, yaş ve yıllık gelirin birimleri değişiktir ve birçok uygulamada bağımlıdırlar. Böyle bir durumda, bir önişleme basamağı olarak, (sıfır ortalama ve birim değişinti elde etmek için) her girdiye ayrı ayrı z normalleştirmesi uygulandıktan sonra Euclid uzaklığı kullanılabilir. Öte yandan, bazen değişkenler bağımlı da olsa, bu bağımlılıkları doğru hesaplayabilecek kadar verimiz yoksa, değişkenlerin bağımsız olduklarını kabul ederek saf Bayes sınıflandırıcısı kullanmak daha iyi olabilir.

Friedman (1989) tüm bunları özel durumlar olarak birleştiren, *düzenlileştirilmiş ayırtaç çözümlemesi* yöntemini önermiştir. Düzenlileştirmenin, yüksek değişintiyle başlayıp, yanlılığı artırmak pahasına, çözümü düşük değişintiye götürecek biçimde



Şekil 5.7: Eşdeğişinti dizeyi için farklı varsayımlar aynı veride farklı sınırlara neden olur.

kısıtlamalar yapmak olduğunu hatırlayalım. Normal dağılımlarla sınıflandırmada, eşdeğişinti dizeyleri üç özel durumun ağırlıklı toplamı olarak yazılabilir:

$$\mathbf{S}_i' = \alpha\sigma^2\mathbf{I} + \beta\mathbf{S} + (1-\alpha-\beta)\mathbf{S}_i \tag{5.29}$$

Eğer $\alpha = \beta = 0$ ise karesel bir sınıflandırıcı elde ederiz. $\alpha = 0$ ve $\beta = 1$ olduğunda eşdeğişinti dizeyleri ortak olduğundan sınıflandırıcı doğrusal olur. $\alpha = 1$ ve $\beta = 0$ olduğundaysa eşdeğişinti dizeyleri, köşegeninde σ^2 bulunan köşegen dizeyler olur ve en yakın ortalama sınıflandırıcısını elde ederiz. Bu uç noktalar arasında, α ve β değerlerini çapraz geçerlemeyle seçip çok farklı sınıflandırıcılar elde edebiliriz.

Veri kümesi küçük olduğunda, düzenlileştirme için başka yaklaşımlar arasında $\boldsymbol{\mu}_i$ ve $\mathbf{S}_i$ için önseller tanımlayarak Bayesçi bir yaklaşım kullanmak ya da tablo 5.1'de verilen dört durumdan en iyisini seçmek için çapraz geçerleme kullanmak da sayılabilir.

5.7 Kesikli Öznitelikler

Bazı uygulamalarda, n farklı değerden birini alabilen kesikli öznitelikler kullanılır. Örneğin, bir öznitelik renk$\in$ {kırmızı, mavi, yeşil, siyah} ya da imge-noktası$\in$ {sıfır, bir} olabilir. x_j değeri ikili (Bernoulli) olduğunda ve j. özniteliğin 1 değerini alma olasılığı

$$p_{ij} \equiv p(x_j = 1|C_i)$$

parametresiyle gösterelim. Eğer farklı x_j değerleri bağımsızsa, tüm yöneyin olasılığı özniteliklerin birer birer olasılıklarının çarpımıdır:

$$p(\boldsymbol{x}|C_i) = \prod_{j=1}^{d} p_{ij}^{x_j}(1-p_{ij})^{(1-x_j)}$$

Bu da girdilerin kesikli olduğu bir başka saf Bayes sınıflandırıcısıdır. Bu olasılık üzerinden Bayes kuralıyla yazılan ayırtaç işlevi doğrusaldır:

$$\begin{aligned} g_i(\boldsymbol{x}) &= \log p(\boldsymbol{x}|C_i) + \log P(C_i) \\ &= \sum_j [x_j \log p_{ij} + (1-x_j)\log(1-p_{ij})] + \log P(C_i) \end{aligned} \tag{5.30}$$

Elimizde bir örneklem olduğunda p_{ij} için en büyük olabilirlik kestiricisini hesaplayabiliriz:

$$\hat{p}_{ij} = \frac{\sum_t x_j^t r_i^t}{\sum_t r_i^t} \tag{5.31}$$

Bu yaklaşım, *belge sınıflandırmada* kullanılır. Örneğin, gazete haberlerini, politika, spor, moda gibi sınıflara ayırmak isteyelim. *Sözcük torbası* gösteriminde önce sınıf hakkında bize bilgi verdiğine inandığımız d sözcüğü seçeriz (Manning ve Schütze

1999). Örneğin, haber sınıflandırmada, "meclis," "futbol," "kumaş" gibi sözcükler farklı sınıfları ayırt etmekte işe yarar ama "galip," ya da "saha" gibi sözcükler muğlaktır. Bu gösterimde her haber d boyutlu ikili bir yöneyle gösterilir ve j. sözcük haberde geçiyorsa $x_j = 1$, geçmiyorsa $x_j = 0$ değerini alır. Bu gösterim sözcüklerin sıra bilgisini kaybeder ve bu yüzden sözcük *torbası* adını almıştır.

Eğitimden sonra, $\hat{p}_{ij}$, j. sözcüğün i. sınıftaki bir haberde geçme olasılığını kestirir. Farklı sınıflarda benzer olasılıklara sahip sözcükler fazla bilgi içermezler; kullanışlı olabilmeleri için olasılığın bir (ya da az sayıda) sınıf için yüksekken ötekiler için düşük olmasını gerekir; bu tür *öznitelik seçimi* yöntemlerini 6. bölümde göreceğiz. Belge sınıflandırmaya başka bir örnek, *elektronik postaların süzülmesidir*; girdi, elektronik posta, çıktıysa istenmeyen ve geçerli iletileri gösteren iki tür sınıftır. Biyoinformatik uygulamalarında da girdiler genelde baz çiftleri ya da aminoasitler gibi ayrık öğelerin dizileridir.

Genel durumda öznitelik, iki değer yerine $\{v_1, v_2, \ldots, v_{n_j}\}$ kümesinden herhangi birini değer olarak alabilen katlıterim x_j olur. Yeni yapay 0/1 değişkenler tanımlayalım:

$$z_{jk}^t = \begin{cases} 1 & \text{eğer } x_j^t = v_k \text{ ise} \\ 0 & \text{değilse} \end{cases}$$

C_i sınıfından x_j özniteliğinin v_k değerini alma olasılığını p_{ijk} ile gösterelim:

$$p_{ijk} \equiv p(z_{jk} = 1|C_i) = p(x_j = v_k|C_i)$$

Öznitelikler bağımsız olduğunda olasılıklar çarpılır:

$$p(\boldsymbol{x}|C_i) = \prod_{j=1}^{d} \prod_{k=1}^{n_j} p_{ijk}^{z_{jk}} \tag{5.32}$$

Ayırtaç işlevi, yine bu olasılığı ve Bayes kuralını kullanır:

$$g_i(\boldsymbol{x}) = \sum_j \sum_k z_{jk} \log p_{ijk} + \log P(C_i) \tag{5.33}$$

p_{ijk} için en büyük olabilirlik kestiricisi

$$\hat{p}_{ijk} = \frac{\sum_t z_{jk}^t r_i^t}{\sum_t r_i^t} \tag{5.34}$$

hesaplanıp denklem 5.33'teki yerine konduğunda ayırtaç elde edilmiş olur.

5.8 Çok Değişkenli Bağlanım

Çok değişkenli doğrusal bağlanımda, sayısal çıktı r, $x_1, \ldots, x_d$ ile gösterdiğimiz d girdi değişkeninin ağırlıklı toplamı artı gürültü biçiminde yazılır:

$$r^t = g(\boldsymbol{x}^t|w_0, w_1, \ldots, w_d) + \epsilon = w_0 + w_1 x_1^t + w_2 x_2^t + \cdots + w_d x_d^t + \epsilon \tag{5.35}$$

Tek değişkenli bağlanımdaki gibi ϵ gürültü rastsal değişkeninin 0 ortalamalı ve sabit değişintili bir normal dağılımdan geldiği varsayılır. Olabilirliği büyütmek yine toplam kare hatayı küçültmeye eştir:

$$E(w_0, w_1, \ldots, w_d|\mathcal{X}) = \frac{1}{2}\sum_t (r^t - w_0 - w_1 x_1^t - w_2 x_2^t - \cdots - w_d x_d^t)^2 \quad (5.36)$$

$w_j, j = 0, \ldots, d$ parametrelerine göre türev alıp sıfıra eşitleyerek *normal denklemleri* elde ederiz:

$$\begin{aligned}
\sum_t r^t &= Nw_0 + w_1\sum_t x_1^t + w_2\sum_t x_2^t + \cdots + w_d\sum_t x_d^t \quad (5.37)\\
\sum_t x_1^t r^t &= w_0\sum_t x_1^t + w_1\sum_t (x_1^t)^2 + w_2\sum_t x_1^t x_2^t + \cdots + w_d\sum_t x_1^t x_d^t\\
\sum_t x_2^t r^t &= w_0\sum_t x_2^t + w_1\sum_t x_1^t x_2^t + w_2\sum_t (x_2^t)^2 + \cdots + w_d\sum_t x_2^t x_d^t\\
&\vdots\\
\sum_t x_d^t r^t &= w_0\sum_t x_d^t + w_1\sum_t x_d^t x_1^t + w_2\sum_t x_d^t x_2^t + \cdots + w_d\sum_t (x_d^t)^2
\end{aligned}$$

Veri dizeyi, parametre ve istenen çıktı yöneylerini tanımlayalım:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \cdots & x_d^1 \\ 1 & x_1^2 & x_2^2 & \cdots & x_d^2 \\ \vdots & & & & \\ 1 & x_1^N & x_2^N & \cdots & x_d^N \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d, \end{bmatrix}, r = \begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^N \end{bmatrix}$$

Normal denklemler, yöney-dizey gösterimiyle yazalım:

$$\mathbf{X}^T\mathbf{X}w = \mathbf{X}^T r \quad (5.38)$$

Çözüm parametre değerleri de yine benzer biçimde yazılabilir:

$$w = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T r \quad (5.39)$$

Biraz dikkatle bakarsak bu yöntemin (doğrusal olmayan) çokterimli bağlanımla aynı olduğunu görürüz. Değişkenlerimizi $x_1 = x, x_2 = x^2, \ldots, x_k = x^k$ olarak tanımladığımızda çokterimli bağlanımla çok değişkenli doğrusal bağlanım eşit olur. Bu aynı zamanda bize *çok değişkenli çokterimli bağlanımın* nasıl gerçeklenebileceği konusunda ipucu da verir (alıştırma 7), ama çok değişkenli bağlanımda, d çok küçük olmadıkça, doğrusaldan daha yüksek derecelere çıkmak istemeyiz.

Girdinin yüksek dereceden terimlerini kullanmak yanında *taban işlevleri* kullanarak girdinin herhangi bir doğrusal olmayan işlevini de ek girdi olarak tanımlayabiliriz. Örneğin, $x_2 = \sin(x)$, $x_3 = \exp(x^2)$ gibi yeni girdileri, bu dönüşümlerin kullanışlı

olacağına inandığımız uygulamalarda tanımlayabiliriz. Ardından bu genişletilmiş uzayda doğrusal bir model, ilk uzayda doğrusal olmayan bir modele karşılık gelir. Aynı hesaplamalar burada da geçerlidir; yalnızca $\mathbf{X}$ yerine taban işlevlerini kullandıktan sonra elde ettiğimiz veri dizeyini koymamız yeterlidir. İleride farklı kılıklarda (örneğin, çokkatmanlı algılayıcılar, destek yöney makineleri, Gauss süreçleri) göreceğimiz gibi, doğrusal modellerin bu biçimde genellenmesi sıkça kullanılmaktadır.

Doğrusal modellerin bir getirisi de bağlanımdan sonra, $w_j, j = 1, \ldots, d$, değerlerine bakarak bilgi çıkartabilmemizdir: Örneğin, w_j ağırlıklarının işaretlerine bakarak, karşılık gelen x_j girdisinin çıktı üzerindeki etkisinin artı mı eksi mi olduğunu görebiliriz. Ayrıca, tüm x_j girdileri aynı aralıkta değişiyorlarsa, w_j ağırlıklarının mutlak değerlerine bakarak özniteliklerin ne kadar önemli olduklarını görebilir, öznitelikleri önem derecelerine göre sıralayabilir, ve hatta w_j değeri sıfıra yakın olan öznitelikleri budayabiliriz.

Birden çok çıktısı olan bir bağlanım uygulamasını genelde bir dizi bağımsız tek çıktılı bağlanım olarak gerçekliyoruz.

5.9 Kısa Kısa

Doğrusal cebir üzerinde iyi bir derleme olarak (Strang 1988) önerilebilir. (Harville 1997), dizey cebirini istatistik açısından ele alan başka bir güzel kitaptır.

Çokdeğişkenli verilerin rahatsız edici bir yanı boyut sayısı yüksek olduğunda görsel çözümlemenin kolayca yapılamamasıdır. İstatistik yazınında çok değişkenli verileri görüntülemek için önerilmiş yöntemler vardır (Rencher 1995). Bir olasılık, değişkenleri ikişerli alıp iki boyutta saçılımlarına bakmaktır: Veri çokdeğişkenli normal dağılmışsa herhangi iki değişkenin çizimi doğrusala yakın olmalıdır; bu yöntem, çokdeğişkenli normalliği basit görsel bir biçimde sınamayı sağlar. Bölüm 6'da veriyi bir ya da iki boyuta izdüşürüp orada görüntülemeyi göreceğiz.

Çoğu örüntü tanıma çalışması çokdeğişkenli normal dağılımlar varsayar. Böyle bir ayırtaca bazen en iyi Bayes sınıflandırıcısı bile dendiği olur, ama bu genelde yanlıştır; bu yaklaşım, yalnızca dağılımlar gerçekten çok değişkenli normalse ve parametre değerlerini doğru kestirebilecek kadar çok verimiz varsa geçerlidir. Rencher (1995), çokdeğişkenli normallik ve eşit eşdeğişinti dizeyleri için sınamaları anlatır. McLahlan (1992), çokdeğişkenli normallerle sınıflandırmayı işler ve bunu doğrusal ve karesel ayırtaçlarla karşılaştırır.

Çokdeğişkenli normallerin bir kısıtı bazı özniteliklerin kesikli olabilmesine olanak tanımamasıdır. Her ne kadar n olası değer alabilen kesikli bir değişken, n tane yapay 0/1 değişkeniyle gösterilebilirse de bunu yaptığımızda boyut sayısı artar. Boyut sayısının artmaması için bu n boyutlu uzayda, 6. bölümde göreceğimiz yöntemlerden biri kullanılarak boyut azaltma gerçeklenebilir. Böylesi karışık türde öznitelikler içeren durumlarda dağılıma bağlı sınıflandırma, (McLahlan 1992)'de işlenmiştir.

5.10 Alıştırmalar

1. Denklem 5.11'i gösterin.
2. Çok değişkenli normal $\mathcal{N}(\mu, \Sigma)$ bir dağılımdan örneklem çekip $\mathbf{m}$ ve $\mathbf{S}$ değerlerini kestirin ve μ ve Σ ile karşılaştırın. Örneklem büyüklüğü değiştiğinde kestirimlerinizin nasıl değiştiğine bakın.
3. İki tane çok değişkenli normal $\mathcal{N}(\mu_i, \Sigma_i), i = 1, 2$ dağılımından örneklem çekip Bayes ayırtaçlarını, tablo 5.1'deki dört durum için hesaplayın.
4. İki sınıflı bir uygulamada, tablo 5.1'deki dört durum için

$$\log \frac{P(C_1|\boldsymbol{x})}{P(C_2|\boldsymbol{x})}$$

oranlarını gösterin.

5. Normal dağılım kullanırken başka bir olasılık da eşdeğişinti dizeylerini köşegen almak ama farklı sınıflar için farklı olmalarına izin vermektir. Bu durum için ayırtacı hesaplayınız.
6. İki boyutta aynı ortalamaya sahip iki sınıfımız olsun. Ne tür sınırlar tanımlanabilir ?
7. x_1 ve x_2 ile gösterdiğimiz iki girdi değişkenimizi kullanarak karesel bir bağlanım gerçeklemek isteyelim:

$$f(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 (x_1)^2 + w_5 (x_2)^2$$

Örneklem, $\mathcal{X} = \{x_1^t, x_2^t, r^t\}$ verildiğinde $w_i, i = 0, \ldots, 5$, ağırlıklarını nasıl buluruz ?

8. Bağlanımda ikinci derece bir model kullanmak, girdinin karesi olan ek bir girdi kullanıp doğrusal model kullanmaya eştir. Aynı yaklaşımı sınıflandırmada da kullanabilir miyiz ?
9. Belge sınıflandırmada sözcüklerin muğlaklığını bağlanımı kullanarak azaltabiliriz. Örneğin, "kokteyl parti" ve "parti meclisi" deyimlerindeki "parti"ler farklıdır. Bunun nasıl gerçeklenebileceğini tartışın.

5.11 Kaynaklar

Duda, R. O., P. E. Hart, D. G. Stork. 2001. *Pattern Classification*, 2nd ed. New York: Wiley.

Friedman, J. H. 1989. "Regularized Discriminant Analysis." *Journal of American Statistical Association* 84: 165–175.

Harville, D. A. 1997. *Matrix Algebra from a Statistician's Perspective.* New York: Springer.

Manning, C. D., H. Schütze. 1999. *Foundations of Statistical Natural Language Processing.* Cambridge, MA: MIT Press.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition.* New York: Wiley.

Rencher, A. C. 1995. *Methods of Multivariate Analysis.* New York: Wiley.

Strang, G. 1988. *Linear Algebra and its Applications*, 3rd ed. New York: Harcourt Brace Jovanovich.

Bölüm 6

Boyut Azaltma

Bir sınıflandırma ya da bağlanım modelinin karmaşıklığı girdi sayısına bağlıdır. Bu değer, böyle bir modeli eğitmenin zaman ve uzay karmaşıklığını ve gerekli öğrenme örneği sayısını belirler. Bu bölümde, özniteliklerin bir altkümesini kullanıp kalanları atan öznitelik seçimi yöntemlerini ve asıl öznitelikleri birleştirerek az sayıda yeni öznitelik oluşturan öznitelik çıkarma yöntemlerini ele alacağız.

6.1 Giriş

İster sınıflandırma, ister bağlanım amaçlı olsun, bir uygulamada, bilgi içerebileceğine inandığımız tüm gözlem verilerini girdi olarak alırız. Kusursuz bir dünyada yaşasaydık bunların içinden gerçekten önemli olanları bulmamıza, bir başka deyişle ayrı bir iş olarak öznitelik seçimi ya da çıkarımı yapmamıza gerek olmazdı; sınıflandırma (ya da bağlanım) modeli kendisi gerekli öznitelikleri kullanıp önemsizleri atabilirdi. Ayrı bir önişleme basamağı olarak boyut azaltmayı istememiz için birkaç neden vardır:

- Çoğu öğrenme algoritmasında karmaşıklık, veri örneklemi büyüklüğüne (N) ve girdi boyut sayısına (d) bağlıdır; gereken bellek ve yapılacak işlem sayısını azaltmak için problemin boyut sayısını azaltmak isteriz. Ayrıca d değerini azaltmak deneme karmaşıklığını da azaltacakır.

- Bir girdinin gereksiz olduğunu anladığımızda onu elde etmek ya da ölçmek için gereken uğraş ve bedelden tasarruf ederiz.

- Küçük veri kümelerinde daha basit modeller daha güvenilir olur. Böylesi modellerin değişintisi düşüktür, yani, gürültü, aykırı gözlem gibi o örneklem kümesine özel etmenlerden daha az etkilenirler.

- Veri kümesi daha az değişkenle açıklandığında bu verileri üreten süreci daha iyi anlayabiliriz; bu da bilgi çıkarımı yapabilmemizi kolaylaştırır.

- Bilgi kaybetmeden birkaç boyutta görselleştirilebildiğinde veriye gözle bakıp yapısı ve aykırı değerler hakkında sonuç çıkarabiliriz.

Boyut azaltmak için iki temel yöntem vardır: öznitelik seçimi ve öznitelik çıkarımı. *Öznitelik seçiminde*, d değişkenden en çok bilgi içeren k tanesi bulunur ve öteki $(d-k)$ tanesi atılır. Bu bölümde öznitelik seçimi için *altküme seçimi* yöntemini ele alacağız.

Öznitelik çıkarımında, asıl d değişkeni birleştirerek $k < d$ tane yeni değişken oluştururuz. Bu yöntemler çıktı bilgisini kullanıp kullanmadıklarına göre gözetimli ya da gözetimsiz olabilirler. Öznitelik çıkarım için en iyi bilenen ve sık kullanılanlar, *temel bileşen çözümlemesi* ve *doğrusal ayırtaç çözümlemesi* yöntemleridir; bunların ikisi de doğrusaldır ve ilki gözetimsiz, ikincisi gözetimlidir. Tartışacağımız öteki iki gözetimsiz doğrusal izdüşüm yöntemi, etmen çözümlemesi ve çok boyutlu ölçekleme, temel bileşen çözümlemesine çok benzerlik taşır. *Doğrusal olmayan* boyut azaltmaya örnek olarak *eşölçümsel öznitelik eşleme* ve *yerel doğrusal gömme* yöntemlerini göreceğiz.

6.2 Altküme Seçimi

Altküme seçiminde amaç, eldeki özniteliklerin en iyi altkümesini bulmaktır. En iyi altküme, başarı miktarını en yüksek yapan en az sayıda boyuttan oluşur. Kalan önemsiz boyutları atarız. Uygun bir hata işleviyle bu yöntem hem sınıflandırmada hem bağlanımda kullanılabilir. d değişkenin 2^d farklı altkümesi olduğundan, d küçük değilse bütün altkümeleri sınayamayız ve bu yüzden en iyi olanı olmasa da, yeterince iyi bir alt kümeyi yeterince hızlı (çokterimli karmaşıklıkta) bulmak için sezgisel yöntemler kullanırız.

İki yaklaşım vardır: *İleriye doğru seçim* yaparken boş bir altkümeyle başlar, ve bir öznitelik daha eklemek hatayı hiç ya da yeterince azaltmayıncaya dek her yinelemede hatayı en çok azaltan özniteliği, birer birer ekleriz. *Geriye doğru seçim* yaparkense tüm özniteliklerle başlar ve bir öznitelik daha çıkarmak hatayı önemli derecede artırıncaya dek her yinelemede hatayı en çok azaltan (ya da çok az artıran) özniteliği, birer birer çıkartırız. Her iki yaklaşımda da genelleme başarısını sınarken hatayı, öğrenme kümesinden farklı bir geçerleme kümesi üzerinde ölçmeliyiz. Daha çok öznitelik kullanmak genelde öğrenme kümesinde daha düşük hata verse de her zaman daha düşük geçerleme hatası vermeyebilir.

$x_i, i = 1, \ldots, d$, ile gösterdiğimiz girdi özniteliklerinin bir kümesini F ile gösterelim. $E(F)$, yalnızca F kümesinin içerdiği öznitelikleri kullanarak eğitilmiş modelin geçerleme kümesindeki hatasını göstersin. Uyguamaya bağlı olarak bu hata, ortalama kare hata ya da sınıflandırma hatası olabilir.

İleriye doğru sırayla seçim yaparken boş bir öznitelik kümesiyle başlarız: $F = \emptyset$. Her adımda, olası tüm x_i modele x_i eklenip öğrenme kümesinde eğitilir ve geçerleme kümesi üstünde $E(F \cup x_i)$ hesaplanır. Ardından en küçük hataya neden olan x_j seçilir

$$j = \arg\min E(F \cup x_i) \tag{6.1}$$

ve modele eklenir:

$$x_j,\ F \text{ kümesine eklenir eğer } E(F \cup x_j) < E(F) \tag{6.2}$$

Eklenen hiçbir öznitelik E değerini azaltmadığında durulur. Eğer hatadaki azalma çok azsa daha önce bile durabiliriz; model karmaşıklığını düşük tutmak için, uygulama kısıtlarına göre kullanıcı tarafından tanımlanan bir eşik değeri tanımlanıp hatadaki azalma bu eşik değerinden az olursa daha erken durulabilir. Eklenen her yeni öznitelik modeli daha karmaşık yapmak yanında, o özniteliği gözlemleme ve ölçme bedeli de getirir.

Bu algoritma karmaşık olabilir, çünkü d boyuttan k boyuta inmek için sistemi $d + (d-1) + (d-2) + \cdots + (d-k)$ kez eğitmek ve geçerlemek gerekir, bu da $\mathcal{O}(d^2)$ demektir. Bu, yerel bir arama yordamıdır ve en iyi altkümeyi, yani en az hatası olan en küçük altkümeyi, bulma güvencesi yoktur. Örneğin, x_i ve x_j, tek başlarına iyi olmayıp birlikte hatayı oldukça azaltabilirler, ancak bu algoritma açgözlü olduğu ve değişkenleri birer birer eklediği için bunu ortaya çıkaramaz. Bu yöntemi genelleyip her adımda daha fazla hesap yapmayı göze alarak bir birden çok öznitelik de aynı anda eklenebilir. Ayrıca, her eklemeden sonra geriye bakıp önceden eklenmiş hangi özniteliğin atılabileceğine de bakılabilir; böylece arama uzayı büyür ama algoritmanın karmaşıklığı da artar. *Kayan arama* yöntemlerinde (Pudil, Novovicova, ve Kittler 1994), eklenen ve atılan öznitelik sayısı adımdan adıma farklı olabilir.

Geriye doğru sırayla seçim yaparken, tüm öznitelikleri içeren F kümesiyle başlar ve benzer bir biçimde, ama bu kez eklemek yerine atarak, her adımda en az hataya neden olan özniteliği bulur

$$j = \arg\min_i E(F - x_i) \tag{6.3}$$

ve modelden atarız:

$$x_j,\ F\text{'den çıkarılır eğer } E(F - x_j) < E(F) \tag{6.4}$$

Herhangi bir özniteliği çıkarmak hatayı azaltmadığında dururuz. Karmaşıklığı azaltmak için bir özniteliği çıkartmak hatayı az bir miktar artırsa da durmayabiliriz.

İleriye doğru aramrken uzayı büyütmek için akla gelen tüm yaklaşımlar geriye doğru aramaya da uyarlanabilir. Geriye doğru aramanın karmaşıklığı ileriye doğru aramayla aynıdır, ancak çok değişkenli bir modeli eğitmek az değişkenli bir modeli eğitmeye göre daha uzun süreceğinden, özellikle de değikenlerin çoğunun gereksiz olduğu beklendiğinde, ileriye doğru arama yeğlenir.

Model çıktıları üzerinden bir hata hesaplanıp kullanıldığı için altküme seçimi gözetimlidir, ama herhangi bir sınıflandırma ya da bağlanım algoritmasıyla kullanılabilir. Çok değişkenli normal dağılımlar özelinde, d değişkenli her normal dağılımın k değişkenli altkümesi normal olduğu için boyut azaltıldıktan sonra da yine dağılıma bağlı yaklaşım, $d \times d$ eşdeğişinti dizeyleri yerine daha küçük $k \times k$ eşdeğişinti dizeyleriyle, kullanılabilir.

Yüz tanıma gibi bir uygulamada öznitelik seçimi boyut azaltmak için iyi bir yöntem değildir, çünkü imge noktaları tek başına ayırt edici bilgi taşımaz; yüzün kimliği hakkında bilgi tekil noktalardan değil birçok noktanın değerinin birleşiminden elde edilebilir. Bu da, az sonra göreceğimiz öznitelik çıkarımı yöntemleriyle yapılır.

6.3 Temel Bileşen Çözümlemesi

İzdüşüm yöntemlerinde verilen d boyutlu uzaydaki girdilerden $k < d$ boyutlu yeni bir uzaya en az bilgi yitimiyle bir eşleme bulmayı amaçlıyoruz. $\boldsymbol{x}$ yöneyinin $\boldsymbol{w}$ yönüne izdüşümü iç çarpımla bulunur:

$$z = \boldsymbol{w}^T \boldsymbol{x} \tag{6.5}$$

Temel bileşen çözümlemesi sınıf bilgisini kullanmadığından gözetimsiz bir yöntemdir; ölçüt, verideki değişintidir. Değişintinin en yüksek olduğu, yani veri noktaları arasındaki farkın en iyi ortaya çıktığı yöney $\boldsymbol{w}_1$ temel bileşenedir. Tek çözüm elde edebilmek ve yöneyin boyunun değil yönünün önemli olduğunu göstermek için $\|\boldsymbol{w}_1\| = 1$ koşulunu koyuyoruz. Denklem 5.14'e göre $z_1 = \boldsymbol{w}_1^T \boldsymbol{x}$ ve $\text{Cov}(\boldsymbol{x}) = \boldsymbol{\Sigma}$ olduğunda verinin tek boyutlu izdüşümünün değişintisini bulabiliriz:

$$\text{Var}(z_1) = \boldsymbol{w}_1^T \boldsymbol{\Sigma} \boldsymbol{w}_1$$

$\boldsymbol{w}_1^T \boldsymbol{w}_1 = 1$ kısıtıyla $\text{Var}(z_1)$ değerini en büyütecek $\boldsymbol{w}_1$ yöneyini arıyoruz. Bunu bir Lagrange problemi olarak yazabiliriz:

$$\max_{\boldsymbol{w}_1} \boldsymbol{w}_1^T \boldsymbol{\Sigma} \boldsymbol{w}_1 - \alpha(\boldsymbol{w}_1^T \boldsymbol{w}_1 - 1) \tag{6.6}$$

$\boldsymbol{w}_1$ yöneyine göre türev alıp sıfıra eşitleyelim:

$$2\boldsymbol{\Sigma} \boldsymbol{w}_1 - 2\alpha \boldsymbol{w}_1 = 0, \text{ ve buradan } \boldsymbol{\Sigma} \boldsymbol{w}_1 = \alpha \boldsymbol{w}_1$$

$\boldsymbol{w}_1$, $\boldsymbol{\Sigma}$ dizeyinin özyöneyi olursa bu sağlanır ve bu durumda α, karşılık gelen özdeğerdir. Değişinti

$$\boldsymbol{w}_1^T \boldsymbol{\Sigma} \boldsymbol{w}_1 = \alpha \boldsymbol{w}_1^T \boldsymbol{w}_1 = \alpha$$

değerini en büyük yapmak için en büyük özdeğerli özyöneyi seçeriz. Sonuç olarak, temel bileşen, verilen örneklemin eşdeğişinti dizeyinin en büyük $\lambda_1 = \alpha$ özdeğerli özyöneyidir.

İkinci temel bileşen $\boldsymbol{w}_2$ yöneyi de birim uzunlukta ve $\boldsymbol{w}_1$ yöneyine dik olmalı ve değişintiyi olabildiğince büyütmelidir. Dik olma koşulu, izdüşüm sonrası $z_2 = \boldsymbol{w}_2^T \boldsymbol{x}$ değeriyle z_1 değerinin ilintisiz olmasını istediğimiz için gerekir. İkinci temel bileşen için ölçütümüz şu şekilde yazılabilir:

$$\max_{\boldsymbol{w}_2} \boldsymbol{w}_2^T \boldsymbol{\Sigma} \boldsymbol{w}_2 - \alpha(\boldsymbol{w}_2^T \boldsymbol{w}_2 - 1) - \beta(\boldsymbol{w}_2^T \boldsymbol{w}_1 - 0) \tag{6.7}$$

Yine $\boldsymbol{w}_2$ yöneyine göre türev alıp sıfıra eşitleriz:

$$2\boldsymbol{\Sigma} \boldsymbol{w}_2 - 2\alpha \boldsymbol{w}_2 - \beta \boldsymbol{w}_1 = 0 \tag{6.8}$$

Soldan $\boldsymbol{w}_1^T$ ile çarpalım:

$$2\boldsymbol{w}_1^T\boldsymbol{\Sigma}\boldsymbol{w}_2 - 2\alpha\boldsymbol{w}_1^T\boldsymbol{w}_2 - \beta\boldsymbol{w}_1^T\boldsymbol{w}_1 = 0$$

$\boldsymbol{w}_1^T\boldsymbol{w}_2 = 0$ koşulu sağlanmalıdır. $\boldsymbol{w}_1^T\boldsymbol{\Sigma}\boldsymbol{w}_2$, bir sayıl (tek boyutlu) olduğu için devriği olan $\boldsymbol{w}_2^T\boldsymbol{\Sigma}\boldsymbol{w}_1$ değerine eşit olur ve $\boldsymbol{w}_1$, $\boldsymbol{\Sigma}$ dizeyinin özyöneyi olduğu için $(\boldsymbol{\Sigma}\boldsymbol{w}_1 = \lambda_1\boldsymbol{w}_1)$

$$\boldsymbol{w}_1^T\boldsymbol{\Sigma}\boldsymbol{w}_2 = \boldsymbol{w}_2^T\boldsymbol{\Sigma}\boldsymbol{w}_1 = \lambda_1\boldsymbol{w}_2^T\boldsymbol{w}_1 = 0$$

olacaktır. Öyleyse $\beta = 0$ olur ve denklem 6.8 basitleşir:

$$\boldsymbol{\Sigma}\boldsymbol{w}_2 = \alpha\boldsymbol{w}_2$$

Buradan, $\boldsymbol{w}_2$ yöneyinin $\boldsymbol{\Sigma}$ dizeyinin ikinci en büyük, $\lambda_2 = \alpha$ özdeğerli özyöneyi olduğu sonucu çıkar. Benzer biçimde, öteki bileşenlerin de, sırayla azalan özdeğerli özyöneyler olduğu gösterilebilir.

$\boldsymbol{\Sigma}$ bakışımlı olduğundan farklı özdeğerli özyöneyler birbirine diktir. Eğer $\boldsymbol{\Sigma}$, kesin artı bir dizeyse ($\boldsymbol{x}^T\boldsymbol{\Sigma}\boldsymbol{x} > 0$, sıfırdan farklı her $\boldsymbol{x}$ için), tüm özdeğerleri artıdır Eğer $\boldsymbol{\Sigma}$ tekilse, kertesi, yani etkin boyutu, $k < d$ olur ve $\lambda_i, i = k+1, \ldots, d$, 0 değerlidir ($\lambda_i$ azalan biçimde sıralıdır). Sıfırdan büyük özdeğeri olan k özyöney, yeni, daha az boyutlu uzayın boyutlarıdır. Temel bileşen denen ilk $\boldsymbol{w}_1$ özyöneyi (yani en büyük özdeğerli özyöney), değişintinin en büyük bölümünü, ikincisi en büyük ikinci bölümünü, ve ötekiler de bu biçimde değişintinin kalanını açıklar.

$\mathbf{W}$ dizeyinin k sütunu, $\boldsymbol{\Sigma}$ dizeyinin kestirimi olan $\mathbf{S}$ dizeyinin en büyük özdeğerli k özyöneyini göstersin. Bu durumda

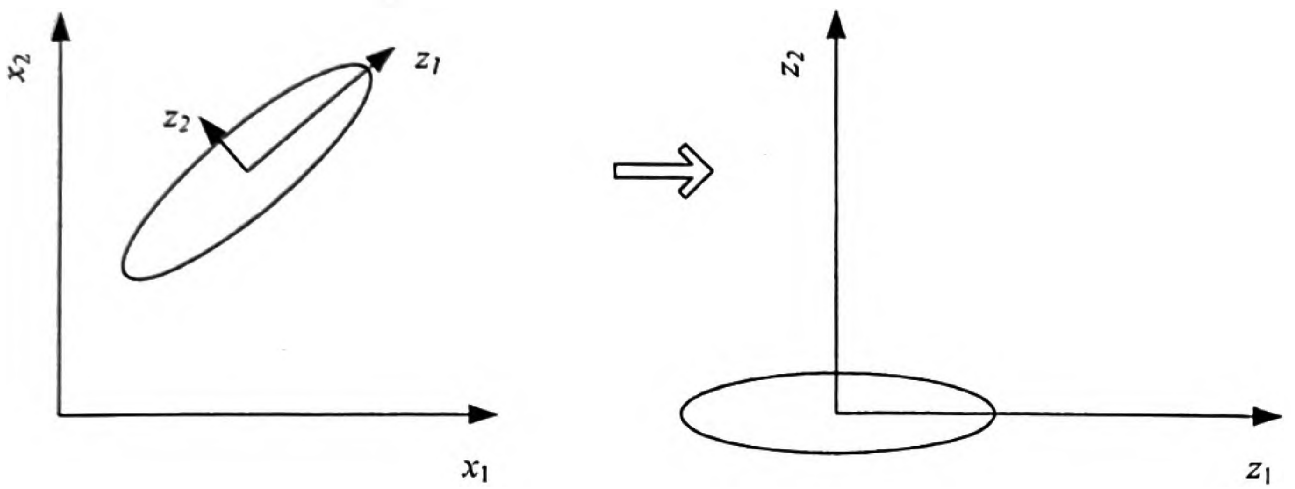
$$\boldsymbol{z} = \mathbf{W}^T(\boldsymbol{x} - \boldsymbol{m}) \tag{6.9}$$

önce $\boldsymbol{m}$ örneklem ortalamasını çıkarıp veriyi sıfır merkezli yapar, ardından doğrusal bir dönüşümle, boyutları bu özyöneylere ve bu boyutlardaki değişintilerin de özdeğerlere karşılık geldiği k boyutlu bir uzaya eşleriz (bkz. şekil 6.1). Değişintileri standartlaştırmak için o boyutun özdeğerinin kareköküne bölebiliriz:

Farklı bir türetim görelim: Öyle bir $\mathbf{W}$ dizeyi bulmak istiyoruz ki $\boldsymbol{z} = \mathbf{W}^T\boldsymbol{x}$ olarak tanımlandığında (verinin merkezlenmiş olduğunu varsayımıyla), $\mathrm{Cov}(\boldsymbol{z}) = \mathbf{D}'$ olarak tanımladığımız $\mathbf{D}'$ köşegen bir dizey olsun, yani, elde ettiğimiz z_i değerleri ilintisiz olsun.

$(d \times d)$ olan bir $\mathbf{C}$ dizeyi tanımlayalım, öyle ki i. sütunu $\boldsymbol{c}_i$, $\mathbf{S}$ dizeyinin standartlaştırılmış özyöneyini göstersin. Bu durumda $\mathbf{C}^T\mathbf{C} = \mathbf{I}$ olur ve

$$\begin{aligned}
\mathbf{S} &= \mathbf{S}\mathbf{C}\mathbf{C}^T \\
&= \mathbf{S}(\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_d)\mathbf{C}^T \\
&= (\mathbf{S}\boldsymbol{c}_1, \mathbf{S}\boldsymbol{c}_2, \ldots, \mathbf{S}\boldsymbol{c}_d)C^T \\
&= (\lambda_1\boldsymbol{c}_1, \lambda_2\boldsymbol{c}_2, \ldots, \lambda_d\boldsymbol{c}_d)\mathbf{C}^T \\
&= \lambda_1\boldsymbol{c}_1\boldsymbol{c}_1^T + \cdots + \lambda_d\boldsymbol{c}_d\boldsymbol{c}_d^T \\
&= \mathbf{C}\mathbf{D}\mathbf{C}^T
\end{aligned} \tag{6.10}$$



Şekil 6.1: Temel bileşen çözümlemesi, örneklemi ortalar ve eksenleri en yüksek değişinti yönlerine rastlayacak biçimde döndürür. Eğer z_2 yöneyinin değişintisi çok düşükse bu boyut göz ardı edilebilir ve böylece boyut sayısı ikiden bire iner.

olarak yazılır, öyle ki $\mathbf{D}$, köşegen değerleri özdeğerler $\lambda_1, \ldots, \lambda_d$ olan köşegen bir dizeydir. Buna, $\mathbf{S}$ dizeyinin *izgesel ayrışımı* denir. $\mathbf{C}$ dikken ve $\mathbf{C}\mathbf{C}^T = \mathbf{C}^T\mathbf{C} = \mathbf{I}$ olduğu için soldan $\mathbf{C}^T$ ve sağdan $\mathbf{C}$ ile çarpıp

$$\mathbf{C}^T\mathbf{S}\mathbf{C} = \mathbf{D} \tag{6.11}$$

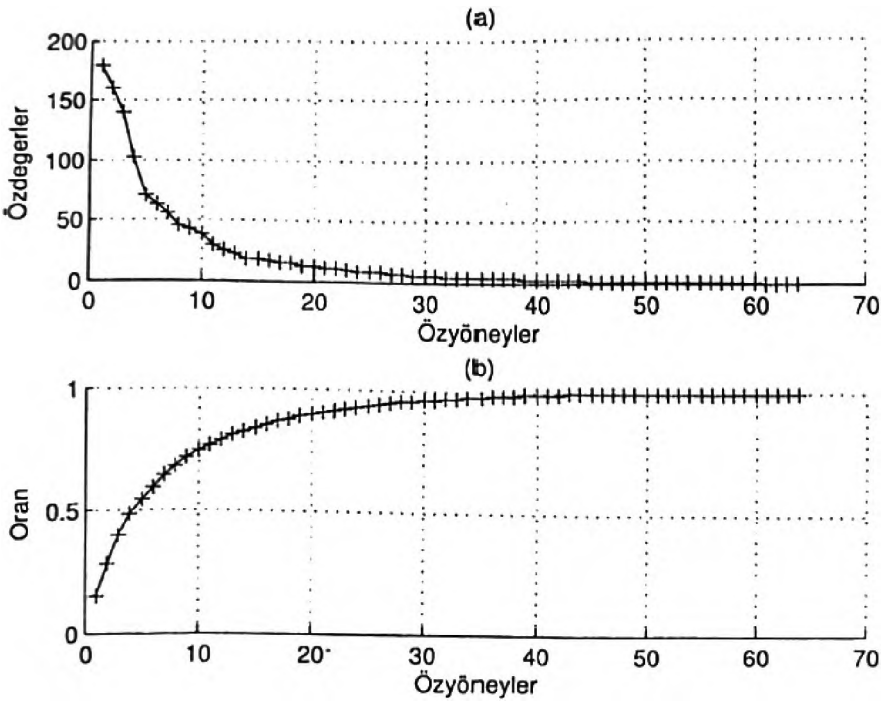
elde ederiz. $\mathbf{z} = \mathbf{W}^T\mathbf{x}$ ise $\text{Cov}(\mathbf{z}) = \mathbf{W}^T\mathbf{S}\mathbf{W}$ olur ve bunun köşegen bir dizey olmasını isteriz. Bu durumda, denklem 6.11'e göre $\mathbf{W} = \mathbf{C}$ alınabilir.

Bir örnek görelim (Rencher 1995): Diyelim ki bir sınıftaki öğrencileri, beş dersten aldıkları notlara göre sıralamak istiyoruz. Yani, verilerin aralarındaki farklar en belirgin olacak biçimde bir boyuta izdüşümlerini almak istiyoruz. Bunun için temel bileşen çözümlemesi kullanabiliriz. En yüksek özdeğere sahip olan özyöney değişintinin en yüksek olduğu, yani öğrencilerin en fazla ayrışacağı, yönü verecektir. Bu yöntem ortalama almaktan daha iyi çalışacaktır, çünkü notlar arasındaki ilintileri ve değişintilerdeki farklılıkları da göz önüne alır.

Uygulamada, tüm özdeğerler sıfırdan büyük de olsa, eğer $|\mathbf{S}|$ küçükse, $|\mathbf{S}| = \prod_{i=1}^{d} \lambda_i$ olduğu için bazı özdeğerlerin değişintiye az katkısı olduğunu görüp onları atabiliriz. Bu durumda değişintinin, örneğin yüzde 90'dan fazlasını açıklayan en büyük k bileşeni alabiliriz. λ_i değerlerini azalan şekilde sıralanmış kabul edersek k temel bileşen tarafından açıklanan *değişinti oranı*

$$\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_k + \cdots + \lambda_d}$$

olarak tanımlıdır. Eğer boyutlar yüksek ölçüde ilintiliyse, büyük özdeğere sahip az sayıda özyöney bulunacak ve k, d değerinden çok daha küçük olacak ve boyut sayısında önemli bir azalma elde edilebilecektir. Birçok görüntü ve ses işleme probleminde, (konum ya da zamana göre) yakın girdiler oldukça ilintili olduğundan bu



Şekil 6.2: UCI veritabanından alınan Optdigits veri üzerinde hesaplanan (üstte) kayşat çiziti ve (altta) açıklanan değişinti oranı. Bu, 10 sınıf ve 64 boyutlu girdiye sahip bir elle yazılmış rakamlar veri kümesidir. İlk 20 özyöney değişintinin yüzde 90'ını açıklamaktadır.

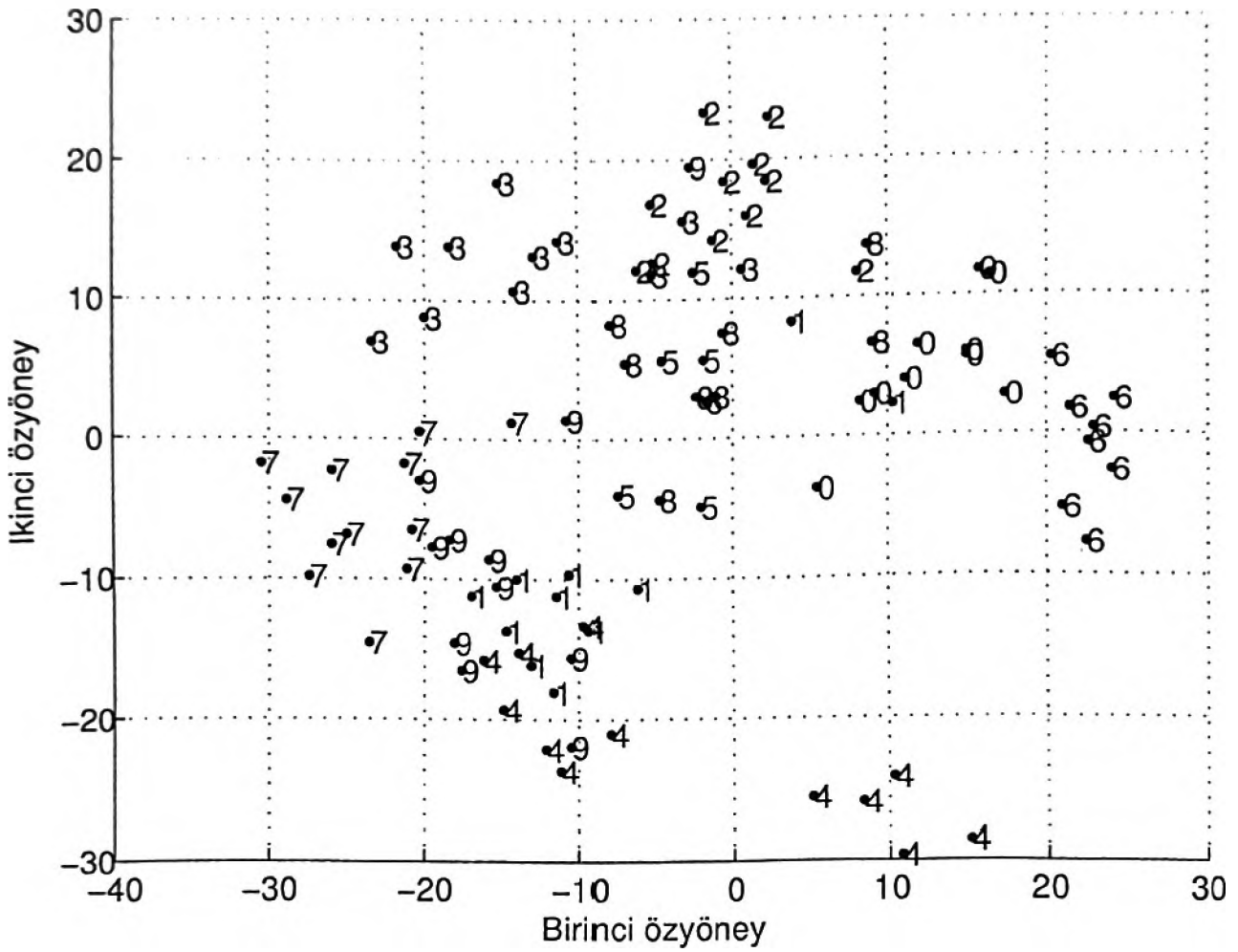
durumla sıklıkla karşılaşılmaktadır. Boyutlar ilintili olmadığında, k, d kadar büyük olacak ve temel bileşen çözümlemesi herhangi bir kazanım getirmeyecektir.

Kayşat çiziti, açıklanan değişinti oranını kullanılan özyöney sayısına göre çizer (bkz. şekil 6.2). Bu çiziti görsel olarak inceleyerek k değerine karar verebiliriz. "Dirsek" noktasında, bir tane daha özyöney eklemek açıklanan değişintiyi önemli oranda artırmaz.

Başka bir olasılık, özdeğerleri ortalama girdi değişintisinden daha düşük olan özyöneyleri atmaktır. $\sum_i \lambda_i = \sum_i s_i^2$ (ki bu, $\mathbf{S}$'nin *izidir* ve $\mathrm{tr}(\mathbf{S})$ olarak gösterilir) olduğu için ortalama özdeğer, ortalama girdi değişintisine eşittir. Ortalama özdeğerden büyük özdeğerli özyöneyleri tuttuğumuzda ortalama girdi değişintisinden yüksek değişinti taşıyanları tutmuş oluruz.

Verilen x_i boyutlarının değişintileri birbirinden çok farklıysa bu değişintiler, temel bileşen yönlerini ilintilerden daha fazla etkiler, ve bu yüzden temel bileşen çözümlemesinden önce veri, tüm girdi boyutları sıfır ortalamalı ve birim değişintili olacak biçimde normalleştirilir. Ya da, değişintiler değil de ilintilerin etkili olması için, eşdeğişinti dizeyi $\mathbf{S}$ yerine, ilinti dizeyi $\mathbf{R}$ dizeyinin özyöneyleri kullanılabilir.

Temel bileşen çözümlemesi değişintiyi açıkladığı için aykırı değerlere karşı hassastır: Merkezden uzakta olan birkaç nokta değişinti üzerinde ve dolayısıyla özyöneyler üzerinde çok etkili olabilir. *Gürbüz kestirim* yöntemleri, aykırı değerlere rağmen parametre değerlerini hesaplamayı sağlar. Basit bir yöntem, Mahalanobis uzaklıklarına



Şekil 6.3: İki temel bileşenin tanımladığı uzayda Optdigits verisinin gösterimi. Okunabilirlik için yalnızca ilk 100 nokta gösterilmiştir.

bakıp öteki noktalara göre yalnız ve uzak kalmış noktaları atmaktır.

Eğer ilk iki temel bileşen değişintinin büyük bir oranını açıklıyorsa *görsel inceleme* yapılabilir: Veriyi bu iki boyutta çizip (bkz. şekil 6.3) yapısal özelliklerine, öbeklere, aykırı örneklere, normalliğine, vs. görsel olarak bakabiliriz. Böylesi bir dağılım, verinin herhangi iki değişkenine göre dağılımından çok daha fazla bilgi taşır. Temel bileşen yöneylerine bakarak veriyi açıklayan anlamlı gizli değişkenleri çıkarmaya da çalışabiliriz. Örneğin, girdilerin görüntüler olduğu görüntü uygulamalarında, özyöneyler görüntü gibi gösterilebilir ve önemli özelliklerin şablonları olarak nitelendirilebilirler; bunlar, sıklıkla "*özyüzler*," "*özrakamlar*," vs. olarak adlandırılırlar (Turk ve Pentland 1991).

d büyük olduğunda $\mathbf{S}$ dizeyini hesaplamak, saklamak, ve işlemek masraflı olabilir. Eşdeğişinti dizeyini hiç hesaplamadan özyöney ve özdeğerleri doğrudan veriden hesaplanabilir (Chatfield ve Collins 1980).

$x \sim \mathcal{N}_d(\mu, \Sigma)$ ise izdüşüm normalliği korur: $\mathbf{W}^T x \sim \mathcal{N}_k(\mathbf{W}^T\mu, \mathbf{W}^T\Sigma\mathbf{W})$ (denklem 5.15). Eğer örneklem d boyutlu normal dağılmışda izdüşümü k boyutlu normal

bir dağılımdır. Bu daha düşük boyutlu uzayda yine dağılıma bağlı bir yaklaşım kullanılabilir. z_j değerleri ilintisiz olacağından yeni eşdeğişinti dizeyleri köşegen olacaktır, ve eğer standartlaştırma da yapılırsa, bu yeni uzayda Euclid uzaklığı kullanılarak basit bir sınıflandırıcı elde edilebilecekir.

$\boldsymbol{x}^t$ örneğinin z uzayına izdüşümünü gördük:

$$\boldsymbol{z}^t = \mathbf{W}^T(\boldsymbol{x}^t - \boldsymbol{\mu})$$

$\mathbf{W}$ dizeyi, $\mathbf{W}\mathbf{W}^T = \mathbf{I}$ olacak biçimde dikgense, geriye izdüşüm de yapılabilir:

$$\hat{\boldsymbol{x}}^t = \mathbf{W}\boldsymbol{z}^t + \boldsymbol{\mu}$$

Burada, $\hat{\boldsymbol{x}}^t$, $\boldsymbol{x}^t$ yöneyinin z uzayındaki gösteriminden geriçatımıdır. Temel bileşen çözümlemesinin tüm doğrusal izdüşümler içinde *geriçatım hatasını*

$$\sum_t \|\hat{\boldsymbol{x}}^t - \boldsymbol{x}\|^2 \tag{6.12}$$

yani, örnek ile örneğin düşük boyutlu uzaydan geriçatımı arasındaki uzaklığı, en aza indiren yöntem olduğu bilinir. Geriçatım hatası tabii ki kullanılan temel bileşen sayısına bağlıdır. Yüz tanıma gibi bir görsel tanıma uygulamasında, $\hat{\boldsymbol{x}}^t$ yöneyinin görselleştirilmesi, temel bileşen çözümlemesinin yol açtığı bilgi kaybının görsel denetlenmesine olanak sağlar.

Temel bileşen çözümlemesi gözetimsizdir ve çıktı bilgisini kullanmaz. Veride tek bir öbek olduğunu varsayar. Ancak sınıflandırmada birden çok öbek vardır. *Karhunen-Loève açılımı* sınıf bilgisini kullanmayı sağlar; örneğin, tüm örneklemin eşdeğişinti dizeyini kullanmak yerine sınıflar için ayrı ayrı eşdeğişinti dizeyleri hesaplayıp ortalamalarını (önsel olasılıklarıyla ağırlıklı) alıp eşdeğişinti dizeyi olarak kullanabiliriz.

Ortak temel bileşen çözümlemesinde (Flury 1988), temel bileşenlerin her sınıf için aynı olduğu ama bu bileşenlerin değişintilerinin sınıftan sınıfa farklılık gösterdiği varsayılır:

$$\mathbf{S}_i = \mathbf{C}\mathbf{D}_i\mathbf{C}^T$$

Böylece tüm veri birlikte kullanılabilir ve bu açıdan bakılınca bu bir düzenlileştirme yöntemi olarak görülebilir. Hem sınıflar için ortak bir eşdeğişinti dizeyi kullanmaya göre daha az karmaşıktır, hem de $\mathbf{S}_i$ dizeylerinin birbirinden farklı olabilmesine olanak tanır. İlgili bir yaklaşım, tüm özniteliklerin ilintisiz olduğu düşük boyutlu bir uzaya doğrusal izdüşüm yapan ve sonra uzaklığa göre sınıflandıran *Esnek ayırtaç çözümlemesidir* (Hastie, Tibshirani ve Buja 1994).

6.4 Etken Çözümlemesi

Temel bileşen çözümlemesiyle asıl $x_i, i = 1, \ldots, d$, değişkenlerinin doğrusal birleşimleri olan z değişkenleri oluşturulur:

$$\boldsymbol{z} = \mathbf{W}^T(\boldsymbol{x} - \boldsymbol{\mu})$$

Etken çözümlemesinde (EÇ), gözlenemeyen *saklı etkenler* $z_j, j = 1, \ldots, k$, olduğu ve birleşerek gözlemlediğimiz x değişkenlerini *ürettikleri* varsayılır. Dolayısıyla etmen çözümlemesi, temel bileşen çözümlemesinin tersi yönde çalışır (bkz. şekil 6.4). Amaç, gözlemlenen değişkenler arasındaki bağımlılıkları az sayıda etkenle açıklamaktır.

Kendi aralarında yüksek ve diğer değişkenlerle düşük ilintisi olan bir küme değişkeni ele alalım. Temelde bu değişkenlere neden olan tek bir etken olabilir. Eğer öteki değişkenler de benzer biçimde altkümelere ayrılabilirlerse birkaç etken bu değişkenlerin tamamını gösterebilir. Her ne kadar etken çözümlemesi değişkenleri her zaman etken kümelerine bölebilse de, bu etkenlerin anlamlı ya da gerçekten var olup olmadıkları tartışmaya açıktır.

Etken çözümlemesi de temel bileşen çözümlemesi gibi veride tek öbek olduğunu varsayar ve gözetimsizdir. Amaç, verinin daha az boyutlu bir uzayda bilgi kaybetmeden modellenmesidir. Etken çözümlemesinde bu, değişkenler arası ilinti olarak ölçülür.

Temel bileşen çözümlemesindeki gibi, $E[x] = \mu$ ve $\text{Cov}(x) = \Sigma$ parametreli bir olasılık dağılımından çekilmiş bir $\mathcal{X} = \{x^t\}_t$ örneklemi verilmiştir. Etkenlerin birim normaller ($E[z_j] = 0, \text{Var}(z_j) = 1$) ve ilintisiz ($\text{Cov}(z_i, z_j) = 0, i \neq j$) olduğunu varsayarız. Verinin etkenler tarafından açıklanmayan bölümü için her girdiye ayrı bir kaynak ϵ_i tanımlıdır, ve bunlar, sıfır ortalamalı ($E[\epsilon_i] = 0$), bilinmeyen değişkenli ($\text{Var}(\epsilon_i) = \psi_i$), ve kendi aralarında ve etkenlerle ilintisizdir ($\text{Cov}(\epsilon_i, \epsilon_j) = 0, i \neq j$, ve $\text{Cov}(\epsilon_i, z_j) = 0, \forall i, j$).

Etken çözümlemesi, her $x_i, i = 1, \ldots, d$ girdisinin daha az sayıda $z_j, j = 1, \ldots, k < d$ etkeninin ağırlıklı toplamı artı bir artık terimi olarak yazılabildiğini varsayar (bkz. şekil 6.5):

$$\begin{aligned} x_i - \mu_i &= v_{i1}z_1 + v_{i2}z_2 + \cdots + v_{ik}z_k + \epsilon_i, \forall i = 1, \ldots, d \\ x_i - \mu_i &= \sum_{j=1}^{k} v_{ij}z_j + \epsilon_i \end{aligned} \tag{6.13}$$

Yöney-dizey gösterimi kullanarak yazalım:

$$x - \mu = \mathbf{V}z + \epsilon \tag{6.14}$$

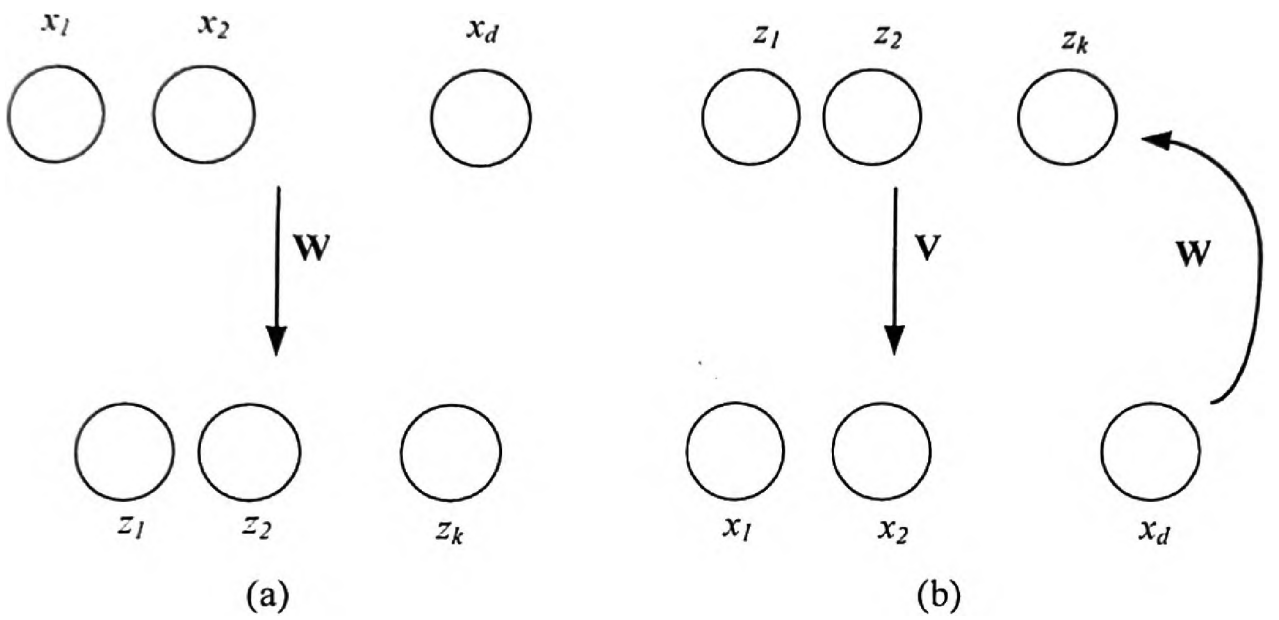
Burada, $\mathbf{V}$, *etken yüklemeleri* denilen $d \times k$ ağırlık dizeyidir. Bu andan sonra, genellikten kaybetmeden $\mu = \mathbf{0}$ olduğunu varsayacağız; değilse her zaman izdüşümden sonra bir μ yöneyi ekleyebiliriz. $\text{Var}(z_j) = 1$ ve $\text{Var}(\epsilon_i) = \psi_i$ varsayıldığı için

$$\text{Var}(x_i) = v_{i1}^2 + v_{i2}^2 + \cdots + v_{ik}^2 + \psi_i \tag{6.15}$$

Burada, $\sum_{j=1}^{k} v_{ij}^2$, değişintinin ortak etkenler tarafından açıklanan bölümü, ψ_i de x_i girdisine özgü değişintidir.

Yöney-dizey gösterimi kullanalım:

$$\Sigma = \text{Cov}(x) = \text{Cov}(\mathbf{V}z + \epsilon) \tag{6.16}$$



Şekil 6.4: (a) Temel bileşen çözümlemesi verilen girdi değişkenlerinin doğrusal birleşimleri olan yeni değişkenler üretir. (b) Etken çözümlemede ise doğrusal olarak birleştirildiklerinde girdi değişkenlerini üreten etkenler olduğu varsayılır.

$$\begin{aligned} &= \text{Cov}(\mathbf{V}\boldsymbol{z}) + \text{Cov}(\boldsymbol{\epsilon}) \\ &= \mathbf{V}\text{Cov}(\boldsymbol{z})\mathbf{V}^T + \boldsymbol{\Psi} \\ &= \mathbf{V}\mathbf{V}^T + \boldsymbol{\Psi} \end{aligned} \tag{6.17}$$

Burada, $\boldsymbol{\Psi}$, köşegenleri ψ_i olan köşegen bir dizeydir. Etkenler ilintisiz, birim normaller olduğu için $\text{Cov}(\boldsymbol{z}) = \mathbf{I}$ olur. Örneğin, iki etken varsa

$$\text{Cov}(x_1, x_2) = v_{11}v_{21} + v_{12}v_{22}$$

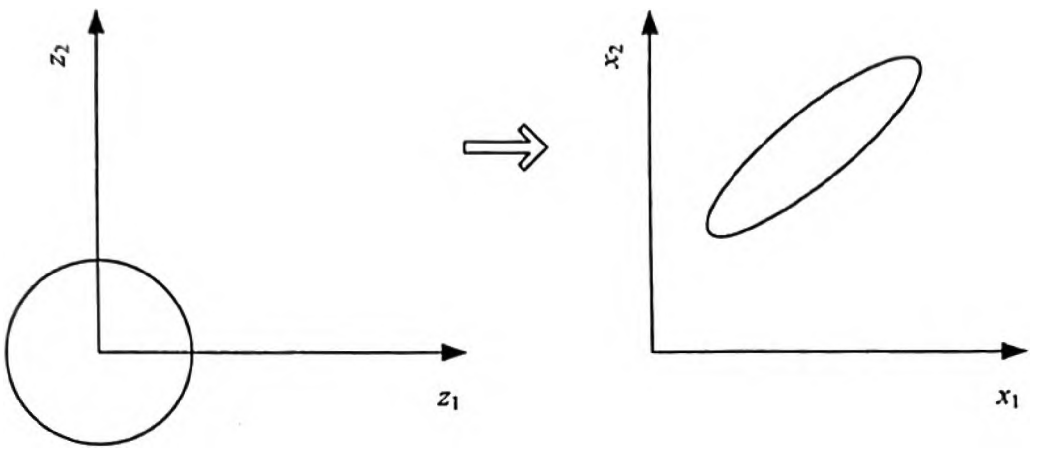
x_1 ve x_2 arasında yüksek eşdeğişinti varsa bir etken üzerinden ilişkililerdir. Eğer ilk etkenden dolayıysa v_{11} ve v_{21} yükleme değerlerinin ikisi de yüksek olacaktır; eğer ikinci etkenden dolayı ise v_{12} ve v_{22} yükleme değerlerinin ikisi birden yüksek olacaktır. Her iki durumda da $v_{11}v_{21} + v_{12}v_{22}$ toplamı yüksek olacaktır. Eşdeğişinti küçük olduğunda ise, x_1 ve x_2 farklı etkenlere bağlıdır ve toplamın içindeki çarpımlarda bir terim yüksekken öbürü düşük olacak ve dolayısıyla toplam düşük olacaktır.

$$\text{Cov}(x_1, z_2) = \text{Cov}(v_{12}z_2, z_2) = v_{12}\text{Var}(z_2) = v_{12}$$

Dolayısıyla, $\text{Cov}(\boldsymbol{x}, \boldsymbol{z}) = \mathbf{V}$ olur, ve buradan yüklemelerin değişkenlerle etkenlerin ilintilerine karşılık geldiğini görürüz.

$\boldsymbol{\Sigma}$ dizeyinin kestiricisi olan $\mathbf{S}$ bilindiğinde

$$\mathbf{S} = \mathbf{V}\mathbf{V}^T + \boldsymbol{\Psi}$$



Şekil 6.5: Etkenler bağımsız birim normallerdir; uzatılıp, döndürülüp, ötelenerek girdileri oluştururlar.

olacak biçimde $\mathbf{V}$ ve $\mathbf{\Psi}$ dizeylerini bulmak isteriz. Az sayıda etken varsa, yani $\mathbf{V}$ dizeyinin az sütunu varsa, $\mathbf{S}$ için, $\mathbf{V}$ dizeyinin $d \times k$ olduğu ve $\mathbf{\Psi}$ dizeyinin d değer içerdiği daha basit bir yapı yazılabilir ve böylece parametre sayısı d^2 değerinden $d \cdot k + d$ değerine iner.

$\mathbf{\Psi}$ köşegen olduğundan eşdeğişintiler $\mathbf{V}$ ile gösterilir. Temel bileşen çözümlemesi, ayrı $\mathbf{\Psi}$ değerleri kullanmaz ve eşdeğişinti *ve* değişintileri birlikte açıklamaya çalışır. Tüm ψ_i değerleri eşit olduğunda, yani, $\mathbf{\Psi} = \psi \mathbf{I}$ ise, elimizdeki yapıya *olasılıksal TBÇ* denir (Tipping ve Bishop 1999); ψ_i değerleri 0 olduğunda geleneksel TBÇ elde edilir.

Şimdi etken yüklemelerini ve girdiye özel değişintileri nasıl bulabileceğimizi görelim: İlk olarak $\mathbf{\Psi}$ değerlerini göz ardı edip izgesel ayrışım kullanırız:

$$\mathbf{S} = \mathbf{C}\mathbf{D}\mathbf{C}^T = \mathbf{C}\mathbf{D}^{1/2}\mathbf{D}^{1/2}\mathbf{C}^T = (\mathbf{C}\mathbf{D}^{1/2})(\mathbf{C}\mathbf{D}^{1/2})^T$$

Açıklanan değişinti oranına bakarak ilk k özyöney alınır ve $\mathbf{C}$, özyöneylerin $d \times k$ dizeyi, $\mathbf{D}^{1/2}$ de özdeğerlerin kareköklerini içeren $k \times k$ dizey olur. Böylece

$$\mathbf{V} = \mathbf{C}\mathbf{D}^{1/2} \tag{6.18}$$

yazılır. ψ_j değerlerini, denklem 6.15'den bulabiliriz:

$$\psi_i = s_i^2 - \sum_{j=1}^{k} v_{ij}^2 \tag{6.19}$$

$\mathbf{V}$, herhangi bir dikgen–$\mathbf{T}\mathbf{T}^T = \mathbf{I}$ koşulunu sağlayan–dizeyle çarpıldığında başka bir geçerli çözüm elde edileceğinden bu yüzden çözüm tek değildir.

$$\mathbf{S} = (\mathbf{V}\mathbf{T})(\mathbf{V}\mathbf{T})^T = \mathbf{V}\mathbf{T}\mathbf{T}^T\mathbf{V}^T = \mathbf{V}\mathbf{I}\mathbf{V}^T = \mathbf{V}\mathbf{V}^T$$

$\mathbf{T}$ dikgen bir dizeyse sıfır noktasına uzaklık değişmez. $\boldsymbol{z} = \mathbf{T}\boldsymbol{x}$ ise

$$\boldsymbol{z}^T\boldsymbol{z} = (\mathbf{T}\boldsymbol{x})^T(\mathbf{T}\boldsymbol{x}) = \boldsymbol{x}^T\mathbf{T}^T\mathbf{T}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{x}$$

olur. Dikgen bir dizeyle çarpmak eksenleri döndürür ve daha kolay yorumlanabilen eksenler bulmamıza izin verir (Rencher 1995). İki boyutta

$$\mathbf{T} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}$$

eksenleri ϕ kadar döndürür. İki tür döndürme vardır: Dikgen döndürmede etkenler birbirine dik kalır; yatık döndürmedeyse etkenler ilintili duruma gelebilir. Etkenlerin yorumlanabilir olması için az sayıda değişken en fazla yükleme taşıyacak biçimde döndürülür. Ancak, yorumlama özneldir ve kişinin kendi önyargısını veriye doğrulatmaya çalışması için kullanılmamalıdır.

Etken çözümlemesini yüklemeleri bularak bilgi çıkarmak ya da değişkenleri az sayıda etkenle göstermek için kullanabiliriz. $k < d$ olduğunda boyut azaltır. Bunlardan ilkinin nasıl yapıldığını gördük, şimdi de etken çözümlemesinin boyut azaltma için nasıl kullanıldığına bakalım.

Boyut azaltma için, asıl x_i değişkenlerinden etken z_j değerlerini bulabilmemiz gerekir. x_i ortalaması 0 olacak biçimde merkezdeyken öyle w_{ji} yüklemeleri bulmak istiyoruz ki

$$z_j = \sum_{i=1}^{d} w_{ji} x_i + \epsilon_i, j = 1, \ldots, k \tag{6.20}$$

sağlansın. Yöney-dizey gösteriyle de yazabiliriz (örnek t için):

$$z^t = \mathbf{W}^T x^t + \epsilon, \forall t = 1, \ldots, N$$

Bu, d girdili ve k çıktılı doğrusal bir modeldir. Devriğini yazalım:

$$(z^t)^T = (x^t)^T \mathbf{W} + \epsilon^T, \forall t = 1, \ldots, N$$

Elimizde N örnek olduğunda

$$\mathbf{Z} = \mathbf{X}\mathbf{W} + \Xi \tag{6.21}$$

$\mathbf{Z}$, $N \times k$ etkenler, $\mathbf{X}$, $N \times d$ (merkeze taşınmış) örnekler ve Ξ, $N \times k$ sıfır ortalamalı gürültüdür ve çok değişkenli doğrusal bir bağlanım yapısı elde ederiz. Bölüm 5.8'de $\mathbf{W}$ değerlerinin

$$\mathbf{W} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Z}$$

olarak bulunabileceğini gördük ama burada $\mathbf{Z}$ değerlerini bilmiyoruz; zaten onları hesaplamak istiyoruz. Eşitliğin her iki yanını $N-1$ ile çarpıp böler

$$\begin{aligned} \mathbf{W} &= (N-1)(\mathbf{X}^T\mathbf{X})^{-1}\frac{\mathbf{X}^T\mathbf{Z}}{N-1} \\ &= \left(\frac{\mathbf{X}^T\mathbf{X}}{N-1}\right)^{-1}\frac{\mathbf{X}^T\mathbf{Z}}{N-1} \\ &= \mathbf{S}^{-1}\mathbf{V} \end{aligned} \tag{6.22}$$

denklem 6.22'yi denklem 6.21'e koyar ve S'nin tekil olmadığını varsayıp

$$\mathbf{Z} = \mathbf{XW} = \mathbf{XS}^{-1}\mathbf{V} \tag{6.23}$$

olarak çözebiliriz. x_i değişkenleri birim değişintili olacak biçimde normalleştirilmemişse **S** yerine **R** kullanılabilir.

Boyut azaltmada, etken çözümlemesi, temel bileşen çözümlemesine göre, ortak etkenlerin saptanması, basit bir açıklama getirme, ve bilgi çıkarımına olanak tanıyan etken yorumlanabilirliği dışında bir üstünlük sağlamaz. Örneğin, konuşma tanımada, $\boldsymbol{x}$ işitimsel sinyali, ciğerlerden çıkan hava geçerken çene, dil, artdamak, dudaklar, ve ağzı denetleyen kas öbeklerinin (doğrusal olmayan) etkileşimiyle oluşur. Ses sinyali, bu *eklemleyiciler* uzayında gösterilebilirse tanıma çok daha kolay olabilir. Böylesi üretici modeller, konuşma tanımada güncel araştırma konularından biridir; 16. bölümde böylesi modellerin çizgesel olarak gösterimi ve işlenmesinden bahsedeceğiz.

6.5 Çok Boyutlu Ölçekleme

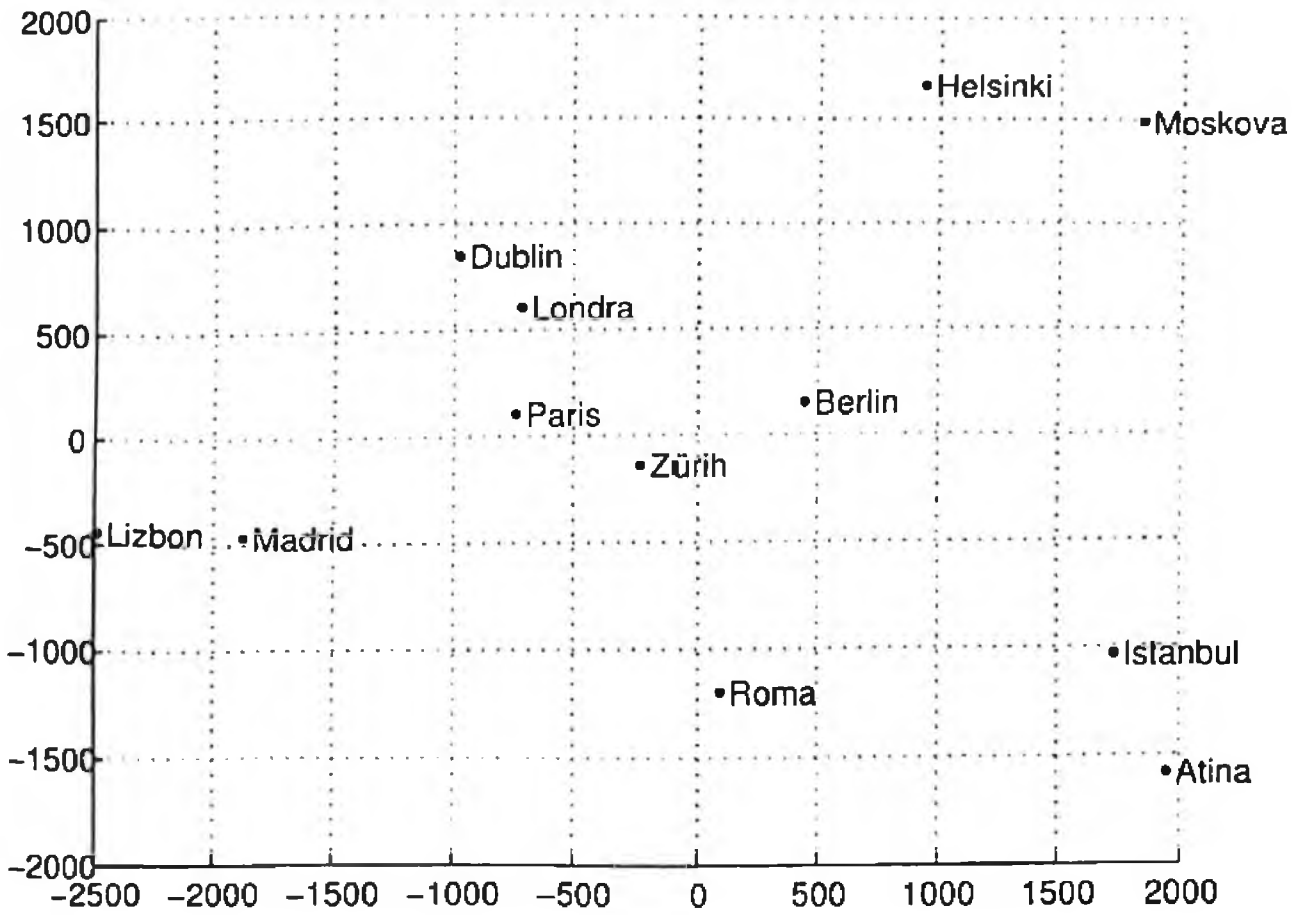
Elimizde N için her nokta çifti arasındaki d_{ij}, for all $i, j = 1, \ldots, N$ uzaklık bilgisi olsun ama bu noktaların kaç boyutlu bir uzayda, nerede konumlandıklarını, ya da uzaklıkların nasıl hesaplandığını bilmiyor olalım. *Çok boyutlu ölçekleme* (ÇBÖ), bu noktaları daha düşük boyutlu bir uzayda öyle bir biçimde yerleştirir ki bu yeni uzayda noktalar arasındaki Euclid uzaklıkları verilen uzaklıklara olabildiğince yakın olsun. Dolayısıyla bilinmeyen boyutlu bir uzaydan başka bir uzaya eşleme gerektirir.

Çok boyutlu ölçeklemenin en bilinen örneği, bir ülkedeki kentler arasındaki karayolu uzaklıklarını alıp bunları iki boyuta eşleyerek ülkenin haritasını çıkarmaktır. Bu harita, dağlar, göller gibi doğal engeller yüzünden karayolunun kuş uçuşu yoldan (yani Euclid uzaklığından) fark ettiği yerlerde çarpıtılmış ve daha büyük uzaklıkları gösterebilecek biçimde uzatılmış olacaktır (bkz. şekil 6.6). Harita merkezlenmiş olacaktır ve çözüm tek değildir; uzaklıkları değiştirmeden herhangi bir açıda döndürülebilir ya da ayna görüntüsü olabilir.

Çok boyutlu ölçekleme boyut azaltma için kullanılabilir. $\boldsymbol{x}$ girdilerinin geldiği d boyutlu uzayda nokta çiftleri arasındaki Euclid uzaklıklarıyla $N \times N$ dizeyi oluşturup çok boyutlu ölçekleme girdi olarak veririz; sonucunda çıkan düşük boyutlu uzaydaki konumu düşük girdili gösterimi olur.

Örneklememizi $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$ ile gösterelim ve $\boldsymbol{x}^t \in \Re^d$ olsun. r ve s noktaları arasındaki Euclid uzaklığının karesini açalım:

$$\begin{aligned} d_{rs}^2 &= \|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2 = \sum_{j=1}^{d}(x_j^r - x_j^s)^2 = \sum_{j=1}^{d}(x_j^r)^2 - 2\sum_{j=1}^{d} x_j^r x_j^s + \sum_{j=1}^{d}(x_j^s)^2 \\ &= b_{rr} + b_{ss} - 2b_{rs} \end{aligned} \tag{6.24}$$



Şekil 6.6: Çok boyutlu ölçeklemeyle oluşturulan Avrupa haritası. Kent çiftleri arasındaki karayolu uzaklıkları girdi olarak verilmiş ve ÇBÖ, bu kentleri, aralarındaki uzaklıkları olabildiğinde koruyacak biçimde iki boyuta yerleştirmiştir.

öyle ki

$$b_{rs} = \sum_{j=1}^{d} x_j^r x_j^s \tag{6.25}$$

Çözümü sınırlamak için veri merkezini sıfır noktası yapalım:

$$\sum_{t=1}^{N} x_j^t = 0, \forall j = 1, \ldots, d$$

Ardından, denklem 6.24'ü, r, s, ve her ikisi üzerinden toplayıp ve

$$T = \sum_{t=1}^{N} b_{tt} = \sum_t \sum_j (x_j^t)^2$$

tanımıyla

$$\begin{aligned} \sum_r d_{rs}^2 &= T + N b_{ss} \\ \sum_s d_{rs}^2 &= N b_{rr} + T \\ \sum_r \sum_s d_{rs}^2 &= 2NT \end{aligned}$$

eşitliklerini elde ederiz.

$$d_{\bullet s}^2 = \frac{1}{N} \sum_r d_{rs}^2 \quad , \quad d_{r\bullet}^2 = \frac{1}{N} \sum_s d_{rs}^2 \quad , \quad d_{\bullet\bullet}^2 = \frac{1}{N^2} \sum_r \sum_s d_{rs}^2$$

tanımları ve denklem 6.24'ü kullanarak

$$b_{rs} = \frac{1}{2}(d_{r\bullet}^2 + d_{\bullet s}^2 - d_{\bullet\bullet}^2 - d_{rs}^2) \tag{6.26}$$

yazabiliriz. b_{rs} değerlerini hesaplamışken, $\mathbf{B} = \mathbf{X}\mathbf{X}^T$ olduğunu da bildiğimize göre denklem 6.25'i kullanarak yaklaşık bir çözüm arayabiliriz. İzgesel ayrışım kullanarak $\mathbf{C}$ dizeyinin, sütunları $\mathbf{B}$ dizeyinin özyöneyleri olan bir dizey ve $\mathbf{D}^{1/2}$ dizeyi, köşegenlerinde özdeğerlerin karekökleri olan köşegen bir dizey olduğunda, $\mathbf{X} = \mathbf{C}\mathbf{D}^{1/2}$ biçiminde yazılabildiğini biliyoruz. TBÇ ve EÇ'de yaptığımız gibi $\mathbf{B}$ dizeyinin özdeğerlerine bakarak d (ve N) değerinden daha düşük bir k boyutuna karar verebiliriz. $\mathbf{c}_j$ özyöney, ve λ_j karşılık gelen özdeğer olsun; $\mathbf{c}_j$ yöneyinin N boyutlu olduğunu unutmayalım. Yeni boyutlar, karşılık gelen özyöneylerin normalleştirilmesiyle bulunur:

$$z_j^t = \sqrt{\lambda_j} c_j^t, j = 1, \ldots, k, t = 1, \ldots, N \tag{6.27}$$

Yani, t. örneğin yeni değerleri, $\mathbf{c}_j, j = 1, \ldots, k$ özyöneylerin t. boyutlarının normalleştirilmişidir.

$N \times N$ boyutlu $\mathbf{X}\mathbf{X}^T$ dizeyinin özdeğerleriyle $d \times d$ boyutlu $\mathbf{X}^T\mathbf{X}$ dizeyinin özdeğerleri aynıdır ve özyöneyleri arasında basit bir doğrusal dönüşüm bağıntısı vardır (Chatfield ve Collins 1980). Bu da TBÇ'nin, ÇBÖ ile aynı işi, daha ucuza yaptığını gösterir. Eşdeğişinti yerine ilinti dizeyi üzerinde yapılan TBÇ, her değişkenin birim değişintiye getirilmesiyle elde edilen standartlaştırılmış Euclid uzaklıklarını kullanan ÇBÖ'ye eşittir.

Genel durumda, $\boldsymbol{z} = g(\boldsymbol{x}|\theta)$ biçiminde bir eşleme bulmak istiyoruz, öyle ki, $\boldsymbol{z} \in \Re^k, \boldsymbol{x} \in \Re^d$, ve θ bu eşlemenin parametreleri olsun. Yukarıda ele aldığımız ÇBÖ, doğrusal bir dönüşüme karşılık gelir:

$$\boldsymbol{z} = g(\boldsymbol{x}|\mathbf{W}) = \mathbf{W}^T\boldsymbol{x} \tag{6.28}$$

ama doğrusal olmayan eşleme de kullanılabilir; buna *Sammon eşlemesi* denir. Normalleştirilmiş eşleme hatasına, *Sammon gerilimi* denir:

$$\begin{aligned} E(\theta|\mathcal{X}) &= \sum_{r,s} \frac{(\|\boldsymbol{z}^r - \boldsymbol{z}^s\| - \|\boldsymbol{x}^r - \boldsymbol{x}^s\|)^2}{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2} \\ &= \sum_{r,s} \frac{(\|g(\boldsymbol{x}^r|\theta) - g(\boldsymbol{x}^s|\theta)\| - \|\boldsymbol{x}^r - \boldsymbol{x}^s\|)^2}{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2} \end{aligned} \tag{6.29}$$

$g(\cdot|\theta)$ için herhangi bir bağlanım yöntemi kullanılabilir ve $\mathcal{X}$ üzerinde gerilimi en aza indiren θ kestirilir. Eğer $g(\cdot)$, $\boldsymbol{x}$ girdisinin doğrusal olmayan bir işleviyse doğrusal olmayan bir boyut azaltma yapılmış olur.

Sınıflandırmada uzaklığa sınıf bilgisi de eklenir; örneğin, (Webb 1999)

$$d'_{rs} = (1-\alpha)d_{rs} + \alpha c_{rs}$$

burada c_{rs}, $\boldsymbol{x}^r$ ve $\boldsymbol{x}^s$ örneklerinin sınıfları arasındaki "uzaklıktır." Sınıflararası böylesi uzaklıklar öznel olarak tanımlanır ve α çapraz geçerlemeyle ayarlanır.

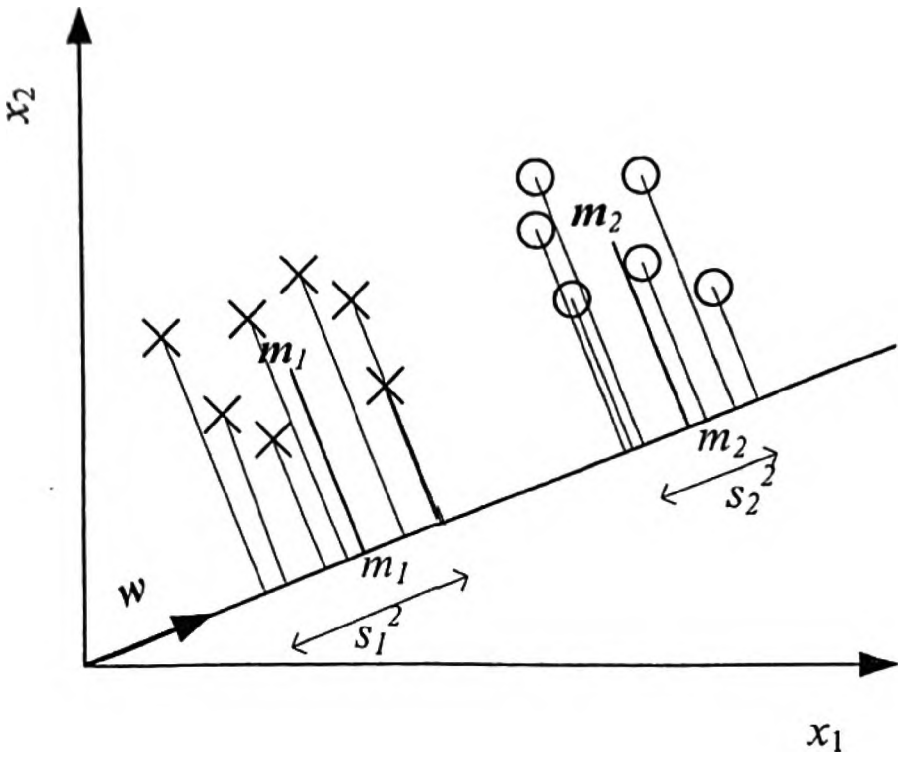
6.6 Doğrusal Ayırtaç Çözümlemesi

Doğrusal ayırtaç çözümlemesi (DAÇ), sınıflandırma için gözetimli bir boyut azaltma yöntemidir. Tartışmaya iki sınıflı durumla başlayalım ve sonra $K > 2$ sınıf için genelleyelim.

C_1 and C_2 sınıflarından örnekler verildiğinde, $\boldsymbol{w}$ yöneyiyle tanımlanan öyle bir yön bulmak istiyoruz ki, $\boldsymbol{w}$ üzerine izdüşümleri alındığında iki sınıfın örnekleri birbirinden olabildiğince iyi ayrılsın. Daha önce gördüğümüz gibi

$$z = \boldsymbol{w}^T\boldsymbol{x} \tag{6.30}$$

$\boldsymbol{x}$ örneğinin $\boldsymbol{w}$ üzerine izdüşümüdür ve sayıl olduğu için boyut sayısını d değerinden bire indirir.



Şekil 6.7: İki boyutlu, iki sınıflı verilerin w üzerine izdüşümü.

$\boldsymbol{m}_1 \in \Re^d$ ve $m_1 \in \Re$, $\mathcal{C}_1$ sınıfından örneklerin izdüşümden önceki ve sonraki ortalamasını gösterir. $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}$ örneklemimizi ($\boldsymbol{x}^t \in \mathcal{C}_1$ ise $r^t = 1$, ve $\boldsymbol{x}^t \in \mathcal{C}_2$ ise $r^t = 0$) kullanarak bunları kestirebiliriz:

$$\begin{aligned} m_1 &= \frac{\sum_t \boldsymbol{w}^T \boldsymbol{x}^t r^t}{\sum_t r^t} = \boldsymbol{w}^T \boldsymbol{m}_1 \\ m_2 &= \frac{\sum_t \boldsymbol{w}^T \boldsymbol{x}^t (1 - r^t)}{\sum_t (1 - r^t)} = \boldsymbol{w}^T \boldsymbol{m}_2 \end{aligned} \tag{6.31}$$

$\mathcal{C}_1$ ve $\mathcal{C}_2$ sınıflarından örneklerin izdüşümden sonraki *saçılımını* tanımlayalım:

$$\begin{aligned} s_1^2 &= \sum_t (\boldsymbol{w}^T \boldsymbol{x}^t - m_1)^2 r^t \\ s_2^2 &= \sum_t (\boldsymbol{w}^T \boldsymbol{x}^t - m_2)^2 (1 - r^t) \end{aligned} \tag{6.32}$$

İzdüşümden sonra iki sınıfın iyi ayrışmış olması için ortalamalarının olabildiğince uzak olmasını ve bir sınıfın örneklerinin olabildiğince küçük bir alana saçılmasını isteriz. Yani $|m_1 - m_2|$ değerini büyük ve $s_1^2 + s_2^2$ değeri küçük olsun isteriz (bkz. şekil 6.7). *Fisher doğrusal ayırtacı*, bunların oranını en büyük yapan $\boldsymbol{w}$ olarak tanımlıdır:

$$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} \tag{6.33}$$

Payı açalım:

$$\begin{aligned}(m_1 - m_2)^2 &= (w^T m_1 - w^T m_2)^2 \\ &= w^T(m_1 - m_2)(m_1 - m_2)^T w \\ &= w^T \mathbf{S}_B w \end{aligned} \tag{6.34}$$

Burada, $\mathbf{S}_B = (m_1 - m_2)(m_1 - m_2)^T$ dizeyine *sınıflar arası saçılım dizeyi* denir. Payda ise sınıf örneklerinin izdüşüm sonrasında merkez çevresindeki saçılımlarının toplamıdır:

$$\begin{aligned} s_1^2 &= \sum_t (w^T x^t - m_1)^2 r^t \\ &= \sum_t w^T (x^t - m_1)(x^t - m_1)^T w r^t \\ &= w^T \mathbf{S}_1 w \end{aligned} \tag{6.35}$$

Burada

$$\mathbf{S}_1 = \sum_t r^t (x^t - m_1)(x^t - m_1)^T \tag{6.36}$$

dizeyine C_1 sınıfı için *sınıf içi saçılım dizeyi* denir. $\mathbf{S}_1 / \sum_t r^t$, $\mathbf{\Sigma}_1$ dizeyinin kestirimidir. Benzer biçimde, $s_2^2 = w^T \mathbf{S}_2 w$ ve $\mathbf{S}_2 = \sum_t (1 - r^t)(x^t - m_2)(x^t - m_2)^T$ tanımlıdır. Buradan

$$s_1^2 + s_2^2 = w^T \mathbf{S}_W w$$

yazılabilir ve $\mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$ dizeyine toplam sınıf içi saçılımı denir. $s_1^2 + s_2^2$ bölü örnek sayısı bize birleşmiş verinin değişintisini verir. Denklem 6.33'ü yeniden yazalım:

$$J(w) = \frac{w^T \mathbf{S}_B w}{w^T \mathbf{S}_W w} = \frac{|w^T (m_1 - m_2)|^2}{w^T \mathbf{S}_W w} \tag{6.37}$$

J işlevinin w yöneyine göre türevini alıp sıfıra eşitleyelim:

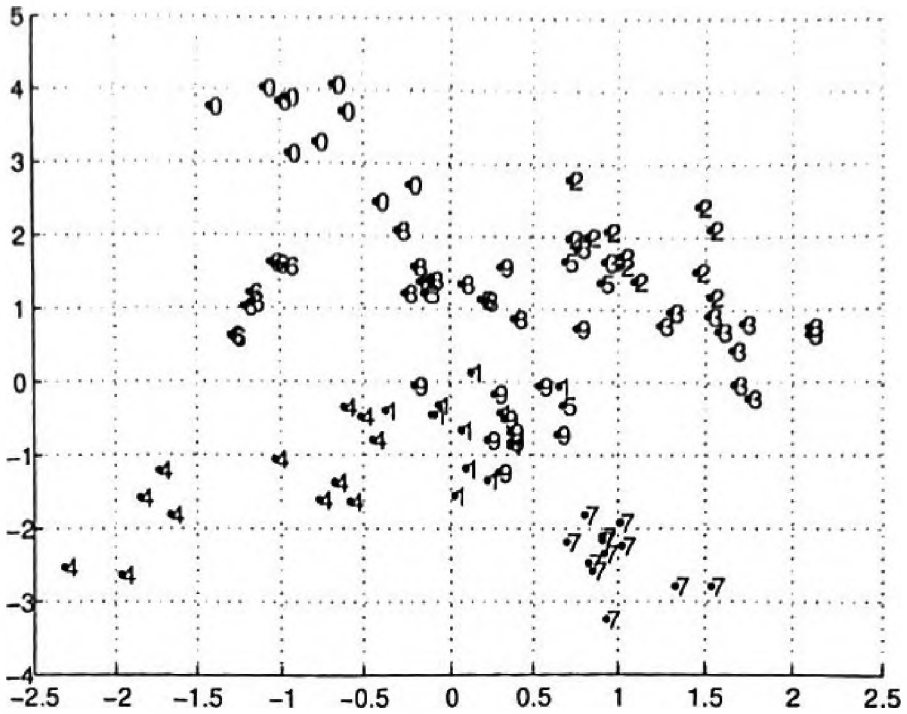
$$\frac{w^T (m_1 - m_2)}{w^T \mathbf{S}_W w} \left(2(m_1 - m_2) - \frac{w^T (m_1 - m_2)}{w^T \mathbf{S}_W w} \mathbf{S}_W w \right) = 0$$

$w^T (m_1 - m_2) / w^T \mathbf{S}_W w$ bir sayıldır ve

$$w = c \mathbf{S}_W^{-1} (m_1 - m_2) \tag{6.38}$$

olarak yazılabilir. c bir sabittir ve bizce önemli olan yöneyin boyu değil yönü olduğu için $c = 1$ alarak w yöneyini bulabiliriz.

$p(x|C_i) \sim \mathcal{N}(\mu_i, \mathbf{\Sigma})$ olduğunda $w = \mathbf{\Sigma}^{-1}(\mu_1 - \mu_2)$ biçiminde doğrusal bir sınıflandırıcı oluştuğunu bildiğimiz için Fisher doğrusal ayırtacının sınıflar normal dağıldığında en iyi sınıflandırıcı olduğunu söyleyebiliriz. Bu varsayımla w_0 değeri



Şekil 6.8: Optdigits verisinin doğrusal ayırtaç çözümlemesiyle bulunan ilk iki boyutlu uzayda gösterimi. Bununla şekil 6.3'ü karşılaştırırsak, beklendiği gibi, doğrusal ayırtaç çözümlemesinin gözetimli olduğu için temel bileşen çözümlemesine göre sınıfları daha iyi ayrıştırdığını görürüz. Bu iki boyutlu uzayda bile (ki toplam 9 boyut var), farklı sınıfların örneklerinin farklı öbekler oluşturduğunu görebiliyoruz.

de hesaplanabilir. Ama Fisher doğrusal ayırtacı sınıflar normal dağılmasa da kullanılabilir; örnekleri d boyutlu uzaydan bir boyuta izdüşürdüğükten sonra herhangi bir sınıflandırıcı kullanılabilir.

$K > 2$ sınıf olduğunda, $\mathbf{W}$ dizeyi $d \times k$ olur, ve

$$z = \mathbf{W}^T x \tag{6.39}$$

k boyutlu bir yöneydir. $\mathcal{C}_i$ için sınıf içi saçılım dizeyini yazalım:

$$\mathbf{S}_i = \sum_t r_i^t (x^t - m_i)(x^t - m_i)^T \tag{6.40}$$

burada, $x^t \in \mathcal{C}_i$ ise $r_i^t = 1$ değilse $r_i^t = 0$ etiketli olduğunu varsayıyoruz. Toplam sınıf içi saçılımını yazalım:

$$\mathbf{S}_W = \sum_{i=1}^{K} \mathbf{S}_i \tag{6.41}$$

$K > 2$ sınıf olduğunda merkezlerin saçılımı genel merkez

$$m = \frac{1}{K} \sum_{i=1}^{K} m_i \tag{6.42}$$

çevresinde ne kadar saçıldıklarına bağlıdır. Sınıflar arası saçılım dizeyini tanımlayalım:

$$\mathbf{S}_B = \sum_{i=1}^{K} N_i (m_i - m)(m_i - m)^T \tag{6.43}$$

öyle ki $N_i = \sum_t r_i^t$. İzdüşüm sonrası sınıflar arası saçılım dizeyi $\mathbf{W}^T\mathbf{S}_B\mathbf{W}$ ve sınıf içi saçılım dizeyi $\mathbf{W}^T\mathbf{S}_W\mathbf{W}$ olur. Bunların ikisi de $k \times k$ dizeylerdir. İlk saçılımın büyük olmasını, yani izdüşüm sonrası yeni k boyutlu uzayda sınıf merkezlerinin birbirinden olabildiğince uzak olmasını isteriz. İkinci saçılımınsa küçük olmasını, yani izdüşüm sonrası aynı sınıftan örneklerin merkezlerine olabildiğince yakın olmasını isteriz. Bir saçılım (ya da eşdeğişinti) dizeyinin yayılımı için bir ölçüt determinanttır çünkü determinant, özdeğerlerin çarpımıdır ve her özdeğer kendi özyöneyi (bileşeni) üzerindeki değişintiyi verir. Dolayısıyla

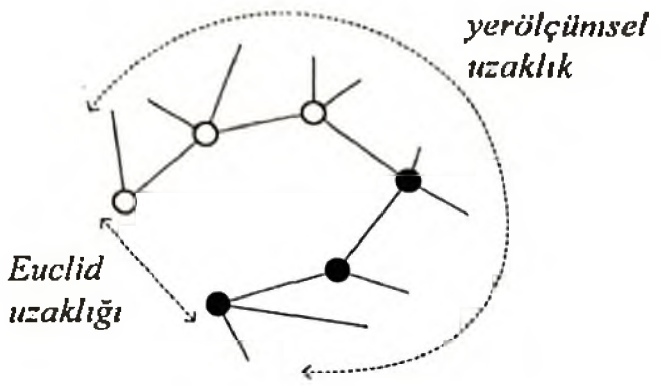
$$J(\mathbf{W}) = \frac{|\mathbf{W}^T\mathbf{S}_B\mathbf{W}|}{|\mathbf{W}^T\mathbf{S}_W\mathbf{W}|} \tag{6.44}$$

işlevini en büyük yapan $\mathbf{W}$ dizeyini arıyoruz. $\mathbf{S}_W^{-1}\mathbf{S}_B$ dizeyinin en büyük özyöneyleri bize çözümü verir. $\mathbf{S}_B$, kertesi 1 olan K tane $(m_i - m)(m_i - m)^T$ dizeyinin toplamıdır ve yalnızca $K-1$ tanesi bağımsızdır, dolayısıyla, $\mathbf{S}_B$ dizeyinin kertesi en fazla $K-1$ olur ve bu yüzden $k = K-1$ alınır. Yani, $(K-1)$ boyutlu bir uzaya eşleme yaparız (bkz. şekil 6.8). Her ne kadar boyut azaltırken sınıf ayrılabilirliğini kıstas alsak da, ardından bu yeni uzayda ayırtaçları belirlemek için herhangi bir sınıflandırıcı kullanabiliriz.

Doğrusal ayırtaç çözümlemesini uygularken $\mathbf{S}_W$ dizeyinin tersini alıyoruz. Eğer bunu yapamıyorsak önce TBÇ kullanarak tekillikten kurtulur, ardından sonuca DAÇ uygularız. Bu durumda, TBÇ yaparken boyut sayısını bilgi kaybına neden olacak kadar çok azaltmadığına dikkat etmemiz gerekir.

6.7 Eşölçümsel Öznitelik Eşleme

6.3. bölümde işlediğimiz temel bileşen çözümlemesi, verinin doğrusal bir altuzayda bulunduğunu varsayar. Ama birçok uygulamada bu varsayım geçerli olmayabilir. Yüz tanımayı ele alalım. Yüzlerin iki boyutta, örneğin, 100×100 imgeler olduğunu varsayarsak, her yüz 10.000 boyutlu uzayda bir nokta olacaktır. Şimdi bir kişinin başını sağdan sola yavaşça çevirdiğini düşünelim; bu sırada kaydedilen imgeler bu 10.000 boyutlu uzayda bir gezinge izleyecektir ama bu gezinge doğrusal değildir. İnsanlar yüzlerini çevirirlerken yüz resimlerinin 10.000 boyuttaki uzayda oluşturduğu gezingeler bir altuzay tanımlar ve istediğimiz bu altuzayı modellemektir. İki yüz resmi arasındaki benzerlik imge noktalarındaki değerlerin farklarının toplamı olarak yazılamaz ve bu yüzden Euclid uzaklığı iyi bir benzerlik ölçüsü değildir. Farklı iki kişinin aynı açıdan çekilmiş imgeleri arasındaki Euclid uzaklığı, aynı kişinin farklı iki açıdan çekilmiş imgeleri arasındaki Euclid uzaklığından daha küçük olabilir. İstediğimiz bu değildir. Önemli olması gereken *yerölçümsel uzaklık* denilen,



Şekil 6.9: Yerölçümsel uzaklık altuzay boyunca hesaplanır; Euclid uzaklığıysa bu bilgiyi kullanmaz. Çok boyutlu ölçeklemeden sonra farklı sınıflardaki bu iki örnek, asıl uzayda yakın bile olsalar yeni uzayda uzak konumlara eşlenecektir.

gezingenin tanımladığı altuzay boyunca olan uzaklıktır. *Eşölçümsel öznitelik eşleme* (Tenenbaum, de Silva, ve Langford 2000) bu uzaklığı kestirip çok boyutlu ölçekleme (bölüm 6.5) ile boyut azaltır.

Verideki nokta çiftleri arasındaki yerölçümsel uzaklıklar kullanılır. Girdi uzayında yakın olan komşu noktalar arasında Euclid uzaklığı kullanılabilir; bakış açısındaki çok küçük değişiklikler için altuzayın yerel olarak doğrusal olduğu varsayılabilir. Uzak noktalar için yerölçümsel uzaklık ise altuzayda üzerinden geçilen noktalar arasındaki yerel uzaklıkların toplamı olarak kestirilir. Bunu yaparken tanımladığımız çizgenin düğümleri N veri noktasına, ayrıtlar da komşuluklara (biri ötekinin n komşusundan biri, ya da aralarındaki Euclid uzaklığı ϵ değerinden küçük) karşılık gelir. İki nokta arasındaki yerölçümsel uzaklık, karşılık gelen düğümlerin arasında çizge içindeki en kısa yolun uzunluğu olarak hesaplanır. Yakın olmayan iki nokta için çizge üstünde ara düğümler üzerinden sıçrayarak gidileceğinden uzaklık, altuzay üzerindeki yerel Euclid uzaklıklarının toplamı olarak kestirilen uzaklık olacaktır (bkz. şekil 6.9).

r ve s düğümleri arasında bir ayrıtla bağ olması için ya $\|\boldsymbol{x}^r - \boldsymbol{x}^s\| < \epsilon$ (çizgenin bağlı olması koşuluyla), ya da $\boldsymbol{x}^s$ örneğinin $\boldsymbol{x}^r$ örneğinin n komşusundan biri olması sağlanmalıdır. Bu durumda ayrıt uzunluğu $\|\boldsymbol{x}^r - \boldsymbol{x}^s\|$ olur. Herhangi iki r ve s düğümü arasında çizge üstündeki en kısa yol uzunluğunu d_{rs} ile gösterelim. Bunlar üzerinde çok boyutlu ölçekleme kullanarak boyut sayısını k değerine (açıklanan değişinti oranına bakarak) indiririz. Bunun sonucu olarak yerölçümsel uzayda uzak olan iki r ve s örneği, asıl d boyutlu girdi uzayında Euclid uzaklığına göre yakın bile olsalar, yeni k boyutlu uzayda uzak olurlar.

Daha çok örnekle çizge uzaklıklarının daha iyi bir yaklaştırma sağlayacağı açıktır, ama bu hesap ve bellek karmaşıklığının da artırır. Bir ödünleşim vardır; zaman önemliyse algoritmayı daha hızlı çalıştırabilmek için noktaların bir altkümesini seçip bir küme "nirengi" noktasıyla çalışılabilir. ϵ değerinin iyi seçilmesi gerekir; çok küçük seçilirse çizge bağlı olmaz ve birden fazla bileşen oluşur, çok büyük seçilirse "kestirme" ayrıtlar ortaya çıkar ve bunlar düşük boyuta indirmeyi bozar (Balasubrama-

nian ve ötekiler 2002).

Çok boyutlu ölçekleme kullanıldığı için N noktayı düşük boyutlu bir uzaya yerleştirsek de yeni bir noktayı bu altuzaya eşleyebilecek genel bir eşleme işlevi öğrenmeyiz; bu örneği eldeki veriye katıp algoritmayı $N+1$ örnekle yeniden çalıştırmak gerekir.

6.8 Yerel Doğrusal Gömme

Yerel doğrusal gömme, küresel doğrusal olmayan yapıyı yerel doğrusal modellerle açıklar (Roweis ve Saul 2000). Verinin dağıldığı altuzayın yerel parçalarla yaklaşık olarak gösterilebileceği, ve yeterince veri varsa her noktanın, komşularının (komşuluk tanımı için yine ya bir komşu sayısı, n, ya da uzaklık eşiği, ϵ, kullanılır) doğrusal, ağırlıklı bir toplamı şeklinde yazılabileceği düşüncesine dayanır. Asıl uzayda x^r örneğinini $x^s_{(r)}$ ile gösterdiğimiz komşularının ağırlıklı ($\mathbf{W}_{rs}$) toplamı olarak yazabildiğimizi varsayalım. Toplam geriçatma hatasını

$$\mathcal{E}^w(\mathbf{W}|\mathcal{X}) = \sum_r \|x^r - \sum_s \mathbf{W}_{rs} x^s_{(r)}\|^2 \tag{6.45}$$

en aza indiren ağırlıkları, en küçük kareler yöntemiyle, $\mathbf{W}_{rr} = 0, \forall r$ ve $\sum_s \mathbf{W}_{rs} = 1$ koşullarını da göz önüne alarak bulabiliriz.

Yerel doğrusal gömmede düşünce, $\mathbf{W}_{rs}$ geriçatma ağırlıklarının verinin özünde bulunan geometrik özellikleri yansıttığı ve bunların hem asıl uzayda, hem de örnekleri eşlediğimiz yeni uzayın yerel parçalarında geçerli olması gerektiğidir (bkz. şekil 6.10). Yerel doğrusal gömmenin ikinci adımında, bu kez $\mathbf{W}_{rs}$ ağırlıklarını sabit tutup yeni z^r yöneylerini, ağırlıkların tanımladığı noktalar arası kısıtlara uyacak biçimde çatarız:

$$\mathcal{E}^z(\mathcal{Z}|\mathbf{W}) = \sum_r \|z^r - \sum_s \mathbf{W}_{rs} z^s\|^2 \tag{6.46}$$

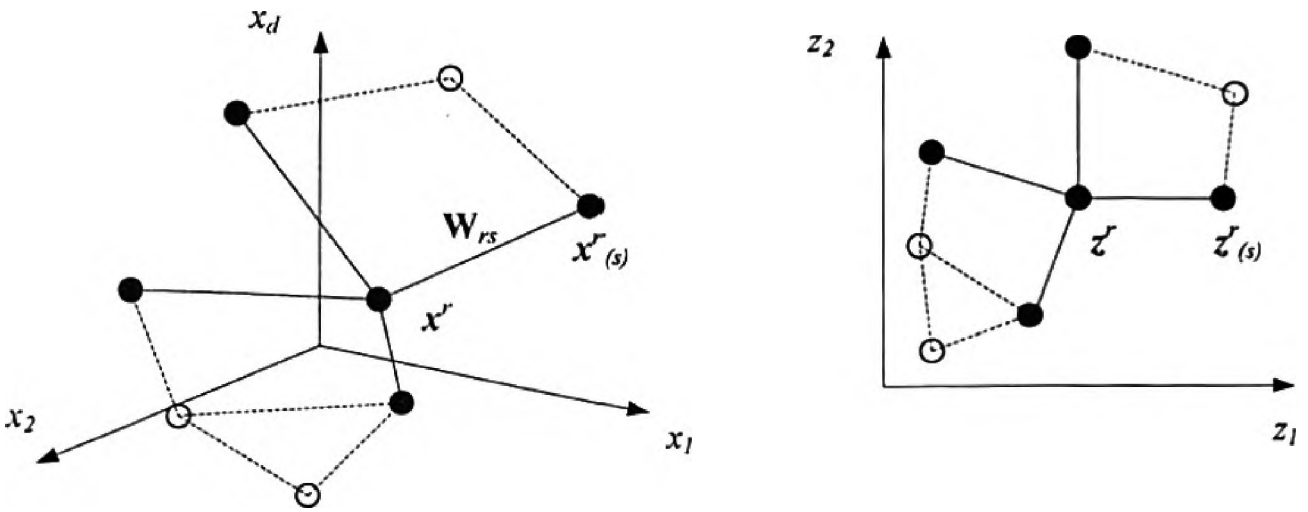
d boyutlu asıl uzayda yakın olan örnekler yakın kalmalı ve $k < d$ boyutlu yeni uzayda benzer biçimde yerleşmelidir. Denklem 6.46'yı yeniden yazalım:

$$\mathcal{E}^z(\mathcal{Z}|\mathbf{W}) = \sum_{r,s} \mathbf{M}_{rs} (z^r)^T z_s \tag{6.47}$$

öyle ki

$$\mathbf{M}_{rs} = \delta_{rs} - \mathbf{W}_{rs} - \mathbf{W}_{sr} + \sum_i \mathbf{W}_{ir} \mathbf{W}_{is} \tag{6.48}$$

$\mathbf{M}$, seyrek (bir noktanın az sayıda, $n \ll N$, komşusu vardır), bakışımlı, ve yarı kesin bir dizeydir. Öbür boyut azaltma yöntemlerinde olduğu gibi, verinin sıfır merkezli ($E[z] = 0$) ve yeni boyutların ilintisiz ve birim uzunlukta ($\mathrm{Cov}(z) = \mathbf{I}$) olması koşullarını koyuyoruz. Bu iki kısıtla denklem 6.47'nin çözümü, en küçük özdeğerli $k+1$ özyöneydir; en küçüğünü atıp öteki k özyöneyi yeni eksen değerleri olarak alırız.



Şekil 6.10: Yerel doğrusal gömme, önce asıl x uzayında noktalar arasındaki kısıtları öğrenir ve ardından noktaları yeni z uzayında bu kısıtlara uygun biçimde yerleştirir. Kısıtlar, noktaların en yakın komşuları (sürekli çizgiler) üzerinden hesaplanır ama ikinci dereceden komşularına (kesikli çizgiler) da yayılır.

İki boyutlu uzayda konumumuzu belirlemek için üç nokta gerekir; benzer biçimde n komşu, $n-1$ boyutlu bir uzayı kapsar ve YDG, boyutu en fazla $k \leq n-1$ değerine düşürebilir. İyi bir gömme elde edebilmek için k ve n arasında biraz pay bırakmak gerekir (Saul ve Roweis 2003). Eğer n (ya da ϵ) küçükse, her noktayı komşularına bağlayarak elde ettiğimiz çizge bağlı olmayabilir ve YDG'yi bileşenler üzerinde ayrı ayrı uygulayıp her biri için farklı altuzaylar bulmak gerekebilir. Öteki yandan, eğer n (ya da ϵ) büyükse, bazı komşular yerel doğrusallık varsayımının geçerli olabileceği uzaklıktan daha ötede olup gömmeyi kötüleştirir. Girdi uzayının farklı yerlerinde, önsel bilgimizie dayanarak farklı n (ya da ϵ) değerleri kullanabiliriz ama bunun tam olarak nasıl yapılacağı bir araştırma konusudur (Saul ve Roweis 2003).

Eşölçümsel öznitelik eşlemede olduğu gibi YDG çözümü de N nokta için bulunan yeni yöneylerdir ama bir eşleme öğrenmediğimiz için yeni bir x' örneği için karşılık gelen z' yöneyini hesaplayamayız. Bunun iki çözümü vardır:

1. Aynı düşünceyle x' yöneyinin asıl d boyutlu uzaydaki n komşusunu bulur ve geriçatma hatasını en aza indiren ağırlıkları öğrenir

$$\mathcal{E}^w(w|\mathcal{X}) = \|x' - \sum_s w_s x^s\|^2 \tag{6.49}$$

ve sonra bunları z' yöneyini yeni k boyutlu uzayda çatmak için kullanabiliriz:

$$z' = \sum_s w_s z^s \tag{6.50}$$

Bu yaklaşımı, eşölçümsel öznitelik eşleme (ya da ÇBÖ) yöntemlerinde de yeni

noktaları eşlemek için kullanabiliriz. Güçlüğüyse tüm $\{x^t, z^t\}_{t=1}^N$ kümesini saklamak gerekmesidir.

2. Öğrenme kümesinden hesapladığımız $\mathcal{X} = \{x^t, z^t\}_{t=1}^N$ çiftlerini bir $g(x^t|\theta) \in \Re^k$ bağlanım modelini—örneğin, çok katmanlı bir algılayıcı (bölüm 11)—eğitmek için kullanabiliriz. Model parametreleri olan θ bağlanım hatasını en aza indirecek biçimde bulunur:

$$\mathcal{E}(\theta|\mathcal{X}) = \sum_t \|z^t - g(x^t|\theta)\|^2 \tag{6.51}$$

Öğrenme bittikten sonra $z' = g(x'|\theta)$ hesaplanabilir. Model $g(\cdot)$ eşlemeyi öğrenebilecek biçimde dikkatli seçilmelidir. Tek çözüm olmayabilir ve bu yüzden en küçük hatalı çözümü ararken ilk değerleri uygun seçme, yerel en küçük çözüme yakınsama, yakınsama hızı, vs. gibi tüm dertler burada da kendini gösterebilir.

Hem eşölçümsel öznitelik eşleme, hem de yerel doğrusal gömmede, yerel bilgi komşular üzerinden çevreye yayılarak küresel çözüm aranır. Yerölçümsel uzaklık yerel uzaklıkların toplamıdır; yerel doğrusal gömmedeyse en iyi z^t değerlerini bulurken tüm yerel $\mathbf{W}_{rs}$ değerleri kullanılır. Diyelim ki, a ile b ve b ile c komşu olsun; a ile c komşu olmasalar da, aralarında ya çizge üzerinden $d_{ac} = d_{ab} + d_{bc}$ biçiminde, ya da $\mathbf{W}_{ab}$ ve $\mathbf{W}_{bc}$ ağırlıkları üzerinden bağımlılık vardır. Her iki algoritma da doğrusal olmayan küresel düzeni, kısmen örtüşen yerel doğrusal kısıtları birleştirerek bulur.

6.9 Kısa Kısa

Öznitelik seçme algoritmalarının bir taraması (Devijer ve Kittler 1982)'de bulunabilir. Öznitelik altküme seçimi algoritmalarına seçim yordamının modeli sarıp bir altprogram gibi kullanmasından dolayı, *dürümcü* olarak da bilinir (Kohavi ve John 1997). Bağlanım için altküme seçimi (Miller 1990)'da ele alınmıştır. Ele aldığımız ileriye ve geriye doğru seçim yöntemleri yerel arama yapar; Fukunaga ve Narendra (1977) bir dallan ve sınırla yöntemi önermiştir. Arama uzayında daha geniş bir inceleme yapmak için, ama çok daha fazla hesaplama yaparak, tavlama benzetimi ya da genetik algoritma gibi olasılıksal yöntemler de kullanılabilir.

Öznitelik seçmek için herhangi bir gözetimli model kullanmadan, bir önişleme aşaması olarak buluşsal ölçütlerle özniteliklerin "ilgililiklerini" hesaplayan *süzme* algoritmaları da vardır. Örneğin bir sınıflandırma uygulamasında, her adımda sıfırdan eğitme ve sınama yerine, yeni uzayın sınıfları birbirinden ayrıştırma kalitesini ölçmek için, doğrusal ayırtaç çözümlemesindeki gibi bir ayrılabilirlik ölçütü kullanılabilir (McLachlan 1992). Hesaplama masrafının gittikçe azalıyor olmasından dolayı modelin işin içinde tutulması en iyisidir çünkü süzgecin kullandığı buluşsal ayrılabilirlik kıstasıyla bulunan öznitelikleri kullanan model aynı varsayımlara göre çalışmayabilir;

hiçbir buluşsal ölçüt, seçilen özniteliklerin iyiliği hakkında bir geçerleme kümesi üzerindeki başarı kadar iyi bir gösterge değildir. Öznitelik seçimi yöntemlerinin bir taraması (Guyon ve Elisseeff 2003)'te verilmiştir.

Altküme seçimi, ayrık değişkenleri kullanılabilse de izdüşüm yöntemleri sayısal girdilerle çalıştığından ayrık değişkenlerin önce 0/1 yapay değişkenlerine çevrilmesi gerekir. Özyöney ve özdeğerleri bulmak oldukça kolaydır; örnek bir program (Press ve ötekiler, 1992)'de verilmiştir. Etken çözümlemesi, değişik zekâ testi puanları arasındaki ilintiyi açıklayan tek bir zekâ etkeni oluşturmak amacıyla İngiliz psikolog Charles Spearman tarafından ilk önerilmiştir; g olarak adlandırılan böyle tek bir etkenin varlığı tartışma konusudur. ÇBÖ hakkında daha çok bilgi (Cox ve Cox 1994)'te bulunabilir.

Bu bölümde ele aldığımız izdüşüm yöntemleri, tüm örneklemin kullanıldığı toplu yordamlardır. Mao ve Jain (1995), TBÇ ve DAÇ için, örneklerin tek tek verildiği ve yeni örnekler geldikçe küçük güncellemelerin yapıldığı çevrimiçi yöntemleri önerir. Doğrusal olmayan izdüşüm yapmanın bir yolu da Sammon eşlemesindeki kestiricinin doğrusal olmayan bir işlevle, örneğin, çok katmanlı algılayıcılarla (bölüm 11.11), gerçeklenmesidir (Mao ve Jain 1995). Doğrusal olmayan etken çözümlemesi de gerçeklenebilir, ama daha zordur. Hem uygun bir doğrusal olmayan model belirlemek zordur, hem de model parametrelerini bulmak için karmaşık iyileme ve yakınsama yöntemleri kullanmak gerekir.

Yüksek dereceden terimlerin ek girdi olduğu doğrusal bir modelle çokterimli bağlanım gerçekleyebildiğimiz gibi (bölüm 5.8), doğrusal olmayan boyut azaltmak için bir yol da doğrusal olmayan taban işlevleri kullanıp yeni bir uzaya geçmek ve orada doğrusal bir boyut azaltma yöntem kullanmaktır. Çekirdek yöntemlerini ele alacağımız 13. bölümde bunun nasıl yapılacağını göreceğiz.

Öznitelik çıkarmayla karar verme arasında bir ödünleşim vardır. Öznitelik çıkarıcı iyi olduğunda sınıflandırıcının (ya da bağlanım modelinin) işi çok basitleşir; en iyi durumda sınıf etiketi asıl özniteliklerden yeni bir öznitelik olarak çıkarılacaktır. Öbür yandan, sınıflandırıcı yeteri kadar iyiyse öznitelik çıkarmaya gerek kalmaz ve sınıflandırıcı, gereken öznitelik seçme ya da birleştirme işlemini kendi içinde yapar. Gerçek yaşam bu iki kusursuz dünyanın ortasındadır.

Bazı öğrenme algoritmaları, sınırlı bir biçimde de olsa, kendi içlerinde boyut azaltır. Karar ağacı yöntemleri (bölüm 9), ağacı büyütürken öznitelik seçer, çok katmanlı algılayıcılar (bölüm 11) gizli düğümlerdine doğrusal olmayan öznitelik çıkarımı yapar. Öznitelik çıkarımını bu biçimde, ardından gelen sınıflandırma/bağlanım modelinin içine gömme üzerinde sıkça çalışılan konulardandır.

6.10 Alıştırmalar

1. Sınıfların normal dağıldığını varsayarsak, altküme seçiminde bir değişken eklendiği ya da çıkartıldığında yeni ayırtacı nasıl hızla bulabiliriz? Örneğin, $\mathbf{S}_{yeni}^{-1}$, $\mathbf{S}_{eski}^{-1}$'den nasıl hesaplanabilir?

2. UCI veri tabanından Optdigits'i kullanarak TBÇ yöntemini gerçekleyin. Farklı sayıda

özyönev kullanarak rakam görüntülerini geriçatın ve geriçatma hatalasını hesaplayın (denklem 6.12).

3. Karayolu uzaklıklarını ÇBÖ'ye girdi olarak verip ülkenizin haritasını çizdirin.
4. Sammon eşlemesinde, eşleme doğrusal ($g(\boldsymbol{x}|\mathbf{W}) = \mathbf{W}^T\boldsymbol{x}$) olduğunda Sammon gerilimini en aza indiren $\mathbf{W}$ nasıl hesaplanabilir?
5. Alıştırma 3'ü bir kez daha, bu kez eşölçümsel öznitelik eşleme kullanarak ve yalnız aralarında doğrudan (başka bir kentten geçmeyen) yol olan kentleri birleştirerek çözün.
6. Eşölçümsel öznitelik eşlemede komşu noktalar arasında Euclid yerine Mahalanobis uzaklığını da kullanabiliriz. Bunun olası getiri ve götürüleri nelerdir?
7. İki boyutta öyle iki sınıf çizin ki (a) TBÇ ve DAÇ aynı yönü bulsun, ve (b) TBÇ ve DAÇ tamamen farklı yönler bulsun.
8. Elimizde nesneler arasında ikili uzaklıklar olduğunda çok boyutlu ölçekleme kullanılabilir. Bir benzerlik ölçütü olduğu sürece nesneleri yöneyler olarak göstermemize gerek yoktur. Bir örnek verebilir misiniz?
9. Aynı sınıftaki örnekleri yeni uzayda yakın yerlere eşleyebilmek için eşölçümsel öznitelik eşleme ve YDG'ye sınıf bilgisini nasıl kullandırabiliriz?
10. Etken çözümlemesinde bazı etkenleri zaten biliyorsak kalanları nasıl bulabiliriz?
11. (Doğrusal olması gerekmeyen) saklı etkenlerin bulunduğu ve etken çözümlemesinin iyi çalışmasını beklediğiniz bir uygulama ne olabilir?

6.11 Kaynaklar

Balasubramanian, M., E. L. Schwartz, J. B. Tenenbaum, V. de Silva, J. C. Langford. 2002. "The Isomap Algorithm and Topological Stability." *Science* 295: 7.

Chatfield, C., A. J. Collins. 1980. *Introduction to Multivariate Analysis.* London: Chapman and Hall.

Cox, T. F., M. A. A. Cox. 1994. *Multidimensional Scaling.* London: Chapman and Hall.

Devijer, P. A., J. Kittler. 1982. *Pattern Recognition: A Statistical Approach.* New York: Prentice-Hall.

Flury, B. 1988. *Common Principal Components and Related Multivariate Models.* New York: Wiley.

Fukunaga, K., P. M. Narendra. 1977. "A Branch and Bound Algorithm for Feature Subset Selection." *IEEE Transactions on Computers* C-26: 917–922.

Guyon, I., A. Elisseeff. 2003. "An Introduction to Variable and Feature Selection." *Journal of Machine Learning Research* 3: 1157–1182.

Hastie, T. J., R. J. Tibshirani, A. Buja. 1994. "Flexible Discriminant Analysis by Optimal Scoring." *Journal of the American Statistical Association* 89: 1255–1270.

Kohavi, R., G. John. 1997. "Wrappers for Feature Subset Selection." *Artificial Intelligence* 97: 273–324.

Mao, J., A. K. Jain. 1995. "Artificial Neural Networks for Feature Extraction and Multivariate Data Projection." *IEEE Transactions on Neural Networks* 6: 296–317.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition.* New York: Wiley.

Miller, A. J. 1990. *Subset Selection in Regression.* London: Chapman and Hall.

Press, W. H., B. P. Flannery, S. A. Teukolsky, W. T. Vetterling. 1992. *Numerical Recipes in C.* Cambridge, UK: Cambridge University Press.

Pudil, P., J. Novovičová, J. Kittler. 1994. "Floating Search Methods in Feature Selection." *Pattern Recognition Letters* 15: 1119–1125.

Rencher, A. C. 1995. *Methods of Multivariate Analysis.* New York: Wiley.

Roweis, S. T., L. K. Saul. 2000. "Nonlinear Dimensionality Reduction by Locally Linear Embedding." *Science* 290: 2323–2326.

Saul, K. K., S. T. Roweis. 2003. "Think Globally, Fit Locally: Unsupervised Learning of Low Dimensional Manifolds." *Journal of Machine Learning Research* 4: 119–155.

Tenenbaum, J. B., V. de Silva, J. C. Langford. 2000. "A Global Geometric Framework for Nonlinear Dimensionality Reduction." *Science* 290: 2319–2323.

Tipping, M. E., C. M. Bishop. 1999. "Probabilistic Principal Components Analysis." *Journal of the Royal Statistical Society Series B* 61: 611–622.

Turk, M., A. Pentland. 1991. "Eigenfaces for Recognition." *Journal of Cognitive Neuroscience* 3: 71–86.

Webb, A. 1999. *Statistical Pattern Recognition.* London: Arnold.

Bölüm 7

Öbekleme

Önceki bölümlerde örneklemin bilinen bir dağılımdan geldiğini varsaydık. Böyle bir varsayımın geçerli olmadığı durumlarda dağılımların karışımını kullanabilir ve öbekleme yöntemleriyle karışım parametrelerini veriden kestirebiliriz. Bu bölümde, olasılık tabanlı modeller yanında, yöney nicemleme ve öbek ağaçlarını da tartışacağız.

7.1 Giriş

4 ve 5. bölümlerde dağılım kestiriminde, $\mathcal{X}$ örnekleminin belirli bir dağılımdan—örneğin, normal—geldiğini varsayan yaklaşımı tartıştık. Sınıflandırmada bu, $p(\boldsymbol{x}|\mathcal{C}_i)$ sınıf dağılımları için varsayım yapmaya karşılık gelir. Eğer belirli bir dağılım ailesi varsayabilirsek problem o dağılımın parametrelerinin kestirimine indirgenir ki bunlar yeterli istatistiklerdir ve örneğin, normal dağılım için ortalama ve eşdeğişintidir.

Dağılıma bağlı yaklaşımlar sıkça kullanılsa da böyle bir varsayım yapmak yanlılığa neden olabilir ve daha esnek modellere gerek duyabiliriz. Örneğin normal dağılım, bir sınıftan örneklerin d boyutlu uzayda tek bir merkez çevresinde öbeklendiklerini varsayar ki 5. bölümde gördüğümüz gibi bu öbeğin merkezi ve biçimi, ortalama yöneyi ve eşdeğişinti dizeyi tarafından tanımlanır.

Birçok uygulamada örneklem tek bir öbekten değil birkaç öbekten gelebilir. Elle yazılmış sayıları düşünelim: 7 rakamını Amerikalılar ortasında yatay çizgi olmadan yazar, Avrupadaysa bu çizgi vardır. Eğer örnek kümemizde bu iki yazı biçiminden de örnekler varsa bu ikisinin ayrı ayrı, ortalamasını alıp tek bir modele indirgemeden gösterilebilmesi gerekir. Eğer her yazım biçimi bir Gauss dağılımıyla modellenirse 7 rakamının sınıfı, bu iki dağılımın bir *karışımı* olarak gösterilir.

Benzer bir örnek konuşma tanıma için de verilebilir. Aynı sözcük, farklı aksan, şive, cinsiyet, yaş ya da başka nedenlerden dolayı farklı insanlar tarafından farklı söylenir. Eğer tek, evrensel bir asıl örnek yoksa istatistiksel doğru olabilmek için tüm farklılıkların gösterilebilmesi gerekir.

Bu yaklaşıma, *dağılıma yarı bağlı kestirim* diyoruz, çünkü öbeklerin sayısında bir kısıtınız olmasa da her öbek için bir dağılım varsayıyoruz. Veride hiçbir yapı olmayıp karışım modeli bile uygulanamadığında kullanılabilecek olan *dağılımdan bağımsız* yaklaşımı 8. bölümde konuşacağız. Bu bölümde yalnızca olasılık dağılım kestirimini görecek, gözetimli öğrenmeyi 12. bölüme erteleyeceğiz.

7.2 Karışım Dağılımı

Karışım dağılımını tanımlayalım:

$$p(x) = \sum_{i=1}^{k} p(x|\mathcal{G}_i)P(\mathcal{G}_i) \tag{7.1}$$

Olasılık, $\mathcal{G}_i$ ile gösterilen *öbek* ya da *karışım bileşenlerinin* ağırlıklı ortalamasıdır. $p(x|\mathcal{G}_i)$, *bileşen dağılımlarını* ve $P(\mathcal{G}_i)$, *karışım oranlarını* gösterir. Bileşen sayısı olan k bir üstparametredir ve önceden tanımlanması gerekir. Bir örneklem ve k değeri verildiğinde öğrenme, bileşen dağılımlarının ve karışım oranlarının kestirilmesine karşılık gelir. Bileşen dağılımlarını belirli bir dağılım ailesinden geldiğini varsayıp yalnızca o dağılımın parametrelerini kestiririz. Örneğin, eğer çok boyutlu Gauss dağılırlarsa, $p(x|\mathcal{G}_i) \sim \mathcal{N}(\mu_i, \Sigma_i)$, $\mathcal{X} = \{x^t\}_t$ örnekleminden kestirilecek parametreler, $\Phi = \{P(\mathcal{G}_i), \mu_i, \Sigma_i\}_{i=1}^k$ olur.

Dağılıma bağlı sınıflandırmayı da bir karışım modeli olarak düşünebiliriz. C_i sınıfları $\mathcal{G}_i$ bileşenlerine, $p(x|\mathcal{G}_i)$ bileşen dağılımları $p(x|C_i)$ sınıf dağılımlarına ve $P(\mathcal{G}_i)$ karışım oranları $P(C_i)$ önsel olasılıklara karşılık gelir:

$$p(x) = \sum_{i=1}^{K} p(x|C_i)P(C_i)$$

Bu *gözetimli* durumda bileşen sayısı, sınıf sayısına karşılık gelir ve değeri bilinir. Her örneğin hangi sınıftan (bileşenden) olduğu da örneklem de verildiği için bileşen parametreleri kolayca hesaplanabilir. 5. bölümde gördüğümüz gibi, etiket r_i^t değerinin $x^t \in C_i$ ise 1 değilse 0 olduğu $\mathcal{X} = \{x^t, r^t\}_{t=1}^N$ örneklemi verildiğinde parametreler olabilirliği büyütecek biçimde hesaplanabilir. Her sınıf Gauss dağıldığında elimizde bir Gauss karışımı olur ve parametreleri

$$\begin{aligned} \hat{P}(C_i) &= \frac{\sum_t r_i^t}{N} \\ m_i &= \frac{\sum_t r_i^t x^t}{\sum_t r_i^t} \\ \mathbf{S}_i &= \frac{\sum_t r_i^t (x^t - m_i)(x^t - m_i)^T}{\sum_t r_i^t} \end{aligned} \tag{7.2}$$

olarak veriden kestirilir.

Bu bölümdeki fark, örneklemin $\mathcal{X} = \{\boldsymbol{x}^t\}_t$ olması ve *gözetimsiz öğrenme* yapmamızdır. Elimizde yalnızca $\boldsymbol{x}^t$ girdileri vardır, r^t etiketleri yoktur; yani $\boldsymbol{x}^t$ girdisinin hangi bileşenden geldiğini bilmeyiz. İkisini birlikte kestirmemiz gerekir: Önce r_i^t etiketlerini, yani hangi girdinin hangi bileşenden geldiğini, ve ardından bu etiketleri kullanarak bileşen parametrelerini kestiririz. Bu amaç için önce en basit k merkezli öbekleme algoritmasını ve ardından onun olasılıksal çatısını tanımlayan *Beklenti Büyütme* algoritmasını göreceğiz.

7.3 k Merkezli Öbekleme

Her noktası 24 bitle gösterilen ve dolayısıyla 16 milyon farklı renk içerebilen bir imgemiz olsun. Ama ekranımızin her nokta için 8 bit kullandığını ve dolayısıyla renk çizelgemizin en fazla 256 renk içerebildiğini varsayalım. Bu durumda 16 milyon rengi görüntüde en az bozulma yaratacak biçimde 256 renge indirmemiz gerekir. Buna *renk nicemlemeye* denir. Genelde, daha yüksek (ya da sonsuz) çözünürlüklü bir uzaydan daha düşük (ya da kesikli) bir uzaya eşlemeye *nicemleme* denir.

Tabii ki renk uzayını eşit aralıklarla 256 parçaya bölebiliriz, ama bu hiç kullanılmayan renkler için yer ayırıp renk çizelgemizin bazı değerlerini boş yere doldurmamıza, ya da çok kullanılan renklerin tonları arasındaki farkları kaybetmemize neden olacaktır. Örneğin, elimizdeki bir deniz manzarasıysa imgede mavinin farklı birçok tonu vardır ve belki hiç kırmızı yoktur. Renk çizelgesindeki öğelerin girdi dağılımını olabildiğince iyi yansıtması gerekir; yüksek olasılıklı bölgeler için çok yer ayırmalı, veri olmayan bölgeleri göz ardı etmeliyiz.

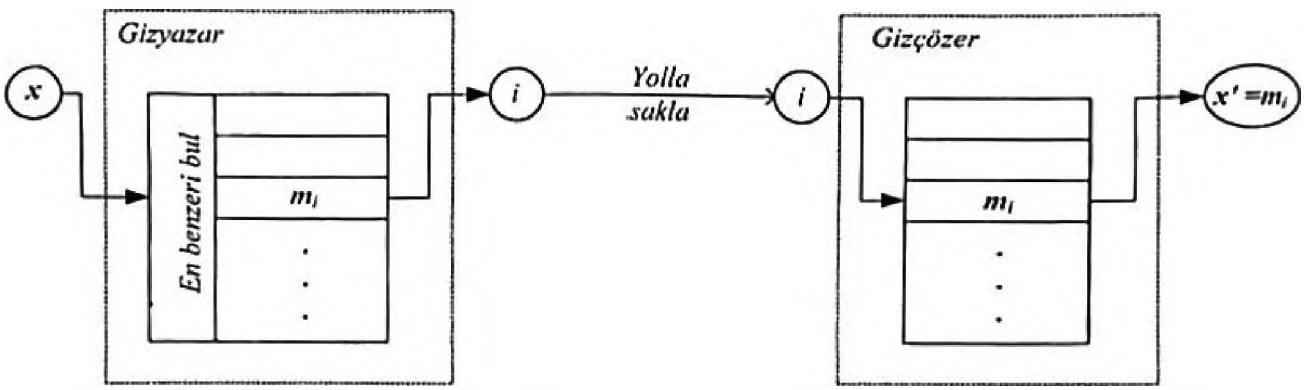
$\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$ örneklemimiz olduğunu varsayalım. k tane *dayanak yöneyimiz*, $\boldsymbol{m}_j, j = 1, \ldots, k$, olsun. Renk nicemleme örneğinde, $\boldsymbol{x}^t$, görüntü dosyasındaki 24 bit noktalar, $\boldsymbol{m}_j$ ise renk çizelgesindeki yine 24 bitlik öğelere karşılık gelir ve $k = 256$ olur.

Önce $\boldsymbol{m}_j$ değerlerinin bilindiğini varsayalım; onları nasıl öğreneceğimizi az sonra göreceğiz. İmgeyi ekranda gösterirken her $\boldsymbol{x}^t$ noktasını ona renk çizelgesinde en çok benzeyen öğeyle gösteririz:

$$\|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\|$$

Yani, veri yerine ona dayanak yöneyleri alfabesinde en çok benzeyeni kullanırız. $\boldsymbol{m}_i$ yöneylerine *gizyazı yöneyleri* de denir, çünkü yapılan bir tür *gizyazma* ve *gizçözme* işlemidir (bkz. şekil 7.1): $\boldsymbol{x}^t$ girdisinden i değerine geçerken girdiyi $\boldsymbol{m}_i, i = 1, \ldots, k$ gizyazı yöneyleriyle gösteririz ve ardından alıcı uçta i değerinden $\boldsymbol{m}_i$ oluşturulur. Nicemleme, *sıkıştırma* da sağlar: Her $\boldsymbol{x}^t$ için 24 bit yerine i dizin değerini saklamak ya da ağ üzerinden yollamak için 256 değerden birini gösterecek 8 bit yeterlidir; bu da neredeyse üçte bir sıkıştırma demektir—renk çizelgesini de saklamamız ya da yollamamız gerekir.

Şimdi $\boldsymbol{m}_i$ yöneylerini nasıl öğreneceğimizi görelim: $\boldsymbol{x}^t$, $\boldsymbol{m}_i$ tarafından gösterildiğinde aradaki $\|\boldsymbol{x}^t - \boldsymbol{m}_i\|$ farkına bağlı bir hata oluşur. Yeniden oluşturulanın



Şekil 7.1: Gizyazar x için ona en benzer dayanak yöneyinin dizinini yollar ve alıcı uçta gizçözer bu dizinle dayanak yöneyleri içinden x' yöneyini geri oluşturur. $\|x' - x\|^2$, geriçatma hatasını verir.

ilk görüntüye benzemesi için tüm noktalarda bu farkların olabildiğince küçük olması gerekir. Toplam *geriçatma hatası*, bu farkların karelerinin tüm örnekler için toplamıdır:

$$E(\{m_i\}_{i=1}^k | \mathcal{X}) = \sum_t \sum_i b_i^t \|x^t - m_i\|^2 \tag{7.3}$$

Her örnek, en yakın dayanak yöneyince gösterilir:

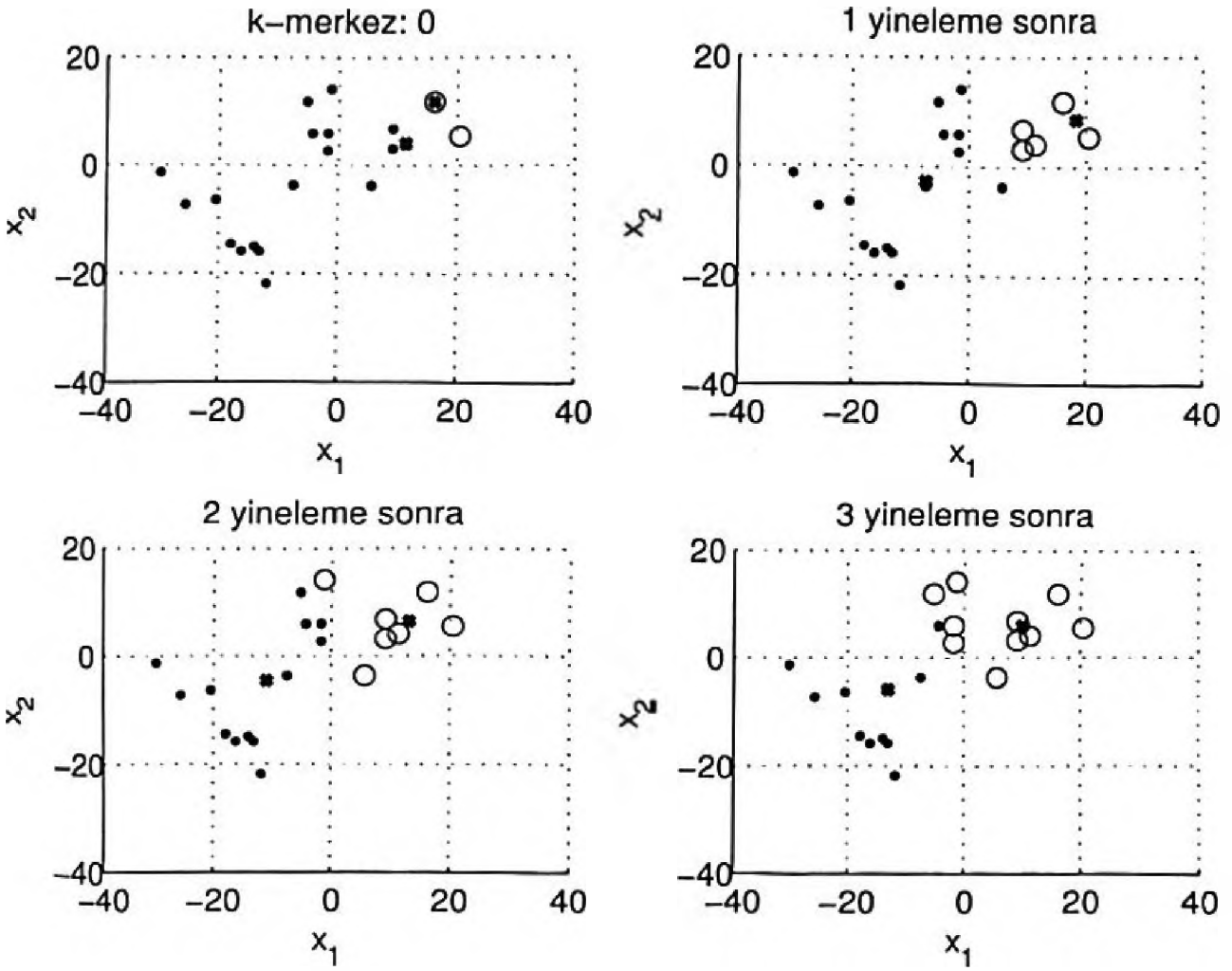
$$b_i^t = \begin{cases} 1 & \text{eğer } \|x^t - m_i\| = \min_j \|x^t - m_j\| \\ 0 & \text{değilse} \end{cases} \tag{7.4}$$

İstediğimiz toplam geriçatma hatasını en küçük yapan dayanak yöneylerini bulmaktır. b_i^t değerleri m_i yöneylerine de bağlıdır ve çözümsel sonuç yoktur. Bu amaç için k *merkezli öbekleme* denilen yinelemeli bir yordam kullanırız: Rastgele seçilen ilk m_i değerleriyle başlarız. Ardından her yinelemede önce denklem 7.4'ü kullanıp tüm x^t için b_i^t değerlerini hesaplarız; bunlar *kestirdiğimiz etiketler* olur: $b_i^t = 1$ ise x^t örneği m_i dayanak yöneyinin öbeğindendir. Ardından bu etiketleri kullanarak denklem 7.3'ü en küçük yapmaya çalışırız. Bunun için m_i yöneyine göre türev alıp sıfıra eşitleriz:

$$m_i = \frac{\sum_t b_i^t x^t}{\sum_t b_i^t} \tag{7.5}$$

Bir dayanak yöneyi gösterdiği örneklerin ortalamasıdır. Dikkat edersek bunun denklem 7.2'yle aynı olduğunu, tek farkın r_i^t etiketleri yerine kestirilen b_i^t etiketlerinin kullanılması olduğunu görürüz. Bu yinelemeli bir yordamdır çünkü yeni m_i dayanak yöneylerini hesapladığımızda, b_i^t üyelikleri değişir ve yeniden hesaplanması gerekir; ki bunlar da m_i yöneylerini etkiler. Bu iki adımı m_i yöneyleri kararlı olana dek yineleriz (bkz. şekil 7.2). k merkezli öbekleme algoritmasının örnek programı şekil 7.3'te verilmiştir.

Bu algoritmanın bir sakıncası yerel arama yapması ve bu yüzden son m_i yöneylerinin ilk m_i değerlerine bağlı olmasıdır. İlk değerleri seçerken farklı olasılıklar vardır:



Şekil 7.2: *k* merkezli öbeklemede dayanak yöneylerinin yakınsaması. Çarpılar merkezleri gösterir. Veriler en yakın oldukları merkeze göre işaretlenmiştir.

$m_i, i = 1, \ldots, k$ yöneylerini, örneğin, k rastsal x^t ile, başlat
Yinele
  Tüm $x^t \in \mathcal{X}$
    $b_i^t \leftarrow \begin{cases} 1 & \text{eğer } \|x^t - m_i\| = \min_j \|x^t - m_j\| \\ 0 & \text{değilse} \end{cases}$
  Tüm $i = 1, \ldots, k$
    $m_i \leftarrow \sum_t b_i^t x^t / \sum_t b_i^t$
Yakınsamaya dek

Şekil 7.3: k merkezli öbekleme için örnek program.

- Öğrenme kümesinden rastgele seçilen k örnek ilk m_i yöneyi olarak alınabilir.
- Tüm verinin ortalaması hesaplanır ve üstüne küçük rastsal yöneyler eklenerek ilk m_i yöneyleri oluşturulur.
- Temel bileşen hesaplanır, tanımladığı aralık k eşit parçaya bölünerek veri k parçaya bölünür. Parçalara düşeny örneklerin ortalamaları ilk merkezleri verir.

Algoritma yakınsadığında her merkezin verinin bir parçasını kapsamasını ve işe yaramasını isteriz; bunun için merkezleri veri olan yerlerden başlatmak gerekir.

Artımlı bir biçimde yeni merkez ekleyen ya da boşları silen algoritmalar da vardır. *Önder öbekleme algoritmasında*, tanımlı merkezlerden (belirli bir eşik değerine göre) çok uzak olan örnek için o noktada yeni bir merkez yaratılır (böyle bir yapay sinir ağı modeli olan ART algoritmasını 12. bölümde göreceğiz). Ya da, çok fazla örnek kapsayan ($\sum_t b_i^t / N > \theta$) bir merkez ikiye bölünebilir (iki kopyadan birine farklı olması için küçük rastsal bir yöney eklenir). Benzer biçimde, çok az örnek kapsayan bir merkez silinip girdi uzayının başka bir yerinden yeniden başlatılabilir.

k merkezli öbekleme, veri içindeki öbeklerin bulunması içindir ve öbekler, asıl örneklerine karşılık gelen merkezleri tarafından gösterilir. Yöney nicemleme, öbeklemenin bir kullanım alanıdır ama öbekleme, sınıflandırma ya da bağlanımdan önce bir *önişleme* adımı olarak da kullanılabilir. x^t için hesapladığımız b_i^t değerleri, ilk x uzayından k boyutlu uzaya, k boyutlu üstünkübün köşelerinden birine, bir eşleme olarak da görülebilir. Bu yeni uzayda bir bağlanım ya da sınıflandırma modeli öğrenilebilir; böylesi yöntemleri 12. bölümde göreceğiz.

7.4 Beklenti Büyütme Algoritması

k merkezli öbeklemede amacımızı, toplam geriçatma hatasını en küçük yapan dayanak yöneylerini bulmak olarak tanımladık. Bu bölümdeyse olasılık tabanlı bir yaklaşım kullanacağız ve örneklemin olabilirliğini en büyük yapan bileşen parametrelerini arayacağız. Denklem 7.1'deki karışım modelini kullanarak $\mathcal{X} = \{x^t\}_t$ örnek-

leminin log olabilirliğini yazalım:

$$\begin{aligned} \mathcal{L}(\Phi|\mathcal{X}) &= \log \prod_t p(x^t|\Phi) \\ &= \sum_t \log \sum_{i=1}^k p(x^t|\mathcal{G}_i)P(\mathcal{G}_i) \end{aligned} \tag{7.6}$$

Burada, Φ parametre yöneyi, $P(\mathcal{G}_i)$ önsel olasılıkları ve $p(x^t|\mathcal{G}_i)$ bileşen dağılımlarının yeterli istatistiklerini içerir. Bu parametreler için çözümsel bir sonuç olmadığından yinelemeli bir biçimde iyileme yapmamız gerekir.

Beklenti Büyütme algoritması (Dempster, Laird, ve Rubin 1977; Redner ve Walker 1984), en büyük olabilirlik kestiriminde, elimizde biri gözlenen X, öbürü saklı Z olan iki küme değişken olduğunda kullanılır. Algoritmanın amacı X gözlenen değişkenlerinde $\mathcal{L}(\Phi|\mathcal{X})$ olabilirliğini en büyük yapan Φ yöneyini bulmaktır. Bunu doğrudan yapamadığımızda Z *saklı değişkenlerini* de işe katar, modeli ikisini de kullanarak tanımlar, X ve Z için birleşik dağılımın olabilirliğini gösteren *tüm* olabilirlik $\mathcal{L}_c(\Phi|\mathcal{X},\mathcal{Z})$ işlevini en büyük yapan Φ yöneyini ararız.

Z değerleri gözlenmediği için doğrudan $\mathcal{L}_c$ ile çalışamayız; yerine $\mathcal{X}$ ve şu anki parametre Φ^l verildiğinde (l yineleme dizin değişkenidir) beklenen değeri olan $\mathcal{Q}$ işlevini kullanırız. Bu algoritmanın *beklenti* adımıdır. Ardından *büyütme* adımında, bunu en büyük yapan yeni Φ^{l+1} parametre yöneyini ararız:

$$\begin{aligned} \text{Beklenti} &: \mathcal{Q}(\Phi|\Phi^l) = E[\mathcal{L}_c(\Phi|\mathcal{X},\mathcal{Z})|\mathcal{X},\Phi^l] \\ \text{Büyütme} &: \Phi^{l+1} = \arg\max_{\Phi} \mathcal{Q}(\Phi|\Phi^l) \end{aligned}$$

Dempster, Laird, ve Rubin (1977), $\mathcal{Q}$ değerindeki iyileşmenin eksik olabilirlikte de bir iyileşmeye karşılık geldiğini göstermiştir:

$$\mathcal{L}(\Phi^{l+1}|\mathcal{X}) \geq \mathcal{L}(\Phi^l|\mathcal{X})$$

Karışım modelinde saklı değişkenler, girdinin etiketini, yani hangi bileşenden geldiğini gösterir. Bunları bildiğimizde (ki bu gözetimli öğrenmede sınıf etiketlerinin bilinmesine denktir), her veri için hangi parametreleri nasıl değiştirmemiz gerektiğini biliyor oluruz. Beklenti büyütme algoritmasının beklenti adımında, bileşenler ile ilgili şu anki bilgimizi kullanarak bu etiketleri kestirir, büyütme adımında da bu kestirilmiş etiketlerle olabilirliği büyütecek biçimde bileşen parametrelerini güncelleriz. Bu iki adım, k merkezli öbeklemenin iki adımı, b_i^t değerlerinin hesaplanması (beklenti adımı) ve m_i yöneylerinin yeniden kestirimi (büyütme adımı) ile aynıdır.

$z^t = \{z_1^t, \ldots, z_k^t\}$ *gösterge değişkenleri* tanımlayalım: eğer x^t, $\mathcal{G}_i$ bileşenindense $z_i^t = 1$, değilse 0 olsun. z, k farklı değerli katlıterimli bir dağılımdan gelir ve olasılıklar için $\pi_i \equiv P(\mathcal{G}_i)$ parametreleri tanımlıdır:

$$P(z^t) = \prod_{i=1}^k \pi_i^{z_i^t} \tag{7.7}$$

x^t gözleminin olabilirliği çekildiği bileşenin olabilirliğine eşittir:

$$p(x^t|z^t) = \prod_{i=1}^{k} p_i(x^t)^{z_i^t} \tag{7.8}$$

Burada, $p(x^t|\mathcal{G}_i)$ yerine kısaca $p_i(x^t)$ yazdık. Birleşik dağılımı

$$p(x^t, z^t) = P(z^t)p(x^t|z^t)$$

ve bunu kullanarak özdeş ve bağımsız $\mathcal{X}$ örneklemi için tüm olabilirliği yazalım:

$$\begin{aligned}
\mathcal{L}_c(\Phi|\mathcal{X}, \mathcal{Z}) &= \log \prod_t p(x^t, z^t|\Phi) \\
&= \sum_t \log p(x^t, z^t|\Phi) \\
&= \sum_t \log P(z^t|\Phi) + \log p(x^t|z^t, \Phi) \\
&= \sum_t \sum_i z_i^t [\log \pi_i + \log p_i(x^t|\Phi)]
\end{aligned}$$

Beklenti adımı:

$$\begin{aligned}
\mathcal{Q}(\Phi|\Phi^l) &\equiv E\left[\log P(X, Z)|\mathcal{X}, \Phi^l\right] \\
&= E\left[\mathcal{L}_c(\Phi|\mathcal{X}, \mathcal{Z})|\mathcal{X}, \Phi^l\right] \\
&= \sum_t \sum_i E[z_i^t|\mathcal{X}, \Phi^l][\log \pi_i + \log p_i(x^t|\Phi^l)]
\end{aligned}$$

öyle ki

$$\begin{aligned}
E[z_i^t|\mathcal{X}, \Phi^l] &= E[z_i^t|x^t, \Phi^l] \quad x^t \text{ örnekleri bağımsızdır} \\
&= P(z_i^t = 1|x^t, \Phi^l) \quad z_i^t \text{ değerleri } 0/1 \text{ rastsal değişkenlerdir} \\
&= \frac{p(x^t|z_i^t = 1, \Phi^l)P(z_i^t = 1|\Phi^l)}{p(x^t|\Phi^l)} \quad \text{Bayes kuralı} \\
&= \frac{p_i(x^t|\Phi^l)\pi_i}{\sum_j p_j(x^t|\Phi^l)\pi_j} \\
&= \frac{p(x^t|\mathcal{G}_i, \Phi^l)P(\mathcal{G}_i)}{\sum_j p(x^t|\mathcal{G}_j, \Phi^l)P(\mathcal{G}_j)} \\
&= P(\mathcal{G}_i|x^t, \Phi^l) \equiv h_i^t
\end{aligned} \tag{7.9}$$

Yani, $E[z_i^t]$ ile gösterdiğimiz saklı değişken beklenti değerinin x^t örneğinin $\mathcal{G}_i$ bileşeninden gelme sonsal olasılığı olduğunu görüyoruz. Bu değer bir olasılıktır ve 0 ile 1 arasındadır; k merkezli öbeklemedeki keskin 0/1 değeri yerine "eşiksiz' bir etikettir.

Büyütme adımı: Bir sonraki Φ^{l+1} parametre yöneyini, $\mathcal{Q}$ işlevini en büyük yapacak biçimde buluyoruz:

$$\Phi^{l+1} = \arg\max_{\Phi} \mathcal{Q}(\Phi|\Phi^l)$$

Açalım:

$$\begin{aligned}\mathcal{Q}(\Phi|\Phi^l) &= \sum_t \sum_i h_i^t [\log \pi_i + \log p_i(\boldsymbol{x}^t|\Phi^l)] \\ &= \sum_t \sum_i h_i^t \log \pi_i + \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\Phi^l) \end{aligned} \tag{7.10}$$

π_i değerlerini bulurken ikinci terimi atabiliriz (çünkü π_i değerlerine bağlı değildir). $\sum_i \pi_i = 1$ kısıtını bir Lagrange terimi olarak ekleyip en büyük yapan π_i değerlerini bulalım:

$$\nabla_{\pi_i} \sum_t \sum_i h_i^t \log \pi_i - \lambda \left(\sum_i \pi_i - 1 \right) = 0$$

Çözelim:

$$\pi_i = \frac{\sum_t h_i^t}{N} \tag{7.11}$$

Bu, denklem 7.2'de önsel olasılıkların kestirimine benzer; tek fark, 0/1 değerli r_i^t yerine esnek h_i^t değerlerinin kullanılmasıdır.

Benzer biçimde denklem 7.10'un birinci terimini bileşen parametrelerinden bağımsız olduğu için atabiliriz:

$$\nabla_{\Phi} \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\Phi) = 0 \tag{7.12}$$

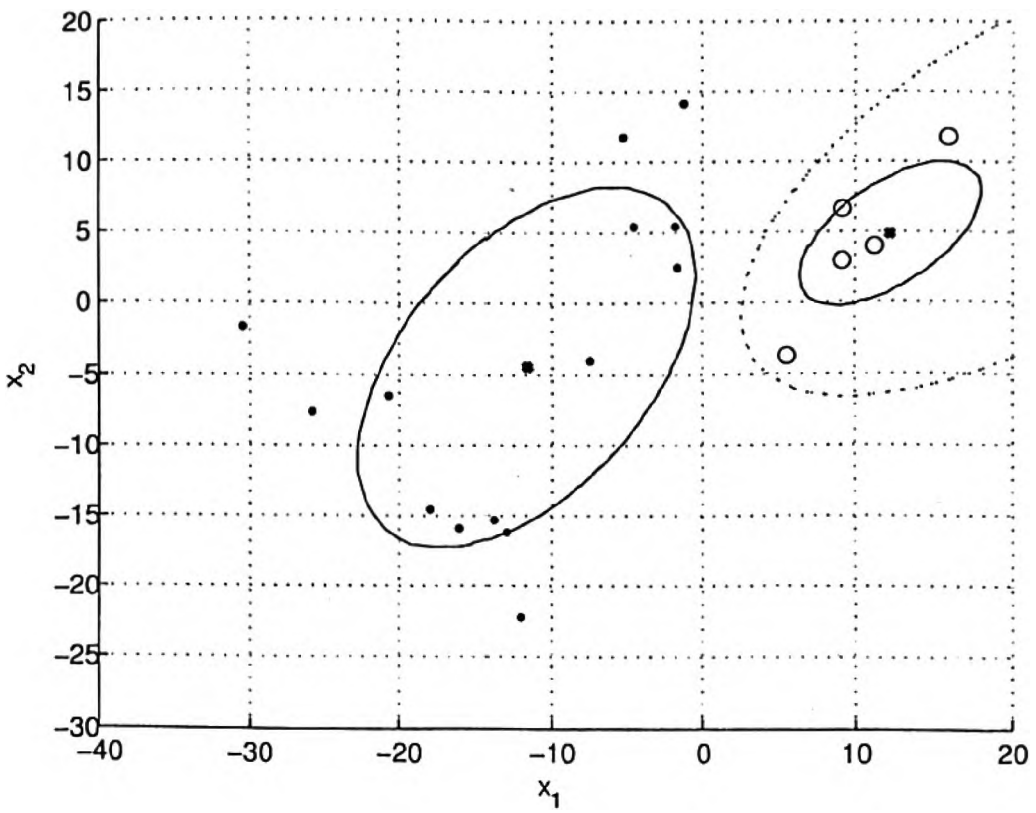
Bileşenlerin Gauss dağıldığını, $\hat{p}_i(\boldsymbol{x}^t|\Phi) \sim \mathcal{N}(\boldsymbol{m}_i, \mathbf{S}_i)$, varsayınca büyütme adımında hesapladığımız parametreler, bileşenlerin ortalama yöneyleri ve eşdeğişinti dizeyleri olur:

$$\begin{aligned}\boldsymbol{m}_i^{l+1} &= \frac{\sum_t h_i^t \boldsymbol{x}^t}{\sum_t h_i^t} \\ \mathbf{S}_i^{l+1} &= \frac{\sum_t h_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i^{l+1})(\boldsymbol{x}^t - \boldsymbol{m}_i^{l+1})^T}{\sum_t h_i^t} \end{aligned} \tag{7.13}$$

Gauss bileşenleri kullandığımızda beklenti adımında hesapladığımız değerleri açık olarak yazalım:

$$h_i^t = \frac{\pi_i |\mathbf{S}_i|^{-1/2} \exp[-(1/2)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T \mathbf{S}_i^{-1} (\boldsymbol{x}^t - \boldsymbol{m}_i)]}{\sum_j \pi_j |\mathbf{S}_j|^{-1/2} \exp[-(1/2)(\boldsymbol{x}^t - \boldsymbol{m}_j)^T \mathbf{S}_j^{-1} (\boldsymbol{x}^t - \boldsymbol{m}_j)]} \tag{7.14}$$

Yine 7.13 ve 7.2 numaralı denklemleri karşılaştırdığımızda farkın gerçek (bilinmeyen) 0/1 r_i^t yerine kestirilmiş eşiksiz h_i^t değerlerinin kullanılması olduğunu görürüz.



Şekil 7.4: k merkezli öbekleme bir kez yinelendikten sonra (şekil 7.2) beklenti büyütme algoritması kullanıldığında kestirilen iki Gauss bileşeni. k merkezli öbeklemeden farklı olarak burada eşdeğişinti dizeyleri de kestirilir. Veri noktaları, daha büyük h_i sonsal olasılığına göre etiketlenmiştir. Kestirilmiş Gauss bileşenlerinin eşolasılık çevritleri ve kesikli çizgiyle $h_i = 0.5$ sınırı çizilmiştir.

Beklenti büyütme algoritmasını rastgele Φ^0 yerine k merkezli öbeklemenin sonucuyla başlatırız. Önce k merkezli öbekleme için birkaç yineleme yapar ve $\boldsymbol{m}_i$ merkezlerini buluruz. Bunların kapsamına giren örneklerin eşdeğişintisiyle $\mathbf{S}_i$ dizeylerini, ve $\sum_t b_i^t/N$ ile π_i değerlerini kestiririz. Bu noktadan sonra, şekil 7.4'te gördüğümüz gibi beklenti büyütme algoritmasını kullanırız.

Gözetimli sınıflandırmada (bölüm 5.5) yaptığımız gibi, eğer boyut sayısı yüksek ama örneklem küçükse basitleştirici varsayımlarla modeli düzenlileştirebiliriz. Örneğin, ortak bir eşdeğişinti dizeyi, $\hat{p}_i(\boldsymbol{x}^t|\Phi) \sim \mathcal{N}(\boldsymbol{m}_i, \mathbf{S})$, varsayarsak denklem 7.12 basitleşir:

$$\min_{\boldsymbol{m}_i,\mathbf{S}} \sum_t \sum_i h_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)^T \mathbf{S}^{-1} (\boldsymbol{x}^t - \boldsymbol{m}_i) \tag{7.15}$$

$\hat{p}_i(\boldsymbol{x}^t|\Phi) \sim \mathcal{N}(\boldsymbol{m}_i, s^2\mathbf{I})$ varsayarsak eşdeğişinti dizeyi köşegen olur:

$$\min_{\boldsymbol{m}_i,s} \sum_t \sum_i h_i^t \frac{\|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2}{s^2} \tag{7.16}$$

ki bu, k merkezli öbeklemede kullandığımız toplam geriçatma hatasına (denklem 7.3)

eşittir. Tek fark, beklenti adımımda hesapladığımız

$$h_i^t = \frac{\exp\left[-(1/2s^2)\|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2\right]}{\sum_j \exp\left[-(1/2s^2)\|\boldsymbol{x}^t - \boldsymbol{m}_j\|^2\right]} \tag{7.17}$$

sonsal değerinin 0 ile 1 arasında bir olasılık olmasıdır. k merkezli öbeklemenin kullandığı b_i^t, keskin 0/1 kararı verir, ama h_i^t, girdiyi öbeklere belirli bir olasılıkla atayan *esnek* bir etikettir. b_i^t yerine h_i^t kullandığımızda bir girdi örneği tüm bileşen parametreleriin güncellenmesine sonsal olasılık değerlerine orantılı olarak etki eder. Özellikle bir girdi iki merkezin ortasına yakın bir yere düşüyorsa böyle bir esnek güncelleme yeğlenmelidir.

Bütün bunlardan, k merkezli öbeklemenin, beklenti büyütme algoritmasının özel bir durumu olduğunu ve bağımsız, eşit değişintili ve ilintisiz girdiler için Gauss karışımları kullanılıp bileşen önsel olasılıkları eşit alınır ve etiketleri 0/1 olacak biçimde keskinleştirilirse iki algoritmanın denk olduğunu görüyoruz. k merkezi öbeklemeyi girdi uzayını eşit çemberlerle örtmek olarak yorumluyorsak Gauss karışımında beklenti büyütmeyi de, büyüklüğü, yönü ve kapsama oranı veriye bağlı olarak belirlenen elipslerle örtmek olarak yorumlayabiliriz.

7.5 Saklı Değişken Modellerinin Karışımı

Gauss karışımlarında basitleştirici varsayımlar yapmadan tüm eşdeğişinti dizeyini kullanırsak girdi boyutu çok, veri az olduğunda tekillik olmasa bile aşırı öğrenme riski oluşur. Parametre sayısını azaltmak için ortak bir eşdeğişinti varsaymak doğru olmaz çünkü öbeklerin farklı biçimleri olabilir. Köşegen dizeyler daha da risklidir çünkü ilintileri yok sayarlar.

Seçenek, öbeklerde boyut azaltmaktır. Bu ilintileri korurken parametre sayısının azaltılmasını sağlar. Serbest parametre sayısı inilen uzayın boyut sayısına bağlı olur.

Öbeklerde etken çözümlemesi (bölüm 6.4) yaptığımızda öbeklerdeki veriyi yaratan *saklı değişkenleri* ya da *etkenleri* ararız (Bishop 1999):

$$p(\boldsymbol{x}^t|\mathcal{G}_i) \sim \mathcal{N}(\boldsymbol{m}_i, \mathbf{V}_i\mathbf{V}_i^T + \boldsymbol{\Psi}_i) \tag{7.18}$$

Burada, $\mathbf{V}_i$ ve $\boldsymbol{\Psi}_i$, sırasıyla $\mathcal{G}_i$ bileşenindeki etken yüklemelerini ve girdiye özel değişintileri gösterir. Rubin ve Thayer (1982), etken çözümlemesine özgü beklenti büyütmenin adımlarını verir. Bu yaklaşımı karışım modelleri için genelleyerek *etken çözümleme karışımları* tanımlanabilir (Ghahramani ve Hinton 1997). Beklenti adımında denklem 7.9'da denklem 7.18'i kullanır, büyütmedeyse denklem 7.12'yi $\mathbf{S}_i$ yerine $\mathbf{V}_i$ ve $\boldsymbol{\Psi}_i$ parametrelerine göre büyütürüz. Benzer biçimde öbeklerde temel bileşen çözümlemesi de yapılabilir ve buna *temel bileşen çözümleme karışımı* denir (Tipping ve Bishop 1999).

Tabii ki önce beklenti büyütme kullanarak $\mathbf{S}_i$ dizeylerini öğrenip ardından her öbekte ayrı ayrı etken ya da temel bileşen çözümlemesi yapılabilir, ama beklenti büyütmeyi doğrudan karışımın etkenlerini kestirmek için kullanmak daha iyidir;

çünkü böylece öbekleme ve boyut azaltma adımlarını birleştirir ve eşiksiz bir paylaşım gerçekleştirebiliriz. Her girdi, tüm öbeklerin gizli değişkenlerinin hesabına, h_i^l ile ağırlıklanmış olarak katkı yapacaktır.

7.6 Öbekleme Ardından Gözetimli Öğrenme

Öbekleme, 6. bölümde gördüğümüz boyut azaltma yöntemleri gibi, iki amaç için kullanılabilir: Veriyi anlayıp yapısını öğrenmek için kullanılabilir. Boyut azaltma yöntemleri değişkenler arasındaki ilintileri bulup değişkenleri bir araya getirmek içindir; öbekleme yöntemleriyse örnekler arasındaki benzerlikleri bulup benzeyen örnekleri bir araya getirir.

Böyle öbekler bulunduğunda uygulama uzmanlarınca adlandırılabilir ve özellikleri tanımlanabilir. Her öbeğin merkezi, o öbeğin üyelerinin asıl örneğini gösterir; ayrıca, öbek üyelerinin her girdi boyutunda hangi aralıkta değer aldığı yazılabilir. Bu da verinin basit bir tanımını çıkarır ve anlamayı kolaylaştırır. Örneğin, öbeklemeyle bir şirketin müşterileri k öbeğe ayrılıp *müşteri bölütleme* yapılabilir. Her bölüte düşen müşterilerin kişisel bilgilerine ve şirketten aldıkları ürün ve servislere bakılarak müşteri tabanı daha iyi anlaşılır ve farklı müşteri türleri için farklı yaklaşımlar, örneğin, farklı kapmanyalar, geliştirilebilir; buna *müşteri ilişkileri yönetimi* denir. Sık olanlar yanında sık olmayanlar da bilgi verebilir: büyük öbeklerden birine girmeyen ve dikkat edilmesi gereken, örneğin kaçmakta olan müşteriler de böylece belirlenebilir.

Öbekleme, bir önişleme adımı olarak da kullanılabilir. 6. bölümde gördüğümüz boyut azaltma yöntemlerinin bizi yeni bir uzaya taşıması gibi öbeklemeden sonra da boyutları h_i (ya da bilgi kaybını göze alıp b_i) olan yeni bir uzaya geçeriz. Ardından gözetimli bir uygulamada ayırtaç ya da bağlanım işlevini bu yeni uzayda öğrenebiliriz. Temel bileşen çözümlemesi gibi boyut azaltma yöntemleriyle karşılaştırdığımızda fark, bu yeni uzayın boyut sayısı olan k değerinin girdi boyut sayısı olan d değerinden daha büyük olabilmesidir.

Temel bileşen çözümlemesi gibi bir yöntem kullandığımızda yeni boyutlar ilk boyutların birleşimi olarak tanımlıdır ve yeni uzayda bir noktayı gösterirken tüm boyutlar katkı yapar, yani z_j değerleri sıfırdan farklıdır. Öbeklemedeyse yeni boyutlar yerel tanımlıdır, çok daha fazla yeni b_j boyutu vardır ve yalnızca bir tanesi (ya da h_j kullanıyorsak birkaç tanesi) sıfırdan farklı değer alır. İlk durumda, yani az boyut olup hepsi kullanıldığında buna *dağıtık gösterim*, sonraki durumda, çok boyut olup azı kullanıldığındaysa *yerel gösterim* denir.

Gözetimli öğrenmenin önüne gözetimsiz öbekleme ya da boyut azaltma koymanın bir çekici yanı, bu ikinciler için etiketli veriye gerek olmamasıdır. Veriyi etiketlemek pahalıdır. Öbek parametrelerini hesaplarken bolca bulunabilen etiketsiz veriyi kullanıp ardından gelen sınıflandırıcı ya da bağlanım modelini daha az etiketli veriyle eğitebiliriz. Gözetimsiz öğrenme, "sık olanı öğrenmek," diye de tanımlanır (Barrow 1989). Ardından gözetimli öğrenme geldiğinde önce sık olanı modeller, sonra onun ne anlama geldiğini öğreniriz. Böyle yöntemleri 12. bölümde göreceğiz.

Sınıflandırmada her sınıf için bileşenlerden oluşan bir karışım modeli tanımlanırsa tüm dağılım, *karışımların karışımı* olur:

$$p(\boldsymbol{x}|C_i) = \sum_{j=1}^{k_i} p(\boldsymbol{x}|\mathcal{G}_{ij})P(\mathcal{G}_{ij})$$

$$p(\boldsymbol{x}) = \sum_{i=1}^{K} p(\boldsymbol{x}|C_i)P(C_i)$$

öyle ki k_i, $p(\boldsymbol{x}|C_i)$ dağılımını oluşturan bileşen sayısını ve $\mathcal{G}_{ij}$, i. sınıfın j. bileşenini gösterir. Farklı sınıfların farklı sayıda bileşeni olabilir. Bileşen parametreleri yukarıda gördüğümüz yöntemlerin biriyle her sınıf için ayrı ayrı öğrenilebilir; veri az olduğunda düzenlileştirme yapmak gerekecektir. Bu yaklaşım, tüm veri üzerinde tek bir öbeklemeyle $K \times k$ bileşen kestirip ardından bunları etiketlemeye çalışmaktan daha iyi sonuç verecektir.

7.7 Öbek Ağaçları

Öbeklemeye olasılık kuramından bakıp veriye bir karışım modeli oturtmak olarak, ya da geriçatma hatasını en aza indiren dayanak yöneyleri bulmak olarak tanımladık. Bunlardan başka yalnızca örnekler arasındaki benzerlikleri kullanan öbekleme yöntemleri de vardır; amaç, verilmiş bir benzerlik ölçütüne göre, aynı öbek içindeki örneklerin benzerliğinin farklı öbeklerdeki örneklerin benzerliğinden daha fazla olduğu bir öbekleme bulmaktır. *Öbek ağaçları* bu düşünceye dayanır.

Bunun için örnekler arasında bir benzerlik, ya da tersi olarak bir uzaklık, ölçütü gerekir. Genelde Euclid uzaklığı kullanılır, ama bunun için tüm girdilerin aynı ölçekte olması gerekir. Genelde *Minkowksi uzaklığı*

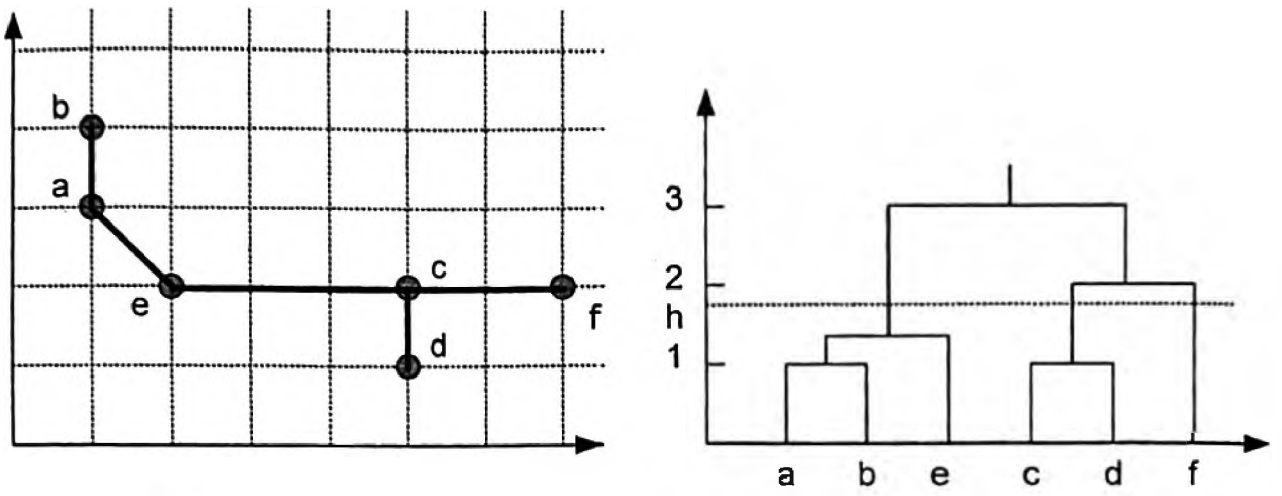
$$d_m(\boldsymbol{x}^r, \boldsymbol{x}^s) = \left[\sum_{j=1}^{d}(x_j^r - x_j^s)^p\right]^{1/p}$$

olarak tanımlıdır ve $p = 2$ için Euclid uzaklığını verir. *Şehiriçi uzaklığını* hesaplamak daha kolaydır:

$$d_{cb}(\boldsymbol{x}^r, \boldsymbol{x}^s) = \sum_{j=1}^{d}|x_j^r - x_j^s|$$

Birleştirmeli öbekleme algoritması, her biri başlangıçta tek bir örnek içeren N öbekle başlar, her adımda en benzeyen iki öbeği birleştirir ve tek bir öbek kaldığında durur. *Bölmeli öbeklemeyse* ters yönde gider, tek öbekle başlayıp küçük öbeklere böle böle ilerler ve her öbekte tek örnek kaldığında durur.

Birleştirmeli algoritmanın her adımında birbirine en benzer iki öbeği birleştiririz. *Tek bağlı öbeklemede* iki öbek arasındaki uzaklık, iki öbekteki olası tüm örnek çiftleri



Şekil 7.5: İki boyutlu bir veri kümesi ve tek bağlı öbeklemenin oluşturduğu öbek ağacı. Dalların kesişmemesi için yapraklar sıralanmıştır. Oluşan ağacı herhangi bir h değerinden kesip öbekleri bulabiliriz.

arasındaki uzaklıkların en küçüğüdür:

$$d(\mathcal{G}_i, \mathcal{G}_j) = \min_{\boldsymbol{x}^r \in \mathcal{G}_i, \boldsymbol{x}^s \in \mathcal{G}_j} d(\boldsymbol{x}^r, \boldsymbol{x}^s) \tag{7.19}$$

Öyle bir çizge olsun ki düğümleri örneklere, ayrıtların ağırlıkları da örnekler arasındaki uzaklıklara karşılık gelsin. Tek bağlı öbekleme algoritması, bu çizgenin en küçük kapsayan ağacını oluşturur.

Tam bağlı öbeklemede iki öbek arasındaki uzaklık, iki öbeklerden olası tüm örnek çiftleri arasındaki uzaklıkların en büyüğüdür:

$$d(\mathcal{G}_i, \mathcal{G}_j) = \max_{\boldsymbol{x}^r \in \mathcal{G}_i, \boldsymbol{x}^s \in \mathcal{G}_j} d(\boldsymbol{x}^r, \boldsymbol{x}^s) \tag{7.20}$$

Tek bağlı ve tam bağlı öbekleme en sık kullanılan iki yaklaşımdır. Ortalama bağlı öbeklemede farklı öbeklerden tüm örnek çiftleri arasındaki uzaklıkların ortalamasını, merkez uzaklığındaysa öbek merkezleri (öbekteki örneklerin ortalamaları) arasındaki uzaklığı kullanırız.

Birleştirme sonucu bir ağaç olarak gösterilebilir. Yapraklar örnekleri gösterir ve birleştirilme sırasına göre bağlanırlar. Bir örnek, şekil 7.5'te verilmiştir. Ağacı herhangi bir düzeyde kesip istediğimiz sayıda öbek elde edebiliriz.

Tek bağlı ve tam bağlı yöntemler öbekler arasındaki uzaklığı farklı ölçtükleri için farklı öbek ağaçları oluşturabilir. Tek bağlı yöntemde, iki örnek arasındaki uzaklık h değerinden azsa, ya da ikisi arasında her adımda h değerinden küçük bir bağ olan bir örnek dizisi varsa, bu iki örnek h düzeyinde aynı öbektedir. Tam bağlı yöntemde öbek içindeki tüm örnek çiftleri arasındaki uzaklık h değerinden küçüktür. Tek bağlı öbekler bu "zincirleme" etkisiyle uzayabilir. (Şekil 7.5'te e ile c örneklerinin ortasında bir örnek olsaydı ne olurdu?) Tam bağlı öbekler daha derli toplu olur.

7.8 Öbek Sayısının Belirlenmesi

Her öğrenme yöntemindeki gibi öbeklemede de karmaşıklığı ayarlamak için bir düğme vardır, o da öbek sayısı olan k üstparametresidir. Bir k değeri verildiğinde öbekleme algoritması her zaman k merkez bulur, ama bunların gerçekten anlamlı öbekler mi olduğunu yoksa kullandığımız algoritmanın mı dayattığını bilemeyiz. k değerini belirlemek için farklı yollar vardır:

- Renk nicemleme gibi uygulamalarda uygulama bir değer gerektirir.
- Temel bileşen çözümlemesi gibi bir yöntemle iki boyuta indirip görselleştirmek verinin yapısını ve öbek sayısını bulmakta faydalı olabilir.
- Artımlı bir yaklaşım işe yarayabilir: İzin verilen en büyük uzaklık, örnek başına izin verilen en fazla geriçatma hatasını tanımlar.
- Bazı uygulamalarda, örneğin veri madenciliğinde, öbekler uzmanlar tarafından geçerlenerek verideki anlamlı birliktelikleri gösterip göstermediğine bakılır. Renk nicemlemede resim gözle incelenebilir (her ne kadar gözlerimiz ve beynimiz resmi nokta nokta incelemese de).

Kullandığımız öbekleme yöntemine göre, geriçatma hatası, ya da log olabilirlik değerini k değerine bağlı olarak çizip "dirsek" noktasını arayabiliriz. k çok büyük olursa algoritma öbekleri sahte öbeklere bölecek ve bu durumda geriçatma hatasında anlamlı bir azalma ya da log olabilirlikte anlamlı bir artış olmayacaktır. Benzer biçimde öbek ağaçlarında ardışık düzeyler arasındaki farka bakarak iyi bir h kesme değeri bulabiliriz.

7.9 Kısa Kısa

Karışım modelleri istatistikte sıklıkla kullanılır. Bu konudaki ders kitapları, Titterington, Smith, ve Makov (1985) ile McLachlan ve Basford (1988)'dir. McLachlan ve Krishnan (1997), beklenti büyütme algoritmasında son yıllarda yapılan çalışmaları, yakınsamanın nasıl iyileştirilebileceğini, ve farklı sürümleri tartışır. Sinyal işlemede, k merkezli öbeklemeye *Linde-Buzo-Gray* (LBG) algoritması (Gersho ve Gray 1992) denir. Bu algoritmalar, hem istatistik, hem de sinyal işlemede birçok uygulamada başarıyla kullanılmaktadır. *Bulanık k merkezli öbekleme*, yine 0 ile 1 arasında bir değer alan *bulanık üyelik* değerini kullanır (Bezdek ve Pal 1995). Alpaydın (1998), k merkezli öbekleme, bulanık k merkezli öbekleme ve Gauss karışımlarında beklenti büyütme algoritmalarını karşılaştırır. Gauss karışım modellerinde beklenti büyütme ve başka algoritmaların karşılaştırılması Xu ve Jordan (1996) tarafından yapılmıştır. Küçük veri kümelerinde basitleştirici varsayımlar yapmak yerine Bayesçi bir yaklaşım da kullanılabilir (Ormoneit and Tresp 1996). Moerland (1999), Gauss karışımlarını ve saklı değişken modellerinin karışımını farklı sınıflandırma verilerinde karşılaştırır ve saklı değişken modellerinin deneysel üstünlüğünü gösterir. Öbekleme hakkında

bir kitap Jain ve Dubes (1988)'dir; Jain, Murty, ve Flynn (1999) ile Xu ve Wunsch (2005), farklı öbekleme algoritmalarını gözden geçirir.

7.10 Alıştırmalar

1. k merkezli öbeklemeyi gGörüntü sıkıştırmada da kullanabiliriz: Her görüntüyü ayrık (örtüşmeyen) $c \times c$ büyüklüğünde pencerelere ayırırız. Bunların oluşturduğu c^2 boyutlu yöneylerle herhangi bir k için (genelde ikinin bir kuvveti alınır) k merkezli öbekleme yaparız. Dayanak yöneylerini ve her pencere için dayanak yöneyinin dizin değerini göndeririz. Alıcı uçta bu dizin değerlerine göre alınan pencereler yan yana konarak görüntü yeniden oluşturulur. Farklı k ve c değerleri için bunu gerçekleyen programı yazın. Her durum için geriçatma hatasını ve sıkıştırma oranını hesaplayın.
2. Önce k merkezli öbeklemeyle örnekleri öbeklere ayırıp ardından her öbek için kendi $\mathbf{S}_i$ değişinti dizeyini kestirebiliriz. Sizce bu neden iyi bir sonuç vermez?
3. $\mathbf{S}$ eşdeğişinti dizeyini (denklem 7.15) ve köşegen dizeyideki s^2 parametresini (denklem 7.16) kestirmek için büyütme adımlarını türetin.
4. Girdilerin ikili olduğu çok değişkenli Bernoulli karışımlarında beklenti büyütme algoritmasını türetin.
5. Sınıflandırmada karışımların karışımı yaklaşımında, C_i sınıfının bileşen sayısı olan k_i parametresini nasıl ayarlayabiliriz?
6. Ortalama bağ öbeklemeyle k merkezli öbekleme arasındaki benzerlik ve farklar nelerdir?
7. Öbek ağaçlarında yerel olarak uyarlanabilir uzaklıkları nasıl kullanabiliriz? Bunun üstünlük ve sakıncaları nedir?
8. k merkezli öbeklemeyi nasıl aykırı gözlemlere gürbüz yapabiliriz?
9. Öbek ağacını "budayabilir miyiz"?

7.11 Kaynaklar

Alpaydın, E. 1998. "Soft Vector Quantization and the EM Algorithm." *Neural Networks* 11: 467–477.

Barrow, H. B. 1989. "Unsupervised Learning." *Neural Computation* 1: 295–311.

Bezdek, J. C., N. R. Pal. 1995. "Two Soft Relatives of Learning Vector Quantization." *Neural Networks* 8: 729–743.

Bishop, C. M. 1999. "Latent Variable Models." *Learning in Graphical Models*, ed. M. I. Jordan, 371–403. Cambridge, MA: MIT Press.

Dempster, A. P., N. M. Laird, D. B. Rubin. 1977. "Maximum Likelihood from Incomplete Data via the EM Algorithm." *Journal of Royal Statistical Society* B 39: 1–38.

Gersho, A., R. M. Gray. 1992. *Vector Quantization and Signal Compression*. Boston: Kluwer.

Ghahramani, Z., G. E. Hinton. 1997. *The EM Algorithm for Mixtures of Factor Analyzers.* Technical Report CRG TR-96-1, Department of Computer Science, University of Toronto (revised Feb. 1997).

Jain, A. K., R. C. Dubes. 1988. *Algorithms for Clustering Data.* New York: Prentice Hall.

Jain, A. K., M. N. Murty, P. J. Flynn. 1999. "Data Clustering: A Review." *ACM Computing Surveys* 31: 264–323.

McLachlan, G. J., K. E. Basford. 1988. *Mixture Models: Inference and Applications to Clustering.* New York: Marcel Dekker.

McLachlan, G. J., T. Krishnan. 1997. *The EM Algorithm and Extensions.* New York: Wiley.

Moerland, P. 1999. "A Comparison of Mixture Models for Density Estimation." *International Conference on Artificial Neural Networks*, ed. D. Willshaw, A. Murray, 25–30. London, UK: IEE Press.

Ormoneit, D., V. Tresp. 1996. "Improved Gaussian Mixture Density Estimates using Bayesian Penalty Terms and Network Averaging." *Advances in Neural Information Processing Systems 8*, ed. D. S. Touretzky, M. C. Mozer, M. E. Hasselmo, 542–548. Cambridge, MA: MIT Press.

Redner, R. A., H. F. Walker. 1984. "Mixture Densities, Maximum Likelihood and the EM Algorithm." *SIAM Review* 26: 195–239.

Rubin, D. B., D. T. Thayer. 1982. "EM Algorithms for ML Factor Analysis." *Psychometrika* 47: 69–76.

Tipping, M. E., C. M. Bishop. 1999. "Mixtures of Probabilistic Principal Component Analyzers." *Neural Computation* 11: 443–482.

Titterington, D. M., A. F. M. Smith, E. E. Makov. 1985. *Statistical Analysis of Finite Mixture Distributions.* New York: Wiley.

Xu, L., M. I. Jordan. 1996. "On Convergence Properties of the EM Algorithm for Gaussian Mixtures." *Neural Computation* 8: 129–151.

Xu, R., D. Wunsch II. 2005. "Survey of Clustering Algorithms." *IEEE Transactions on Neural Networks* 16: 645–678.

Bölüm 8

Dağılımdan Bağımsız Yöntemler

Önceki bölümlerde verinin bilinen bir dağılım ya da dağılım karışımından geldiğini varsayan dağılıma bağımlı ya da yarı bağımlı yaklaşımları gördük. Şimdiyse böyle bir varsayım yapılamadığında kullanılan dağılım bağımsız yaklaşımı tartışacağız. Olasılık kestirimi, sınıflandırma, ve bağlanımda dağılımdan bağımsız yöntemleri görecek, ayrıca zaman ve bellek karmaşıklığını nasıl denetim altında tutabileceğimizi tartışacağız.

8.1 Giriş

Dağılıma bağımlı yöntemlerde, ister olasılık kestirimi, sınıflandırma, isterse bağlanım için olsun, tüm girdi uzayında geçerli bir model varsayarız. Bağlanımda, örneğin, doğrusal bir model varsaydığımızda, her girdi için aynı doğrusal işlevi kullanırız. Sınıflandırmada, bir sınıfın tüm örneklerinin aynı dağılımdan geldiğini varsayarız. Dağılıma bağımlı bir yöntemin üstünlüğü, bir olasılık dağılımını, ayırtacı ya da bağlanım işlevini kestirme problemini, o modelin az sayıdaki parametresini kestirmeye çevirmesi ve böylece işi kolaylaştırmasıdır. Ama ne yazık ki böyle varsayımlar her zaman geçerli olmaz ve yanlılığa ve dolayısıyla hataya neden olabilir.

Eğer böylesi varsayımlarla az sayıda parametreyle tanımlanabilen bir model yazamazsak bir olasılık, 7. bölümde gördüğümüz gibi, dağılımı az sayıda dağılımın karışımı olarak tanımlayan yarı bağımlı bir yöntem kullanmaktır.

Dağılımdan bağımsız kestirimde tek varsayımımız benzer girdilerin çıktılarının da benzer olduğudur. Bu akla yakın bir varsayımdır: dünyamız düzenlidir ve işlevler, ister olasılık, ister ayırtaç, isterse bağlanım için olsun, yavaşça değişir. Benzer örneklerin anlamları benzerdir. Hepimiz komşularımızı severiz, çünkü bize benzerler.

Bu yaklaşımda algoritma uygun bir uzaklık ölçütüyle öğrenme kümesinden benzer geçmiş örneklerin bulunması, ve bunlar üzerinden ara değerlemeyle çıktının bu-

lunması biçiminde iki adımdan oluşur. Farklı yöntemler, ya kullandıkları uzaklık ölçütüyle, ya da benzer öğrenme örnekleri üzerinden yaptıkları ara değerlemede birbirinden ayrılır. Dağılıma bağımlı bir modelde çıktıyı tüm öğrenme örnekleri etkiler, ama dağılımdan bağımsız durumda böyle tek bir model yoktur; gerektiğinde yalnızca yakındaki öğrenme örnekleri kullanılarak yerel modeller geçici olarak kestirilir, kullanılır, ve unutulur.

Daha gevşek bir tanımla bakarsak dağılımdan bağımsız yöntemler karmaşıklığı öğrenme öncesinde sabitlenmemiş yöntemlerdir; karmaşıklıkları öğrenme kümesinin büyüklüğüne, ya da daha doğrusu, öğrenme kümesinin altında yatan problemin karmaşıklığına bağlıdır.

Yapay öğrenme yazınında dağılımdan bağımsız yöntemlere *örnek tabanlı* ya da *bellek tabanlı* algoritmalar da denir, çünkü yaptıkları örnekleri bellekte saklayıp üzerinden ara değerleme yapmaktır. Bunun için tüm öğrenme örneklerini saklamamız gerekir ki bu $\mathcal{O}(N)$ bellek gerektirir. Ayrıca, bir girdi verildiğinde ona benzer olanların bulunması gerekir ki bu da $\mathcal{O}(N)$ hesap ister. Böyle yöntemlere *tembel* öğrenme algoritmaları da denir, çünkü *çalışkan* olanlar gibi öğrenme kümesini gördüklerinde hemen bir model oluşturmaz, deneme sorgusu gelmesini bekler, ancak o zaman çıktı üretirler. Dağılıma bağlı yaklaşımda model basittir ve az, $\mathcal{O}(d)$, ya da $\mathcal{O}(d^2)$ parametrelidir; bir kez bu parametreleri veriden hesaplayınca öğrenme kümesini artık saklamak gerekmez. N değeri, genelde d (ya da d^2) değerinden daha büyüktür, ve bellek ve hesaptaki bu artış dağılımdan bağımsız yöntemlerin en önemli sakıncasıdır.

Dağılımdan bağımsız yöntemleri bir olasılık dağılımının kestirilmesinde kullanmaya başlayıp önce sınıflandırma, ardından bağlanım için genelleyeceğiz.

8.2 Dağılımdan Bağımsız Olasılık Kestirimi

$\mathcal{X} = \{x^t\}_{t=1}^N$ örnekleminin bilinmeyen bir $p(\cdot)$ olasılık dağılımından bağımsız örnekler içerdiğini varsayalım. Kestiricimiz $\hat{p}(\cdot)$ olsun. Önce x^t örneğnin tek boyutlu sayıl olduğu durumla başlayıp ardından çok boyutlu duruma genelleyeceğiz.

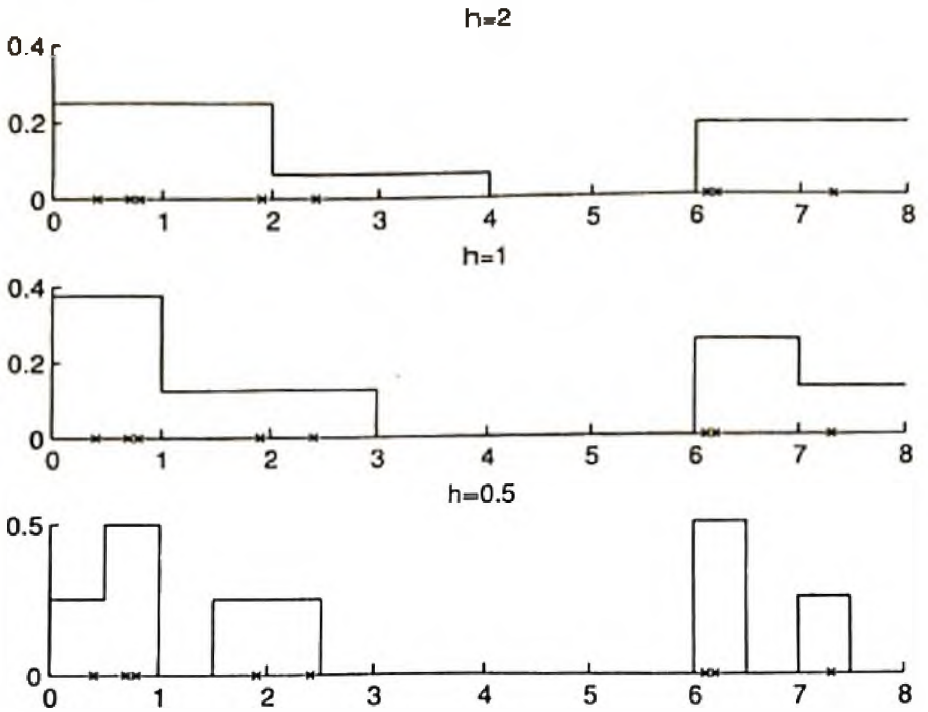
Birikimli dağılım işlevi, $F(x)$, için x noktasındaki dağılımdan bağımsız kestirim değeri x ya da daha küçük olan örneklerin oranıdır:

$$\hat{F}(x) = \frac{\#\{x^t \leq x\}}{N} \tag{8.1}$$

Burada, $\#\{x^t \leq x\}$, değeri x ya da küçük olan x^t öğrenme örneklerinin sayısını gösterir. Olasılık yoğunluğu, birikimli dağılımın türevi olarak yazılabildiği için yoğunluk kestirimi de benzer biçimde hesaplanabilir:

$$\hat{p}(x) = \frac{1}{h}\left[\frac{\#\{x^t \leq x+h\} - \#\{x^t \leq x\}}{N}\right] \tag{8.2}$$

h parametresi aralığın boyudur ve bu aralığa düşen x^t örnekler, "yakın" ya da "benzerdir." Bu bölümde göreceğimiz yöntemler, yakınlık ve kestirime etkileri için kullandıkları farklı işlevlerle birbirinden ayrılır.



Şekil 8.1: Farklı kutucuk genişlikleri için çubukçizitler. '×', veri noktalarını gösterir.

8.2.1 Çubukçizit Kestirimi

En eski ve en yaygın yöntem olan *çubukçizit* kestiriminde girdi uzayı eşit boydaki *kutucuklara* bölünür. Bir sıfır noktası x_o ve kutu genişliği h verildiğinde kutucuklar, artı ve eksi m tamsayıları için, $[x_o+mh, x_o+(m+1)h)$ aralıklarını kapsar ve kestirim, kutucuk içine düşün örneklerin oranıdır:

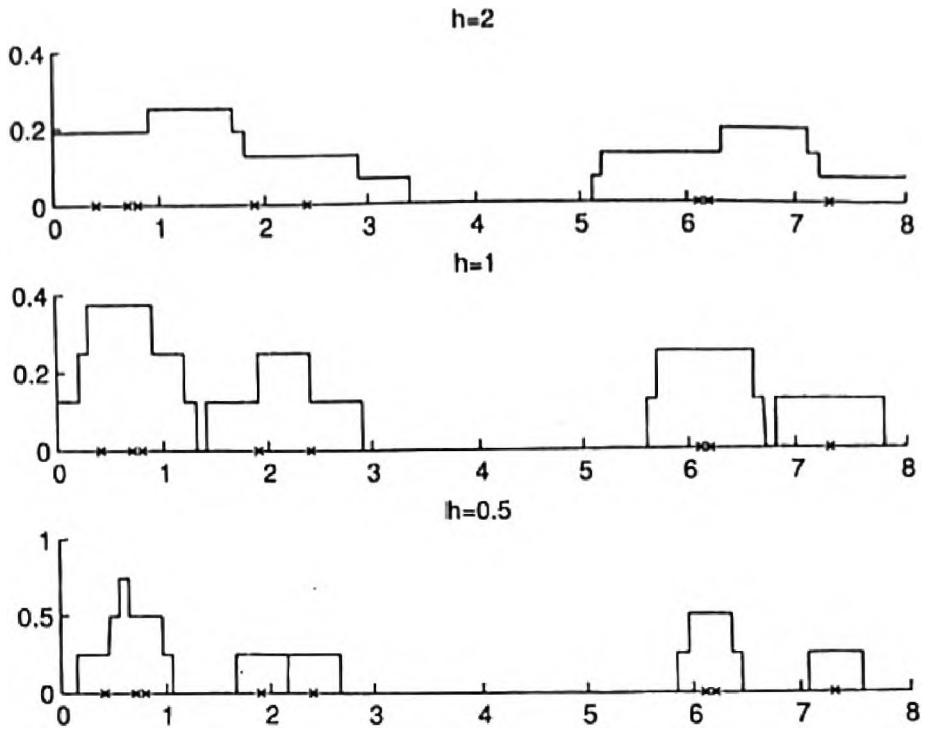
$$\hat{p}(x) = \frac{\#\{x\text{'le aynı kutucuğa düşen } x^t\}}{Nh} \tag{8.3}$$

Çubukçiziti oluşturmadan önce sıfır noktası ve kutucuk genişliğini belirlemek gerekir. Sıfır noktası kutucuk sınırları yakınındaki kestirimi etkiler ama kutu genişliğinin kestirime etkisi daha fazladır. Küçük kutucuklarla kestirim fazla dalgalı, büyük kutucuklarla fazla düzleştirilmiş olabilir (bkz. şekil 8.1). Eğer bir kutucuğa örnek düşmezse o aralıktaki kestirim 0 olur; bir başka sakınca da kutucuk sınırlarındeki süreksizliklerdir. Yine de çubukçizitin üstünlüğü, bir kez kutucuklar için kestirimler hesaplanınca artık öğrenme kümesini saklamaya gerek kalmamasıdır.

Saf kestirici (Silverman 1986) için sıfır noktası belirlemeye gerek yoktur:

$$\hat{p}(x) = \frac{\#\{x - h/2 < x^t \leq x + h/2\}}{Nh} \tag{8.4}$$

Bu yaklaşım, x girdisinin her zaman h genişliğinde bir kutucuğun ortasına geldiği bir çubukçizit kestiricisine eşittir (bkz. şekil 8.2). Kestirim, öğrenme örneklerinin



Şekil 8.2: Farklı kutucuk genişlikleri için saf kestiriciler.

ağırlıklı etkilerinin bir toplamı olarak da yazılabilir:

$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^{N} w\left(\frac{x - x^t}{h}\right) \tag{8.5}$$

Ağırlık işlevi, örneğin yakın olup olmamasına göre kesikli olarak 1 ya da 0 değeri alır:

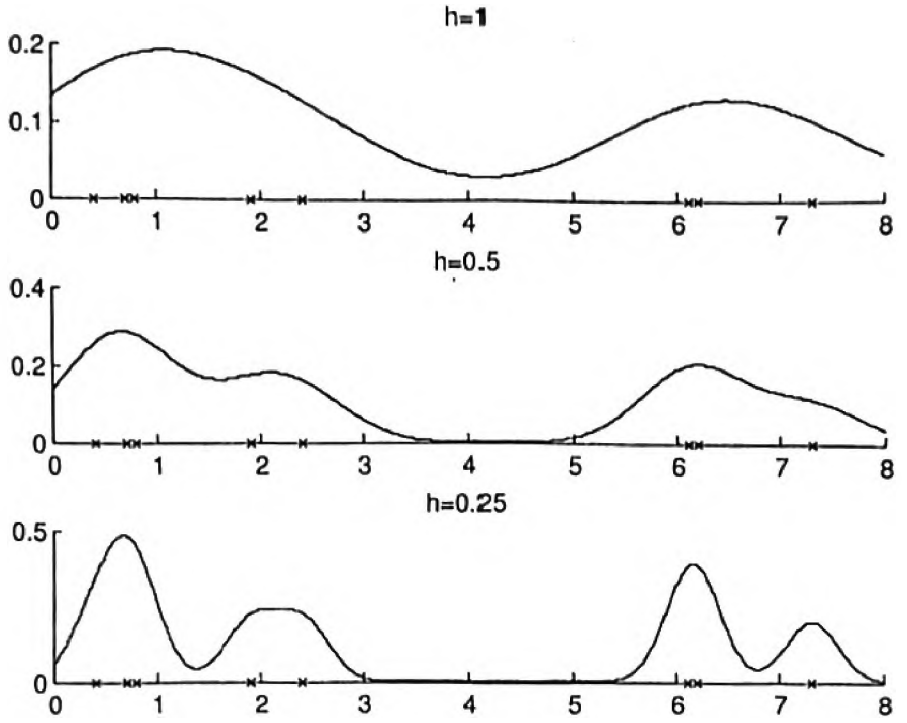
$$w(u) = \begin{cases} 1 & \text{eğer } |u| < 1/2 \\ 0 & \text{değilse} \end{cases}$$

Her x^t örneği çevresinde h genişliğinde bir etki alanı tanımlar ve bu alana düşen x girdisi için ağırlığı 1, dışarıda ise ağırlığı 0 olur. Dağılım bağımsız kestirim, x girdisinin etki alanına düştüğü x^t öğrenme örneklerinin ağırlıklarının toplamıdır. Bu etki alanı, "keskin" (0 ya da 1) olduğu için kestirim sürekli bir işlev değildir ve $x^t \pm h/2$ noktalarında sıçramalar gösterir.

8.2.2 Çekirdek Kestiricisi

Sınırlarda sıçramalar yapmayan daha düzleştirilmiş bir kestirim için *çekirdek işlevi* denilen eşiksiz bir ağırlık işlevi kullanırız. En sık kullanılan Gauss çekirdeğidir:

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{u^2}{2}\right] \tag{8.6}$$



Şekil 8.3: Farklı pencere genişlikleri için çekirdek kestiriciler.

Çekirdek kestiricisine, *Parzen penceresi* de denir:

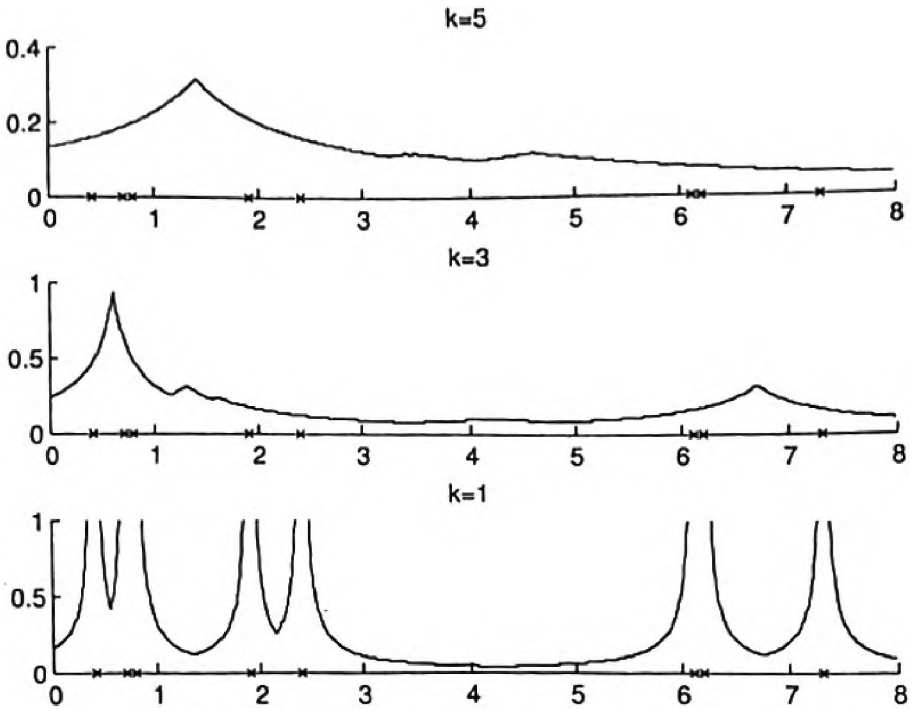
$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^{N} K\left(\frac{x - x^t}{h}\right) \tag{8.7}$$

$K(\cdot)$ çekirdek işlevi ağırlıkların biçimini, pencere genişliği de etki alanı genişliğini belirler. Saf kestirim "kutucukların" toplamıysa çekirdek kestirimi de "tümseklerin" toplamıdır. Tüm x^t örneklerinin x noktasındaki kestirime etkisi vardır ve bu etki $|x - x^t|$ arttıkça azalır.

Hesabı basitleştirmek için $|x - x^t| > 3h$ olduğunda $K(\cdot)$ 0 alınabilir. Hesaplaması daha basit başka çekirdekler de kullanılabilir; tek kısıt $K(u)$ değerinin $u = 0$ için en büyük olması ve $|u|$ artarken azalmasıdır.

h küçükken her öğrenme örneğinin daha küçük bir alanda daha fazla etkisi vardır ve uzak noktalarda etkisi yoktur. h büyüdükçe çekirdekler daha fazla örtüşür ve daha düzleştirilmiş bir kestirim oluşur (bkz. şekil 8.3). Eğer $K(\cdot)$ hiçbir yerde eksi değilse ve integrali 1 ise, yani geçerli bir dağılımsa, $\hat{p}(\cdot)$ de öyle olacaktır. Üstelik, $\hat{p}(\cdot)$ çekirdeğin tüm süreklilik ve türev alınabilirlik özelliklerini taşır, örneğin, $K(\cdot)$ Gauss ise, $\hat{p}(\cdot)$ tüm türevleri tanımlı düzleştirilmiş bir işlev olur.

Burada bir sorun tüm girdi uzayında pencere genişliğinin aynı olmasıdır. h değerini x çevresindeki dağılıma göre uyarlayan farklı yöntemler önerilmiştir.



Şekil 8.4: Farklı k değerleri için en yakın k komşu kestiricileri.

8.2.3 En Yakın k Komşu Kestirimi

En yakın komşu kestiricileri düzleştirmeyi verinin *yerel* yoğunluluğuna göre uyarlar. Ne kadar düzleştirildiği göz önüne alınan komşu sayısını gösteren k parametresine bağlıdır; bu değer, örneklem büyüklüğü olan N değerinden çok küçüktür. İki veri noktası, a ve b arasında bir uzaklık ölçütümüz, örneğin, $|a-b|$, tanımlı olsun ve her x için

$$d_1(x) \leq d_2(x) \leq \cdots \leq d_N(x)$$

biçiminde öğrenme örneklerini x noktasına olan uzaklıklarına göre artan sırada dizelim, öyle ki $d_1(x)$ en yakın örneğe olan uzaklık, $d_2(x)$ ikinci en yakın örneğe uzaklık, vs olsun. x^t örnek verileri gösterdiğinde, $d_1(x) = \min_t |x - x^t|$ olur, ve i, en yakın örneğin dizin değeriyse, $i = \arg\min_t |x - x^t|$, $d_2(x) = \min_{j \neq i} |x - x^j|$ olur, vs.

En yakın k komşu olasılık kestiricisini tanımlayalım:

$$\hat{p}(x) = \frac{k}{2Nd_k(x)} \tag{8.8}$$

Saf kestiricide kutucuk genişliğin en yakın k. komşuya uzaklık $h = 2d_k(x)$ olarak alsak aynı kestiriciyi elde ederiz. Pencere genişliği olan h değerini sabitleyip içine kaç öğrenme örneği düşüyor diye saymak yerine, içine düşmesi gereken örnek sayısını k olarak sabitleyip bu kadar örneğin içine düşmesi için pencerenin ne kadar geniş olması gerektiğine bakıyoruz. Yoğunluğun yüksek olduğu yerlerde pencereler dar, düşük olduğu yerlerde pencereler geniş olacaktır (bkz. şekil 8.4).

En yakın k komşu kestirimi sürekli değildir; türevi tüm $\frac{1}{2}(x^{(j)} + x^{(j+k)})$ noktalarında süreksizdir ($x^{(j)}$ sıra istatistiklerini gösterir). En yakın k komşu kestirimi bir olasılık dağılımı üretmez, çünkü integrali 1 değil ∞ olur.

Daha düzleştirilmiş bir kestirim için uzaklaştıkça etkisi azalan bir çekirdek kullanılabilir:

$$\hat{p}(x) = \frac{1}{N d_k(x)} \sum_{t=1}^{N} K\left(\frac{x - x^t}{d_k(x)}\right) \tag{8.9}$$

Bu, $h = d_k(x)$ biçiminde pencere genişliği veriye uyarlanan bir çekirdek kestiricisi olarak görülebilir. $K(\cdot)$, sıklıkla Gauss çekirdeği olarak alınır.

8.3 Çok değişkenli Veriye Genelleme

d boyutlu örneklerden oluşan $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$ kümesi verildiğinde çok boyutlu çekirdek kestirimi

$$\hat{p}(\boldsymbol{x}) = \frac{1}{N h^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}^t}{h}\right) \tag{8.10}$$

olarak yazılır; tek koşul

$$\int_{\Re^d} K(\boldsymbol{x}) d\boldsymbol{x} = 1$$

olmasıdır. Hemen akla gelen aday çok değişkenli Gauss çekirdeğidir:

$$K(\boldsymbol{u}) = \left(\frac{1}{\sqrt{2\pi}}\right)^d \exp\left[-\frac{\|\boldsymbol{u}\|^2}{2}\right] \tag{8.11}$$

Dağılım bağımsız kestiricileri çok boyutlu uzaylarda kullanırken *çok boyutluluğun lanetinden* sakınmak gerekir: $\boldsymbol{x}$ girdileri sekiz boyutlu olsun ve her boyutu ona böldüğümüzü düşünelim. Bu durumda 10^8 tane kutucuk olur ve çok verimiz yoksa bu kutucukların çoğu boş kalacak ve içlerindeki kestirim 0 olacaktır. Yüksek boyutlarda "yakınlık" kavramı da bulanıklaşır ve h değerini seçerken dikkat gerekir.

Denklem 8.11'de Euclid uzaklığını kullanmak çekirdeğin tüm boyutlarda aynı ölçekte olması demektir ve eğer bu doğru değilse tüm boyutta aynı değişintili olacak biçimde girdilerin önceden normalleştirilmesi gerekir. Ama bu da ilintileri göz önüne almaz; çekirdek veriyle aynı özellikler taşıdığında daha iyi sonuçlar alınacaktır:

$$K(\boldsymbol{u}) = \frac{1}{(2\pi)^{d/2}|\mathbf{S}|^{1/2}} \exp\left[-\frac{1}{2}\boldsymbol{u}^T \mathbf{S}^{-1} \boldsymbol{u}\right] \tag{8.12}$$

Burada $\mathbf{S}$, örneklem eşdeğişinti dizeyidir ve yakınlık için Euclid yerine Mahalanobis uzaklığını kullanmaya karşılık gelir.

Uzaklık ölçütü yerel de olabilir ve bunun için $\mathbf{S}$, $\boldsymbol{x}$ girdisinin çevresindeki örneklerle, örneğin, en yakın k komşusu kullanılarak hesaplanır. Yerel hesaplanan $\mathbf{S}$ dizeyi

tekil olabilir ve temel bileşen çözümlemesi (ya da sınıflandırmada, doğrusal ayırtaç çözümlemesi) kullanmak gerekebilir.

Girdiler kesikli olduğunda, aynı olmayan öznitelikleri sayan *Hamming uzaklığı* kullanılabilir:

$$HD(\boldsymbol{x}, \boldsymbol{x}^t) = \sum_{j=1}^{d} 1(x_j \neq x_j^t) \tag{8.13}$$

öyle ki

$$1(x_j \neq x_j^t) = \begin{cases} 1 & \text{eğer } x_j \neq x_j^t \\ 0 & \text{değilse} \end{cases}$$

$HD(\boldsymbol{x}, \boldsymbol{x}^t)$, uzaklık ölçütü olarak $\|\boldsymbol{x} - \boldsymbol{x}^t\|$ ya da $(\boldsymbol{x} - \boldsymbol{x}^t)^T \mathbf{S}^{-1} (\boldsymbol{x} - \boldsymbol{x}^t)$ yerine çekirdek içinde, ya da en yakın k komşuyu bulurken kullanılır.

8.4 Dağılımdan Bağımsız Sınıflandırma

Sınıflandırmada, sınıf koşullu $p(\boldsymbol{x}|C_i)$ dağılımlarını kestirmekte dağılımdan bağımsız bir yaklaşım kullanabiliriz. Çekirdek kestiricisi kullanırken

$$\hat{p}(\boldsymbol{x}|C_i) = \frac{1}{N_i h^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}^t}{h}\right) r_i^t \tag{8.14}$$

burada, $\boldsymbol{x}^t \in C_i$ ise $r_i^t = 1$, değilse $r_i^t = 0$ olur. N_i, C_i sınıfından örnek sayısıdır: $N_i = \sum_t r_i^t$. Öncel olasılık için en büyük olabilirlik kestirimi $\hat{P}(C_i) = N_i/N$ olduğuna göre ayırtaç

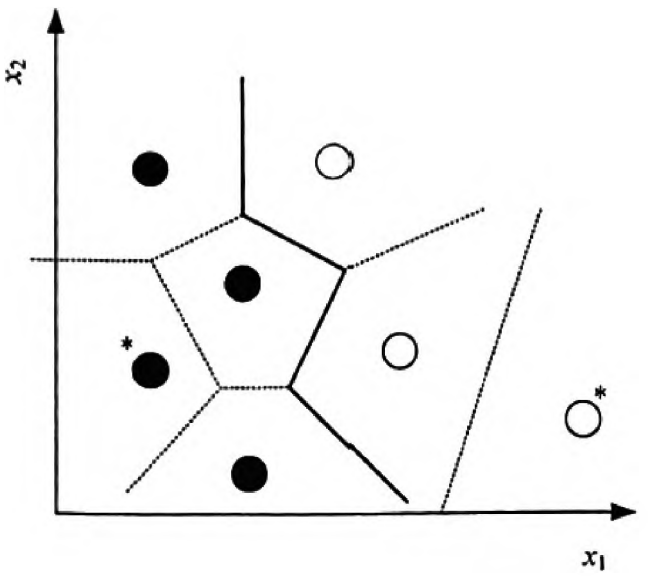
$$\begin{aligned} g_i(\boldsymbol{x}) &= \hat{p}(\boldsymbol{x}|C_i)\hat{P}(C_i) \\ &= \frac{1}{Nh^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}^t}{h}\right) r_i^t \end{aligned} \tag{8.15}$$

olarak yazılır ve $\boldsymbol{x}$ girdisi için ayırtaç değeri en yüksek olan sınıf seçilir. Ortak çarpan $1/(Nh^d)$ göz ardı edilebilir. Her öğrenme örneği kendi sınıfı için oy verir ve öteki sınıflara etkisi yoktur; verdiği oyun ağırlığı, çekirdek $K(\cdot)$ işlevine göre, genelde yakın örneklere daha fazla ağırlık verecek biçimde belirlenir.

En yakın k komşu kestirimi de kullanılabilir:

$$\hat{p}(\boldsymbol{x}|C_i) = \frac{k_i}{N_i V^k(\boldsymbol{x})} \tag{8.16}$$

Burada, k_i, k komşu içinde C_i sınıfından olanların sayısı, $V^k(\boldsymbol{x})$ ise merkezi $\boldsymbol{x}$, yarıçapı ($\boldsymbol{x}_{(k)}$, tüm sınıflardan tüm komşular içinde $\boldsymbol{x}$ örneğinin k. en yakın komşusunu gösterirken) $r = \|\boldsymbol{x} - \boldsymbol{x}_{(k)}\|$ olan d boyutlu üstünkürenin hacmidir: $V^k = r^d c_d$



Şekil 8.5: Noktalı çizgiler Voronoi mozaiğini, düz çizgi de sınıf ayırtacını gösterir. Seyrek en yakın komşu algoritmasında ayırtacı tanımlamayan ('*' ile işaretlenmiş) örnekler öğrenme hatasını etkilemez ve atılabilir.

olduğunda c_d, d boyutlu birimkürenin hacmini gösterir; örneğin, $c_1 = 2, c_2 = \pi, c_3 = 4\pi/3$, vs. Bu durumda sınıf onsal olasılığı basitleşir:

$$\hat{P}(C_i|\boldsymbol{x}) = \frac{\hat{p}(\boldsymbol{x}|C_i)\hat{P}(C_i)}{\hat{p}(\boldsymbol{x})} = \frac{k_i}{k} \tag{8.17}$$

En yakın k komşu sınıflandırıcısı, girdiyi, k komşu içinde en fazla örneği olan sınıfa atar. Tüm komşuların oyu eşittir ve k komşudan en fazla oy alan sınıf seçilir. Eşitlikte herhangi biri seçilebilir, ya da ağırlıklı oylama alınabilir. k genelde tek sayı alınır çünkü eşitlikler daha çok iki sınıf arasında olur.

Bir kez daha yinelemek gerekirse Euclid uzaklığının kullanılması girdilerin eşit değişintili ve ilintisiz olduğu varsayımına karşılık gelir ve bu sağlanmıyorsa uygun bir ölçüt kullanmak gerekir; bu örneğin, yerel ve gözetimli biçimde hesaplanabilir (Hastie ve Tibshirani 1996).

Bir özel örnek, $k = 1$ olan *en yakın komşu sınıflandırıcısıdır*; girdi ona en yakın örneğin sınıfına atanır. Bu karar kuralı girdi uzayını, *Voronoi mozaiği* denilen bir biçimde parçalara ayırır (bkz. şekil 8.5).

8.5 Seyrek En Yakın Komşu

Dağılımdan bağımsız yöntemlerin zaman ve bellek karmaşıklığı öğrenme kümesinin büyüklüğüyle doğru orantılıdır ve sınıflandırma başarısını azaltmadan öğrenme kümesini küçültmek için *seyreltme* yöntemleri önerilmiştir. $\mathcal{X}$ örnekleminin öyle bir $\mathcal{Z}$ altkümesini bulmak isteriz ki hem $\mathcal{Z}$ olabildiğince küçük olsun, hem de $\mathcal{X}$ yerine kullanıldığında hata yaratmasın (Dasarathy 1991).

$\mathcal{Z} \leftarrow \emptyset$
Yinele
  Tüm $x \in \mathcal{X}$ (rastgele sırada)
    Bul $x' \in \mathcal{Z}$ öyle ki $\|x - x'\| = \min_{x^j \in \mathcal{Z}} \|x - x^j\|$
    Eğer sınıf(x)$\neq$sınıf(x') ise $\mathcal{Z} \leftarrow \mathcal{Z} \cup x$
$\mathcal{Z}$ değişmeyene dek

Şekil 8.6: Seyrek en yakın komşu algoritması.

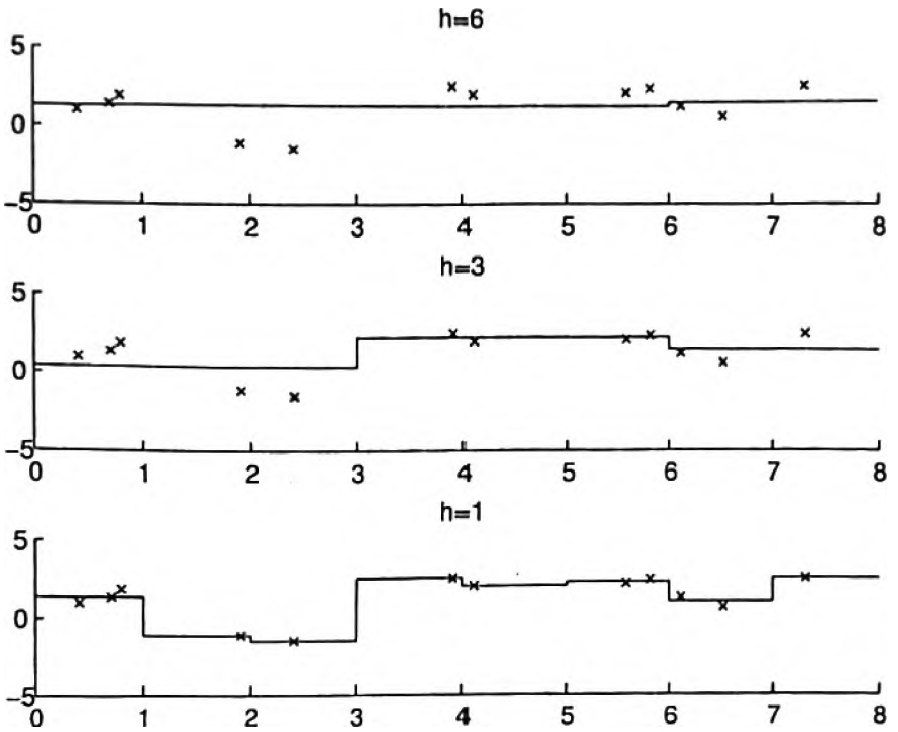
En erken ve yaygın bilinen yöntem olan *seyrek en yakın komşu*, sınıflandırma için en yakın komşu algoritmasını kullanır (Hart 1968). En yakın komşu, ayırtacı parçalı doğru biçimde kestirir ve yalnızca ayırtacı tanımlayan örneklerin saklanması yeter; sınıf bölgeleri içindeki bir örneğin komşuları kendisiyle aynı sınıftansa saklanmasına gerek yoktur ve atılması (öğrenme kümesi üzerinde) hataya neden olmayacaktır (şekil 8.5). Böyle bir altkümeye tutarlı altküme denir ve amacımız tutarlı en küçük altkümeyi bulmaktır.

Hart, $\mathcal{Z}$ altkümesini bulmak için açgözlü bir algoritma önermiştir (şekil 8.6). Algoritma, boş küme $\mathcal{Z}$ ile başlar. $\mathcal{X}$ içindeki örneklerden rastgele sırayla birer birer geçerken her örnek, yalnızca $\mathcal{Z}$ altkümesindeki örnekler içinden en yakın komşuları kullanıldığında doğru sınıflandırılır mı diye bakar. Eğer yanlış sınıflandırılıyorsa o örnek de $\mathcal{Z}$ altkümesine eklenir; eğer doğru sınıflandırılıyorsa bir şey yapılmaz. Artık hiçbir örnek eklenmemesi için $\mathcal{X}$ üzerinden birkaç kez geçmek gerekir. Bu algoritma yerel bir arama yaptığından öğrenme kümesi üzerinden farklı sırada geçildiğinde, hepsi öğrenme kümesi üzerinde hatasız olsa da geçerleme kümesi üzerinde farklı başarımı olan farklı altkümeler bulunabilir. Bu da, bu algoritmanın en küçük tutarlı altkümeyi bulmadığını gösterir ama zaten bunun çokterimli zamanda bulunamadığı gösterilmiştir (Wilfong 1992).

Seyrek en yakın komşu, öğrenme kümesi üzerindeki hatayı ve saklanan altkümenin büyüklüğü olarak ölçülen karmaşıklığı birlikte azaltan açgözlü bir algoritmadır. Bu amaç için genişletilmiş bir hata işlevi yazabiliriz:

$$E'(\mathcal{Z}|\mathcal{X}) = E(\mathcal{X}|\mathcal{Z}) + \lambda|\mathcal{Z}| \tag{8.18}$$

Burada, $E(\mathcal{X}|\mathcal{Z})$, $\mathcal{Z}$ üzerinde $\mathcal{X}$ sınıflandırıldığında oluşan hatadır. $|\mathcal{Z}|$, $\mathcal{Z}$ altkümesinin büyüklüğüdür ve bu ikinci terim karmaşıklığı cezalandırır. Her düzenlileştirme uygulamasında olduğu gibi λ, hata ve karmaşıklık arasındaki ödünleşimi tanımlar. Küçük λ değeri için hata önemlidir; λ büyüdükçe karmaşık modeller daha fazla cezalandırılır. Seyrek en yakın komşu, denklem 8.18'i azaltmanın bir yoludur ama başka algoritmalar da düşünülebilir.



Şekil 8.7: Farklı kutucuk genişlikleri için çubukçizit düzleştirici sonuçları. '×', veri noktalarını gösterir.

8.6 Modelden Bağımsız Bağlanım: Düzleştiriciler

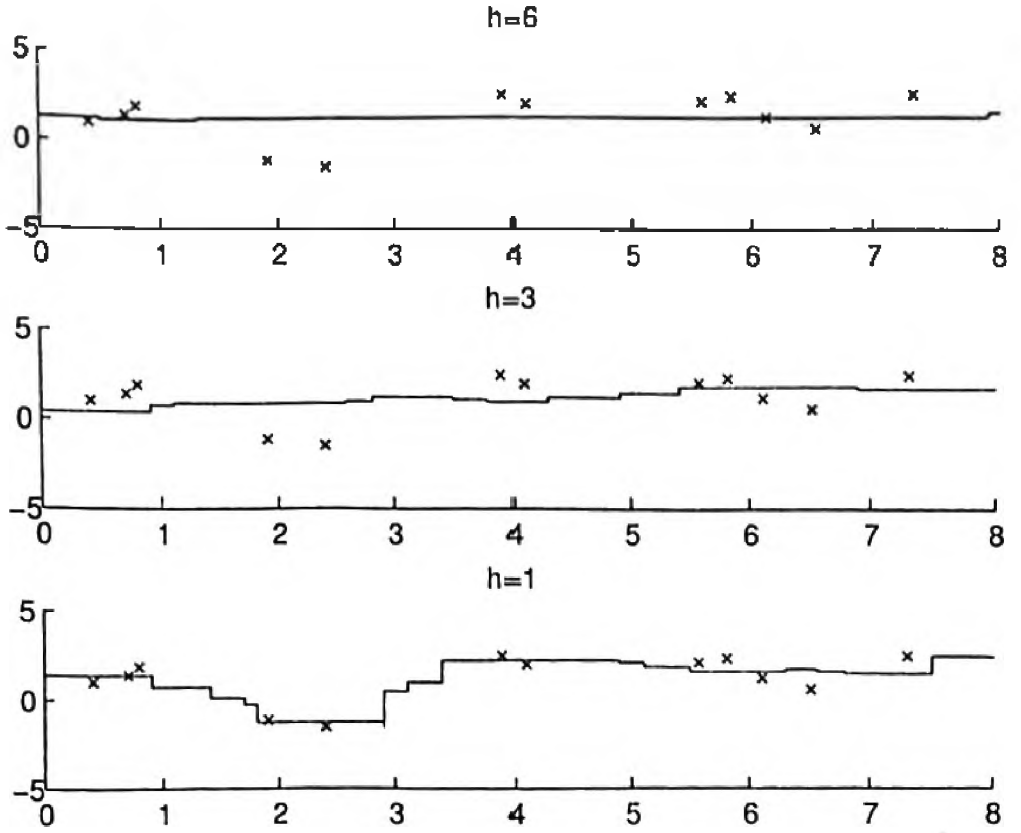
Bağlanımda $r^t \in \Re$ olduğunu hatırlayalım. $\mathcal{X} = \{x^t, r^t\}$ öğrenme kümesinin

$$r^t = g(x^t) + \epsilon$$

biçiminde oluştuğunu varsayarız. Model bağımlı bağlanımda, belirli bir dereceden bir çokterimli varsayar ve öğrenme kümesi üzerindeki kare hatayı en aza indiren katsayıları hesaplarız. Model bağımsız bağlanımda böyle bir çokterimli, ya da az sayıda parametreyle tanımlanabilen başka bir model varsayamayız; yalnızca benzer x girdilerinin benzer $g(x)$ değerleri olduğunu varsayabiliriz. Dağılımdan bağımsız olasılık kestiriminde yaptığımız gibi x verildiğinde onun komşuluğunu bulur ve bu komşuluk içindeki örneklerin r çıktı değerlerini kullanarak $\hat{g}(x)$ değerini kestiririz. Modelden bağımsız bağlanıma *düzleştirici* de denir (Härdle 1990). Komşuluğu ve bu komşuluk içinde kestirimi bulmak için, olasılık kestirimindeki gibi, farklı yöntemler vardır. Burada tek boyutlu x kullanıyoruz, ama çok boyutlu çekirdeklerle kolaylıkla çok değişkenli bağlanıma genelleyebiliriz.

8.6.1 Akan Ortalama Düzleştiricisi

Olasılık kestiriminde yaptığımız gibi bir sıfır noktası ve kutucuk genişliği tanımlayıp her kutucuk içindeki örneklerin r değerlerinin ortalamasını kullanırsak *çubukçizit*



Şekil 8.8: Farklı kutucuk genişlikleri için akan ortalama düzleştiricileri.

düzleştiricisini elde ederiz (bkz. şekil8.7)

$$\hat{g}(x) = \frac{\sum_{t=1}^{N} b(x, x^t) r^t}{\sum_{t=1}^{N} b(x, x^t)} \tag{8.19}$$

öyle ki

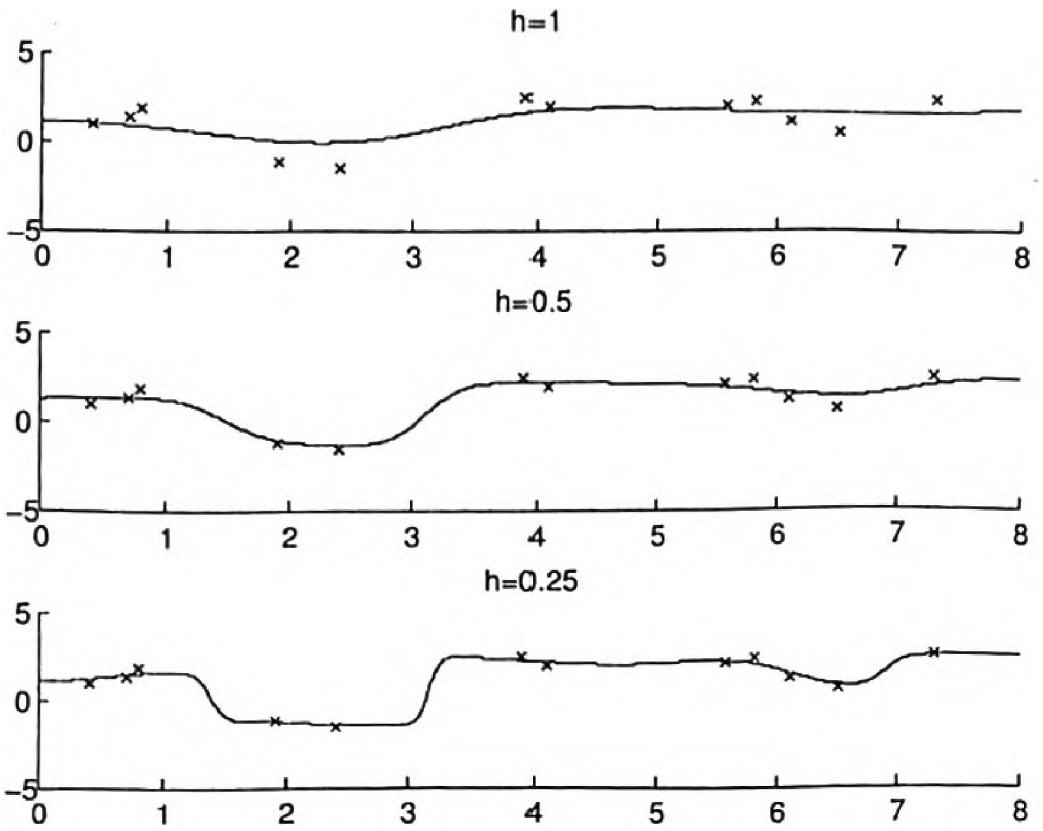
$$b(x, x^t) = \begin{cases} 1 & \text{eğer } x^t, x \text{ ile aynı kutucuktaysa} \\ 0 & \text{değilse} \end{cases}$$

Kutucuk sınırlarındaki süreksizlikler ve bir sıfır noktası belirlemek gerekmesi can sıkıcıdır. Saf kestiricideki gibi, *akan ortalama düzleştiricide*, x girdisini merkez alan bir kutucuk belirler ve içinde ortalama alırız (şekil 8.8):

$$\hat{g}(x) = \frac{\sum_{t=1}^{N} w\left(\frac{x-x^t}{h}\right) r^t}{\sum_{t=1}^{N} w\left(\frac{x-x^t}{h}\right)} \tag{8.20}$$

Komşu ağırlığı 0/1 değer alan kesikli bir işlevdir:

$$w(u) = \begin{cases} 1 & \text{eğer } |u| < 1 \\ 0 & \text{değilse} \end{cases}$$



Şekil 8.9: Farklı genişlikler için Gauss çekirdek düzleştiricileri.

Bu yöntem zaman serilerindeki gibi eşit aralıklı veride yaygın olarak kullanılır. Gürültü olduğunda kutu içindeki r^t değerlerinin ortalaması yerine ortancasını da kullanabiliriz.

8.6.2 Çekirdek Düzleştiricisi

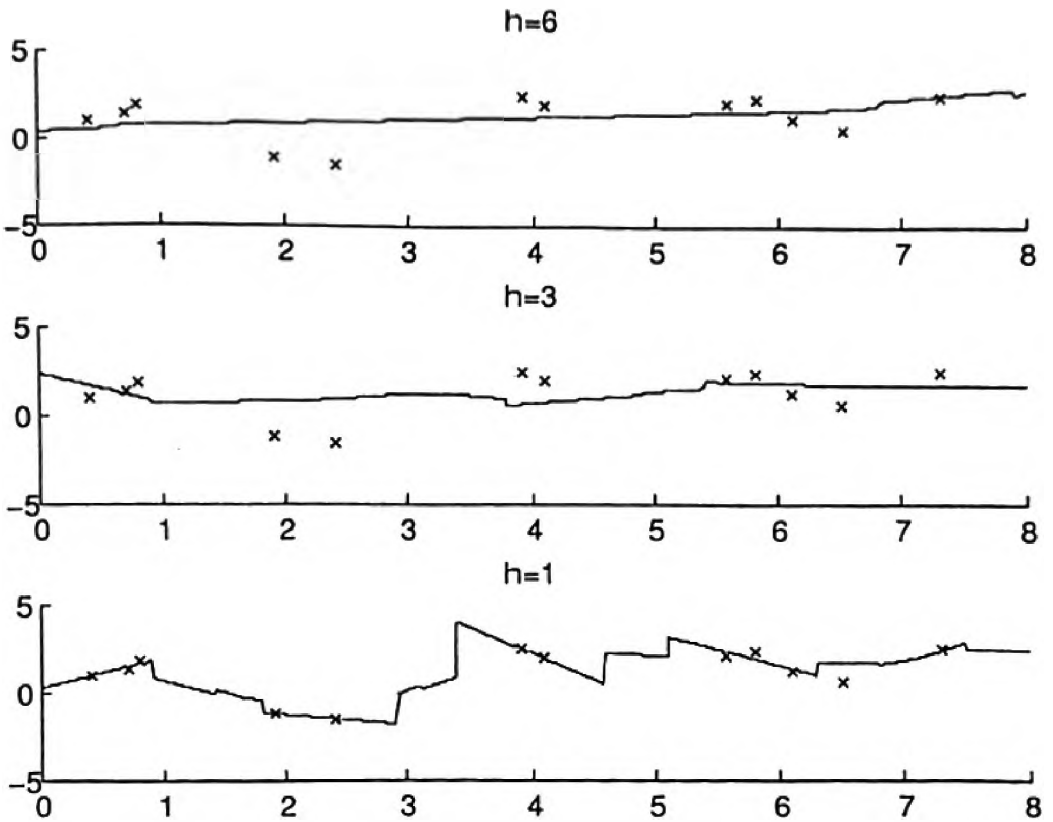
Olasılık kestiriminde yaptığımız gibi bir çekirdek işleviyle daha uzaktaki noktalara daha az ağırlık veren *çekirdek düzleştiricisini* kullanabiliriz (bkz. şekil 8.9):

$$\hat{g}(x) = \frac{\sum_t K\left(\frac{x-x^t}{h}\right) r^t}{\sum_t K\left(\frac{x-x^t}{h}\right)} \tag{8.21}$$

Genelde $K(\cdot)$, Gauss çekirdeği olarak alınır. h değerini sabitlemek yerine komşu sayısı olan k değeri sabitlenip düzleştirici, x çevresindeki veri yoğunluğuna uyarlanabilir; buna *en yakın k komşu düzleştiricisi* denir.

8.6.3 Akan Doğru Düzleştiricisi

Ortalama almak yerine Taylor açılımında bir terim daha kullanıp yerel doğrusal bir kestirim de yapılabilir. Buna *akan doğru düzleştiricisi* denir. h ya da k ile tanımlanan



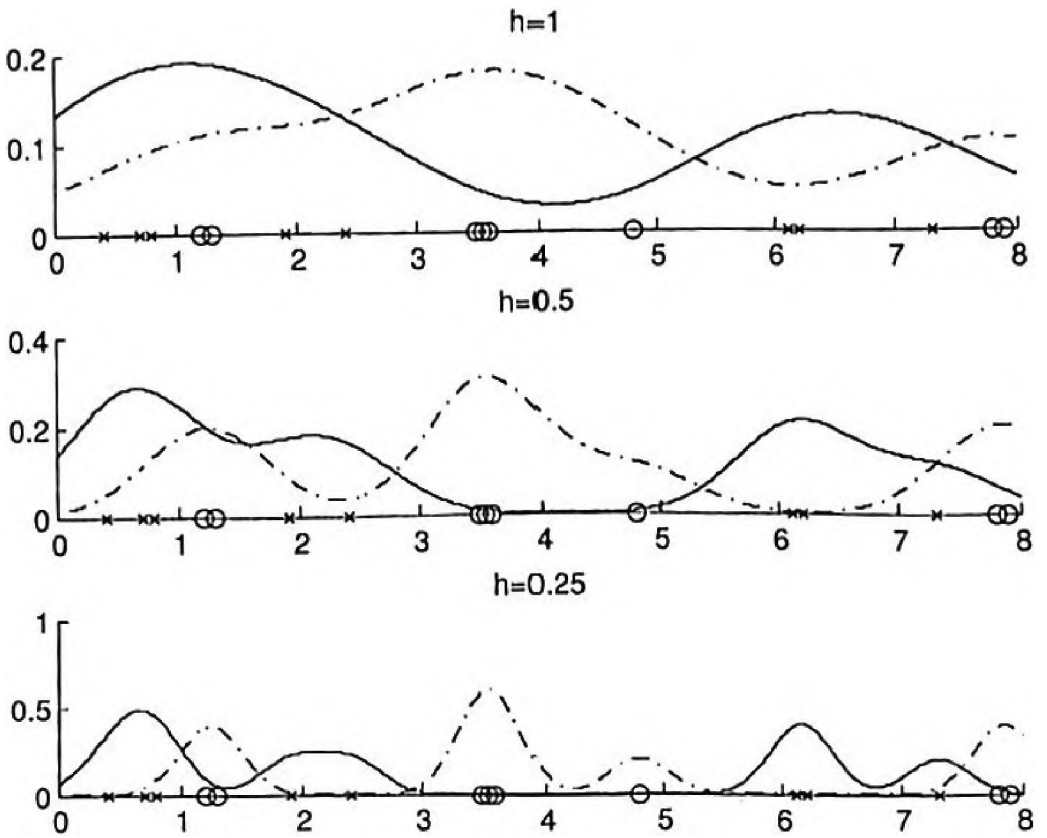
Şekil 8.10: Farklı genişlikler için akan doğru düzleştiricileri.

bir komşuluk içindeki noktaları kullanarak yerel bir doğrusal bağlanım hesaplarız (bkz. şekil 8.10).

Yerel ağırlıklandırılmış akan doğru düzleştiricide, komşuluğu keskin tanımlamak yerine esnek bir çekirdek işlevi kullanarak uzaktaki noktaların bağlanım hatasında daha az etkisi olmasını sağlarız.

8.7 Düzleştirme Parametresinin Seçimi

İster olasılık kestirimi, ister bağlanım için olsun, kestirimin en hassas olduğu parametre, kutucuk ya da çekirdek genişliğini gösteren h, ya da komşu sayısı k değeridir. Amacımız her zaman verinin değişkenliğinden daha az değişkenlik gösteren bir kestirim oluşturmaktır. Daha önce konuştuğumuz gibi verideki değişkenlik, altta yatan ve kestirmek istediğimiz işlevdeki değişkenlik yanında eklenen gürültüden dolayı da olabilir. Gürültünün etkisini yok edecek kadar düzleştirmeliyiz—ne daha az, ne daha fazla. Fazla büyük h ya da k değeri kullanırsak bir noktadaki kestirime çok uzaktaki örnekler de katılır ve işlevdeki değişkenliği de düzleştiririz; küçük h ya da k değerinde örneklerin birer birer etkisi fazla olur ve gürültüden kurtulamayız. Bir başka deyişle, küçük h ya da k ile yanlılık az ama değişinti yüksek olur. h ya da k değerini artırmaksa değikenliği azaltırken yanlılığı artırabilir. Geman, Bienenstock, ve Doursat (1992), yanlılık ve değişintiyi dağılım bağımsız kestiriciler için tartışır.



Şekil 8.11: İki sınıflı bir örnekte farklı genişlikler için çekirdek kestirimleri. Çizilenler $p(x|C_i)$ koşullu olasılıklarıdır. En üstteki fazla, en alttaki ise az düzleştirmiş görünüyor; en uygun olan h değeri geçerleme kümesindeki örneklerin nereden geldiğine bağlıdır.

Bu gereksinim, *düzleştiren eğrilerde* hata işlevine eklenmiştir:

$$\sum_t \left[r^t - \hat{g}(x^t)\right]^2 + \lambda \int_a^b [\hat{g}''(x)]^2 dx \tag{8.22}$$

İlk terim, kestirimin hatasıdır. $[a, b]$, girdi aralığı, $\hat{g}''(\cdot)$, kestirilmiş $\hat{g}(\cdot)$ işlevinin eğrilik derecesidir ve değişkenliği ölçer. Eklenmiş bu ikinci terimle hızlı değişen kestirimleri cezalandırmayı amaçlarız. λ, değişkenlikle hata arasındaki ödünleşimi belirler ve örneğin λ büyüdükçe, daha düz kestirimler elde ederiz.

h, k, ya da λ değerini belirlemek için çapraz geçerleme kullanılır. Olasılık kestiriminde geçerleme kümesinin olabilirliğini en yüksek yapan parametre değerini seçeriz. Gözetimli bir uygulamada, farklı parametre değerleriyle öğrenme kümesi üzerinde model geliştirip geçerleme kümesinde en düşük hatayı vereni seçeriz (bkz. şekil 8.11).

8.8 Kısa Kısa

En yakın k komşu ve çekirdek tabanlı kestirim algoritmaları önerileli elliden fazla yıl geçmiş olmasına rağmen yakın zamana dek büyük bellek ve hesap gereksinimi

yüzünden sık kullanılamazlardı (Aha, Kibler, ve Albert 1991). Koşut işlemin gelişmesi ve bilgisayarların ucuzlamasıyla bu yöntemler gittikçe daha yaygın olarak kullanılıyor. Dağılımdan bağımsız kestirim hakkında daha fazla bilgi (Silverman 1986) ve (Scott 1992) gibi ders kitaplarında, ya da (Dasarathy 1991) ve (Aha 1997) gibi makale derlemelerinde bulunabilir.

Dağılımdan bağımsız yöntemler, tek komutlu çok verili koşut bilgisayarlar üstünde kolayca programlanabilir; her işlemci bir öğrenme örneğini saklar ve öteki işlemcilerle aynı anda verilen bir girdi için çekirdek değerini hesaplayabilir (Stanfill ve Waltz 1986). Çekirdek işleviyle çarpmak bir evrişim işlemi olarak görülebilir ve Fourier dönüşümüyle kestirim kolayca hesaplanabilir (Silverman 1986). Düzleştirici eğrilerle çekirdek düzleştirmenin benzeştiği de gösterilmiştir.

Dağılımdan bağımsız kestirimde önemli bir etken kullanılan uzaklık ölçütüdür. Kesikli öznitelikler için eşleşmeyen öznitelikleri saydığımız Hamming uzaklığı kullanılabilir. Daha gelişmiş uzaklık ölçütleri (Wettschereck, Aha, ve Mohri 1997) ve (Webb 1999)'da tartışılır.

Yapay zekâda, dağılımdan bağımsız yaklaşıma *örnek tabanlı akıl yürütme* denir. Çıktı, geçmiş benzer "örneklerden" ara değerlemeyle bulunur. Böylece bilgi çıkarımı da yapılabilir: verilen çıktı benzer geçmiş örnekler listelenerek haklı çıkarılabilir.

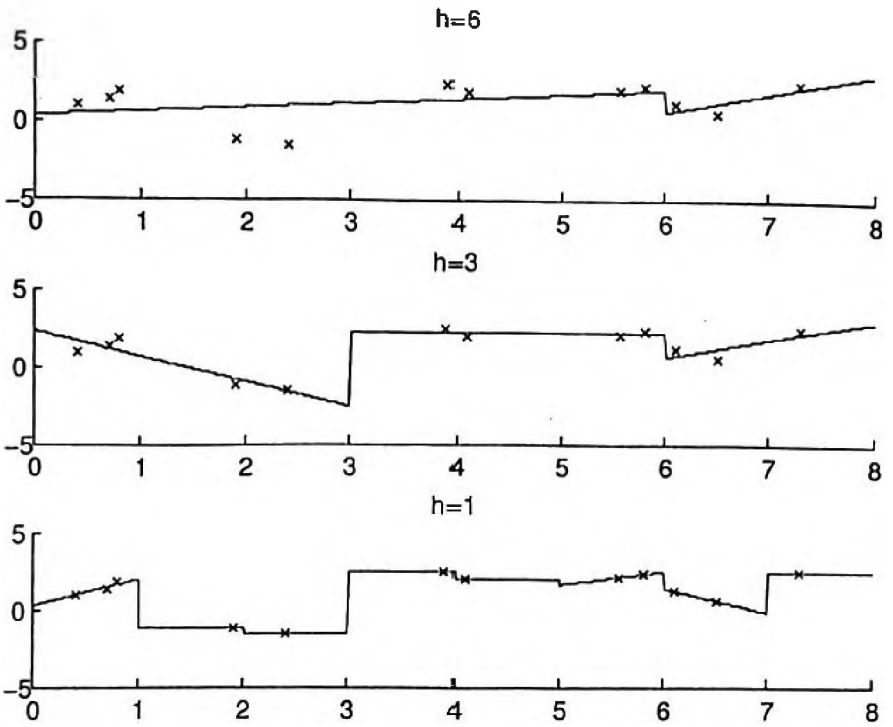
En yakın k komşu algoritması basitliğinden dolayı en fazla kullanılan algoritmalardan biridir ve birçok farklı uygulamada oldukça başarılıdır. $N \to \infty$, yani örnek kümesi çok büyükken, en yakın komşu ($k = 1$) algoritmasının riskinin hiçbir zaman Bayes riskinin (olabilecek en düşük risk) iki katından daha kötü olmadığı gösterilmiştir (Cover ve Hart 1967; Duda, Hart, ve Stork 2001) ve bundan dolayı, "'bir veri kümesindeki bilginin yarısı en yakın komşudadır," denir (Cover ve Hart 1967, 21). Atalarımız, "Bana komşunu söyle, sana kim olduğunu söyleyeyim," demiştir. En yakın k komşu algoritmasında k sonsuza giderken risk, Bayes riskine yakınsar.

Düzeltici bağlanım yöntemleri (Härdle 1990)'da ayrıntılı anlatılır. Hastie ve Tibshirani (1990), düzeltici modelleri tartışır ve çok değişkenli bir işlevin birden çok tek değişkenli işlevin toplamı olarak yazılabildiği *toplanır modelleri* önerir. Yerel ağırlıklı bağlanım, (Atkeson, Moore, ve Schaal 1997)'de tartışılır. Bu yöntemler, 12. bölümde göreceğimiz dairesel taban işlevleri ve uzmanların karışımı modellerine çok benzer.

Seyrek en yakın komşu algoritması öğrenme örneklerinin yalnızca bir altkümesini, sınıra yakın olup ayırtacı tanımlayanları saklar. Bu düşünce, 13. bölümdeki *destek yöney makinelerinde* de vardır. Orada, örnekler arasındaki benzerliği ölçmek için farklı çekirdek işlevleri ve en iyisinin nasıl seçilebildiğini de göreceğiz. Çıktıyı öğrenme örneklerinin etkilerinin birleşimi olarak yazma düşüncesini, çekirdek işlevi yerine *eşdeğişinti işlevi* adının kullanıldığı *Gauss süreçlerinde* (bölüm 14) görürüz.

8.9 Alıştırmalar

1. Olasılık kestiriminde daha düzleştirilmiş bir çubukçiziti nasıl oluşturabiliriz?
2. Denklem 8.17'yi gösterin.
3. $k > 1$ için seyrek en yakın komşu algoritması nasıl davranır?



Şekil 8.12: Çubukçizit düzleştiricide her kutuda ortalama almak yerine bir doğru da hesaplayabiliriz.

4. Seyrek en yakın komşu algoritmasında $\mathcal{Z}$ altkümesine daha önce eklenmiş bir örnek, başka örnekler eklendikten sonra artık gerekmeyebilir. Böylesi gereksiz örnekleri nasıl bulup atabiliriz?
5. Çubukçizit düzleştiricide ortalama almak yerine doğrusal bir model de oturtabiliriz (bkz. şekil 8.12). Bunun için programı yazın ve sonucu ortalama almakla karşılaştırın.
6. 8.6.3. bölümde gördüğümüz yerel ağırlıklandırılmış akan doğru düzleştirici için hata işlevini yazın.
7. Akan ortalama düzleştiricisi için, seyrek en yakın komşu benzeri, örneklerin gerektiğinde eklendiği artımlı bir algoritma önerin.
8. Çekirdek düzleştiricisini çok değişkenli veri için genelleyin.
9. Akan düzleştiricide her deneme noktasında komşuluğa sabit, doğru, ya da daha üst dereceden bir çokterimli oturtabiliriz. Aralarında nasıl seçim yapmalıyız?
10. Akan ortalama kestiricide, bir değer yanında kestirimimizin değişintisini gösteren bir güven aralığı hesaplayabilir miyiz?

8.10 Kaynaklar

Aha, D. W., ed. 1997. Special Issue on Lazy Learning, *Artificial Intelligence Review* 11(1–5): 7–423.

Aha, D. W., D. Kibler, M. K. Albert. 1991. "Instance-Based Learning Algorithm." *Machine Learning* 6: 37–66.

Atkeson, C. G., A. W. Moore, S. Schaal. 1997. "Locally Weighted Learning." *Artificial Intelligence Review* 11: 11–73.

Cover, T. M., P. E. Hart. 1967. "Nearest Neighbor Pattern Classification." *IEEE Transactions on Information Theory* 13: 21–27.

Dasarathy, B. V. 1991. *Nearest Neighbor Norms: NN Pattern Classification Techniques.* Los Alamitos, CA: IEEE Computer Society Press.

Duda, R. O., P. E. Hart, D. G. Stork. 2001. *Pattern Classification*, 2nd ed. New York: Wiley.

Geman, S., E. Bienenstock, R. Doursat. 1992. "Neural Networks and the Bias/Variance Dilemma." *Neural Computation* 4: 1–58.

Härdle, W. 1990. *Applied Nonparametric Regression.* Cambridge, UK: Cambridge University Press.

Hart, P. E. 1968. "The Condensed Nearest Neighbor Rule." *IEEE Transactions on Information Theory* 14: 515–516.

Hastie, T. J., R. J. Tibshirani. 1990. *Generalized Additive Models.* London: Chapman and Hall.

Hastie, T. J., R. J. Tibshirani. 1996. "Discriminant Adaptive Nearest Neighbor Classification." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18: 607–616.

Scott, D. W. 1992. *Multivariate Density Estimation.* New York: Wiley.

Silverman, B. W. 1986. *Density Estimation in Statistics and Data Analysis.* London: Chapman and Hall.

Stanfill, C., D. Waltz. 1986. "Toward Memory-Based Reasoning." *Communications of the ACM* 29: 1213–1228.

Webb, A. 1999. *Statistical Pattern Recognition.* London: Arnold.

Wettschereck, D., D. W. Aha, T. Mohri. 1997. "A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms." *Artificial Intelligence Review* 11: 273–314.

Wilfong, G. 1992. "Nearest Neighbor Problems." *International Journal on Computational Geometry and Applications* 2: 383–416.

Bölüm 9

Karar Ağaçları

Karar ağacı, böl ve yönet yaklaşımını gerçekleyen ağaç yapılı bir veri yapısıdır. Hem sınıflandırma, hem de bağlanımda kullanılabilecek etkin bir yöntemdir ve girdi dağılımından bağımsızdır. Gözetimli bir veri kümesinden ağacın nasıl öğrenilebileceğini ve ağacın anlaşılması kolay kurallara nasıl çevrilebileceğini göreceğiz. Bir başka olasılık da doğrudan bir kural tabanı öğrenmektir.

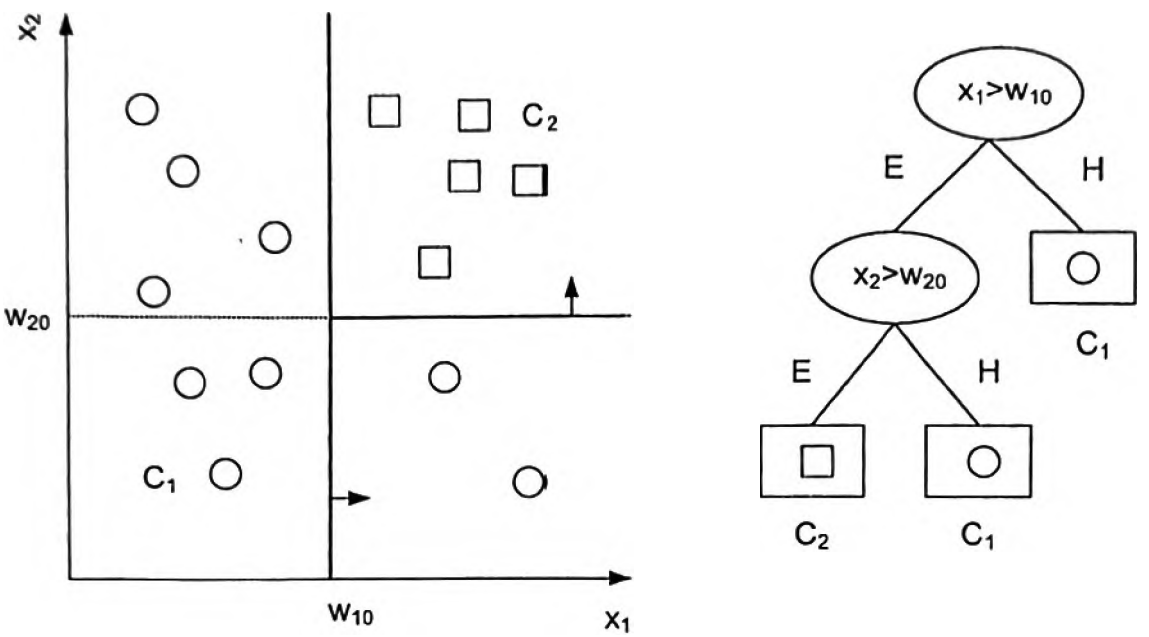
9.1 Giriş

Dağılıma bağlı kestirimde, tüm girdi uzayında geçerli bir model varsayar ve parametrelerini tüm veriyi kullanarak öğreniriz. Ardından her deneme girdisi için aynı yapı ve parametre kümesini kullanırız. Dağılımdan bağımsız kestirimde, Euclid uzaklığı gibi bir ölçütle girdi uzayını yerel parçalara ayırır ve her girdi için kendi bölgesindeki veriyle eğitilmiş bir yerel model kullanırız. 8. bölümde gördüğümüz bu örnek tabanlı yöntemlerde, bir girdiyle ilgili yerel modeli bulmak masraflıdır; girdiyle tüm öğrenme örnekleri arasındaki uzaklıkları hesaplamak gerekir ki bunun karmaşıklığı $\mathcal{O}(N)$ olur.

Karar ağacı, gözetimli öğrenme için kullanılır. İlgili yerel bölgenin özçağrılı bir biçimde, bir dizi bölmeyle az sayıda adımda bulunduğu ağaç yapılı bir modeldir. Karar ağacı, iç karar düğümleri ve uç yapraklardan oluşur (bkz. şekil 9.1). Her *karar düğümü* m, kesikli değeri dallara karşılık gelen bir $f_m(x)$ denetim işlevi gerçekler. Bir girdi için her düğümde denetim işlevi uygulanır ve sonuca göre dallardan biri seçilir. Bu süreç kökte başlar ve bir *yaprak düğümüne* gelene dek özçağrılı bir biçimde sürer; yaprakta yazılı değer çıktıyı oluşturur.

Karar ağacı, dağılımdan bağımsızdır çünkü sınıf dağılımlarıyla ilgili bir varsayım yapmaz; ağaç yapısı baştan sabit değildir, verinin altında yatan işlevin karmaşıklığına göre dallar ve yapraklar eklenerek ağaç büyür.

Her $f_m(x)$, d boyutlu girdi uzayında bir sınır tanımlar ve kökten aşağı inerken



Şekil 9.1: Örnek bir veri kümesi, ayırtaç, ve karşılık gelen karar ağacı. Yuvarlaklar karar düğümlerini, dikdörtgenler yaprakları gösterir. Tek boyutlu bir karar düğümü eksenlerin birinden böler ve ardışık bölmeler birbirinde diktir. İlk bölmeden sonra $\{x|x_1 < w_{10}\}$ saf olduğu için artık bölünmez.

ardışık düğümler girdi uzayını gittikçe daha küçük alanlara bölerler. $f_m(\cdot)$, basit bir işlevdir ve ağaç olarak gerçeklendiğinde, karmaşık bir işlev, bir dizi basit karara ayrılmış olur. Farklı karar ağaç yöntemleri, $f_m(\cdot)$ için farklı modeller varsayar ve varsayılan model ayırtacın ve alanların biçimini belirler. Her yaprağın bir çıktı değeri vardır ki bu, sınıflandırmada bir sınıf etiketi, bağlanımdaysa sayısal bir değerdir. Her yaprak, girdi uzayında bir alan tanımlar ve bu alana düşen tüm örneklerin sınıflandırmada aynı etiketi, bağlanımdaysa benzer sayıl çıktısı vardır. Alanların sınırları kökten yaprağa inen yol üstündeki iç karar düğümlerince tanımlanır.

Kararların bir ağaç yapısı oluşturması girdiyi kapsayan alanın hızla bulunmasını sağlar. Örneğin kararlar ikiliyse, en iyi durumda her karar, örneklerin yarısını eler. Eğer b alan varsa, en iyi durumda doğru alan $\log_2 b$ kararla bulunabilir. Karar ağacının bir başka üstünlüğü yorumlanabilirliktir. Az sonra göreceğimiz gibi ağaçı kolayca anlaşılabilen bir küme *EĞER-İSE* kuralı olarak yazılabilir. Bu yüzden karar ağaçları, yaygın olarak kullanılır ve bazen daha başarılı, ama daha zor yorumlanan yöntemlere yeğlenir.

Karar düğümündeki işlevlerde yalnız bir girdi değişkeni kullanılan tek değişkenli karar ağaçlarıyla başlayıp böyle ağaçların sınıflandırma ya da bağlanım için nasıl büyütüldüğünü göreceğiz. Ardından bir iç düğümde tüm girdilerin kullanılabildiği çok değişkenli ağaçlar için genelleme yapacağız.

9.2 Tek Değişkenli Ağaç

Tek değişkenli karar ağacında, her iç düğümdeki karar işlevi girdilerden yalnızca birini kullanır. Eğer bu x_j girdi boyutu, n değerden birini alan kesikli bir değişkense, karar düğümü x_j değişkeninin olası değerlerine karşılık gelen dallardan birini seçer ve girdi uzayını n parçaya böler. Örneğin, renk$\in$ {kırmızı, mavi, yeşil} özniteliğinden bölen bir karar düğümünün her biri olası değerlerden biri gerçek olduğunda seçilen üç dalı vardır.

Karar ağacının dalları kesikli olduğundan sürekli bir değerin kesiklileştirilmesi gerekir. Bu genelde bir w_{m0} eşik değeriyle karşılaştırma olarak tanımlanır:

$$f_m(x) : x_j > w_{m0} \tag{9.1}$$

Bu karar düğümü girdi uzayını iki parçaya, $L_m = \{x|x_j > w_{m0}\}$ ve $R_m = \{x|x_j \leq w_{m0}\}$ böler; buna *ikili bölme* denir. Kökten yaprağa doğru aynı yolda birbirini izleyen düğümler öteki girdileri de kullanarak birbirine dik bölmeler oluşturur ve bu durumda yapraklar girdi uzayında üstündikdörtgenler tanımlar (bkz. şekil 9.1).

Karar ağacı öğrenmek, bir öğrenme kümesi verildiğinde bir ağaç oluşturmak demektir. Bir öğrenme kümesini hatasız öğrenen birden çok ağaç olabilir ve basitlik için bu ağaçların en küçüğünü bulmak isteriz. Bir ağacın büyüklüğü, düğüm sayısına ve bu düğümlerin karmaşıklığına bağlıdır. En küçük ağacın çokterimli zamanda bulunamadığı gösterildiğinden (Quinlan 1986) akla yakın zamanlarda akla yakın ağaçlar büyütebilen sezgisel, yerel arama yordamlarına başvururuz.

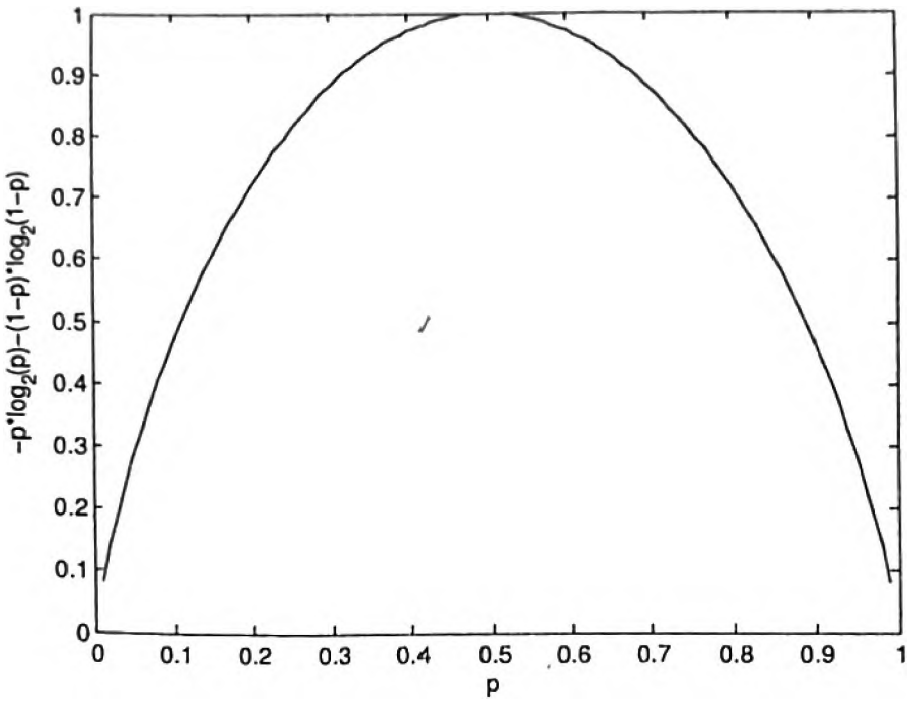
Ağaç öğrenme algoritmaları açgözlüdür; tüm öğrenme kümesiyle kökten başlayıp her adımda en iyi nereden bölebileceğimize bakarız. Bu bölmeler, kullanılan girdinin sürekli ya da kesikli olmasına göre öğrenme kümesini iki ya da n parçaya böler. Her bölümü kendi içinde özçağrılı bir biçimde nasıl en iyi bölebileceğimizi araştırır, artık bölme gerekmediğinde durur, ve bir yaprak düğümü oluştururuz.

9.2.1 Sınıflandırma Ağaçları

Sınıflandırma ağacında bir bölmenin iyiliği bir *katışıklık ölçütü* kullanılarak hesaplanır. Eğer bir bölmeden sonra tüm dallar için, o dala düşen örnekler hep aynı sınıftansa o bölme katışıksız ya da saftır. m düğümüne ulaşan öğrenme örneği sayısı N_m olsun (kök için bu sayı N olur). N_m örneğin N_m^i tanesi C_i sınıfından olsun, öyle ki, $\sum_i N_m^i = N_m$. Bir örnek m düğümüne ulaşıyorsa, C_i sınıfı için dağılımdan bağımsız sonsal olasılık kestirimi

$$\hat{P}(C_i|x, m) \equiv p_m^i = \frac{N_m^i}{N_m} \tag{9.2}$$

olur. Eğer tüm p_m^i değerleri, 0 ya da 1 ise m katışıksız bir düğümdür; ulaşan örneklerin hiçbiri C_i sınıfından değilse 0, hepsi C_i sınıfındansa 1 olur. Eğer bölme katışıksızsa bir daha bölmeye gerek yoktur ve hangi sınıf için p_m^i 1 ise o sınıfla etiketlenmiş bir



Şekil 9.2: İki sınıf için düzensizlik işlevi.

yaprak ekleyip dururuz. Katışıklılık ölçümü için en sık olarak *düzensizlik* kullanılır (Quinlan 1986) (bkz. şekil 9.2):

$$\mathcal{I}_m = -\sum_{i=1}^{K} p_m^i \log_2 p_m^i \tag{9.3}$$

Burada, $0 \log 0 \equiv 0$ olarak alınır. Düzensizlik, bilgi kuramında bir örneğin sınıfını göstermek için gereken en az bit sayısı olarak tanımlanır. İki sınıflı bir uygulamada, $p^1 = 1$ ve $p^2 = 0$ ise tüm örnekler $\mathcal{C}^1$ sınıfındandır ve bir şey yollamamıza gerek yoktur; düzensizlik 0 olur. Eğer $p^1 = p^2 = 0.5$ ise iki sınıftan biri için 1 bit göndermemiz gerekir ve düzensizlik 1 olur. Bu iki uç arasında daha olası sınıfı daha kısa biçimde gösteren gösterimler bulabiliriz. $K > 2$ sınıf olduğunda da aynı tartışma geçerlidir ve en yüksek düzensizlik olan $\log_2 K$ değerine $p^i = 1/K$ olduğunda ulaşılır.

Olası tek ölçüt düzensizlik değildir. İki sınıflı bir problemde, $p^1 \equiv p$ ve $p^2 = 1 - p$ olduğunda eksi değer almayan $\phi(p, 1-p)$ işlevinin uygun bir katışıksızlık ölçütü olması için bazı koşulları sağlaması gerekir (Devroye, Györfi, ve Lugosi 1996):

- $\phi(1/2, 1/2) \geq \phi(p, 1-p)$, tüm $p \in [0, 1]$ için.
- $\phi(0, 1) = \phi(1, 0) = 0$.
- $\phi(p, 1-p)$, $p \in [0, 1/2]$ için artar ve $p \in [1/2, 1]$ için azalır.

Örnekler:

1. Düzensizlik:
$$\phi(p, 1-p) = -p \log_2 p - (1-p) \log_2 (1-p) \tag{9.4}$$
Denklem 9.3 bunun $K > 2$ sınıf için genellenmişidir.

2. *Gini ölçütü*: (Breiman ve ötekiler 1984)
$$\phi(p, 1-p) = 2p(1-p) \tag{9.5}$$

3. Sınıflandırma hatası:
$$\phi(p, 1-p) = 1 - \max(p, 1-p) \tag{9.6}$$

Bu ölçütler, $K > 2$ sınıf için ve bir kayıp dizeyi verildiğinde sınıflandırma hatasından riske (alıştırma 1) genellenebilir. Araştırmalar, bu ölçütler arasında uygulamada çok büyük fark olmadığını göstermiştir.

m düğümüne düşen örnekler saf değilse, katışıklılığı azaltmak için bölmek ve bunun için olası girdiler içinden birini seçmek gerekir. Üstelik sayısal bir girdi için birden çok olası eşik değeri vardır. Bütün bu olasılıklar için bölmeden sonra katışıklılığın ne olacağına bakar ve en az katışıklı olanı seçeriz. Olabildiğine küçük bir ağaç büyütmek istediğimiz için en az katışıklı olanın sonrasında daha az bölmeye neden olacağını düşünürüz. Bu tabii ki bu bizi yerel iyi bir çözüme götürür ama en küçük ağacı bulmamızı güvence altına almaz.

m düğümünde, N_m örneğin N_{mj} tanesi j dalını seçiyor olsun; bunlar $f_m(\boldsymbol{x}^t)$ karar işlevinin j sonucunu verdiği $\boldsymbol{x}^t$ örneklerinin sayısıdır. Olası n değer alan kesikli bir öznitelik için n sonuç, sayısal bir öznitelik içinse iki sonuç ($n = 2$) vardır ve her iki durumda da $\sum_{j=1}^{n} N_{mj} = N_m$ sağlanır. N_{mj} örneğin N_{mj}^i tanesi C_i sınıfındandır: $\sum_{i=1}^{K} N_{mj}^i = N_{mj}$. Benzer biçimde, $\sum_{j=1}^{n} N_{mj}^i = N_m^i$ yazabiliriz.

m düğümünde karar işlevi eğer j. sonucu verirse, C_i sınıfından olma olasılığını yazalım:
$$\hat{P}(C_i | \boldsymbol{x}, m, j) \equiv p_{mj}^i = \frac{N_{mj}^i}{N_{mj}} \tag{9.7}$$
olur Bölmeden sonraki toplam katışıklılık yine düzensizlik kullanılarak hesaplanabilir:
$$\mathcal{I}'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p_{mj}^i \log_2 p_{mj}^i \tag{9.8}$$

Sayısal bir girdi için bölmenin nereden yapıldığını da bilmemiz, yani denklem 9.1'de p_{mj}^i değerini hesaplayabilmek için w_{m0} değerini de bulmamız gerekir. N_m veri arasında $N_m - 1$ tane olası bölme eşiği vardır; olası tüm (sonsuz sayıda) noktayı denememize gerek yoktur, yalnızca örneğin, orta noktalarını denemek yeter. Üstelik en iyi bölmenin farklı sınıflardan noktaların arasından geçtiğini biliriz. Tüm bu olasılıkları dener ve katışıksızlık açısından en iyisini seçer ve değerini o girdi için en iyi değer alır ve öbür girdiler için hesapladıklarımızla karşılaştırırız. Kesikli bir girdi için böyle bir arama yapmaya gerek yoktur.

```
AğaçÖğren(𝒳)
    Eğer DüğümDüzensizliği(𝒳)< θ_I /* denklem 9.3 */
        𝒳'te en çok görülen sınıfla etiketli yaprak yarat
        Dön
    i ← EnİyiBölenÖznitelik(𝒳)
    x_i'nin tüm dalları için
        O dala düşen 𝒳_i örneklerini bul
        AğaçÖğren(𝒳_i)

EnİyiBölenÖznitelik(𝒳)
    MinEnt← MAX
    Tüm öznitelikler i = 1,...,d
        Eğer x_i, n değerli kesikliyse
            𝒳'i x_i değerine göre 𝒳_1,...,𝒳_n'e ayır
            e ← BölmeDüzensizliği(𝒳_1,...,𝒳_n) /* denklem 9.8 */
            Eğer e<MinEnt ise MinEnt ← e; bestf ← i
        Değilse /* x_i sayısalsa */
            Tüm olası eşikler
                𝒳'i x_i değerine göre 𝒳_1, 𝒳_2'ye ayır
                e←BölmeDüzensizliği(𝒳_1, 𝒳_2)
                Eğer e<MinEnt ise MinEnt ← e; bestf ← i
    bestf değerini döndür
```

Şekil 9.3: Sınıflandırma ağacı büyütme için örnek algoritma.

Demek ki tüm öznitelikler için, kesikli ya da sayısal, ve bir sayısal girdi için olası tüm eşikler üzerinden, olası bölmelere bakıyor, (örneğin, denklem 9.8'deki gibi) katışıklılıklarını ölçüyor ve en iyi olanını seçiyoruz. Ağacın büyütülmesi özçağrılıdır ve bu bölmeden sonra saf olmayan her parça için saflığa ulaşılana dek yine en iyi bölme aranır. *CART* (Breiman ve ötekiler 1984) ve *ID3* (Quinlan 1986), ve gelişmiş sürümü *C4.5* (Quinlan 1993), bu biçimde çalışan ağaç öğrenme algoritmalarıdır. Örnek program, şekil 9.3'te verilmiştir.

Ağaç büyütülürken her adımda katışıklılığı en fazla azaltan bölme seçilir ki bu, m düğümüne ulaşan verinin katışıklılığıyla (denklem 9.3), bölmeden sonra dallarına düşen verinin toplam katışıklılığı (denklem 9.8) arasındaki farktır.

Bir sorun, böyle bir bölmenin çok sayıda değer alabilen öznitelikleri yeğlemesidir. Çok değer olduğunda, çok dal olur ve katışıklılık daha düşük olabilir. Örneğin, örnek dizin değişkeni t girdi olarak alınsa N dala bölen bir düğümle her dalda saf bir örnek olur ve toplam katışıklılık 0 olur; ama tabii ki bu akla yakın bir bölme olmaz. Çok dalı olan düğümler karmaşıktır ve sınıf ayırtaçlarını basit kararlara indirme yaklaşımımıza ters düşer. Böylesi karmaşık öznitelikleri cezalandırararak katışıklılıktaki azalmayla dallanmayı dengeleyen yöntemler önerilmiştir.

Gürültü olduğunda, her uç saf olana dek bölmek veriyi ezberleyen çok büyük bir ağaç büyütebilir—çok sayıda doğru örnek arasında yanlış etiketli bir örnek düşünün. Bunu engellemek için düğümler yeterince saf olduğunda durulabilir; yani, bir küme veri $\mathcal{I} < \theta_I$ sağladığında durulabilir. Bu, p^i_{mj} olasılıklarının ille de 0 ya da 1 olmaması ama belirli bir θ_p değeri kadar yakın olması demektir. Bu durumda bir yaprak yaratılır ve en yüksek p^i_{mj} olasılığı olan sınıfla etiketlenir.

θ_I (ya da θ_p) karmaşıklık parametresidir ve etkisi dağılımdan bağımsız kestirimdeki h ya da k ile aynıdır. Küçük olduğunda değişinti yüksektir ve ağaç öğrenme kümesini iyi yansıtabilmek için büyüyebilir; büyük olduğundaysa değişinti küçüktür ama oluşan küçük ağaç yanlı olabilir. En iyi değer, yanlış sınıflandırmanın bedeliyle bellek ve hesabın bedelinin birbirlerini nasıl dengelediğine bağlıdır.

Genelde bir yaprakta, (sonsal) olasılığı en yüksek sınıfın etiketini saklamak yerine sonsal olasılıkları saklamak önerilir. Bu olasılıklar sonraki adımlarda, örneğin risk hesaplamakta, kullanılabilir. Yaprağa ulaşan örneklerin sayısını saklamamıza gerek yokur; yalnızca oranları saklamak yeter.

9.2.2 Bağlanım Ağacı

Bağlanım ağacı, sınıflandırma ağacı gibi öğrenilir; tek farkla ki, sınıflandırma için uygun olan düzensizlik ölçütü yerine bağlanım için uygun bir ölçüt kullanılır. $\mathcal{X}$ öğrenme kümesinin m düğümüne ulaşan bölümünü $\mathcal{X}_m$ ile gösterelim; yani $\mathcal{X}_m$, kökten m düğümüne dek tüm kararları sağlayan $\boldsymbol{x} \in \mathcal{X}$ örneklerinin kümesi olsun:

$$b_m(\boldsymbol{x}) = \begin{cases} 1 & \text{eğer } \boldsymbol{x} \in \mathcal{X}_m \text{ ise: } \boldsymbol{x}, m \text{ düğümüne ulaşıyorsa} \\ 0 & \text{değilse} \end{cases} \tag{9.9}$$

Bağlanımda, bölmenin iyiliği kestirilen değerin ortalama kare hatasına bağlı olarak hesaplanır:

$$E_m = \frac{1}{N_m} \sum_t (r^t - g_m)^2 b_m(\boldsymbol{x}^t) \tag{9.10}$$

Burada, g_m, m düğümündeki kestirimimizi gösterir. Düğüme ulaşan örnek sayısı $N_m = |\mathcal{X}_m| = \sum_t b_m(\boldsymbol{x}^t)$ olarak hesaplanabilir.

Kestirim için düğüme ulaşan örneklerin istenen çıktı değerlerinin ortalaması (çok gürültü varsa ortancası) kullanılır:

$$g_m = \frac{\sum_t b_m(\boldsymbol{x}^t) r^t}{\sum_t b_m(\boldsymbol{x}^t)} \tag{9.11}$$

Denklem 9.10, m düğümündeki değişintiye karşılık gelir. Bir düğümde bu hata hoşgörülebilecek kadar azsa, yani, $E_m < \theta_r$ ise, bir yaprak yaratılır ve içinde g_m değeri saklanır. Bu durumda, bölüm 8'deki düzleştiriciler gibi, yaprak sınırlarında süreksizlikler olan parçalı sabit bir yaklaşıklama oluşur.

Hata çoksa m düğümüne ulaşan verinin daha da bölünmesi gerekir. Sınıflandırmada yaptığımız gibi her düğümde, olası tüm öznitelikler (ve sayısal bir öznitelikte olası

tüm eşik değerleri) için bölmelere bakar, kare hatası en az olanı seçer, ve özçağrılı biçimde dallanmayı sürdürürüz.

$\mathcal{X}_{mj}$, $\mathcal{X}_m$ kümesinin j dalını seçen altkümesini göstersin: $\cup_{j=1}^{n} \mathcal{X}_{mj} = \mathcal{X}_m$.

$$b_{mj}(\boldsymbol{x}) = \begin{cases} 1 & \text{eğer } \boldsymbol{x} \in \mathcal{X}_{mj} \text{ ise: } \boldsymbol{x},\ m \text{ düğümüne ulaşır ve } j \text{ dalını seçerse} \\ 0 & \text{değilse} \end{cases} \tag{9.12}$$

g_{mj}, m düğümünün j dalındaki kestirimimizi göstersin:

$$g_{mj} = \frac{\sum_t b_{mj}(\boldsymbol{x}^t) r^t}{\sum_t b_{mj}(\boldsymbol{x}^t)} \tag{9.13}$$

Bölmeden sonraki hatayı yazalım:

$$E'_m = \frac{1}{N_m} \sum_j \sum_t (r^t - g_{mj})^2 b_{mj}(\boldsymbol{x}^t) \tag{9.14}$$

Hatadaki azalma, denklem 9.10 ile denklem 9.14 arasındaki farktır. Bu farkı en büyük yapan, ya da denklem 9.14'ü en küçük yapan bölme en iyi bölmedir. Şekil 9.3'teki örnek program, düzensizlik yerine kare hata ve sınıf etiketi yerine ortalama çıktı kullanılarak bağlanım ağacı büyütmeye uyarlanabilir.

Ortalama kare hata, olası hata işlevlerinden yalnızca biridir; bir başkası en büyük hatadır:

$$E_m = \max_j \max_t |r^t - g_{mj}| b_m(\boldsymbol{x}^t) \tag{9.15}$$

ve bunu kullandığımızda herhangi bir örnek için hatanın hiçbir zaman verilen eşik değerinden büyük olmamasını sağlarız.

Kabul edilen hata eşiği, karmaşıklık parametresidir; küçük olduğunda ağaç büyük olur ve aşırı öğrenme olabilir; büyük olduğunda fazla düzleştirebiliriz (bkz. şekiller 9.4 ve 9.5).

Düzleştiricilerde akan ortalama yerine akan doğru kullanabildiğimiz gibi bağlanım ağacında da yapraklarda doğrusal modeller kullanabiliriz:

$$g_m(\boldsymbol{x}) = \boldsymbol{w}_m^T \boldsymbol{x} + w_{m0} \tag{9.16}$$

Böylece, o yapraktaki kestirim $\boldsymbol{x}$ girdisine bağlı olur ve parçalı doğrusal bir yaklaşıklama oluşur; daha küçük bir ağaç çıkabilir ama yaprakları daha karmaşık olur.

9.3 Budama

Bir düğüme ulaşan örnek sayısı, öğrenme kümesinin belli bir yüzdesinden (örneğin yüzde beşinden) azsa o düğüm artık bölünmez. Az sayıda örneğe bağlı olarak yapılan kararların değişintiyi ve dolayısıyla genelleme hatasını artıracağı düşünülür. Bu biçimde erken durmaya *erken budama* denir.

Basit ağaçlar büyütmenin bir başka yoluysa *geç budamadır*, ki genelde uygulamalarda erken budamadan daha iyi sonuç verir. Ağaç büyütmenin açgözlü bir arama yaptığını ve her adımda bir düğüm ekleyip hiçbir zaman geri adım atarak başka bir seçenek düşünmediğimizi gördük. Bunun tek istisnası gerekmeyen altağaçları bulup budadığımız geç budamadır.

Geç budamada, ağacı tüm yapraklar saf olana ve öğrenme kümesinde hata yapmayana dek büyütürüz. Ardından ezberlemeye neden olan altağaçları bulur ve budarız. İlk öğrenme kümesinin bir parçasını *budama kümesi* olarak ayırırız. Her altağaç yerine o altağacın öğrenme kümesinde kapsadığı örneklerle (sınıflandırma ya da bağlanım için uygun biçimde) eğitilmiş bir yaprak koyar ve budama kümesi üzerinde bu iki seçeneği karşılaştırırız. Eğer yaprak, budama kümesi üzerinden daha çok hataya neden olmuyorsa altağacı budayıp yaprağı kullanır, yoksa altağacı tutarız.

Örneğin, 9.5'in üçüncü ağacında kökünde $x < 6.31$ koşulu olan altağacı düşünelim. Bu altağaç yerine (ikinci ağaçtaki gibi) $y = 0.9$ olan bir yaprak, eğer bu yer değiştirme budama hatasını artırmazsa konabilir. Budama işlemi bir tür geçerleme yapar ama bir uygulamada farklı amaçlar için hem budama, hem de geçerleme kümesi gerekebilir.

Erken budamayla geç budamayı karşılaştırdığımızda erken budamanın daha hızlı, geç budamanınsa daha başarılı ağaçlar oluşturduğunu görüyoruz.

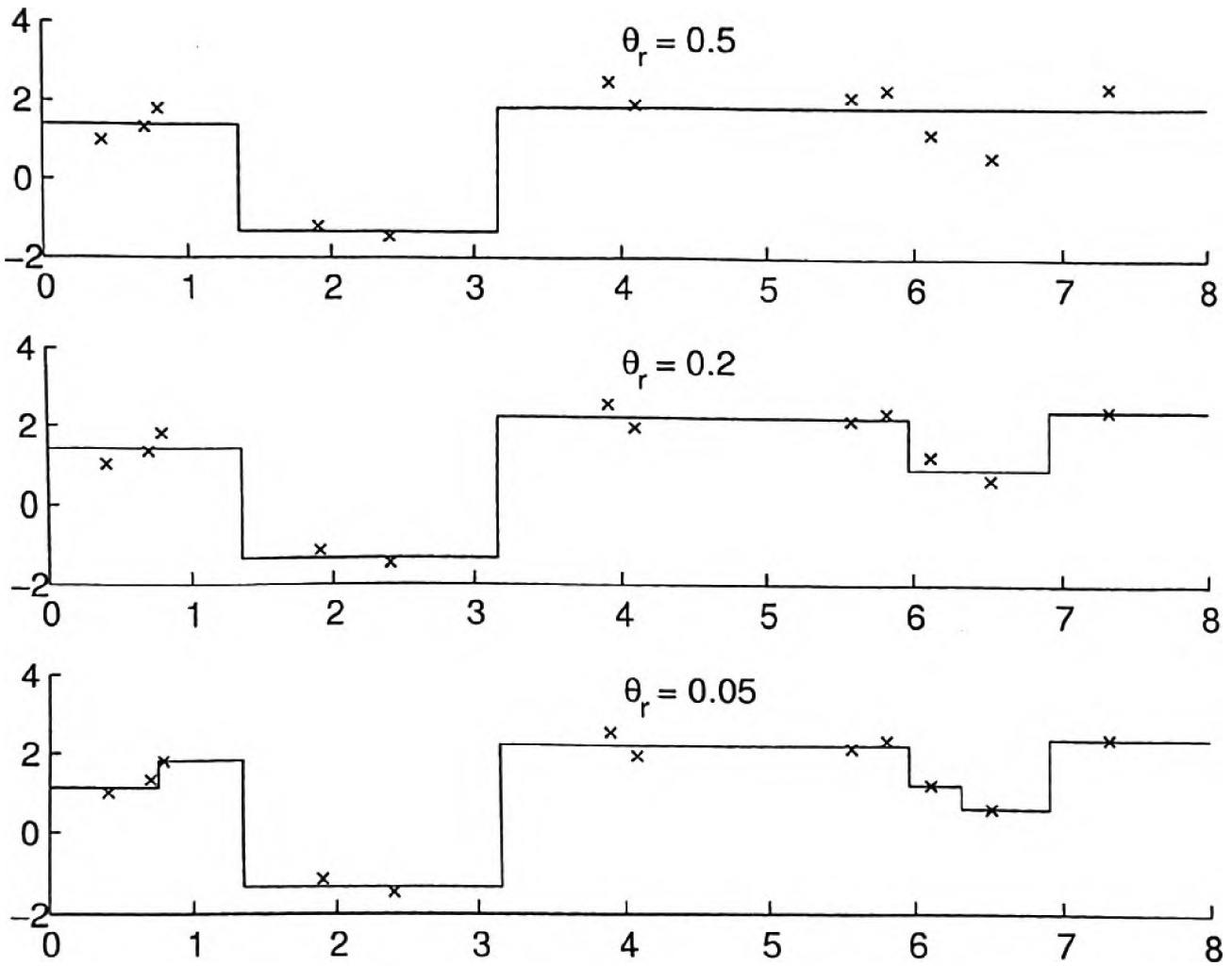
9.4 Ağaçtan Kural Çıkarma

Karar ağacı boyut azaltma işini kendi yapar. Tek değişkenli ağaç, gerek duyduğu değişkenleri kullanır ve ağaç büyüdükten sonra bazı öznitelikler hiç kullanılmıyor olabilir. Ayrıca, köke yakın özniteliklerin genelde daha önemli olduğu söylenebilir. Örneğin, şekil 9.6'deki ağaç, x_1, x_2, ve x_4 girdilerini kullanır ama x_3 girdisini kullanmaz. Karar ağacını öznitelik seçimi ve boyut azaltma için de kullanabiliriz: ağacı büyütür, sonra başka bir öğrenme algoritmasına yalnızca ağacın kullandığı öznitelikleri girdi olarak veririz.

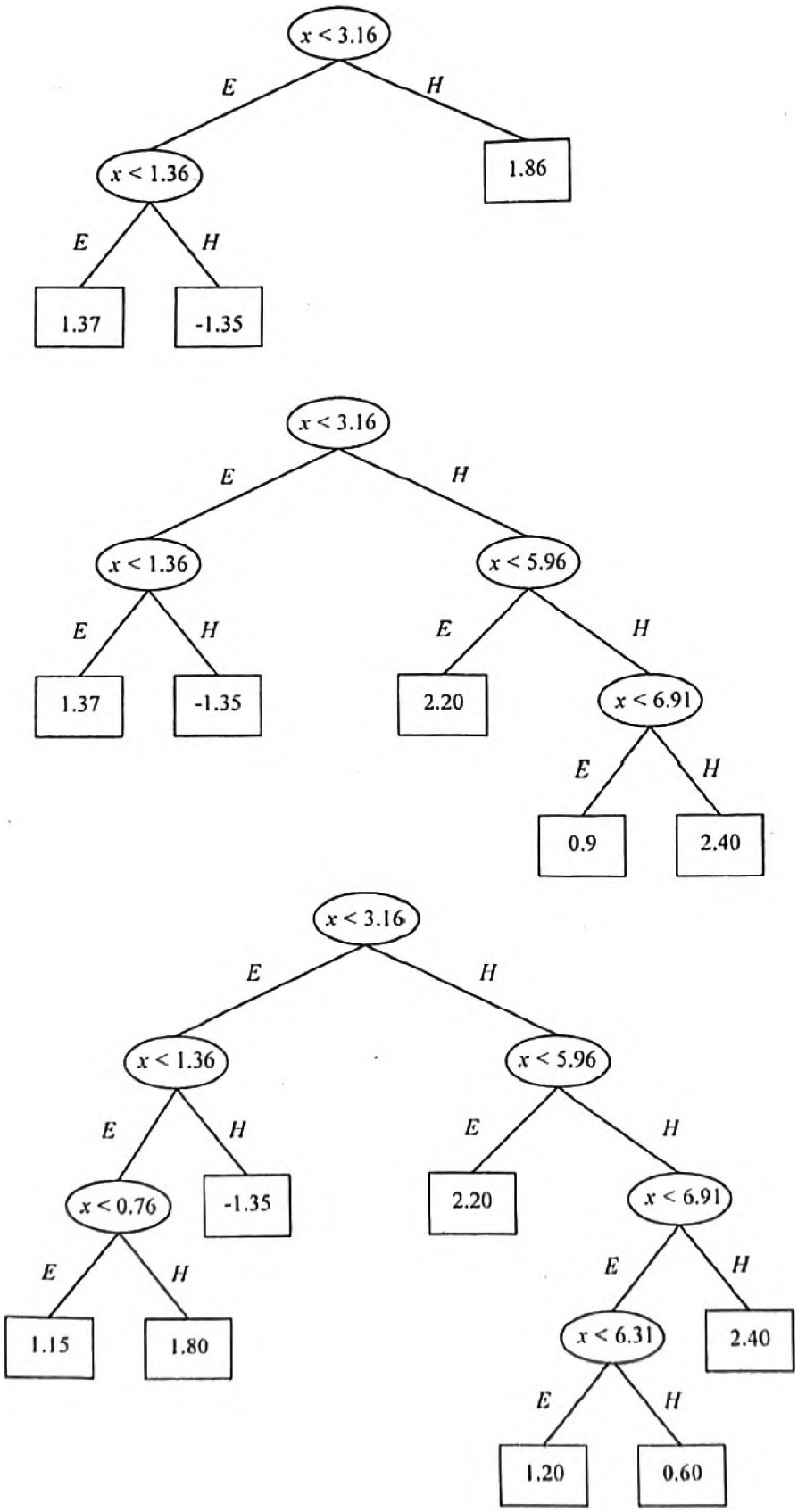
Ağacın bir başka üstünlüğü de *yorumlanabilir* olmasıdır: Karar düğümleri kolay anlaşılan koşulları gerçekler. Kökten yaprağa giden her yol, aynı anda sağlanması gereken (VE ile birleştirilen) koşullar tanımlar. Bu yollar, *EĞER-İSE kuralları* olarak bir *kural tabanı* oluşturur. Bunun için bir yöntem, *C4.5Rules* algoritmasıdır (Quinlan 1993).

Örneğin, şekil 9.6'deki ağaç için kuralları yazalım:

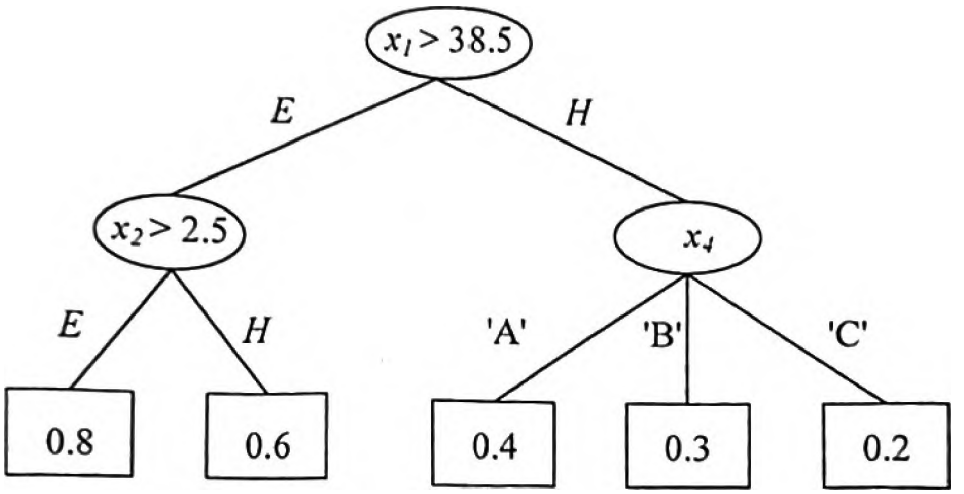
R1: EĞER (yaş>38.5) VE (işindeki-yıl-sayısı>2.5) İSE y =0.8
R2: EĞER (yaş>38.5) VE (işindeki-yıl-sayısı≤2.5) İSE y =0.6
R3: EĞER (yaş≤38.5) VE (meslek-türü='A') İSE y =0.4
R4: EĞER (yaş≤38.5) VE (meslek-türü='B') İSE y =0.3
R5: EĞER (yaş≤38.5) VE (meslek-türü='C') İSE y =0.2



Şekil 9.4: Farklı θ_r değerleri için bağlanım ağaçları. Karşılık gelen ağaçlar, şekil 9.5'te gösterilmiştir.



Şekil 9.5: Şekil 9.4'deki kestirimlere karşılık gelen, farklı θ_r değerleri için bağlanım ağaçları.



Şekil 9.6: Örnek (hayali) bir karar ağacı. Girdiler—x_1: yaş, x_2: işindeki yıl sayısı, x_3: cinsiyet, x_4: meslek türü. Kökten yaprağa giden her yol bir kural oluşturur; yol üstündeki tüm koşulların birlikte sağlanması gerekir.

Böyle bir kural tabanı, *bilgi çıkarmaya* izin verir; kurallar, kolayca anlaşılabilir ve uzmanlarca incelenip geçerlenebilir. Her kuralın öğrenme kümesinin yüzde kaçını kapsadığına bakıp *kural desteği* hesaplanabilir. Kurallar, verinin önemli özelliklerini, örneğin önemli öznitelikleri ve değerleri gösterir. Örneğin, bu (hayali) örnekte, otuz sekiz ve daha genç insanlarla otuz sekiz yaşından büyük insanlar arasında uygulama açısından bir fark olduğunu ve gençlerde şu anki işinde kaç yıl çalıştığı önemliyken, yaşlılarda meslek türünün ayırt etmede önemli olduğunu anlıyoruz.

Bir sınıflandırma ağacında aynı sınıfla etiketlenmiş birden çok yaprak olabilir. Bu durumda, her biri farklı bir yola karşılık gelen bu kuralların VEYA ile birleştirilmesi gerekir, çünkü herhangi birinin sağlanması yeterlidir. Bu da sınıf alanını farklı yaprakların alanlarının birleşimi olarak tanımlar. Örneğin, şekil 9.1'de C_1 sınıfı

$$\text{EĞER } (x \leq w_{10}) \text{ VEYA } ((x_1 > w_{10}) \text{ VE } (x_2 \leq w_{20})) \text{ İSE } C_1$$

biçiminde yazılır.

Kural budama, kuralları basitleştirmemizi sağlar. Bir altağacın budanması birden çok kural içinden bazı koşulların birlikte budanmasına neden olur. Ama bir koşulu bir kuraldan başka kurallara dokunmadan budayabiliriz. Örneğin, yukarıdaki kural tabanında, `meslek-türü='A'` olan herkes için, yaş değeri ne olursa olsun, çıktı 0.4'e yakınsa, *R3* kuralından yaş değişkenini içeren koşul budanabilir:

$$R3' : \text{EĞER (meslek-türü='A') İSE } y = 0.4$$

Dikkat: Kurallar budandıktan sonra yeniden bir ağaç olarak yazılamayabilir.

9.5 Kural Öğrenme

EĞER-İSE kuralları oluşturmanın bir yolu, yukarıda gördüğümüz gibi karar ağacı eğitip kurallara çevirmektir. Ötekiyse doğrudan kuralları öğrenmektir. *Kural öğrenme* ağaç öğrenmeye benzer, ama kural öğrenme derinlik öncelikli çalışarak kuralları birer birer oluştururken ağaç öğrenme genişlik öncelikli çalışarak tüm yolları (kuralları) birlikte çıkarır.

Kurallar birer birer öğrenilir. Her kural (karar ağaçlarındaki gibi), kesikli ya da sayısal öznitelikleri kullanan koşul terimlerinin VE ile birleşmesinden oluşur. Bu terimler bir kıstasa göre, örneğin düzensizliği azaltacak biçimde, birer birer eklenerek kural büyütülür. Bir örnek kuraldaki terimlerin hepsini sağlıyorsa kuralın o örneği *kapsadığı* söylenir. Bir kural büyütülür, budanır, kural tabanına eklenir ve kapsadığı tüm örnekler öğrenme kümesinden çıkarılır. Bu süreç, yeterince kural eklenene dek sürer. Buna, *sırayla kapsama* denir. İç içe iki döngü vardır: Dışarıdaki döngü, kural tabanına kuralları birer birer eklerken içerideki döngü, bir kurala terimleri birer birer ekler. Her iki döngünün adımları da açgözlüdür ve en iyiyi bulma güvencesi yoktur. Her iki döngüde de daha iyi genelleme için budama yapılır.

Kural öğrenme için bir örnek, *Ripper* (Cohen 1995) algoritmasıdır, ve daha önce önerilmiş *Irep* (Fürnkranz ve Widmer 1994) algoritmasına dayanır. Önce, artı ve eksi örnekler içeren iki sınıflı sürümünden bahsedecek, ardından $K > 2$ sınıf için genelleyeceğiz. Kurallar artı örnekleri açıklamak için eklenir; bir örnek hiçbir kural tarafından kapsanmıyorsa eksi olarak sınıflandırılır. Bir örnek bir kuralla eşleştiğinde sonuç ya doğrudur (doğru artı), ya da yanlış artıdır. Ripper algoritmasının dış döngüsü için örnek program şekil 9.7'de verilmiştir.

Ripper algoritmasında koşullar, Quinlan'ın (1990) *Foil* algoritmasındaki gibi bir bilgi kazanç ölçütünü artıracak biçimde eklenir. Kuralı R, koşul eklendikten sonraki aday kuralı da R' ile gösterelim. Kazançtaki artış

$$Gain(R', R) = s \cdot \left(\log_2 \frac{N'_+}{N'} - \log_2 \frac{N_+}{N} \right) \tag{9.17}$$

olarak ölçülür. N, R tarafından kapsanan örneklerin sayısıdır ve içinden N_+ tanesi doğru artıdır; N' ve N'_+ benzer biçimde R' için tanımlıdır. s, hem R içinde, hem de koşu eklendikten sonra R' içinde doğru artıların sayısıdır. Bilgi kuramı açısından bakarsak kazançtaki artış, artı bir örneği göstermek için gereken bit sayısındaki azalmaya karşılık gelir.

Koşullar kurala hiçbir eksi örneği kapsamayıncaya dek eklenir. Kuralı önce büyütür, sonra koşulları ters sırada silerek budarız. Buradaki amacımız *kural değer ölçevini* artırmaktır:

$$rvm(R) = \frac{p - n}{p + n} \tag{9.18}$$

p ve n sırasıyla, budama kümesindeki doğru ve yanlış artıların sayısıdır; genelde öğrenme verisinin üçte ikisi kural büyütme, üçte biri de budama için kullanılır.

Kural büyütülüp budandıktan sonra kapsadığı artı ve eksi örnekler öğrenme kümesinden çıkarılır. Artı örnek kaldıysa kural öğrenme sürer. Gürültü olduğunda

```
Ripper(Artı,Eksi,k)
    KuralKümesi ← KuralKümesiÖğren(Artı,Eksi)
    Yinle k kez
        KuralKümesi ← KuralKümesiİyileştir(KuralKümesi,Artı,Eksi)
KuralKümesiÖğren(Artı,Eksi)
    KuralKümesi ← ∅
    DL ← DescLen(KuralKümesi,Artı,Eksi)
    Yinele
        Kural ← KuralÖğren(Artı,Eksi)
        Kural'ı KuralKümesi'ne ekle
        DL' ← TanımUzunluğu(KuralKümesi,Artı,Eksi)
        EĞER DL'¿DL+64
                KuralKümesiBuda(KuralKümesi,Artı,Eksi)
                KuralKümesi döndür
        EĞER DL'¡DL İSE DL ← DL'
                Kural tarafından kapsanan örnekleri Artı ve Eksi'den çıkar
    Pos = ∅ olana dek
    KuralKümesi döndür
KuralKümesiBuda(KuralKümesi,Artı,Eksi)
    Tüm Kural∈KuralKümesi için ters sırada
        DL ← TanımUzunluğu(KuralKümesi,Artı,Eksi)
        DL' ← TanımUzunluğu(KuralKümesi-Kural,Artı,Eksi)
        EĞER DL'¡DL İSE Kural'ı KuralKümesi'nden çıkar
    Return KuralKümesi
KuralKümesiİyileştir(KuralKümesi,Artı,Eksi)
    Tüm Kural ∈ KuralKümesi
        DL0 ← TanımUzunluğu(KuralKümesi,Artı,Eksi)
        DL1 ← TanımUzunluğu(KuralKümesi-Kural+
                SıfırdanKural(KuralKümesi,Artı,Eksi),Artı,Eksi)
        DL2 ← TanımUzunluğu(KuralKümesi-Rule+
                DüzeltilmişKural(KuralKümesi,Kural,Artı,Eksi),Artı,Eksi)
        EĞER DL1=min(DL0,DL1,DL2) İSE
                KuralKümesi'nden Kural'ı çıkar ve
                    SıfırdanKural(KuralKümesi,Artı,Eksi)'ı ekle
        DEĞLSE EĞER DL2=min(DL0,DL1,DL2) İSE
                KuralKümesi'nden Kural'ı çıkar ve
                    DüzeltilmişKural(KuralKümesi,Kural,Artı,Eksi)'ı ekle
    KuralKümesi'ni döndür
```

Şekil 9.7: Kural öğrenme için Ripper algoritması. Yalnızca dış döngü gösterilmiştir; iç döngü, karar ağacında düğümlerin eklenmesine benzer.

daha erken, örneğin bir kural çok az sayıda örnek kapsadığında da, durabiliriz. Bir kuralın değerini en kısa tanım uzunluğuyla (bölüm 4.8)ölçeriz (Quinlan 1995). Genelde, kuralın tanımı kapsadığı örneklerin tanımından daha kısa değilse dururuz. Bir kural tabanının uzunluğu, içerdiği kuralların tanım uzunluklarıyla kural tabanının kapsamadığı örneklerin tanımını toplayarak bulunur. Ripper, kural tabanının uzunluğuyla o ana kadar bulunan en iyi uzunluk arasındaki fark 64 bitten daha fazla olduğunda artık kural eklemez. Kural tabanı büyütüldükten sonra kuralların üstünden ters sırada geçer ve tanım uzunluğunu artırmadan budayabilir miyiz diye bakarız.

Kural tabanındaki kurallar öğrenildikten sonra iyileştirilir. Ripper her kural için iki farklı seçenek dener: Birinde sıfırdan, boş bir kuraldan başlar, büyütür, ve budar. İkincisinde eldeki kuralla başlar, büyütür, ve budar. Bu ikisi, kuralın ilk biçimiyle karşılaştırılır ve en kısa olan seçilir ve kural tabanına eklenir. Kural tabanında bu biçimde istendiği kadar, ama sıklıkla iki kez, yapılır.

$K > 2$ sınıf olduğunda sınıfları önce önsel olasılıklarına göre sıralarız, öyle ki $\mathcal{C}_1$ en düşük, $\mathcal{C}_K$ en yüksek olasılıklı sınıfı gösterir. Ardından bir dizi iki sınıf uygulaması tanımlanır: İlkinde $\mathcal{C}_1$ sınıfının örnekleri artı, öteki tüm sınıfların örnekleri birlikte eksidir. $\mathcal{C}_1$ öğrenilip tüm örnekleri öğrenme kümesinden çıkarıldığında bu kez $\mathcal{C}_2$ sınıfını $\mathcal{C}_3, \ldots, \mathcal{C}_K$ sınıflarından ayırmayı öğreniriz. Bu süreç yalnız $\mathcal{C}_K$ kalana dek sürer. Sona boş bir kural ekleyip $\mathcal{C}_K$ ile etiketleriz; böylece önceki kurallar tarafından kapsanmayan örnekler $\mathcal{C}_K$ sınıfına atanmış olur.

N büyüklüğünde bir kümede Ripper algoritmasının karmaşıklığı $\mathcal{O}(N \log^2 N)$ olur; bu sayede çok büyük veri kümelerinde bile kullanılabilir (Dietterich 1997). Çğrendiklerimiz *önermeli kurallardır*. Daha güçlü, *Birinci derece kurallar*, *belirtim* denilen ve değişken alabilen koşullar kullanır. Belirtim, kullandığı değişkenin değerine göre doğru ya da yanlış değer alan bir işlevdir. Bu sayede belirtimler değişkenlerin değerleri arasında ilişkiler tanımlayabilir ki bu, önermelerle yapılamaz (Mitchell 1997):

$$\text{EĞER Baba}(y, x) \text{ VE Erkek}(y) \text{ İSE Oğul}(x, y)$$

Böyle kurallar, Prolog gibi bir mantık programlama dilinde yazılmış programlara benzer ve bu kuralları veriden öğrenmeye *öğrenen mantık programlama* denir. *Foil* (Quinlan 1990) böyle bir algoritmadır.

Değişkene bir değer atamaya *bağlama* denir. Değişkenler için uygun bağlamalar öğrenme kümesinde varsa bir kural eşleşir. Birinci derece kuralları öğrenmek önermeli kuralları öğrenmeye benzer; yine dışta kural ekleyen bir döngü, içte de kurala koşul ekleyen başka bir döngü, ve her ikisinde döngü sonunda budama vardır. İçteki döngüde (önerme yerine) birer birer belirtim eklerken kural başarımını artırmaya çalışırız. (Mitchell 1997). Kuralın başarımını ölçmek için değişkenlerin olası tüm bağlamalarına bakar, artı ve eksi bağlanımları sayar, ve örneğin denklem 9.17'yi kullanırız. Birinci derece kurallarda önerme yerine kullandığımız belirtimler önceden tanımlanmıştır ve öğrenme kümesi, değerler olarak değil doğru olduğunu bildiğimiz belirtimler olarak verilmiştir.

9.6 Çok Değişkenli Ağaç

Tek değişkenli ağaçta her düğüm (bölme) tek bir girdi kullanır. *Çok değişkenli ağaçta* ise bir karar düğümünde tüm girdi boyutları kullanılabilir; bu onu daha genel yapar. Tüm girdiler sayısal olduğunda çok değişkenli doğrusal bir düğüm kullanılabilir:

$$f_m(\boldsymbol{x}) : \boldsymbol{w}_m^T \boldsymbol{x} + w_{m0} > 0 \tag{9.19}$$

Çok değişkenli doğrusal düğüm, ağırlıklı toplam aldığı için kesikli özniteliklerin 0/1 yapay değişkenleriyle gösterilmesi gerekir. Denklem 9.19, herhangi bir eğim alabilen bir üstündüzlem tanımlar (bkz. şekil 9.8). Kökten yaprağa giden yol üstündeki ardışık düğümler bunları gittikçe daha küçük parçalara böler ve yapraklar girdi uzayında çokyüzlüler tanımlar. Sayısal tek değişkenli düğüm, biri dışında öteki tüm w_{mj} değerlerinin 0 olduğu özel durumdur. Denklem 9.1'deki tek değişkenli sayısal düğüm, x_j eksenine dik, onu w_{m0} noktasında kesen ve tüm öteki x_i eksenlerine dik doğrusal bir ayırtaç tanımlar. Dolayısıyla tek değişkenli bir düğümde yalnızca d olası yön ($\boldsymbol{w}_m$) ve her biri için $N_m - 1$ olası eşik değeri ($-w_{m0}$) vardır ve bu çok büyük bir sayı olmadığı için olası tüm değerleri deneyip en iyisini seçebiliyoruz. Çok değişkenli bir düğümde olası $2^d \begin{pmatrix} N_m \\ d \end{pmatrix}$ tane üstündüzlem (Murthy, Kasif, ve Salzberg 1994) olduğundan tüm olasılıkları deneyemeyiz.

Tek değişkenli düğümden çok değişkenli doğrusal düğüme geçtiğimizde modelimiz daha esnek olur. Doğrusal olmayan bir karar işleviyle daha esnek de yapabiliriz. Örneğin, ikinci dereceye çıkabiliriz:

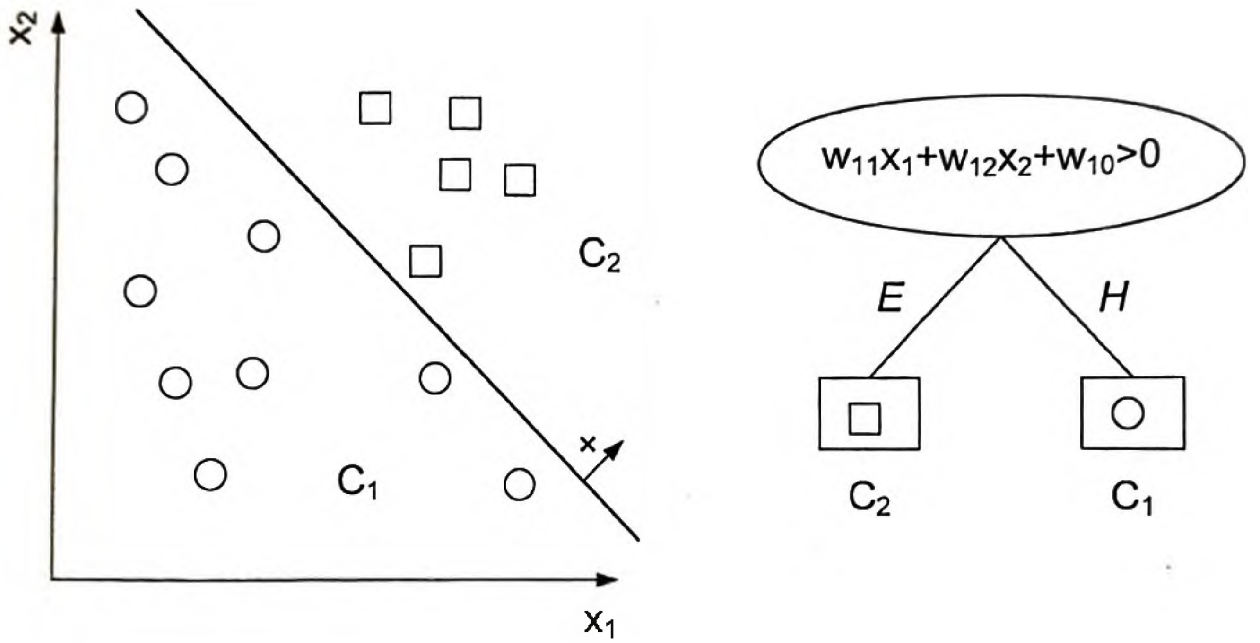
$$f_m(\boldsymbol{x}) : \boldsymbol{x}^T \mathbf{W}_m \boldsymbol{x} + \boldsymbol{w}_m^T \boldsymbol{x} + w_{m0} > 0 \tag{9.20}$$

Guo ve Gelfand (1992), doğrusal olmayan taban işlevlerinin doğrusal bir toplamı olan çok katmanlı algılayıcı (bölüm 11) kullanmayı önermiştir, ki bu da doğrusal olmayan karar düğümü elde etmek için başka bir yoldur. Bir başka olasılık, *küre düğümüdür* (Devroye, Györfi, ve Lugosi 1996):

$$f_m(\boldsymbol{x}) : \|\boldsymbol{x} - \boldsymbol{c}_m\| \leq \alpha_m \tag{9.21}$$

Burada, $\boldsymbol{c}_m$ merkezi, α_m ise yarıçapı gösterir.

Sınıflandırma için çok değişkenli karar ağacı öğrenen algoritmalar önerilmiştir. En erkeni *CART* algoritmasının (Breiman ve ötekiler 1984) çok değişkenli sürümüdür ve w_{mj} ağırlıklarını katışıklılığı azaltacak biçimde birer birer ayarlar. CART algoritmasının öznitelik seçerek boyut sayısını ve dolayısıyla düğüm karmaşıklığı azaltan (bölüm 6) bir önişleme aşaması da vardır. *OC1* algoritması (Murthy, Kasif, ve Salzberg 1994), CART'a bazı ekler içerir. Bir olasılık (Loh and Vanichsetakul 1988), sınıfların Gauss dağıldığını ve eşdeğişinti dizeylerinin aynı olduğunu varsayıp doğrusal ayırtaçları bu biçimde bulmaktır (bölüm 5). K sınıf olduğunda her düğümün K dalı vardır ve her dal, bir sınıfı ötekilerden ayıran ayırtacı tanımlar. Brodley ve Utgoff (1995), doğrusal ayırtaçları sınıflandırma hatasını azaltacak biçimde



Şekil 9.8: Örnek çok değişkenli doğrusal karar ağacı. Çok değişkenli doğrusal düğüm herhangi bir üstündüzlem kullanabileceği için daha geneldir; tek değişkenli düğüm yalnızca eksenlerle koşut bölmeler tanımlayabilir.

eğiten bir yöntem (bölüm 10) önerir. Guo ve Gelfand (1992), $K > 2$ sınıfın iki üstsınıfa bölündüğü ve ikili çok değişkenli ağaç kullanan sezgisel bir algoritma önerir. Loh ve Shih (1997), veriyi ikiye bölmek için 2 merkezli öbekleme (chapter 7) kullanır. Yıldız ve Alpaydın (2000), sınıflar ikiye ayrıldıktan sonra üstündüzlemi doğrusal ayırtaç çözümlemesiyle (bölüm 6) bulur.

Herhangi bir sınıflandırıcı, gerçek (bilinmeyen) ayırtacı denence sınıfından bir denenceyle kestirir. Tek değişkenli düğümler kullandığımızda kestirim, eksenlere koşut parçalı üstündüzlemlerden oluşur. Çok değişkenli düğümlerle daha az düğümle daha iyi yaklaşıklama sağlayabiliriz. Eğer gerçek sınır eğriyse doğrusal olmayan düğümler daha iyi genelleme yapacaktır. Dallanma değerinin de, düğümün tanımladığı ayırtaç sayısını belirlediğinden, benzer etkisi vardır. İkili bir karar düğümünün iki dalı uzayı ikiye böler. n dallı bir düğümse n parçaya böler. Yani, düğüm karmaşıklığı, dallanma değeri, ve ağaç büyüklüğü arasında bir bağımlılık vardır. Basit düğümler ve az sayıda dal olduğunda ağaçlar büyük olabilir ama örneğin tek değişkenli ikili ağaçların yorumlanabilmesi daha kolaydır. Çok değişkenli düğümleri yorumlamak daha zordur. Karmaşık düğümler daha çok veri ister; üstelik ağaçta alt düzeylere indikçe daha az verimiz kalır. Eğer düğümler karmaşık ve ağaç küçükse, ağacın canlandırdığı "karmaşık bir işi basit işlere bölme" düşüncesinden de uzaklaşırız. Kökte tek bir sınıflandırıcı kullanarak tüm sınıfları birbirinden tek düğümde ayırabiliriz, ama bu bir ağaç olmaz!

9.7 Kısa Kısa

"Böl ve yönet," Sezar zamanından beri, karmaşık bir problemi, örneğin, Galya'nın yönetimini, birden çok ve daha basit işe çevirmek için kullanılan sezgisel bir yöntem olmuştur. Bilgisayar bilimlerinde ağaçlar karmaşıklığı doğrusaldan logaritmaya indirmek için kullanılır. Karar ağaçlarının öncüleri, istatistikte (Breiman ve ötekiler 1984), yapay öğrenme de (Quinlan 1986) ve (Quinlan 1993) olmuştur. Çok değişkenli ağaçlar daha yakın yıllarda yaygınlaşmıştır; bu konuda bir özet ve karşılaştırma (Yıldız ve Alpaydın 2000)'de bulunabilir. Birçok araştırmacı, örneğin, (Guo ve Gelfand 1992), ağaçların basitliğiyle çok katmanlı algılayıcıların başarımını (bölüm 11) birleştirmeyi önermiştir. Ama birçok çalışma, tek değişkenli ağaçların oldukça başarılı ve yorumlanabilir olduğu ve bu yüzden çok değişkenli doğrusal (ya da doğrusal olmayan) düğümlerin neden olduğu ek karmaşıklığın haklı çıkarılamadığı sonucuna varmıştır. Yakın zamanlı bir özet, (Rokach ve Maimon 2005)'te verilmiştir.

Tüm değişkenli karar ağacında (Yıldız ve Alpaydın 2001), tek değişkenli, doğrusal çok değişkenli, ya da doğrusal olmayan çok değişkenli karar düğümleri birlikte bulunabilir. Ağaç büyürken her karar düğümünde o düğüme ulaşan veri farklı bir problem tanımlar ve farklı bir model gerekebilir. Her düzeyde aynı tür düğümü kullanmak girdi uzayının her yerinde aynı varsayımları yapmaya karşılık gelir. Tüm değişkenli karar ağacının her düğümünde üç farklı tür düğüm eğitilir, istatistiksel bir sınama (bölüm 19) ile geçerleme kümesi üzerindeki başarılarına göre hangisinin en iyi genelleme yapıldığı bakılır. Daha karmaşık bir model istatistiksel olarak anlamlı bir iyileştirme sağlamıyorsa en basit olan seçilir. Sonuçlar, karmaşık düğümlerin ağacın üst düzeylerinde, köke yakın yerlerinde kullanıldığını ve aşağı indikçe basit, tek değişkenli düğümlerin yeterli olduğunu göstermektedir. Yapraklara yaklaştıkça problem basitleşmekte ve aynı anda elimizdeki veri azalmaktadır. Böyle bir durumda karmaşık bir model aşırı öğrenir ve istatistiksel sınama tarafından reddedilir. Ağaçta aşağı indikçe düğüm sayısı üstel arttığından düğümlerin çoğu tek değişkenli olur ve toplam karmaşıklık fazla olmaz.

Karar ağaçları, sınıflandırma için bağlanıma göre daha yaygın olarak kullanılır. Hızlı öğrenirler, deneme zamanı kısadır, ve birçok uygulamada başarılı sonuç verirler (Murthy 1998). Yorumlanabilir olduğundan bazen karar ağacı daha başarılı başka bir modele yeğlenebilir. EĞER-İSE kuralları, uygulama alanı konusunda uzman olan kişilerce geçerlenebilir.

Herhangi bir uygulamada, daha karmaşık algoritmalar denenmeden önce bir karar ağacının kullanılması ve sonucunun kıyaslama için kullanılması önerilir. Ağaç bize önemli öznitelikleri gösterir ve tek değişkenli ağaç, öznitelik seçimi gerçekleştirir. Tek değişkenli ağacın bir başka üstünlüğü kesikli ve sayısal öznitelikleri birini ötekine çevirmeye gerek görmeden birlikte kullanabilmesidir.

Karar ağacı, 8. bölümde gördüğümüz örnek tabanlı yöntemler gibi dağılımdan bağımsızdır, ama aralarında bazı farklar vardır:

- Her yaprak bir "kutucuğa" karşılık gelir ama yaprakların aynı genişlikte (Parzen pencerelerindeki gibi) ya da eşit sayıda örnek içermesi (en yakın k komşu

algoritmasındaki gibi) gerekmez.

- Girdi uzayını kutucuklara bölerken girdideki benzerlikler yanında düzensizlik ya da ortalama kare hata içinde gözetimli çıktı bilgisi de kullanılır.
- Ağaç yapısı sayesinde gereken yaprak ("kutucuk") daha az hesap yapılarak bulunur.
- Bir kez karar ağacı büyütüldükten sonra öğrenme kümesini saklamaya gerek yoktur; ağacın yapısı, düğümlerin parametrelerini ve yaprakların etiketlerini saklamak yeter. Bu da tüm öğrenme kümesini saklayan örnek tabanlı algoritmalara göre bellek karmaşıklığının da daha düşük olduğunu gösterir.

Karar ağacında bir sınıfın tek bir tanımı olması gerekmez; girdi uzayında birbirinden ayrık farklı tanımları olabilir.

Ağaç, önceki bölümlerde gördüğümüz istatistiksel modellerden farklıdır. Ağaç, sınıfları ayıran sınırları modeller ama bu sınırların ötesinde nasıl dağıldıklarıyla ilgilenmez. Bu bağlamda karar ağacı *ayırtaç tabanlı*, önceki gördüklerimizse *olabilirlik tabanlıdır*. Olabilirlik tabanlı sınıflandırıcılar önce $p(x|C_i)$ dağılımlarını kestirir ve sonra Bayes kuralıyla ayırtaçları oluştururlar. Ayırtaç tabanlı yöntemlerse sınıf dağılımlarını hesaplamadan doğrudan ayırtaçları kestirir. Benzer başka yöntemleri gelecek bölümlerde tartışacağız.

9.8 Alıştırmalar

1. Gini ölçütü (denklem 9.5) ve sınıflandırma hatasını (denklem 9.6) $K > 2$ sınıf için genelleyin. Kayıp dizeyi kullanarak sınıflandırma hatasından riske genellemenin nasıl yapılacağını gösterin.
2. Sayısal bir girdiyi ikiye bölmek yerine iki eşik değeriyle üçe de bölebiliriz:

$$x_j < w_{ma}, \; w_{ma} \leq x_j < w_{mb}, \; x_j \geq w_{mb}$$

   Bu iki, w_{ma} ve w_{mb} eşik değerini öğrenmek için bir algoritma önerin. Böyle bir düğümün ikili bir düğüme göre üstünlükleri ve sakıncaları nelerdir?
3. Gerektiğinde geri adım atıp daha önce oluşturduğu bir düğümden vazgeçebilen bir ağaç büyütme algoritması önerin.
4. Tek değişkenli bir ağaçta, n değer alan kesikli bir değişken, n farklı 0/1 değerli yapay değişkenle gösterilip, n ayrı sayısal değişken gibi de kullanılabilir. Bunun üstünlük ve sakıncaları nelerdir?
5. Küresel ağaçları (denklem 9.21) büyütmek için bir algoritma önerin. Elips ağaçlara genelleyin.
6. Bağlanım ağacının yaprağında ortalama için bir değer yerine doğrusal bağlanım gerçekleyerek girdiye bağlı bir çıktı hesaplayabileceğimizi gördük. Benzer bir yöntemi sınıflandırma ağacında nasıl gerçekleyebiliriz?
7. Bağlanım için bir kural öğrenme algoritması önerin.

8. Bağlanım ağacında yaprak sınırlarındaki süreksizliklerden nasıl kurtulabiliriz?
9. Elimizde veri yanında o veriyle eğitilmiş bir karar ağacı da olsa, en yakın k komşu algoritmasını gerçeklerken bu ek bilgiyi nasıl kullanabiliriz?
10. Çok değişkenli bir karar ağacının düğümlerinde büyük olasılıkla her zaman her değişkene gerek duymayız. Bir düğümde nasıl boyut azaltabiliriz?

9.9 Kaynaklar

Breiman, L., J. H. Friedman, R. A. Olshen, C. J. Stone. 1984. *Classification and Regression Trees.* Belmont, CA: Wadsworth International Group.

Brodley, C. E., P. E. Utgoff. 1995. "Multivariate Decision Trees." *Machine Learning* 19: 45–77.

Cohen, W. 1995. "Fast Effective Rule Induction." *Twelfth International Conference on Machine Learning*, ed. A. Prieditis, S. J. Russell, 115–123. San Mateo, CA: Morgan Kaufmann.

Devroye, L., L. Györfi, G. Lugosi. 1996. *A Probabilistic Theory of Pattern Recognition.* New York: Springer.

Dietterich, T. G. 1997. "Machine Learning Research: Four Current Directions." *AI Magazine* 18: 97–136.

Fürnkranz, J., G. Widmer. 1994. "Incremental Reduced Error Pruning." In *Eleventh International Conference on Machine Learning*, ed. W. Cohen, H. Hirsh, 70–77. San Mateo, CA: Morgan Kaufmann.

Guo, H., S. B. Gelfand. 1992. "Classification Trees with Neural Network Feature Extraction." *IEEE Transactions on Neural Networks* 3: 923–933.

Loh, W.-Y., Y. S. Shih. 1997. "Split Selection Methods for Classification Trees." *Statistica Sinica* 7: 815–840.

Loh, W.-Y., N. Vanichsetakul. 1988. "Tree-Structured Classification via Generalized Discriminant Analysis." *Journal of the American Statistical Association* 83: 715–725.

Mitchell, T. 1997. *Machine Learning.* New York: McGraw-Hill.

Murthy, S. K. 1998. "Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey." *Data Mining and Knowledge Discovery* 4: 345–389.

Murthy, S. K., S. Kasif, S. Salzberg. 1994. "A System for Induction of Oblique Decision Trees." *Journal of Artificial Intelligence Research* 2: 1–32.

Quinlan, J. R. 1986. "Induction of Decision Trees." *Machine Learning* 1: 81–106.

Quinlan, J. R. 1990. "Learning Logical Definitions from Relations." *Machine Learning* 5: 239–266.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. 1995. "MDL and Categorical Theories (continued)." *Twelfth International Conference on Machine Learning*, ed. A. Prieditis, S. J. Russell, 467–470. San Mateo, CA: Morgan Kaufmann.

Rokach, L., O. Maimon. 2005. "Top-Down Induction of Decision Trees Classifiers—A Survey." *IEEE Transactions on Systems, Man, and Cybernetics–Part C* 35: 476–487.

Yıldız, O. T., E. Alpaydın. 2000. "Linear Discriminant Trees." In *Seventeenth International Conference on Machine Learning*, ed. P. Langley, 1175–1182. San Francisco: Morgan Kaufmann.

Yıldız, O. T., E. Alpaydın. 2001. "Omnivariate Decision Trees." *IEEE Transactions on Neural Networks* 12: 1539–1546.

Bölüm 10

Doğrusal Sınıflandırma

Doğrusal sınıflandırmada bir sınıfın öteki sınıflardan doğrusal ayrıldığını varsayarız. Ayırtaç tabanlı bu yaklaşımda sınıf dağılımları modellenmez, etiketli bir öğrenme kümesinden ayırtaç parametreleri doğrudan kestirilir.

10.1 Giriş

Önceki bölümlerde sınıflandırmada $g_j(\boldsymbol{x}), j = 1, \ldots, K$ ile gösterdiğimiz ayırtaç işlevlerini tanımlıyor ve girdiyi en yüksek ayırtaç değerli sınıfa atıyorduk:

$$\text{seç } C_i \text{ eğer } g_i(\boldsymbol{x}) = \max_{j=1}^{K} g_j(\boldsymbol{x})$$

Bunun için önce $\hat{P}(C_i)$ önsel olasılıklarını ve $\hat{p}(\boldsymbol{x}|C_i)$ sınıf olabilirliklerini kestiriyor, ardından Bayes kuralıyla sonsal olasılıkları hesaplıyor ve ayırtaçları bu sonsalları kullanarak tanımlıyorduk. Örneğin,

$$g_i(\boldsymbol{x}) = \log \hat{P}(C_i|\boldsymbol{x})$$

Buna *olabilirlik tabanlı sınıflandırma* diyoruz; önceki bölümlerde dağılıma bağlı (bölüm 5), yarı bağlı (bölüm 7) ve dağılımdan bağımsız (bölüm 8) yaklaşımları kullanarak $p(\boldsymbol{x}|C_i)$ olabilirliklerini nasıl kestirdiğimizi gördük.

Şimdiyse *ayırtaç tabanlı sınıflandırma* yaklaşımını göreceğiz. Bu yaklaşımda olabilirlikleri ya da sonsal dağılımları kestirmiyor, doğrudan ayırtaç için bir model yazıyoruz. 9. bölümdeki karar ağaçlarında da gördüğümüz gibi varsayımlarımız ve tanımladığımız modeller sınıflar arasındaki sınırların biçimiyle ilgili; bu sınırların içinde girdilerin nasıl dağıldığı bizi hiç ilgilendirmiyor.

Ayırtaç için Φ_i parametreli bir model

$$g_i(\boldsymbol{x}|\Phi_i)$$

tanımlıyoruz. Olabilirlik tabanlı kestirimde dağılımların parametrelerini kestiriyorduk, burada varsayımımız farklı: Sınıf dağılımlarının biçimi yerine sınıfları ayıran sınırların biçimi için bir model ailesi seçiyoruz.

Bu durumda öğrenmeyle kastettiğimiz Φ_i parametrelerinin kestirilmesi, bunun için ölçüt de ayrımın ne kadar iyi yapıldığıdır, ki bunu, bir öğrenme kümesindeki sınıflandırma başarısı olarak ölçeriz. Olabilirlik tabanlı yaklaşımdaysa dağılım parametrelerini ayrı ayrı ve yalnız o sınıfın örneklerini kullanarak hesaplıyorduk.

Ayırtaç tabanlı sınıflandırmada sınırların içindeki sınıf bölgelerinde örneklerin nasıl dağıldığı önemli değildir; bizi tek ilgilendiren sınıflar arasındaki *sınırların* kestirimidir. Ayırtaç tabanlı sınıflandırmayı savunanlar (örneğin, Vapnik 1995), sınıf dağılımlarını kestirmenin sınırları kestirmekten daha zor bir iş olduğunu ve kolay bir problemi çözmek için daha zor bir problemi çözmemek gerektiğini söyler. Bu tabii ki yalnızca sınırlar basit bir model olarak yazılabiliyorsa geçerlidir.

Bu bölümde ayırtaçların $\boldsymbol{x}$ girdisine bağlı doğrusal olduğunu varsayıyoruz:

$$g_i(\boldsymbol{x}|\boldsymbol{w}_i, w_{i0}) = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0} = \sum_{j=1}^{d} w_{ij} x_j + w_{i0} \tag{10.1}$$

Doğrusal ayırtaç, daha çok basitliği nedeniyle sık sık kullanılır: Hem bellek, hem de hesaplama karmaşıklığı $O(d)$ olur. Doğrusal modeli anlamak kolaydır: Çıktı, x_j girdilerinin ağırlıklı toplamıdır. w_j ağırlığının büyüklüğü x_j girdisinin önemini, işareti de etkisinin artı mı, eksi mi olduğunu belirtir. Çoğu işlev eklemelidir ve çıktı, farklı özniteliklerin etkilerinin toplamı olarak yazılabilir; bu etkiler destekleyici (artı) ya da engelleyici (eksi) olabilir. Örneğin bankalar bir müşterinin riskini çoğu zaman farklı özelliklerin toplamı olarak yazar; örneğin, yıllık gelirin eksi bir etkisi vardır (gelirin artması riski azaltır).

Çoğu uygulamada doğrusal sınıflandırma başarılıdır da. Örneğin sınıf dağılımları normal olup eşdeğişinti dizeyleri ortak olduğunda en iyi ayırtacın doğrusal olduğunu biliyoruz. Ama doğrusal ayırtaç bu varsayımlar doğru olmadığında bile kullanılabilir ve parametreleri sınıf dağılımlarıyla ilgili varsayımlar yapmadan kestirilebilir. Daha karmaşık bir model kullanmadan önce her zaman daha ucuz doğrusal modeli kullanıp karşılaştırıp ek karmaşıklığı haklı çıkarmak gerekir.

Her zaman yaptığımız gibi doğrusal ayırtaç işlevini öğrenmeyi, bir hata işlevini en küçük yapan parametrelerin bulunması olarak tanımlayacağız. Bunu yaparken hata işlevinin eğimini kullanan yöntemlere sık sık başvuracağız.

10.2 Doğrusal Modelin Genelleştirilmesi

Doğrusal (birinci dereceden) model yeterince esnek değilse, *ikinci dereceden ayırtaç* kullanabiliriz:

$$g_i(\boldsymbol{x}|\mathbf{W}_i, \boldsymbol{w}_i, w_{i0}) = \boldsymbol{x}^T \mathbf{W}_i \boldsymbol{x} + \boldsymbol{w}_i \boldsymbol{x} + w_{i0} \tag{10.2}$$

Sakıncası karmaşıklığın $O(d^2)$ olmasıdır. Burada yanlılık-değişinti ikilemi devreye girer: ikinci (ya da daha yüksek) dereceden model daha geneldir ama daha büyük

öğrenme kümesi gerektirir ve küçük örneklemleri aşırı öğrenme riski vardır.

Buna denk bir yaklaşım girdiyi önişleyip *üst düzey terimler*, ya da bir başka deyişle, *çarpım terimleri* eklemektir. Örneğin, iki girdi x_1 ve x_2 için yeni girdiler ekleyebilir

$$z_1 = x_1, z_2 = x_2, z_3 = x_1^2, z_4 = x_2^2, z_5 = x_1 x_2$$

ve $\boldsymbol{z} = [z_1, z_2, z_3, z_4, z_5]^T$ yöneyini girdi olarak kullanabiliriz. Beş boyutlu $\boldsymbol{z}$ uzayında yazacağımız doğrusal ayırtaç, iki boyutlu $\boldsymbol{x}$ uzayında doğrusal olmayan bir ayırtaç tanımlayacaktır. Yaptığımız ilk uzayda doğrusal olmayan bir ayırtaç ya da bağlanım işlevi öğrenmek yerine doğrusal olmayan bir dönüşümle yeni bir uzaya geçip orada doğrusal bir işlev tanımlamaktır.

Ayırtacı $\phi_{ij}(\boldsymbol{x})$ ile gösterdiğimiz *taban işlevlerinin* ağırlıklı bir toplamı olarak yazabiliriz:

$$g_i(\boldsymbol{x}) = \sum_{j=1}^{k} w_j \phi_{ij}(\boldsymbol{x}) \tag{10.3}$$

Yüksek dereceden terimler olası taban işlevlerinden yalnızca biridir; başka örnekler de yazılabilir:

- $\sin(x_1)$
- $\exp(-(x_1 - m)^2/c)$
- $\exp(-\|\boldsymbol{x} - \boldsymbol{m}\|^2/c)$
- $\log(x_2)$
- $1(x_1 > c)$
- $1(ax_1 + bx_2 > c)$

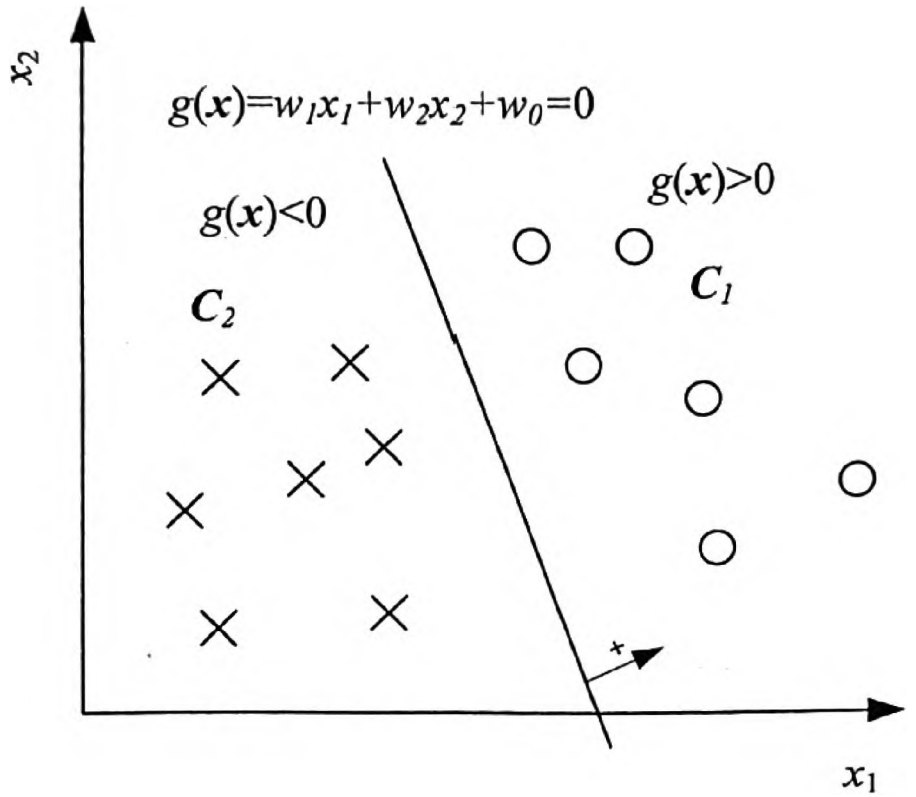
Burada, m, a, b, c sabit sayıllar ve $\boldsymbol{m}$, d boyutlu bir yöneydir. $1(b)$, b doğru olduğunda 1, yanlış olduğunda 0 döndürür. Doğrusal olmayan bir işlevi, doğrusal olmayan taban işlevlerinin doğrusal bir toplamı olarak yazmak eski bir yaklaşımdır (Aizerman, Braverman, ve Rozonoer 1964). Çok katmanlı algılayıcılar (bölüm 11) ve dairesel taban işlevlerinde (bölüm 12) parametreler vardır ve bunlar bir öğrenme kümesinden öğrenilebilir. 13. bölümde göreceğimiz destek yöney makinelerindeki çekirdek işlevleri yine taban işlevleri kullanılarak tanımlanır.

10.3 Doğrusal Ayırtacın Geometrisi

10.3.1 İki Sınıf

Daha kolay olan iki sınıflı durumla başlayalım. İki sınıf varsa bir ayırtaç yeterlidir:

$$g(\boldsymbol{x}) = g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$$



Şekil 10.1: İki boyutlu uzayda iki sınıfı ayıran ayırtaç bir doğrudur.

$$\begin{aligned} &= (\boldsymbol{w}_1^T\boldsymbol{x} + w_{10}) - (\boldsymbol{w}_2^T\boldsymbol{x} + w_{20}) \\ &= (\boldsymbol{w}_1 - \boldsymbol{w}_2)^T\boldsymbol{x} + (w_{10} - w_{20}) \\ &= \boldsymbol{w}^T\boldsymbol{x} + w_0 \end{aligned}$$

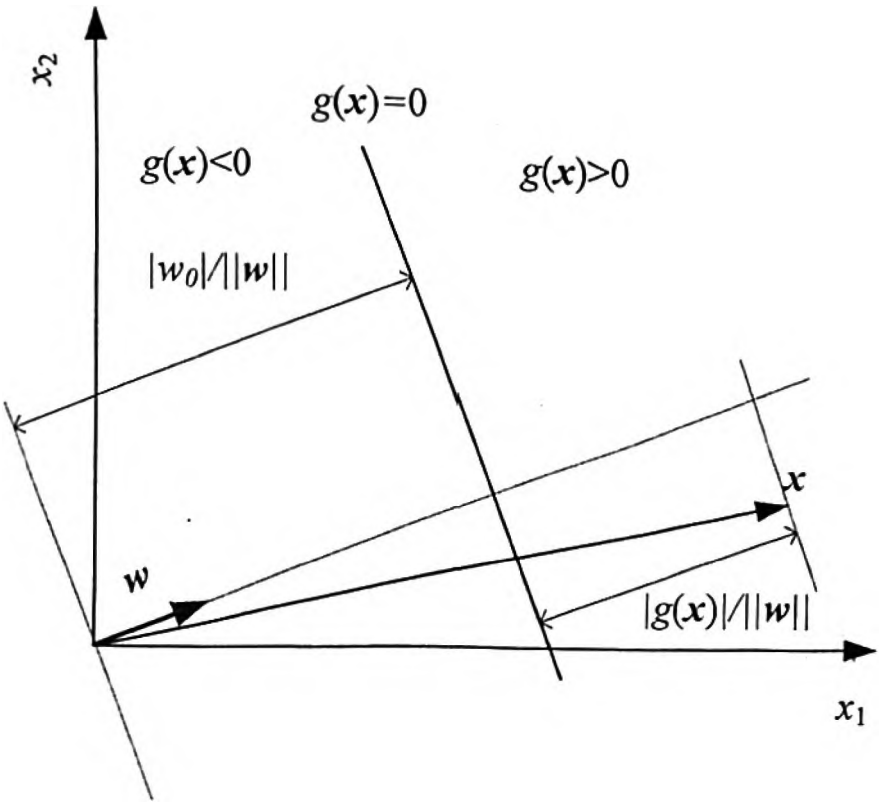
ve ayırtacın işaretine göre karar veririz:

$$\text{Seç} \begin{cases} C_1 & \text{eğer } g(\boldsymbol{x}) > 0 \\ C_2 & \text{değilse} \end{cases}$$

Bu ayırtaç, *ağırlık yöneyi* $\boldsymbol{w}$ ve *eşik* değeri w_0 olan bir üstündüzlem tanımlar. Karar kuralını, $\boldsymbol{w}^T\boldsymbol{x} > -w_0$ ise C_1 sınıfını seç, değilse C_2 sınıfını seç, biçiminde de yazabiliriz ve w_0 değerine "eşik" dememizin nedeni budur. Üstündüzlem girdi uzayını, C_1 için $\mathcal{R}_1$ ve C_2 için $\mathcal{R}_2$ olacak biçimde iki yarım uzaya böler. $\boldsymbol{x} \in \mathcal{R}_1$ sağlayan girdiler üstündüzlemin *artı*, $\boldsymbol{x} \in \mathcal{R}_2$ sağlayanlar da *eksi* yanındadır. $\boldsymbol{x} = \boldsymbol{0}$ olduğunda $g(\boldsymbol{x}) = w_0$ olur; dolayısıyla $w_0 > 0$ ise sıfır noktası üstündüzlemin artı yanında, $w_0 < 0$ ise sıfır noktası üstündüzlemin eksi yanındadır ve $w_0 = 0$ ise üstündüzlem sıfır noktasından geçer (bkz. şekil 10.1).

İkisi de karar düzlemi üzerinde olan iki $\boldsymbol{x}_1$ ve $\boldsymbol{x}_2$ noktası alalım; yani, $g(\boldsymbol{x}_1) = g(\boldsymbol{x}_2) = 0$ olsun. Bu durumda

$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{x}_1 + w_0 &= \boldsymbol{w}^T\boldsymbol{x}_2 + w_0 \\ \boldsymbol{w}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2) &= 0 \end{aligned}$$



Şekil 10.2: Doğrusal ayırtacın geometrik yorumlanması.

Buradan w yöneyinin üstündüzlem üzerindeki herhangi bir yöneye dik olduğunu görürüz. x yöneyini

$$x = x_p + r\frac{w}{\|w\|}$$

olarak da yazabiliriz (Duda, Hart, ve Stork 2001). x_p, x yöneyinin üstündüzleme normal izdüşümü ve r, x yöneyinin üstündüzleme uzaklığı olup x eksi yandaysa eksi, artı yandaysa artı olur (bkz. şekil 10.2). $g(x_p) = 0$ olduğu için

$$r = \frac{g(x)}{\|w\|} \tag{10.4}$$

Sıfır noktasına olan uzaklığı yazalım:

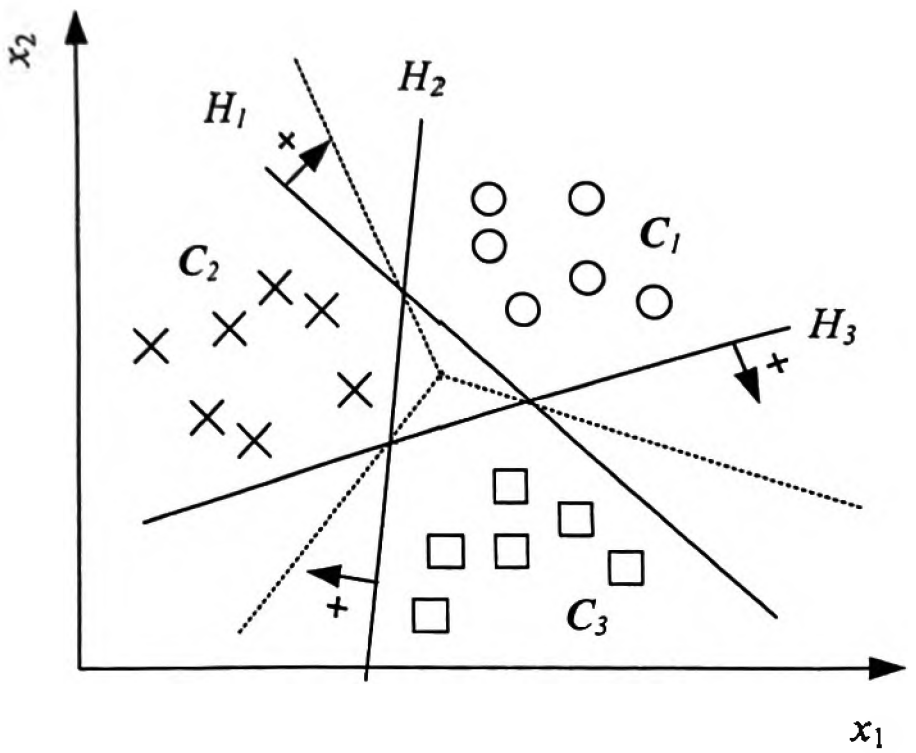
$$r_0 = \frac{w_0}{\|w\|} \tag{10.5}$$

Demek ki, w_0 üstündüzlemin sıfır noktasına görece yerini, w da yönünü belirler.

10.3.2 $K > 2$ Sınıf

$K > 2$ sınıf olduğunda K tane ayırtaç gerekir. Doğrusal olduklarını varsayalım:

$$g_i(x|w_i, w_{i0}) = w_i^T x + w_{i0} \tag{10.6}$$



Şekil 10.3: Doğrusal sınıflandırmada H_i üstündüzlemi, C_i sınıfının örneklerini öteki tüm sınıfların örneklerinden ayırır. Başarılı olabilmesi için sınıfların doğrusal ayrılabilmesi gerekir. Kesikli çizgiler, sınıflandırıcın karar sınırlarını gösterir.

Öğrenmeden az sonra bahsedeceğiz; şimdilik, $\boldsymbol{w}_i, w_{i0}$, parametrelerinin

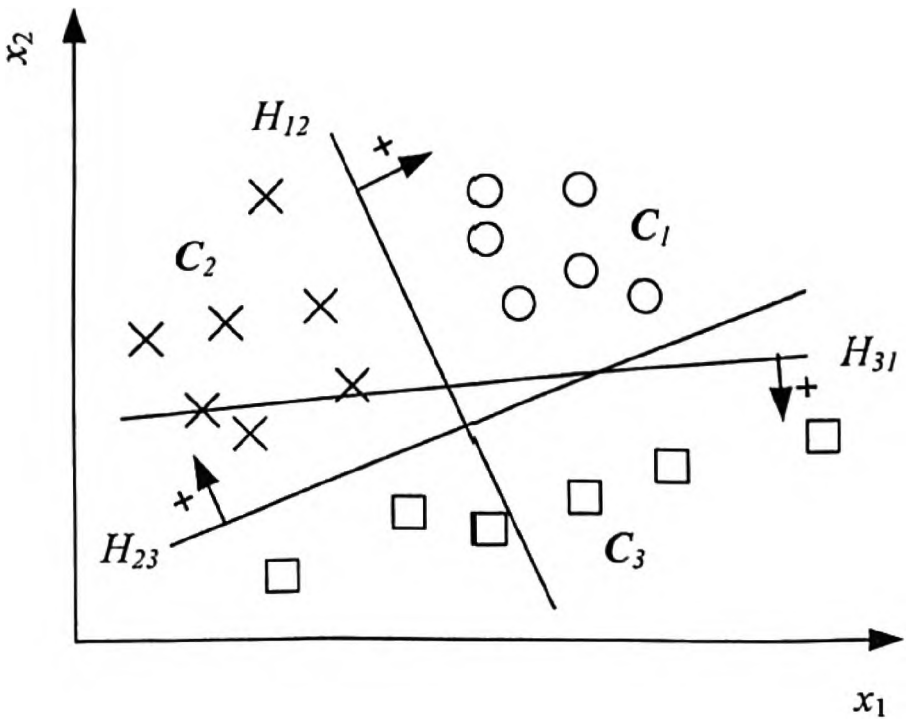
$$g_i(\boldsymbol{x}|\boldsymbol{w}_i, w_{i0}) = \begin{cases} > 0 & \text{eğer } \boldsymbol{x} \in C_i \\ \leq 0 & \text{değilse} \end{cases} \tag{10.7}$$

koşullarını öğrenme kümesindeki tüm $\boldsymbol{x}$ için sağladığını varsayalım. Böyle ayırtaçların kullanılması sınıfların *doğrusal ayrılabilir* olduğunu varsayar; yani, her C_i için öyle bir üstündüzlem H_i vardır ki tüm $\boldsymbol{x} \in C_i$ örnekleri onun artı yanına, tüm $\boldsymbol{x} \in C_j, j \neq i$ örnekleri de eksi yanına düşer (bkz. şekil 10.3).

Deneme sırasında, $\boldsymbol{x}$ verildiğinde istenen $g_j(\boldsymbol{x}), j = 1, \ldots, K$ ayırtaçlarından birinin sıfırdan büyük ötekilerin sıfırdan küçük değer vermesidir ama bu her zaman sağlanmaz: Üstündüzlemlerin artı yarım uzayları kısmen örtüşebilir, ya da bir örnek için tüm $g_j(\boldsymbol{x}) < 0$ olabilir. Böyle örnekler için bir sınıf seçmek yerine onları *reddedebiliriz*. En sık yapılansa $\boldsymbol{x}$ örneğini en yüksek ayırtaç değerli sınıfa atamaktır:

$$\text{Seç } C_i \text{ eğer } g_i(\boldsymbol{x}) = \max_{j=1}^{K} g_j(\boldsymbol{x}) \tag{10.8}$$

Girdinin sınıra uzaklığının $|g_i(\boldsymbol{x})|/\|\boldsymbol{w}_i\|$ olduğunu hatırlayalım. Eğer tüm $\boldsymbol{w}_i$ yöneyleri yaklaşık aynı büyüklükteyse, bu biçimde karar vermek girdiyi, $g_j(\boldsymbol{x}) > 0$ koşulu sağlayan sınıflar içinde sınırının en içine düşen sınıfa atar. Buna, *doğrusal sınıflandırıcı* denir ve girdi uzayını K dışbükey $\mathcal{R}_i$ alanına ayırır (bkz. şekil 10.3).



Şekil 10.4: İkili doğrusal sınıflandırmada her sınıf çifti için bir üstündüzlem vardır. Girdinin C_1 sınıfına atanması için H_{12} ve H_{13} sınırlarının artı (ya da H_{31} sınırının eksi) yanında olması gerekir; H_{23} bizi ilgilendirmez. Bu örnekte, C_1 tüm öteki sınıflardan doğrusal ayrılamasa sa ikili olarak doğrusal ayrılabilir.

10.4 İkili Sınıflandırma

Sınıflar doğrusal ayrılamıyorsa bir yaklaşım problemi birden çok doğrusal alt probleme bölmektir. Bir olasılık, *ikili sınıflandırmadır* (Duda, Hart, ve Stork 2001). Her sınıf çifti için bir tane olmak üzere $K(K-1)/2$ tane ayırtaç (örneğin, doğrusal) $g_{ij}(\boldsymbol{x})$, tanımlanır (bkz. şekil 10.4):

$$g_{ij}(\boldsymbol{x}|\boldsymbol{w}_{ij}, w_{ij0}) = \boldsymbol{w}_{ij}^T \boldsymbol{x} + w_{ij0}$$

$\boldsymbol{w}_{ij}, j \neq i$ ağırlık parametreleri ayrımı sağlayacak biçimde eğitilir:

$$g_{ij}(\boldsymbol{x}) = \begin{cases} > 0 & \text{eğer } \boldsymbol{x} \in C_i \\ \leq 0 & \text{eğer } \boldsymbol{x} \in C_j \quad i,j = 1,\ldots,K \text{ ve } i \neq j \\ \text{kullanılmaz} & \text{değilse} \end{cases} \tag{10.9}$$

Ayrılacak iki sınıftan da olmayan girdiler, $\boldsymbol{x}^t \in C_k, k \neq i, k \neq j$, eğitimde kullanılmaz.

Deneme sırasında bir sınıfın seçilmesi için girdinin o sınıf için tanımlı tüm ikili ayırtaçların artı yanında olması istenir:

$$\text{seç } C_i \text{ eğer } \forall j \neq i, g_{ij}(\boldsymbol{x}) > 0$$

Bazı örneklerde bu herhangi bir i için sağlanamayabilir ve eğer bu örnekleri reddetmek istemiyorsak VE işlemi yerine toplama kullanıp en büyüğü seçerek daha esnek bir seçim kuralı belirleyebiliriz:

$$g_i(\boldsymbol{x}) = \sum_{j \neq i} g_{ij}(\boldsymbol{x}) \tag{10.10}$$

Sınıflar doğrusal olarak ayrılamasa da eğer ikişer ikişer doğrusal ayrılabiliyorsaki bunun sağlanma olasılığı daha yüksektir—ikili sınıflandırma kullanarak sınıfları doğrusal olmayan biçimde ayırabiliriz (bkz. şekil 10.4). Bu yaptığımız, karmaşık (örneğin, doğrusal olmayan) bir işi daha basit (örneğin, doğrusal) alt işlere bölmeye bir örnektir. Daha önce gördüğümüz karar ağaçları da (bölüm 9) aynı düşünceye dayanıyordu; başka örneklerini 17. bölümde model birleştirmeyi tartışırken, örneğin, hata düzelten çıktı kodları ve uzmanların karışımında ($\mathcal{O}(K^2)$ taneden daha az doğrusal modeli birleştirerek) göreceğiz.

10.5 Dağılıma Bağlı Sınıflandırmaya Yeniden Bir Bakış

5. bölümde, $p(\boldsymbol{x}|C_i)$ sınıf dağılımları Gauss ve eşdeğişinti dizeyleri ortaksa, en iyi ayırtacın doğrusal olduğunu görmüştük:

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0} \tag{10.11}$$

Ayırtacın parametrelerini dağılımın parametrelerini kullanarak hesaplayabiliyorduk:

$$\begin{aligned} \boldsymbol{w}_i &= \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i \\ w_{i0} &= -\frac{1}{2} \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i + \log P(C_i) \end{aligned} \tag{10.12}$$

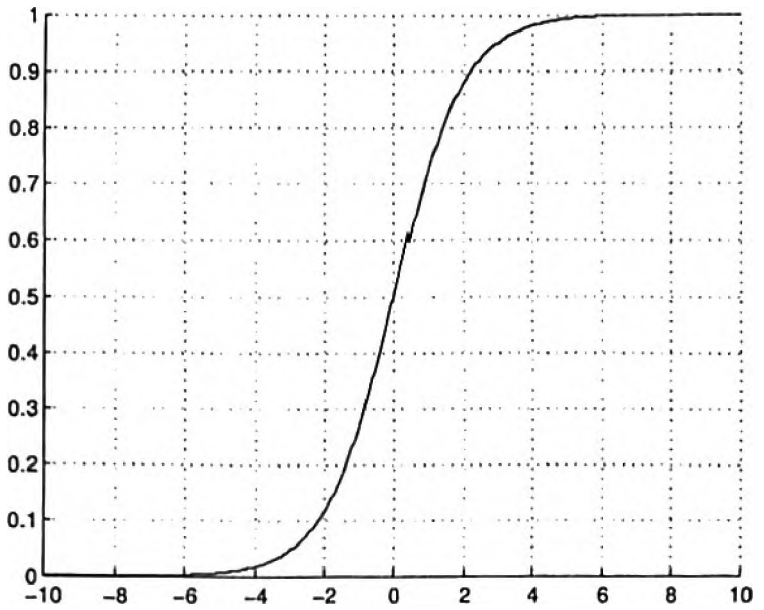
Öğrenme verisi verildiğinde önce $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}$, ve $\hat{P}(C_i)$ için en büyük olabilirlik kestirimlerini hesaplıyor, ardından bunları denklem 10.12'ye yerleştirip doğrusal ayırtacı yazıyorduk.

İki sınıflı durumu bir kez daha görelim. $y \equiv P(C_1|\boldsymbol{x})$ ve $P(C_2|\boldsymbol{x}) = 1 - y$ olsun. Sınıflandırma yaparken sonsal olasılığı yüksek sınıfı seçeriz:

$$\text{seç } C_1 \text{ eğer } \begin{cases} y > 0.5 \\ \frac{y}{1-y} > 1 \\ \log \frac{y}{1-y} > 0 \end{cases} \quad \text{değilse } C_2 \text{ sınıfını seç.}$$

$\log y/(1-y)$, y değerinin *ters S* ya da *logit* dönüşümü, ya da *log olasılık oranı* olarak bilinir. Ortak eşdeğişinti dizeyli iki normal sınıf için log olasılık oranı doğrusaldır:

$$\text{logit}(P(C_1|\boldsymbol{x})) = \log \frac{P(C_1|\boldsymbol{x})}{1 - P(C_1|\boldsymbol{x})} = \log \frac{P(C_1|\boldsymbol{x})}{P(C_2|\boldsymbol{x})}$$



Şekil 10.5: sigmoid (S) işlevi.

$$
\begin{aligned}
&= \log \frac{p(\boldsymbol{x}|C_1)}{p(\boldsymbol{x}|C_2)} + \log \frac{P(C_1)}{P(C_2)} \\
&= \log \frac{(2\pi)^{-d/2}|\boldsymbol{\Sigma}|^{-1/2} \exp[-(1/2)(\boldsymbol{x}-\boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1)]}{(2\pi)^{-d/2}|\boldsymbol{\Sigma}|^{-1/2} \exp[-(1/2)(\boldsymbol{x}-\boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2)]} + \log \frac{P(C_1)}{P(C_2)} \\
&= \boldsymbol{w}^T \boldsymbol{x} + w_0 \qquad (10.13)
\end{aligned}
$$

öyle ki

$$
\begin{aligned}
\boldsymbol{w} &= \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\
w_0 &= -\frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) + \log \frac{P(C_1)}{P(C_2)} \qquad (10.14)
\end{aligned}
$$

Logit işlevinin tersi

$$\log \frac{P(C_1|\boldsymbol{x})}{1 - P(C_1|\boldsymbol{x})} = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

sigmoid ya da *S işlevidir* (bkz. şekil 10.5):

$$P(C_1|\boldsymbol{x}) = \text{sigmoid}(\boldsymbol{w}^T \boldsymbol{x} + w_0) = \frac{1}{1 + \exp[-(\boldsymbol{w}^T \boldsymbol{x} + w_0)]} \qquad (10.15)$$

Öğrenme sırasında $\boldsymbol{m}_1, \boldsymbol{m}_2, \mathbf{S}, \hat{P}(C_1)$ değerlerini kestirip denklem 10.14'e koyup ayırtaç parametrelerini belirleriz. Deneme sırasında, yeni $\boldsymbol{x}$ verildiğinde

1. $g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$ ayırtaç değerini hesaplayıp $g(\boldsymbol{x}) > 0$ ise C_1 sınıfını seçebiliriz, ya da

2. Sonsal olasılığı $y = \text{sigmoid}(\mathbf{w}^T\mathbf{x} + w_0)$ olarak kestirip $y > 0.5$ ise C_1 sınıfını seçebiliriz ($\text{sigmoid}(0) = 0.5$).

İkinci seçenekte, sigmoid işlevi ayırtaç değerini bir sonsal olasılık değerine çevirir. Burada iki sınıf ve tek ayırtaç vardır; $K > 2$ sınıf olduğunda sonsal olasılıkların ayırtaçlardan nasıl kestirilebildiğini 10.7. bölümde göreceğiz.

10.6 Eğim İniş

Olabilirlik tabanlı sınıflandırmada parametrelerimiz, $p(\mathbf{x}|C_i)$ ve $P(C_i)$ dağılımlarının yeterli istatistikleriydi ve bunları en büyük olabilirlik kestirimiyle hesaplıyorduk. Ayırtaç tabanlı sınıflandırmada parametreler ayırtacı tanımlar ve öğrenme kümesindeki sınıflandırma hatasını en küçük yapacak biçimde kestirilir. $E(\mathbf{w}|\mathcal{X})$, $\mathbf{w}$ parametreleriyle $\mathcal{X}$ öğrenme kümesi üzerinde yapılan hatayı göstersin:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w}|\mathcal{X})$$

Bazılarını az sonra göreceğimiz birçok durumda en iyi parametre dğerleri çözümsel hesaplanamaz ve yinelemeli yöntemler kullanmamız gerekir. En sık ve kolay kullanılan *eğim iniştir*. Eğer $E(\mathbf{w})$ hata işlevinin türevi alınabiliyorsa *eğim yöneyi* kısmi türevlerden oluşur:

$$\nabla_{\mathbf{w}} E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_d}\right]^T$$

ve *eğim iniş* algoritması, rastgele bir ilk değerden başlayıp her adımda $\mathbf{w}$ yöneyini eğim yöneyinin tersi yönde günceller:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \forall i \tag{10.16}$$

$$w_i = w_i + \Delta w_i \tag{10.17}$$

Burada, η ile gösterilen *adım büyüklüğü* ya da *öğrenme çarpanıdır*, ve eğim yönünde ne kadar gidileceğini belirler. Eğim çıkış, bir işlevin değerini büyütmek için kullanılır ve parametreleri eğim yöneyi yönünde günceller. En küçük (ya da en büyük) noktada türev 0 olacağından algoritma durur. Bu da bu algoritmanın en yakın en küçük noktayı bulacağını gösterir; durulan nokta yerel en küçük bir nokta olabilir ve küresel en küçük nokta bulma güvencemiz yoktur. İyi bir η değeri kullanmak önemlidir; küçük olursa yakınsama yavaş olur, büyük olduğunda salınımlar, hatta ıraksama olabilir.

Bu kitap boyunca basit ve çoğunlukla etkili olduğu için yinelemeli öğrenmede eğim iniş yöntemlerini sıkça kullanacağız, ama bir model ve hata işlevi verildiğinde onu en küçük yapan model parametrelerini bulmak için kullanılabilecek birçok yöntem olduğunu hep aklımızda tutalım. İkinci dereceden ya da eşlenik yöntemler daha fazla bellek ve hesapla daha hızlı yakınsayabilir. Tavlama benzetimi ya da genetik algoritmalar parametre uzayını daha kapsamlı araştırabilir ve ilk değerlerden göreceli olarak daha bağımsızdır.

10.7 S Biçimli Sınıflandırma

10.7.1 İki Sınıf

S biçimli sınıflandırmada $p(\boldsymbol{x}|C_i)$ sınıf dağılımlarını değil oranlarını modelleriz. İki sınıfla başlayalım ve ve log olabilirlik oranının doğrusal olduğunu varsayalım:

$$\log \frac{p(\boldsymbol{x}|C_1)}{p(\boldsymbol{x}|C_2)} = \boldsymbol{w}^T\boldsymbol{x} + w_0^o \tag{10.18}$$

Sınıf dağılımları normal olduğunda bu sağlanır (denklem 10.13), ama S biçimli sınıflandırmanın uygulama alanı daha geniştir; örneğin, $\boldsymbol{x}$ girdisi kesikli özniteliklerden ya da kesikli ve sürekli özniteliklerin karışımından oluşabilir.

Bayes kuralıyla sonsal olasılıkları yazalım:

$$\begin{aligned}\text{logit}(P(C_1|\boldsymbol{x})) &= \log \frac{P(C_1|\boldsymbol{x})}{1 - P(C_1|\boldsymbol{x})} \\ &= \log \frac{p(\boldsymbol{x}|C_1)}{p(\boldsymbol{x}|C_2)} + \log \frac{P(C_1)}{P(C_2)} \\ &= \boldsymbol{w}^T\boldsymbol{x} + w_0 \end{aligned} \tag{10.19}$$

öyle ki

$$w_0 = w_0^o + \log \frac{P(C_1)}{P(C_2)} \tag{10.20}$$

Buradan $P(C_1|\boldsymbol{x})$ sonsal olasılığını S biçimli sigmoid işlevini kullanarak elde ettiğimizi görüyoruz:

$$y = \hat{P}(C_1|\boldsymbol{x}) = \frac{1}{1 + \exp[-(\boldsymbol{w}^T\boldsymbol{x} + w_0)]} \tag{10.21}$$

Şimdi de $\boldsymbol{w}$ ve w_0 parametrelerini nasıl öğrenebileceğimizi görelim. Elimizde iki sınıflı bir $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}$ örneklemi olsun ($\boldsymbol{x} \in C_1$ ise $r^t = 1$, $\boldsymbol{x} \in C_2$ ise $r^t = 0$). r^t rastsal değişkeninin Bernoulli dağıldığını ve olasılığının ($\boldsymbol{x}^t$ girdisine denklem 10.21'deki gibi bağlı) $y^t \equiv P(C_1|\boldsymbol{x}^t)$ olduğunu varsayıyoruz:

$$r^t|\boldsymbol{x}^t \sim \text{Bernoulli}(y^t)$$

Burada, $p(\boldsymbol{x}|C_i)$ dağılımını modellediğimiz olabilirlik tabanlı yaklaşımla farkı görüyoruz: Ayırtaç tabanlı yaklaşımda, $\boldsymbol{x}$ dağılımını modellemeden, doğrudan $r|\boldsymbol{x}$ için dağılımı modelliyoruz. Örneklem olabilirliğini yazalım:

$$l(\boldsymbol{w}, w_0|\mathcal{X}) = \prod_t (y^t)^{(r^t)}(1 - y^t)^{(1-r^t)} \tag{10.22}$$

Büyütmek istediğimiz bir olabilirlik işlevini, $E = -\log l$ yazıp küçültülecek bir hata işlevine çevirebildiğimizi daha önce gördük. Bu durumda, böyle tanımlanan hataya *çapraz düzensizlik* denir:

$$E(\boldsymbol{w}, w_0|\mathcal{X}) = -\sum_t r^t \log y^t + (1 - r^t)\log(1 - y^t) \tag{10.23}$$

S işlevi doğrusal olmadığı için doğrudan çözüm yoktur ama çapraz düzensizlik üzerinde eğim iniş kullanabiliriz, ki bu olabilirlik, ya da log olabilirlik, üzerinde eğim çıkış yapmaya denktir. $y = \text{sigmoid}(a) = 1/(1+\exp(-a))$ işlevinin türevini yazalım:

$$\frac{dy}{da} = y(1-y)$$

ve eğim inen güncellemeleri türetelim:

$$\begin{aligned}
\Delta w_j &= -\eta \frac{\partial E}{\partial w_j} = \eta \sum_t \left(\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t} \right) y^t(1-y^t)x_j^t \\
&= \eta \sum_t (r^t - y^t)x_j^t, j = 1, \ldots, d \\
\Delta w_0 &= -\eta \frac{\partial E}{\partial w_0} = \eta \sum_t (r^t - y^t)
\end{aligned} \tag{10.24}$$

w_j ağırlıklarının ilk değerleri 0 çevresinde alınır; örneğin $[-0.01, 0.01]$ aralığından rastgele değerler seçebiliriz. Çünkü ilk w_j değerleri büyük olursa ağırlıklı toplam da büyük olabilir ve S işlevinin sağ ya da sol ucunda, türevin sıfıra yakın olduğu yerde kalabiliriz. Şekil 10.5'te gördüğümüz gibi eğer ilk ağırlıklar sıfıra yakın olursa ağırlıklı toplam sıfıra yakın ve türev sıfırdan farklı olacak, böylece güncelleme olabilecektir. Eğer ağırlıklı toplamın mutlak değeri büyükse (-5'ten küçük ya da $+5$'ten büyükse), türev sıfıra yakın olur ve ağırlıklar güncellenmez.

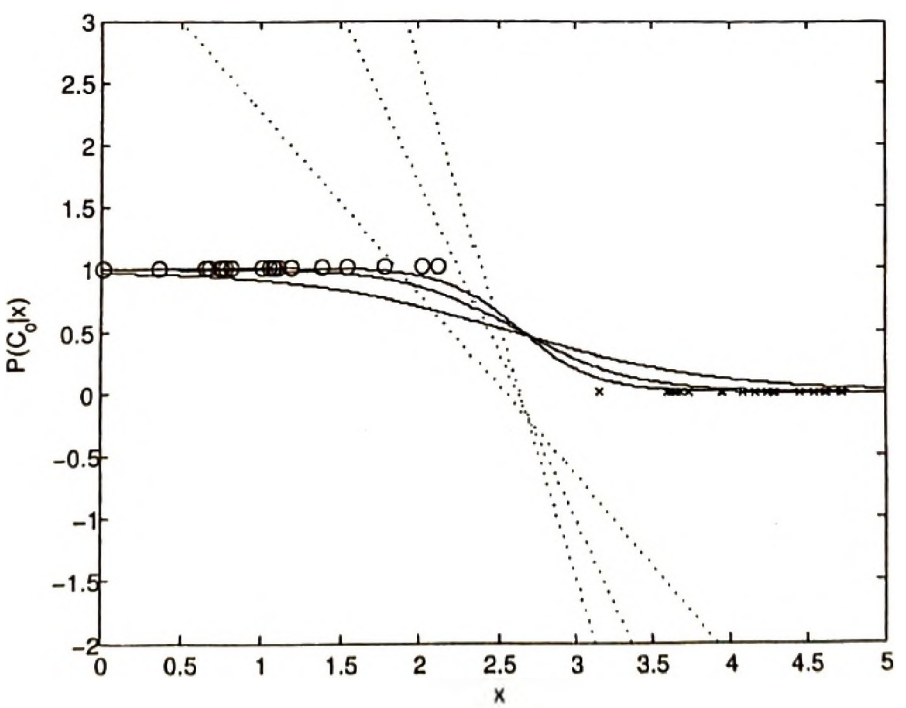
Örnek program şekil 10.6'da verilmiştir. Tek boyutlu bir girdi için örnek şekil 10.7'de görülebilir. Hem $wx+w_0$ doğrusu, hem de S işlevinin çıktısı öğrenme sürdükçe çizilmiştir. 0 ya da 1 çıktı değerlerini elde edebilmek için S işlevinin gittikçe keskinleştiğini görüyoruz, ki bu w değerinin mutlak değerini, ya da çok boyutlu durumda $\|w\|$ değerini artırarak sağlanır.

Öğrenme bitip elimize son w ve w_0 değerleri geçtiğinde x^t deneme örneği için $y^t = S(w^T x^t + w_0)$ çıktısını hesaplar ve $y^t > 0.5$ ise C_1, değilse C_2 sınıfını seçeriz. Yani sınıflandırma hatasını azaltmak istiyorsak tüm y^t çıktılarının 0 ya da 1 olmasını beklemeye gerek yoktur, 0.5 eşiğinin, yani ayırtacın, doğru yanında olmaları yeterlidir. Bu noktadan sonra öğrenmeyi sürdürdüğümüzde çapraz düzensizlik azalır ($|w_j|$ artar), ama yanlış sınıflandırmalar azalmaz. Genelde yanlış sınıflandırılan örneklerin sayısı azalmayıncaya dek öğrenmeyi (eğim inişi) sürdürürüz; eğer sınıflar doğrusal olarak ayrılabiliyorsa hiç yanlış sınıflandırma olmaz. 0 öğrenme hatasına ulaşmadan *erken durmak* bir tür düzenlileştirme sağlar. Ağırlıkları 0 çevresinden başlattığımız ve öğrenme sürdükçe mutlak değerleri büyüdüğü için erken durmak daha çok ağırlığın sıfıra yakın olmasını, yani bir açıdan daha az parametresi olan bir modelin oluşmasını sağlar.

Burada her ne kadar ayırtacı türetirken sınıfların log oranlarının doğrusal olduğunu varsaydıysak da sonsal olasılığı kestirdiğimize ve ara bir aşama olarak $p(x|C_i)$ ya da $P(C_i)$ dağılımlarını kestirmediğimize bir kez daha dikkat edin.

```
Tüm j = 0, ..., d
    w_j ←rand(-0.01,0.01)
Yinele
    Tüm j = 0, ..., d
        Δw_j ← 0
    Tüm t = 1, ..., N
        o ← 0
        Tüm j = 0, ..., d
            o ← o + w_j x_j^t
        y ← S(o)
        Tüm j = 0, ..., d
            Δw_j ← Δw_j + (r^t − y) x_j^t
    Tüm j = 0, ..., d
        w_j ← w_j + η Δw_j
Yakınsamaya dek
```

Şekil 10.6: İki sınıflı S biçimli sınıflandırmada eğim iniş algoritması kullanan örnek program. w_0 için her zaman +1 değerini alan ek bir x_0 girdisi olduğunu varsayıyoruz: $x_0^t \equiv +1, \forall t$.



Şekil 10.7: Tek boyutlu ve 'o' ve '×' ile gösterilen iki sınıflı bir örnekte, $wx + w_0$ doğrusunun (noktalı) ve S çıktısının (sürekli) 10, 100, ve 1,000 eğim iniş yinelemesi sonrasındaki çizimleri.

10.7.2 $K > 2$ Sınıf

Şimdi S biçimli sınıflandırıcıyı $K > 2$ sınıf için genelleyelim. Sınıflardan birini, örneğin, C_K sınıfını, dayanak sınıfı olarak alıp öteki sınıfların ona olan oranını modelliyoruz:

$$\log \frac{p(\boldsymbol{x}|C_i)}{p(\boldsymbol{x}|C_K)} = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}^o \tag{10.25}$$

Bu durumda sonsal olasılıkların oranını yazalım:

$$\frac{P(C_i|\boldsymbol{x})}{P(C_K|\boldsymbol{x})} = \exp[\boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}] \tag{10.26}$$

öyle ki $w_{i0} = w_{i0}^o + \log P(C_i)/P(C_K)$.

$$\begin{aligned} \sum_{i=1}^{K-1} \frac{P(C_i|\boldsymbol{x})}{P(C_K|\boldsymbol{x})} &= \frac{1 - P(C_K|\boldsymbol{x})}{P(C_K|\boldsymbol{x})} = \sum_{i=1}^{K-1} \exp[\boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}] \\ \Rightarrow \quad P(C_K|\boldsymbol{x}) &= \frac{1}{1 + \sum_{i=1}^{K-1} \exp[\boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}]} \end{aligned} \tag{10.27}$$

ve $i = 1, \ldots, K-1$ için

$$\begin{aligned} \frac{P(C_i|\boldsymbol{x})}{P(C_K|\boldsymbol{x})} &= \exp[\boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}] \\ \Rightarrow \quad P(C_i|\boldsymbol{x}) &= \frac{\exp[\boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}]}{1 + \sum_{j=1}^{K-1} \exp[\boldsymbol{w}_j^T \boldsymbol{x} + w_{j0}]} \end{aligned} \tag{10.28}$$

Bütün sınıfları aynı biçimde tanımlayabiliriz:

$$y_i = \hat{P}(C_i|\boldsymbol{x}) = \frac{\exp[\boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}]}{\sum_{j=1}^{K} \exp[\boldsymbol{w}_j^T \boldsymbol{x} + w_{j0}]}, \quad i = 1, \ldots, K \tag{10.29}$$

buna, *eşiksiz en büyük işlevi* diyoruz (Bridle 1990). Eğer sınıflardan biri için ağırlıklı toplam ötekilere göre yeterince büyükse, üstel işlevden geçip normalleştirildiktem sonra y_i değeri bire, ötekilerinkiyse sıfıra yakın olacaktır. Yani bu işlev en büyük değeri seçen max işlevi gibi çalışır, ama farkı değerin 0 ya da 1 değil eşiksiz olması ve bu sayede türevinin alınabilmesidir; eşiksiz en büyük işlev adı da buradan gelir. Paydadaki normalleştirme $\sum_i y_i = 1$ olmasını sağlar.

Şimdi parametrelerin nasıl öğrenilebildiğini görelim. $K > 2$ sınıf durumunda her örnek katlıterimli dağılımdan tek bir çekiştir, yani, $\boldsymbol{r}^t|\boldsymbol{x}^t \sim \text{Mult}_k(1, \boldsymbol{y}^t)$, ve burada $y_i^t \equiv P(C_i|\boldsymbol{x}^t)$ ile sonsal dağılım değerlerini kestiririz. $K > 2$ sınıf için örneklem olabilirliğini

$$l(\{\boldsymbol{w}_i, w_{i0}\}_i|\mathcal{X}) = \prod_t \prod_i (y_i^t)^{r_i^t} \tag{10.30}$$

ve çapraz düzensizliği yazalım:

$$E(\{w_i, w_{i0}\}_i|\mathcal{X}) = -\sum_t \sum_i r_i^t \log y_i^t \tag{10.31}$$

Eğim inişi kullanmak için bize eşiksiz en büyük işlevinin türevi gerekiyor. $y_i = \exp(a_i)/\sum_j \exp(a_j)$ için

$$\frac{\partial y_i}{\partial a_j} = y_i(\delta_{ij} - y_j) \tag{10.32}$$

burada δ_{ij}, *Kronecker delta* olarak adlandırılır: $i = j$ için 1, $i \neq j$ için 0 değerini alır (alıştırma 3). $\sum_i r_i^t = 1$ bilgisini kullanarak $j = 1, \ldots, K$ için eğim inen güncelleme kurallarını türetelim:

$$\begin{aligned}
\Delta w_j &= \eta \sum_t \sum_i \frac{r_i^t}{y_i^t} y_i^t (\delta_{ij} - y_j^t) x^t \\
&= \eta \sum_t \sum_i r_i^t (\delta_{ij} - y_j^t) x^t \\
&= \eta \sum_t \left[\sum_i r_i^t \delta_{ij} - y_j^t \sum_i r_i^t \right] x^t \\
&= \eta \sum_t (r_j^t - y_j^t) x^t \\
\Delta w_{j0} &= \eta \sum_t (r_j^t - y_j^t)
\end{aligned} \tag{10.33}$$

Eşiksiz en büyük işlevinin paydasındaki normalleştirme yüzünden w_j ve w_{j0}, yalnızca $x^t \in C_j$ örneklerine değil $x^t \in C_i, i \neq j$ örneklerine de bağlıdır. Ayırtacın parametreleri, doğru sınıfın örnekleri için çıktıları olabildiğince büyük, öteki sınıfların örnekleri için çıktılarıysa olabildiğince küçük yapacak biçimde bulunur. Örnek program şekil 10.8'de verilmiştir. Üç sınıflı, iki boyutlu bir örnek için sınırlar şekil 10.9'da, ayırtaç düzlemleri ve sonsal olasılıklar şekil 10.10'da gösterilmiştir.

Deneme sırasında yeni bir girdi için tüm $y_k, k = 1, \ldots, K$ değerlerini hesaplayıp $y_i = \max_k y_k$ ise C_i sınıfını seçeriz. Burada da çapraz düzensizliği azaltırken erken durabiliriz; doğru sınıfın ağırlıklı toplamı en yüksek olduğunda durmak onun sonsal olasılığının ötekilerinkinden yüksek olmasına yetecektir.

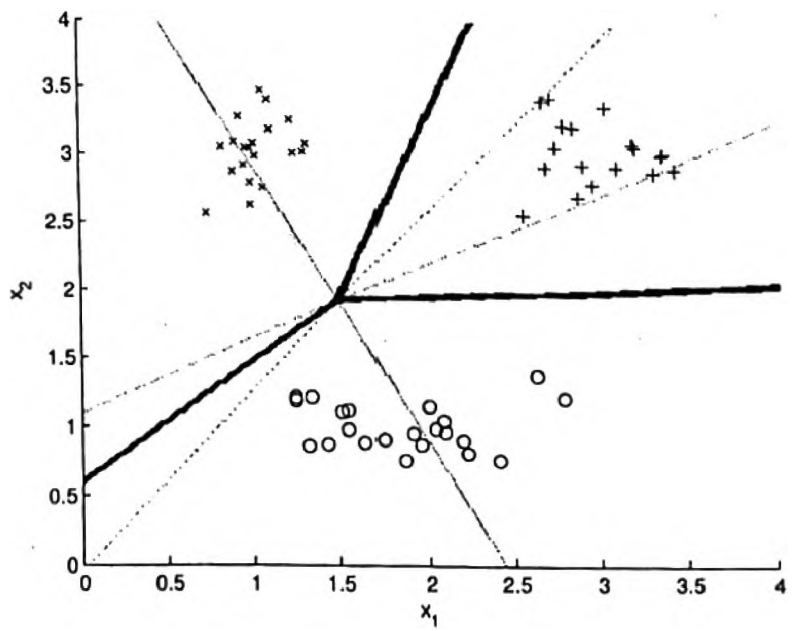
Veri normal dağıldığında S biçimli sınıflandırıcının hatası yaklaşık olarak dağılıma bağlı hesaplanan doğrusal sınıflandırıcının hatası kadardır (McLachlan 1992). Ama sınıflar ortak eşdeğişinti dizeyli olmasa bile (normal olmasalar ya da tek tepeli olmasalar) bazen doğrusal olarak ayrılabilirler ve S biçimli sınıflandırma başarıyla kullanılabilir.

Sınıf dağılımlarının oranları ille de doğrusal olacak diye bir zorunluluk yoktur (Anderson 1982; McLachlan 1992). Örneğin, ikinci dereceden olabilirler:

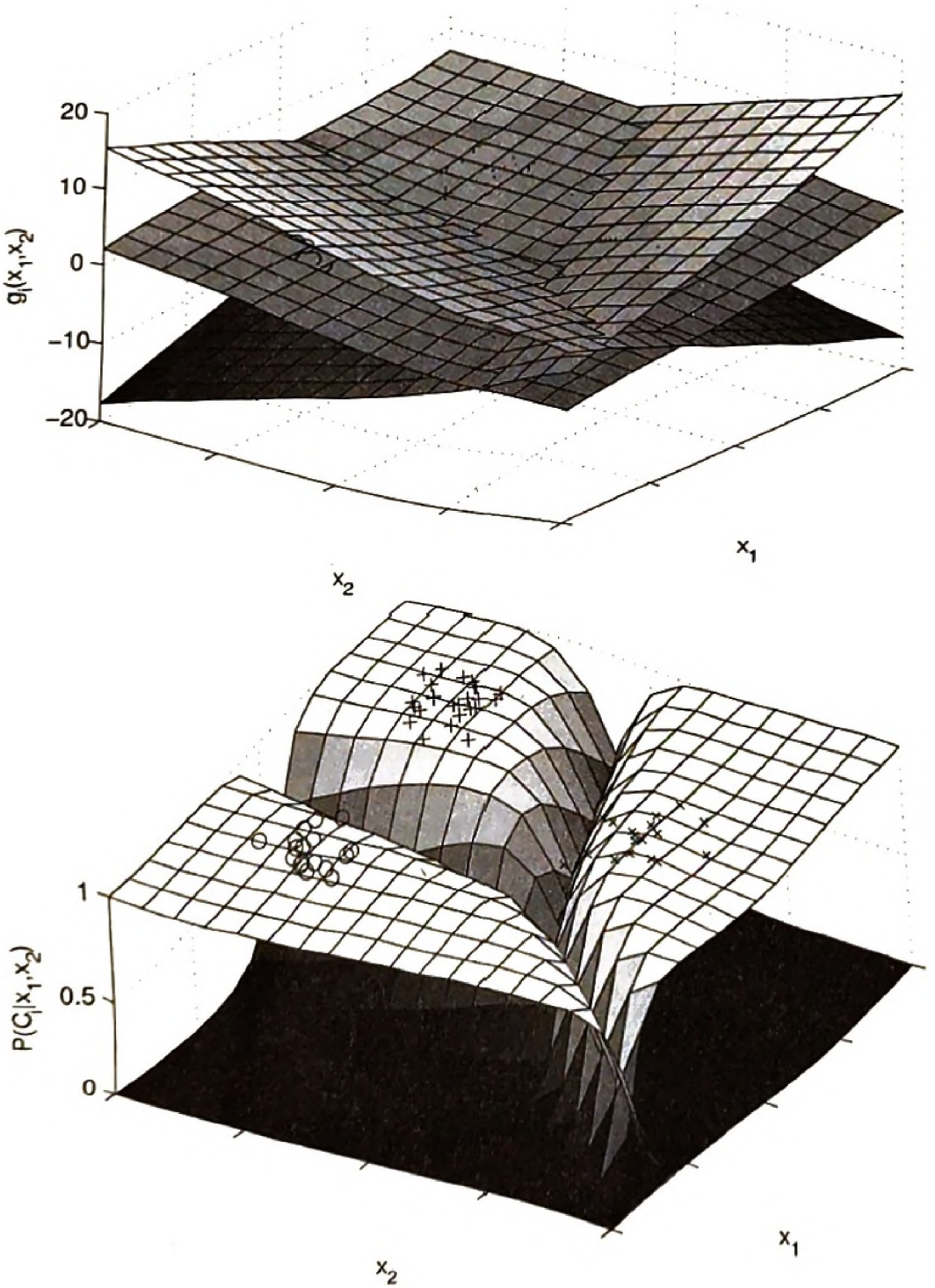
$$\log \frac{p(x|C_i)}{p(x|C_K)} = x^T \mathbf{W}_i x + w_i^T x + w_{i0} \tag{10.34}$$

```
Tüm i = 1,...,K, For j = 0,...,d, w_ij ← rand(−0.01, 0.01)
Yinele
    Tüm i = 1,...,K, Tüm j = 0,...,d, Δw_ij ← 0
    Tüm t = 1,...,N
        Tüm i = 1,...,K
            o_i ← 0
            Tüm j = 0,...,d
                o_i ← o_i + w_ij x_j^t
        Tüm i = 1,...,K
            y_i ← exp(o_i) / Σ_k exp(o_k)
        Tüm i = 1,...,K
            Tüm j = 0,...,d
                Δw_ij ← Δw_ij + (r_i^t − y_i) x_j^t
    Tüm i = 1,...,K
        Tüm j = 0,...,d
            w_ij ← w_ij + η Δw_ij
Yakınsamaya dek
```

Şekil 10.8: $K > 2$ sınıf için S biçimli sınıflandırmayı eğim inişle gerçekleyen örnek program. $x_0^t \equiv 1, \forall t$ olarak tanımlıdır.



Şekil 10.9: İki boyutlu ve üç sınıflı bir veri için S biçimli sınıflandırma çözümü. İnce çizgiler $g_i(\boldsymbol{x}) = 0$ hatlarını, kalın çizgi de en büyüğü seçince oluşan sınırı gösterir.



Şekil 10.10: Şekil 10.9'daki örnek için üstte doğrusal ayırtaçlar, altta da eşiksiz en büyük işlevinin çıkışı olan sonsal olasılıklar çizilmiştir.

Bu durumda, farklı eşdeğişinti dizeyli normal sınıfların oluşturduğu gibi ikinci dereceden bir ayırtaç oluşturur. d büyük olduğunda $\mathbf{\Sigma}_i$ dizeyini basitleştirdiğimiz (düzenlileştirdiğimiz) gibi, yalnızca en büyük özyöneylerini kullanarak $\mathbf{W}_i$ dizeyini de basitleştirebiliriz.

10.2. bölümde tartıştığımız gibi x girdilerinin herhangi bir işlevini yeni girdi olarak tanımlayabiliriz. Örneğin ayırtacı doğrusal olmayan taban işlevlerinin doğrusal bir toplamı olarak yazabiliriz:

$$\log \frac{p(\boldsymbol{x}|C_i)}{p(\boldsymbol{x}|C_K)} = \boldsymbol{w}_i^T \boldsymbol{\phi}(\boldsymbol{x}) + w_{i0} \tag{10.35}$$

Burada, $\boldsymbol{\phi}(\cdot)$ taban işlevlerini gösterir ve dönüştürülmiş yeni değişkenlere karşılık gelir. Yapay sinir ağlarında buna *çok katmanlı algılayıcı* (bölüm 11) denir ve S işlevi en yaygın kullanılan taban işlevidir. Gauss taban işlevi kullanıldığında bu modele *dairesel tabanlı işlev* (bölüm 12) denir. Tümüyle dağılımdan bağımsız bir yaklaşım, örneğin Parzen penceresi (bölüm 8) bile kullanılabilir.

10.8 Bağlanımla Sınıflandırma

Bağlanımda olasılıksal modeli

$$r^t = y^t + \epsilon \tag{10.36}$$

olarak yazdığımızı ve $\epsilon \sim \mathcal{N}(0, \sigma^2)$ gürültü varsaydığımızı hatırlayalım. y^t çıktısına S işlevi uygulanarak $r^t \in \{0, 1\}$ gibi 0 ile 1 arasında olması sağlanabilir. Doğrusal bir model ve iki sınıf olsun:

$$y^t = S(\boldsymbol{w}^T \boldsymbol{x}^t + w_0) = \frac{1}{1 + \exp[-(\boldsymbol{w}^T \boldsymbol{x}^t + w_0)]} \tag{10.37}$$

$r|\boldsymbol{x} \sim \mathcal{N}(y, \sigma^2)$ varsayımıyla bağlanım için örnek olabilirliğini yazalım:

$$l(\boldsymbol{w}, w_0|\mathcal{X}) = \prod_t \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(r^t - y^t)^2}{2\sigma^2}\right] \tag{10.38}$$

Log olabilirliği artırmaya çalışmak toplam kare hatayı azaltmaya denktir:

$$E(\boldsymbol{w}, w_0|\mathcal{X}) = \frac{1}{2}\sum_t (r^t - y^t)^2 \tag{10.39}$$

Eğim iniş kullanarak güncelleme kurallarını türetelim:

$$\begin{aligned} \Delta \boldsymbol{w} &= \eta \sum_t (r^t - y^t) y^t (1 - y^t) \boldsymbol{x}^t \\ \Delta w_0 &= \eta \sum_t (r^t - y^t) y^t (1 - y^t) \end{aligned} \tag{10.40}$$

Bu yöntemi $K > 2$ sınıf olduğunda da kullanabiliriz. Olasılıksal model bu durumda

$$\boldsymbol{r}^t = \boldsymbol{y}^t + \boldsymbol{\epsilon} \tag{10.41}$$

olur ve $\boldsymbol{\epsilon} \sim \mathcal{N}_K(0, \sigma^2 \mathrm{I}_K)$ gürültü yöneyidir. Her sınıf için doğrusal bir model varsayalım:

$$y_i^t = \text{sigmoid}(\boldsymbol{w}_i^T \boldsymbol{x}^t + w_{i0}) = \frac{1}{1 + \exp[-(\boldsymbol{w}_i^T \boldsymbol{x}^t + w_{i0})]} \tag{10.42}$$

Örnek olabilirliği

$$l(\{\boldsymbol{w}_i, w_{i0}\}_i | \mathcal{X}) = \prod_t \frac{1}{(2\pi)^{K/2} |\Sigma|^{1/2}} \exp\left[-\frac{\|\boldsymbol{r}^t - \boldsymbol{y}^t\|^2}{2\sigma^2}\right] \tag{10.43}$$

ve hatayı yazıp

$$E(\{\boldsymbol{w}_i, w_{i0}\}_i | \mathcal{X}) = \frac{1}{2} \sum_t \|\boldsymbol{r}^t - \boldsymbol{y}^t\|^2 = \frac{1}{2} \sum_t \sum_i (r_i^t - y_i^t)^2 \tag{10.44}$$

$i = 1, \ldots, K$ için güncellemeleri türetelim:

$$\begin{aligned} \Delta \boldsymbol{w}_i &= \eta \sum_t (r_i^t - y_i^t) y_i^t (1 - y_i^t) \boldsymbol{x}^t \\ \Delta w_{i0} &= \eta \sum_t (r_i^t - y_i^t) y_i^t (1 - y_i^t) \end{aligned} \tag{10.45}$$

Bu öğrenme yönteminde y_i değerlerinden yalnızca birinin 1, ötekilerin 0 olması gerektiğini, ya da $\sum_i y_i = 1$ bilgisini kullanmadığımıza dikkat edin. 10.29. denklemdeki eşiksiz en büyük işlevi sonsal olasılıkları kestirdiği için bu kısıtları içerir, buradaysa $K > 2$ sınıfta S çıktıları kullandığımızda sanki y_i değerleri bağımsızmış gibi öğrenme yaparız.

Bağlanım kullandığımızda bu bağımsızlıktan dolayı doğru çıktı 1, ötekiler 0 olana dek öğrenme yapmamız gerekir. Bu gerekli değildir çünkü doğru sınıflandırma için doğru sınıfın çıktısının ötekilerden büyük olması yeter, ki eşiksiz en büyük işlev de zaten bunu yapar.

Dememiz o ki, birden çok S çıktılı yaklaşım ancak sınıflar bağımsız olduğunda uygundur; herhangi bir $\boldsymbol{x}^t$ için tüm r_i^t değerleri 0 olabilir ve $\boldsymbol{x}^t$ hiçbir sınıftan değildir, ya da sınıflar örtüşür ve birden çok r_i^t 1 değerini alır.

10.9 Kısa Kısa

Doğrusal ayırtaç basitliğinden dolayı örüntü tanımada en çok kullanılan sınıflandırıcı türüdür (Duda, Hart, ve Stork 2001; McLachlan 1992). Ortak eşdeğişinti dizeyli normal dağılımları 4. bölümde ve Fisher'ın doğrusal ayırtacını 6. bölümde görmüştük; bu bölümde de S biçimli sınıflandırmayı gördük. 11. bölümde doğrusal ayırtacın

yapay sinir ağı gerçeklemesi olan algılayıcıyı göreceğiz. 13. bölümde tartışacağımız destek yöney makinesi de yine doğrusal bir ayırtaçtır.

S biçimli sınıflandırma (Anderson 1982) ve (McLachlan 1992)'de ayrıntılı olarak işlenir. S (sigmoid) işlevi, logit işlevin tersidir ve Bernoulli örneklemlerinde *asal bağdır*. Eşiksiz en büyük işlevi, bunun katlıterimli dağılıma genellenmesidir. Böylesi *genellenmiş doğrusal modeller*, (McCullogh ve Nelder 1989)'da tartışılır.

Doğrusal olmayan taban işlevleriyle doğrusal modelleri genellemek eskiden beri sık kullanılan bir yaklaşımdır. Çok katmanlı algılayıcılar (bölüm 11) ve dairesel tabanlı işlevlerde (bölüm 12) taban işlevinin parametreleri de ayırtacı öğrenirken veriden öğrenilebilir. Destek yöney makinelerinin (bölüm 13) kullandığı çekirdek işlevleri yine böylesi taban işlevlerinden hesaplanır.

10.10 Alıştırmalar

1. Aşağıdaki taban işlevlerinin nerede sıfırdan farklı çıktı verdiğini tanımlayın:
   a. $\sin(x_1)$
   b. $\exp(-(x_1 - a)^2/c)$
   c. $\exp(-\|\boldsymbol{x} - \boldsymbol{a}\|^2/c)$
   d. $\log(x_2)$
   e. $1(x_1 > c)$
   f. $1(ax_1 + bx_2 > c)$
2. Şekil 10.2'deki iki boyutlu örnek için 10.4 ve 10.5 denklemlerini gösterin.
3. Eşiksiz en büyük işlevi, $y_i = \exp(a_i)/\sum_j \exp(a_j)$, için $\partial y_i/\partial a_j = y_i(\delta_{ij} - y_j)$ olduğunu gösterin: $i = j$ için $\delta_{ij} = 1$, değilse $\delta_{ij} = 0$.
4. $K = 2$ için eşiksiz en büyük işlevli iki çıktı kullanmakla S işlevli tek çıktı kullanmanın aynı olduğunu gösterin.
5. Denklem 10.34'te, $\mathbf{W}_i$ dizeyini nasıl öğrenebiliriz?
6. Denklem 10.34'teki gibi ikinci (ya da daha yüksek) dereceden ayırtaçlar kullandığımızda değişintiyi nasıl denetim altında tutabiliriz?
7. Eğim inişte tüm x_j için aynı η kullanmak ne anlama gelir?
8. Tek boyutlu sınıflandırmada (örneğin, şekil 10.7), w ve w_0 neye karşılık gelir?
9. Tek boyutlu x için, $x \in (2, 4)$ aralığının C_1 sınıfından ve $x < 2$ ya da $x > 4$ aralıklarının C_2 sınıfından olduğunu varsayalım. Bu iki sınıfı doğrusal bir sınıflandırıcıyla nasıl ayırabiliriz?

10.11 Kaynaklar

Aizerman, M. A., E. M. Braverman, L. I. Rozonoer. 1964. "Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning." *Automation and Remote Control* 25: 821-837.

Anderson, J. A. 1982. "Logistic Discrimination." *Handbook of Statistics*, Vol. 2, *Classification, Pattern Recognition and Reduction of Dimensionality*, ed. P. R. Krishnaiah L. N. Kanal, 169–191. Amsterdam: North Holland.

Bridle, J. S. 1990. "Probabilistic Interpretation of Feedforward Classification Network Outputs with Relationships to Statistical Pattern Recognition." *Neurocomputing: Algorithms, Architectures and Applications*, ed. F. Fogelman-Soulie J. Herault, 227–236. Berlin: Springer.

Duda, R. O., P. E. Hart, D. G. Stork. 2001. *Pattern Classification*, 2nd ed. New York: Wiley.

McCullagh, P., J. A. Nelder. 1989. *Generalized Linear Models*. London: Chapman and Hall.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition*. New York: Wiley.

Vapnik, V. 1995. *The Nature of Statistical Learning Theory*. New York: Springer.

Bölüm 11

Çok Katmanlı Algılayıcılar

Çok katmanlı algılayıcı, sınıflandırma ve bağlanımda kullanılabilen bir yapay sinir ağı yapısıdır ve dağılımdan bağımsız bir yaklaşımdır. Çok katmanlı algılayıcının eğitiminde kullanılan hata geri yayma algoritmasını ve farklı uygulamalarını göreceğiz.

11.1 Giriş

Bir örneği bu bölümde incelediğimiz algılayıcı olan yapay sinir ağları beyinden esinlenmiştir. Bilişsel bilim ve sinirbilim konusunda çalışan bilim insanlarının amacı beynin nasıl çalıştığını anlamaktır (Posner 1989; Thagard 2005). Bu amaç doğrultusunda beyindeki sinir ağlarının modellerini oluşturur ve benzetim çalışmaları yaparlar.

Ama mühendislikte amacımız yalnızca beyni anlamak değil işe yarar makineler yapmaktır. *Yapay sinir ağlarıyla* ilgilenmemizin nedeni de bize daha iyi bilgisayarlar yapmakta faydalı olacaklarına inanmamızdır. Beyin, görüntü ve konuşma tanıma, öğrenme gibi bazı yetenekleri açısından şu anki mühendislik ürünlerinden çok üstündür ve bu yetenekler, bilgisayarlar üzerinde gerçeklendiğinde ekonomik açıdan önemli olacak uygulamalardır. Eğer beynin bu işlevleri nasıl yerine getirdiğini anlayabilirsek bu işler için algoritmalar oluşturabilir ve yapay olarak gerçekleyebiliriz.

İnsan beyni bilgisayardan oldukça farklıdır. Bilgisayarın genelde tek bir işlemcisi olmasına rağmen beyin, *sinir hücresi* denen ve aynı anda çalışan çok sayıda (10^{11}) işlemci birimden oluşur. Her ne kadar ayrıntılı bilinmese de bu işlemci hücrelerin bilgisayarın işlemcisine göre daha basit ve yavaş olduğu, beyni farklı yapan ve ona hesap gücünü verenin yüksek bağlantısallığı olduğu düşünülmektedir. Beyindeki her sinir hücresinin ortalama olarak yaklaşık 10^4 farklı sinir hücresine bağlantısı vardır. Bilgisayarda işlemci etkin, bellek edilgenken beyinde işlem ve belleğin birlikte ağ üzerine dağıtılmış olduğu düşünülür; işlem, ağdaki hücrelerce yapılırken bellek, aralarındaki kavşaklardadır.

11.1.1 Beyni Anlamak

Marr'a göre (1982), bilgi işlem dizgesini anlamanın, *çözümleme düzeyleri* denilen üç düzeyi vardır:

1. *Hesaplama kuramı*, hesabın amacı ve işin soyut tanımı demektir.

2. *Gösterim ve algoritma*, girdi ve çıktının nasıl gösterildiğine ve girdiden çıktı oluşturulurken izlenecek algoritmanın tanımlanmasına karşılık gelir.

3. *Donanım gerçeklemesi*, dizgenin fiziksel olarak gerçeklenmesidir.

Sayıların sıralanması örneğini alalım: Hesaplama kuramı kümenin öğelerinin sıraya konulmasını söyler. Gösterim, sayılar olup algoritma, örneğin *quicksort* olabilir. Derlemeden sonra belirli bir işlemci için oluşan ve örneğin tamsayıları ikili gösterimde sıralayan program örnek bir donanım gerçeklemesidir.

Aynı hesaplama kuramı için olası farklı gösterim ve algoritmalar olabilir. Benzer biçimde, bir gösterim ve algoritma için farklı donanım gerçeklemeleri oluşturulabilir. Sıralama için geliştirilmiş birçok algoritmadan herhangi birini seçebiliriz ve herhangi bir algoritma farklı işlemciler için derlendiğinde farklı donanım gerçeklemeleri oluşur.

Bir başka örnek alalım: '6', 'VI', and '110', altı rakamının üç farklı gösterimidir. Toplama yaparken kullanılacak algoritma her gösterim için farklıdır. Sayısal bilgisayarlar ikili gösterimi kullanır ve bu gösterimde toplama yapmak için tasarlanmış devreleri vardır; bu, olası bir donanım gerçeklemesidir. Abaküste sayılar farklı gösterilir, toplama için başka işlemler yapılır; bu da başka bir donanım gerçeklemesidir. Kafamızda toplama yaptığımızda başka bir gösterim ve ona uygun, sinir hücrelerince gerçeklenen başka bir algoritma kullanırız. Ama tüm bu farklı donanım gerçeklemeleri—örneğin, biz, abaküs, bilgisayar—aynı hesaplama kuramını uygular, ki buna toplama deriz.

Aynı iş için doğanın ve mühendisliğin farklı çözümlerine en sık verilen örnek uçmadır. Serçe kanatlarını çırpar, uçağınsa pervanesi ya da jet motoru vardır. Serçe ve uçak farklı kısıtlar altında farklı amaçlar için üretilmiş farklı donanımlardır; ama ikisi de aynı hesaplama kuramını, hava devinim bilimini, kullanır.

Beyin, öğrenme ya da örüntü tanıma için olası donanım gerçeklemelerinden biridir. Eğer bu gerçeklemeden tersine mühendislik yaparak kullanılan gösterim ve algoritmayı, oradan da hesaplama kuramını çıkarabilirsek, elimizdeki kaynak ve kısıtlara uygun kendi gösterim ve algoritmamızı ve ardından kendi donanım gerçeklememizi oluşturabiliriz. Oluşturduğumuz bu yeni gerçeklemenin daha ucuz, hızlı, ve doğru olmasını umarız.

Nasıl ki hava devinim bilimini keşfedene dek yapılan uçma makineleri kuşlara benzediyse, zekanın hesaplama kuramını bulana dek beyni taklit eden denemelerin de beyin gibi işlemci ağlarından oluşması doğaldır. Yani beyni anlarken şimdilik gösterim ve algoritma düzeyinde olduğumuz söylenebilir.

Bir gün sinir hücrelerinin ve aralarındaki bağlantıların zekaya etkisinin tüylerin uçmaya etkisi gibi bir donanım ayrıntısı olduğunu anlayabiliriz. Ama beynin nasıl

çalıştığını anlamaya ilgi duymamızın bir başka nedeni daha vardır, o da beynin koşut işlem gücüdür.

11.1.2 Koşut İşlem İçin Bir Örnek Olarak Yapay Sinir Ağları

1980'lerden beri içinde binlerce işlemci olan bilgisayarlar üretilebilmektedir ama bu koşut yapılar için yazılım donanımla aynı hızla gelişmemiştir. Bunun nedeni de hesaplama kuramının o ana dek neredeyse tümüyle tek işlemci varsayımına dayanmasıdır. Elimizdeki koşut bilgisayarları etkili bir biçimde programlayamadığımız için yüksek verimle kullanamıyoruz.

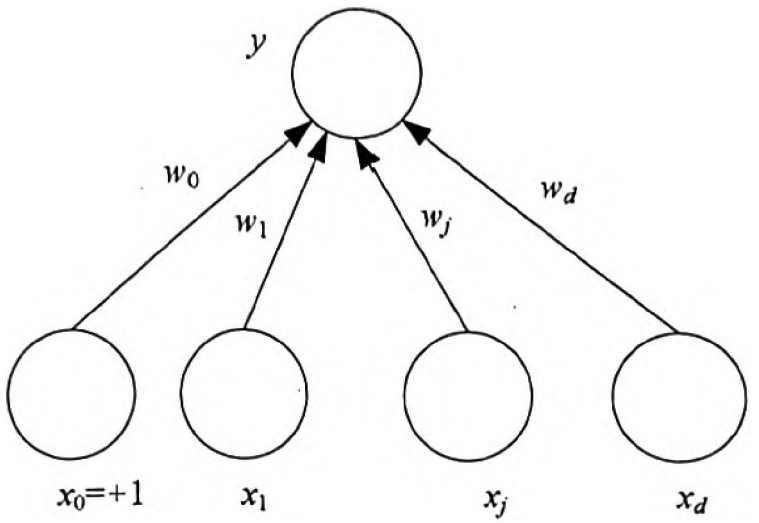
Koşut işlem için temelde iki yaklaşım vardır: Tek komutlu, çok verili hesaplamada tüm işlemciler aynı komutu, farklı veriler üzerinde gerçekler. Çok komutlu, çok verili hesaplamadaysa farklı işlemciler farklı veriler üzerinde farklı komutları gerçekleyebilir. Tek komutlu bilgisayarları programlamak daha kolaydır çünkü tek bir program yazmak yeterlidir. Ama bu biçimde koşut olarak gerçeklenebilmek için problemin çok düzgün bir yapısı olması gerekir, ki buna sık rastlanmaz. Çok komutlu bilgisayarların olası kullanım alanı daha geniştir ama her işlemci için farklı program yazmak kolay bir iş değildir; üstelik eşzamanlama, işlemciler arasında veri iletişimi, vs. gibi ek kısıtlar vardır. Tek komutlu koşut bilgisayarların üretimi de daha kolaydır ve tek komutlularsa bigisayarlar daha çok işlemci içerebilir. Çok komutlu koşut bilgisayarlar daha karmaşıktır ve işlemciler arasında veri iletişimi için daha karmaşık bir bağlantı ağı gerekir.

Şimdi işlemcileri tek komutlulardan biraz daha çok, ama çok komutlular kadar da karmaşık olmayan bir bilgisayar düşünelim. İşlemcilerin içinde bazı parametreler saklayabildiği yerel bellekleri olsun. Her işlemci, tek komutlulardaki gibi basit ve sabit bir işlevi gerçeklesin, ama yerel belleğe farklı değerler yüklenerek işlemciler farklı işler yapabilsin ve uygulama bu işlemciler üzerine dağıtılabilsin. Bu yapıya "sinir komutlu çok verili" bilgisayar diyebiliriz; her işlemci bir sinir hücresine, yerel parametreler bağlantı ağırlıklarına ve tüm yapı bir sinir ağına karşılık gelir. Eğer işlemcilerin gerçeklediği işlev basit ve yerel bellek küçükse, tek bir yonga üstüne böyle çok sayıda işlemci sığdırılabilir.

Bu durumda yapmamız gereken problemin işlemci ağına dağıtılması ve yerel parametrelerin belirlenmesidir. İşte öğrenme de burada devreye girer: Eğer böylesi bilgisayarlar örneklerden öğrenebilirse bunları programlamayı ve yerel parametreleri belirlemeyi bizim yapmamız gerekmez.

Yani, yapay sinir ağları şu anki mühendislik bilgimizle üretebildiğimiz koşut bilgisayarları kullanmak için bir yoldur ve öğrenme sayesinde programlama gerektirmezler.

Bu bölümde böylesi yapıları ve nasıl eğitildiklerini tartışacağız. Yapay sinir ağı çoğunlukla tek işlemcili bir bilgisayar tarafından gerçeklenen matematiksel bir işlevi gerçekler ve bu yapıları eğitmek önceki bölümlerde gördüğümüz istatistiksel yöntemlerden çok farklı değildir. Bu işlemin bir işlemci ağınca gerçeklendiğini düşünmek ancak elimizde koşut bir bilgisayar varsa ve tek işlemciyle gerçeklenemeyecek kadar büyük bir ağ söz konusuysa anlamlıdır.



Şekil 11.1: Algılayıcı. $x_j, j = 1, \ldots, d$, girdi birimlerini gösterir. x_0, her zaman 1 değerini alan ek girdidir. y, çıktı birimini ve w_j, x_j girdisinden çıktıya olan bağlantının ağırlığını gösterir.

11.2 Algılayıcı

Algılayıcı, temel yapay sinir ağı birimidir. Girdisi ya doğrudan dış dünyadan gelir, ya da öteki algılayıcıların çıktılarını girdi olarak alır. Her $x_j \in \Re, j = 1, \ldots, d$ girdisi için bir $w_j \in \Re$ *bağlantı ağırlığı* tanımlıdır. y çıktısı en basit durumda girdilerin ağırlıklı toplamıdır (bkz. şekil 11.1):

$$y = \sum_{j=1}^{d} w_j x_j + w_0 \tag{11.1}$$

w_0 değeri modeli daha genel yapmak için eklenmiştir ve x_0 ile gösterdiğimiz ve hep +1 değerini alan ek girdi biriminin ağırlığı olarak gösterilir. Girdileri ve ağırlıkları yöneyler olarak yazarsak çıktı bir iç çarpım olarak da tanımlanabilir:

$$y = \boldsymbol{w}^T \boldsymbol{x} \tag{11.2}$$

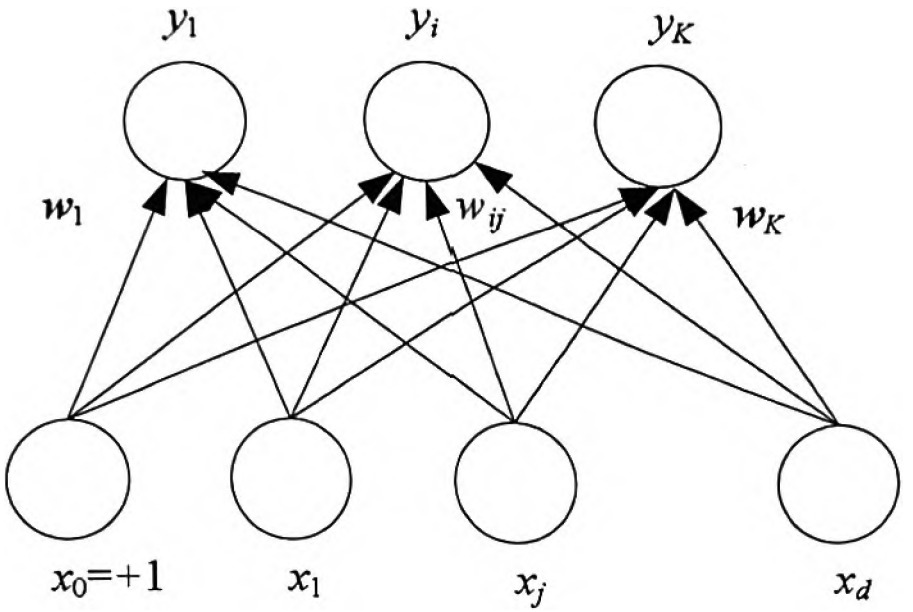
öyle ki, $\boldsymbol{w} = [w_0, w_1, \ldots, w_d]^T$ ve $\boldsymbol{x} = [1, x_1, \ldots, x_d]^T$, ek girdiyi ve ağırlığını içeren genişletilmiş yöneylerdir.

Deneme sırasında, $\boldsymbol{w}$ ağırlıkları elimizdeyken $\boldsymbol{x}$ girdisi için y çıktı değerini hesaplarız. Bu modelin parametreleri $\boldsymbol{w}$ ağırlıklarıdır ve *öğrenme*, onların her girdi için doğru çıktı vermek üzere belirlenmesidir.

$d = 1$ olduğunda ve çevreden gelen tek x girdisi olduğunda

$$y = wx + w_0$$

bir doğru denklemidir ve w, eğimi, w_0 da y eksenini kesim noktasıdır. Yeni, tek girdili ve tek çıktılı algılayıcı bir doğru gerçekler. Birden çok girdi varsa doğru,



Şekil 11.2: K koşut algılayıcı. $x_j, j = 0, \ldots, d$, girdileri ve $y_i, i = 1, \ldots, K$, çıktıları gösterir. x_j girdisinden y_i çıktısına olan bağlantının ağırlığı w_{ij} ile gösterilir. Her çıktı, girdilerin ağırlıklı toplamıdır. $K > 2$ sınıflı sınıflandırmada bir son işlem olarak çıktıların en büyüğü ya da sonsal olasılıklar gerekiyorsa eşiksiz en büyüğü hesaplanır.

(üstün)düzlem olur ve çok girdili algılayıcı doğrusal model gerçekler. Verilmiş bir örneklem için w_j parametreleri doğrusal bağlanımla hesaplanabilir (bkz. bölüm 5.8).

Denklem 11.1 tarafından tanımlanan algılayıcı bir üstündüzlem tanımlar ve uzayı, çıktının artı ya da eksi olduğu iki parçaya ayırmak için kullanılabilir (bkz. bölüm 10). Doğrusal bir ayırtacı gerçeklemek için kullanıldığında algılayıcı çıktısının işaretine bakarak iki sınıf arasında seçim yapabiliriz. $e(\cdot)$ ile bir *eşik işlevi* tanımlayalım:

$$e(a) = \begin{cases} 1 & \text{eğer } a > 0 \\ 0 & \text{değilse} \end{cases} \tag{11.3}$$

ve

$$\text{seç} \begin{cases} C_1 & \text{eğer } e(\boldsymbol{w}^T \boldsymbol{x}) > 0 \\ C_2 & \text{değilse} \end{cases}$$

Doğrusal ayırtacın sınıfların doğrusal ayrıldığını varsaydığını hatırlayalım. Yani, $\boldsymbol{x}^t \in C_1$ ve $\boldsymbol{x}^t \in C_2$ kümelerini ayıran bir $\boldsymbol{w}^T \boldsymbol{x} = 0$ üstündüzlem olması gerekir. Eğer bize sonrasında bir sonsal olasılık gerekiyorsa—örneğin, risk hesaplamak için—çıktda S işlevini kullanabiliriz:

$$\begin{aligned} o &= \boldsymbol{w}^T \boldsymbol{x} \\ y &= S(o) = \frac{1}{1 + \exp[-\boldsymbol{w}^T \boldsymbol{x}]} \end{aligned} \tag{11.4}$$

$K > 2$ çıktı için her birinin kendi $\boldsymbol{w}_i$ ağırlık yöneyi olan K algılayıcı vardır (bkz.

şekil11.2)

$$\begin{aligned} y_i &= \sum_{j=1}^{d} w_{ij} x_j + w_{i0} = \mathbf{w}_i^T \mathbf{x} \\ \mathbf{y} &= \mathbf{W}\mathbf{x} \end{aligned} \tag{11.5}$$

ve burada w_{ij}, x_j girdisinden y_i çıktısına olan bağlantının ağırlığını gösterir. $\mathbf{W}$, w_{ij} ağırlıklarını içeren $K \times (d + 1)$ ağırlık dizeyidir ve satırları K algılayıcının bağlantı yöneyleridir. Sınıflandırma için kullanıldığında denemede en yüksek değerli sınıf seçilir:

$$\text{seç } C_i \text{ eğer } y_i = \max_k y_k$$

Bir yapay sinir ağı yapısı olarak ele aldığımızda her algılayıcı, girdileri ve bağlantı ağırlıklarının *yerel* bir işlevidir. Sınıflandırmada eğer bize yalnızca kazanan sınıf değil sonsal olasılıklar da gerekiyorsa eşiksiz en büyük değeri hesaplayabiliriz, ama bunun için bize öteki sınıfların çıktı değerleri de gerekir. Bunu bir yapay sinir ağı olarak gerçeklerken, birincisi ağırlıklı toplamları, ikincisi eşiksiz en büyük işlevi hesaplayan iki aşama olarak görsek de yine de tek bir çıktı katmanı olarak gösteririz:

$$\begin{aligned} o_i &= \mathbf{w}_i^T \mathbf{x} \\ y_i &= \frac{\exp o_i}{\sum_k \exp o_k} \end{aligned} \tag{11.6}$$

Ek girdiler, örneğin, $x_3 = x_1^2, x_4 = x_2^2, x_5 = x_1 x_2$, tanımlayarak doğrusal modelin çokterimli yaklaşıklama için de kullanılabileceğini daha önce görmüştük (bölüm 10.2). Aynısı algılayıcılarla da kullanılabilir (Durbin ve Rumelhart 1989). 11.5. bölümde, girdinin böylesi doğrusal olmayan işlevlerinin sabit olarak önceden tanımlanmak yerine veriden öğrenildiği "saklı" katmanlı çok katmanlı algılayıcıları göreceğiz.

10. bölümde doğrusal sınıflandırma için gördüğümüz herhangi bir yöntemle $\mathbf{w}_i, i = 1, \ldots, K$ yöneyleri hesaplanıp ağa ağırlıklar olarak yerleştirilebilir. Örneğin, ortak eşdeğişinti dizeyli normal dağılıma bağlı yaklaşım, S biçimli sınıflandırma, ya da bağlanımlı sınıflandırma kullanılabilir. Bazı uygulamalarda öğrenme başladığında elimizde tüm öğrenme örnekleri olmayabilir ve örnekler geldikçe parametreleri güncellememiz gerekebilir; böylesi *çevrimiçi* öğrenme yöntemlerini 11.3. bölümde göreceğiz.

Denklem 11.5, d boyutlu uzaydan K boyutlu bir uzaya doğrusal bir dönüşüm tanımlar ve $K < d$ ise bu bir boyut azaltmadır. 6. bölümde gördüğümüz yöntemlerin biriyle $\mathbf{W}$ hesaplanarak algılayıcılar doğrusal dönüşümü, örneğin, temel bileşen çözümlemesini gerçeklemek için de kullanılabilir. Bu durumda iki katmanlı bir ağ yazabiliriz, öyle ki, birinci algılayıcı katman bu doğrusal boyut azaltma dönüşümünü, ikincisi de bu yeni düşük boyutlu uzayda sınıflandırma ya da bağlanımı gerçekler. Gerçekte tabii ki her iki katman da doğrusal olduğu için bu tek bir doğrusal katman olarak da gerçeklenebilir; birinci katmanın *doğrusal olmayan* bir biçimde boyut azalttığı daha ilginç durumu bölüm 11.5'te göreceğiz.

11.3 Algılayıcının Eğitilmesi

Doğrusal sınıflandırıcı bir üstündüzlem tanımlar ve bir yapay sinir ağı yapısı olan algılayıcı bu üstündüzlemi *gerçeklemenin* bir yoludur. Bir örneklem için ağırlıklar *çevrimdışı* hesaplanıp algılayıcıya yerleştirilerek algılayıcı, çıktı değerlerini hesaplamak için kullanılabilir.

Yapay sinir ağlarını eğitirken sıklıkla çevrimiçi öğrenme kullanırız. Elimizde tüm öğrenme kümesi yoktur, örnekler birer birer gelir, ve her örnekte, ağın parametrelerini biraz güncellemek ve zaman içinde veriye uyarlamak isteriz. Böylesi bir yaklaşım birkaç nedenden dolayı ilginçtir:

1. Öğrenme kümesini, ya da en iyi parametre değerlerini ararken oluşan ara sonuçları, saklamak için ek bir bellek gerekmez. Destek yöney makineleri (bölüm 13) gibi bir yaklaşım büyük örneklemlerde çok maliyetli olabilir ve bazı uygulamalarda, tüm veriyi saklamadığımız ve karmaşık bir iyileme gerektirmeyen bir yaklaşım yeğlenebilir.

2. Öğrenmek istediğimiz iş zaman içinde değişebilir ve bu yüzden önceden bir örneklem oluşturmak anlamlı olmayabilir. Örneğin, konuşmacısına kendini uyarlayan bir konuşma tanıma programı gerçeklemek isteyebiliriz.

3. Fiziksel ortam zamanla değişebilir. Örneğin robotikte mekanik bileşenler aşınır ya da alıcıların hassassiyeti azalır.

Çevrimiçi öğrenmede hata işlevini tüm örneklem üzerinde değil tekil örnekler üzerinde tanımlarız. Rastgele seçilmiş ilk ağırlıklarla başlar ve her örnekte parametreleri azıcık, o örnekteki hatayı azaltacak ama daha önceden öğrendiklerimizi de unutmayacak kadar değiştiririz. Eğer hata işlevinin türevi alınabiliyorsa eğim iniş kullanabiliriz.

Örneğin bağlanımda t dizinli tek örnek çifti, (x^t, r^t), üzerindeki hatayı

$$E^t(w|x^t, r^t) = \frac{1}{2}(r^t - y^t)^2 = \frac{1}{2}[r^t - (w^T x^t)]^2$$

çevrimiçi eğim inen güncellemeyle azaltabiliriz:

$$\Delta w_j^t = \eta(r^t - y^t)x_j^t, j = 0, \ldots, d \tag{11.7}$$

Burada, η ile gösterdiğimiz yakınsama için zaman içinde azaltılan öğrenme çarpanıdır. Bu yönteme *rastgele eğim inişi* denir.

Benzer biçimde, S biçimli sınıflandırmada her örnekte güncelleme yapan kurallar da türetilebilir. İki sınıf olduğunda (x^t, r^t) örneği ($x^t \in C_1$ ise $r_i^t = 1$ ve $x^t \in C_2$ ise $r_i^t = 0$) üzerinde tek çıktı S işleviyle hesaplanır:

$$y^t = \text{sigmoid}(w^T x^t)$$

Tek örnek için çapraz düzensizliği yazıp

$$E^t(w|x^t, r^t) = -r^t \log y^t - (1 - r^t) \log(1 - y^t)$$

```
Tüm i = 1, ..., K
    Tüm j = 0, ..., d
        w_ij ← rand(−0.01, 0.01)
Yinele
    Tüm (x^t, r^t) ∈ 𝒳 rastgele sırada
        Tüm i = 1, ..., K
            o_i ← 0
            Tüm j = 0, ..., d
                o_i ← o_i + w_ij x_j^t
        Tüm i = 1, ..., K
            y_i ← exp(o_i) / Σ_k exp(o_k)
        Tüm i = 1, ..., K
            Tüm j = 0, ..., d
                w_ij ← w_ij + η(r_i^t − y_i) x_j^t
Yakınsamaya dek
```

Şekil 11.3: $K > 2$ sınıf için rastgele eğim inişi gerçekleyen algılayıcı öğrenme algoritması. Bu, şekil 10.8'deki algoritmanın çevrimiçi sürümüdür.

eğim inen güncelleme kuralını türetelim:

$$\Delta w_j^t = \eta(r^t - y^t)x_j^t, j = 0, \ldots, d \tag{11.8}$$

$K > 2$ sınıf olduğunda, (x^t, r^t) örneği ($x^t \in C_i$ ise $r_i^t = 1$ değilse $r_i^t = 0$) üzerinde çıktıları eşiksiz en büyük işlevle hesaplarız:

$$y_i^t = \frac{\exp \boldsymbol{w}_i^T \boldsymbol{x}^t}{\sum_k \exp \boldsymbol{w}_k^T \boldsymbol{x}^t}$$

$K > 2$ sınıf için çapraz düzensizliği

$$E^t(\{w_i\}_i | x^t, r^t) = -\sum_i r_i^t \log y_i^t$$

eğim inişle azaltan güncelleme kuralları

$$\Delta w_{ij}^t = \eta(r_i^t - y_i^t)x_j^t, i = 1, \ldots, K,\ j = 0, \ldots, d \tag{11.9}$$

10.7. bölümdeki denklemlere benzer; tek fark, tüm örnekler üzerinden toplamak yerine her örnekten sonra güncelleme yapmamızdır. Örnek program şekil 11.3'te verilmiştir ve şekil 10.8'deki programın çevrimiçi sürümüdür.

11.7 ve 11.9. denklemlerin ikisi de

$$\text{Güncelleme} = \text{ÖğrenmeÇarpanı} \cdot (\text{İstenenÇıktı} - \text{GerçekÇıktı}) \cdot \text{Girdi} \tag{11.10}$$

biçimindedir. Gelin şimdi, böyle bir güncellemenin ne yaptığını anlamaya çalışalım. İlk olarak, eğer gerçek çıktı zaten istenen çıktıya eşitse güncelleme yapılmadığını görüyoruz. Aradaki fark arttıkça güncellemenin büyüklüğü de artar. Eğer gerçek çıktı istenen çıktıdan küçükse, girdi artı olduğunda güncelleme artı, eksi olduğunda eksidir. Böylece gerçek çıktının değeri artırılır ve istenen çıktıyla farkı azaltılır. Eğer gerçek çıktı istenen çıktıdan büyükse, girdi artı olduğunda güncelleme eksi, eksi olduğunda artıdır; böylece gerçek çıktının değeri azalır ve yine istenen çıktıya yaklaşır.

Güncelleme yapıldığında büyüklüğü girdiye da bağlıdır. Girdi sıfıra yakınsa gerçek çıktıya etkisi küçük olduğundan sorumluluğu da azdır ve ağırlığındaki güncelleme de küçüktür. Girdi büyüdükçe ağırlığı daha büyük oranda güncellenir.

Güncellemenin büyüklüğü son olarak η ile gösterdiğimiz öğrenme çarpanına bağlıdır. Eğer bu değer çok büyük olursa son örneklerin etkisi çok fazla olur ve kısa soluklu bir öğrenme gerçeklenir. Bu çarpan küçükse yakınsama için daha çok sayıda güncelleme gerekir. 11.8.1. bölümde yakınsamayı hızlandırmak için kullanabileceğimiz bazı yöntemlerden bahsedeceğiz.

11.4 Mantık İşlevlerinin Öğrenilmesi

Bir mantık işlevinde girdiler ve çıktı ikilidir; doğru ve yanlış sırasıyla 1 ve 0 değerleriyle gösterilir. Yani iki sınıflı bir uygulamadır. Örnek olarak iki girdili VE kapısı için örneklem tablo 11.1'de verilmiştir. VE kapısını gerçekleyen örnek bir algılayıcı ve iki boyutta geometrik yorumu şekil 11.4'te gösterilmiştir. Ayırtaç

$$y = e(x_1 + x_2 - 1.5)$$

olarak yazılır; $\boldsymbol{x} = [1, x_1, x_2]^T$ ve $\boldsymbol{w} = [-1.5, 1, 1]^T$, genişletilmiş girdi ve ağırlık yöneyleridir. $y = e(x_1 + x_2 - 1.5)$ denkleminin tablo 11.1'deki dört satırı sağladığını kolayca görebiliriz; örneğin, $x_1 = 1, x_2 = 0$ için $y = e(-0.5) = 0$ olur. Benzer biçimde, $y = e(x_1 + x_2 - 0.5)$ denklemi VEYA kapısını gerçekler.

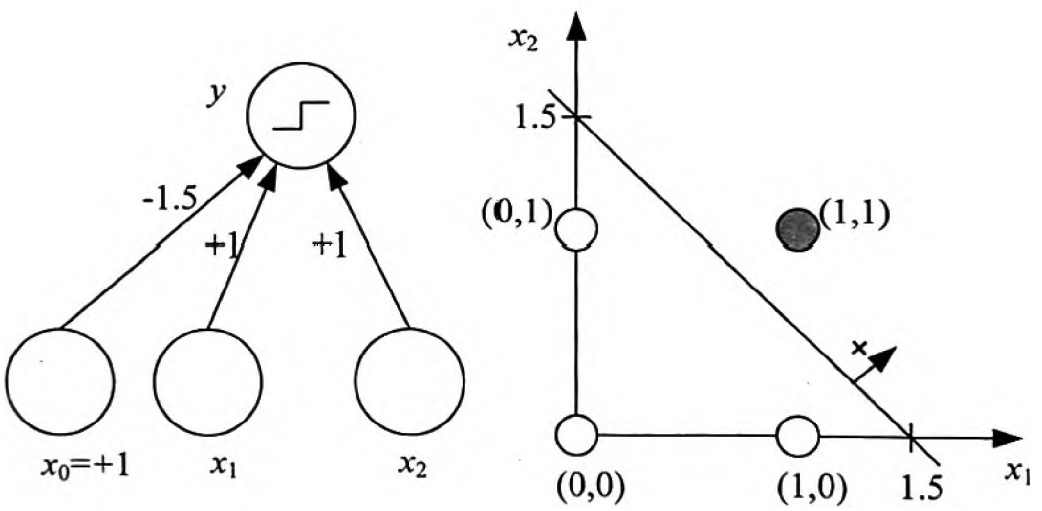
Her ne kadar VE ve VEYA gibi mantık işlevleri doğrusal ayrılabilir ve algılayıcıyla gerçeklenebilirlerse de, dışlamalı VEYA gibi bazıları gerçeklenemez. Dışlamalı VEYA için girdi ve istenen çıktı değerleri tablo 11.2'de verilmiştir. Şekil 11.5'te gördüğümüz gibi bu problemde iki sınıf doğrusal ayrılabilir değildir. Aşağıdaki dört koşulu sağlayan w_0, w_1, ve w_2 değeri yoktur:

$$\begin{array}{rrrl} & & w_0 & \leq 0 \\ & w_2+ & w_0 & > 0 \\ w_1+ & & w_0 & > 0 \\ w_1+ & w_2+ & w_0 & \leq 0 \end{array}$$

Bu sonuç bizi hiç şaşırtmamalı, çünkü daha önce iki boyutta doğrunun VC boyutunun üç olduğunu görmüştük. İki ikili girdi için olası dört durum olduğundan iki girdili bazı problemlerin doğru tarafından gerçeklenemediğini bu sonuçtan anlaşılır; dışlayıcı VEYA bunlardan biridir.

Tablo 11.1: VE işlevi için girdi ve çıktılar.

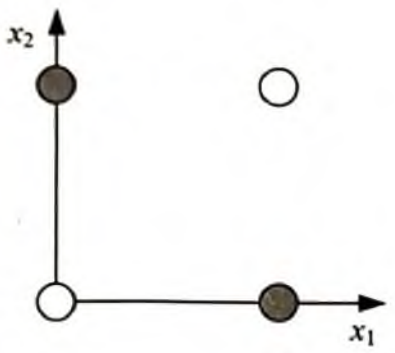
x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1



Şekil 11.4: VE kapısını gerçekleyen algılayıcı ve geometrik yorumu.

Tablo 11.2: Dışlayıcı VEYA için girdi ve çıktılar.

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0



Şekil 11.5: Dışlayıcı VEYA doğrusal ayrılabilir bir sınıflandırma problemi değildir. Bu uzayda boş dairelerin bir yanında, dolu dairelerin öbür yanında kaldığı bir doğru çizemeyiz.

11.5 Çok Katmanlı Algılayıcılar

Tek ağırlık katmanı olan algılayıcı girdinin yalnızca doğrusal işlevlerini öğrenebilir ve dışlayıcı VEYA gibi ayırtacın doğrusal olmadığı durumu gerçekleyemez. Benzer biçimde algılayıcıyı doğrusal olmayan bağlanımda kullanamayız. Bu sınırlama, arada *saklı katmanı* olan *ileriye doğru yayılımlı* ağlar için geçerli değildir. Böylesi *çok katmanlı algılayıcılar* sınıflandırmada doğrusal olmayan ayırtaçları öğrenmede ya da doğrusal olmayan bağlanım gerçeklemede kullanılabilir.

$\boldsymbol{x}$ girdisi $x_0 = +1$ ek girdisiyle genişletilmiş olarak girdi katmanına verilir ve "etkilenim" ileriye doğru yayılıp z_h ile gösterilen saklı birim değerleri hesaplanır (bkz. şekil11.6). Her saklı birim kendi başına bir algılayıcıdır ve ağırlıklı toplamına S işlevini uygular:

$$z_h = \text{sigmoid}(\boldsymbol{w}_h^T \boldsymbol{x}) = \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}, \quad h = 1, \ldots, H \tag{11.11}$$

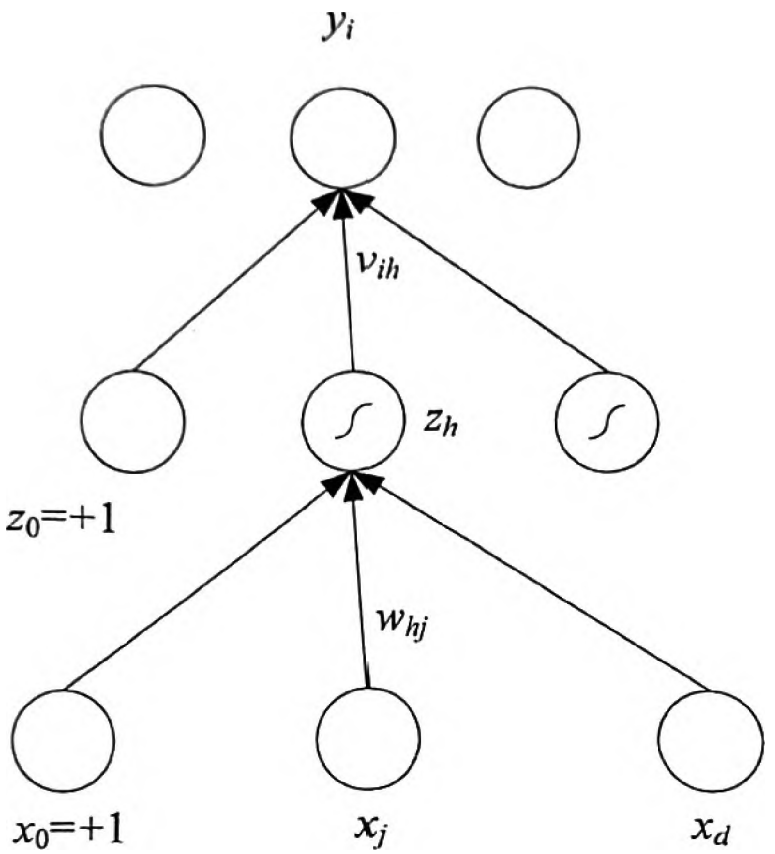
y_i çıktıları, saklı birimleri girdi alan ikinci algılayıcı katmanını oluşturur:

$$y_i = \boldsymbol{v}_i^T \boldsymbol{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0} \tag{11.12}$$

Saklı katmanda da z_0 ile gösterdiğimiz ve hep $+1$ değeri olan ek bir birim vardır; v_{i0}, ondan çıktılara giden ağırlıkları gösterir. Hesaplama yapılmadığı için x_j girdi katmanını saymaz ve bir saklı katmanı olduğundan bu yapıya iki katmanlı ağ deriz.

Bağlanımda çıktı katmanındaki y değerini hesaplarken doğrusal olmayan bir işlev kullanılmaz. İki sınıflı bir uygulamada S işlevli tek bir çıktı, $K > 2$ sınıf olduğundaysa eşiksiz en büyük işlevinin kullanıldığı K çıktı birimi vardır.

Saklı birimlerde S işlevi olmasaydı ve doğrusal olsalardı saklı katman hiçbir işe yaramazdı, çünkü doğrusal birleşimin doğrusal birleşimi yine bir doğrusal birleşimdir. S işlevi, eşik işlevine benzer ama süreklidir ve türevi alınabilir. Eğim bilgisini kullanan öğrenme algoritmaları kullandığımızdan türev alabilmek bizim için gereklidir. S

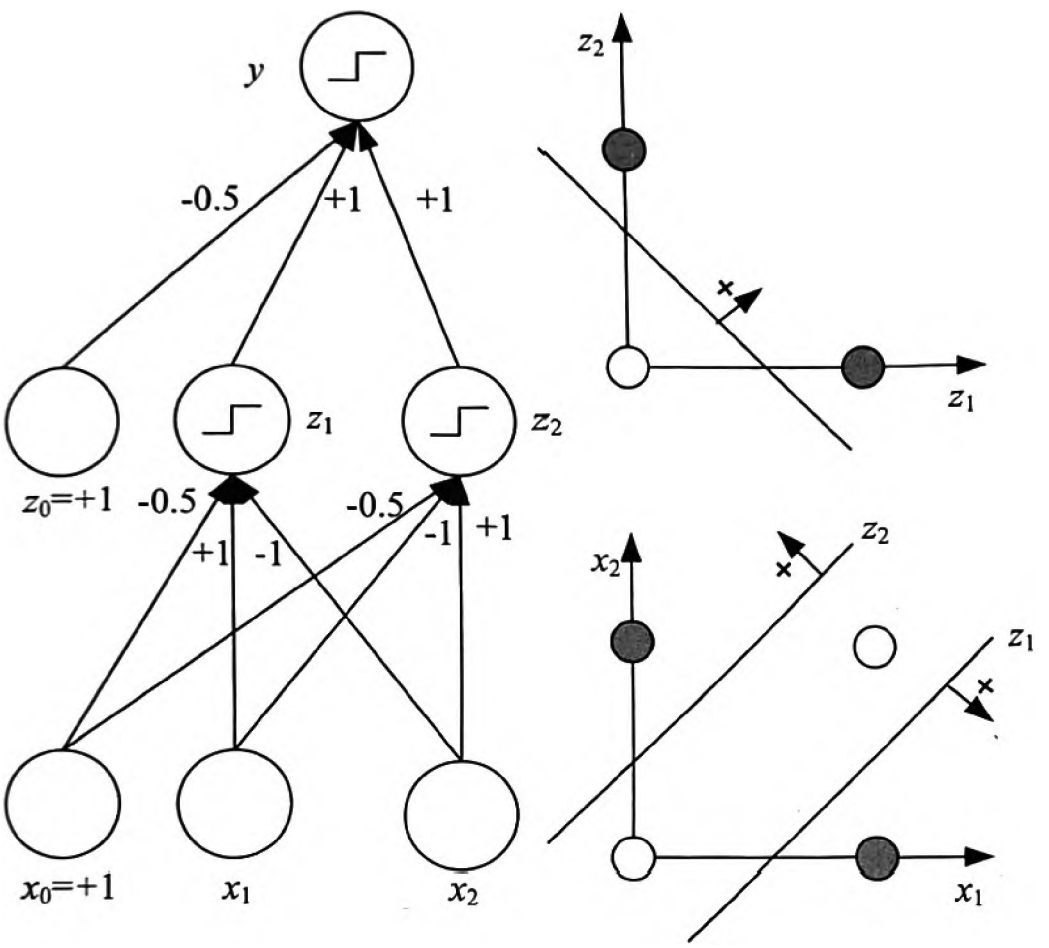


Şekil 11.6: Çok katmanlı algılayıcının yapısı. $x_j, j = 0, \ldots, d$, girdileri ve $z_h, h = 1, \ldots, H$, saklı birimleri gösterir; H saklı uzayın boyut sayısıdır. z_0, saklı katmandaki ek birimdir. $y_i, i = 1, \ldots, K$, çıktı birimleridir. w_{hj} ile birinci, v_{ih} ile ikinci katmandaki ağırlıkları gösteririz.

işlevi yerine bazen tanh işlevi de kullanılır; tek farkı, 0 ile 1 arasında değil, -1 ile $+1$ arasında değer döndürmesidir; uygulamada aralarında fark yoktur. Bir başka olasılık da iç çarpım yerine Euclid uzaklığını kullanan Gauss işlevidir; böylesi dairesel taban işlevli ağları, 12. bölümde göreceğiz.

Çıktı, saklı birimlerce hesaplanan doğrusal olmayan taban işlevlerinin doğrusal bir birleşimidir. Saklı birimlerin d boyutlu girdi uzayından saklı birimlerin tanımladığı H boyutlu uzaya doğrusal olmayan bir dönüşüm yaptığı ve bu yeni uzayda ikinci katmanın sınıflandırma ya da bağlanım için doğrusal bir işlev gerçeklediği söylenebilir.

Birden çok, ardı ardına yerleştirilmiş S işlevli ve hepsinin kendi bağlantı ağırlıkları olan saklı katmanlar girdinin daha karmaşık işlevlerini hesaplayabilir. Uygulamada bir saklı katmandan daha fazlası çok seyrek kullanılır çünkü çok saklı katmanlı ağların çözümlenmesi zordur. Bazen tek saklı katmanda çok birim kullanmak yerine az birimli ama çok saklı katmanlı, "kısa ve şişman" yerine "uzun ve ince" ağlar yeğlenebilir.



Şekil 11.7: Dışlayıcı VEYA için çok katmanlı algılayıcı. Saklı birimler ve çıktı, 0 eşik değerli bir eşik işlevi kullanır.

11.6 Evrensel bir Yaklaşıklayıcı olarak Çok Katmanlı Algılayıcı

Herhangi bir mantıksal işlevi, VEYA ile birleştirilen VE kapıları kullanarak yazabiliriz. Böyle bir mantık ifadesi tek saklı katmanlı bir algılayıcıyla gerçeklenebilir. Her VE kapısını bir saklı birim, VEYA kapısı da çıktı birimi gerçekler. Örneğin,

$$x_1 \text{ dışlayıcı VEYA } x_2 = (x_1 \text{ VE } \sim x_2) \text{ VEYA } (\sim x_1 \text{ VE } x_2)$$

Daha önce, VE ve VEYA kapılarının algılayıcılarla nasıl gerçeklendiğini gördük. İki algılayıcı yan yana iki VE kapısını gerçeklerken onların çıktısına bağlı bir algılayıcı ikinci katmanda onların çıktılarına VEYA işlemi uygular. (bkz. şekil 11.7). Birinci katman, (x_1, x_2) girdi uzayından (z_1, z_2) uzayına eşleme yapar. Hem (0,0), hem de (1,1) girdileri saklı katmanda (0,0) noktasına eşlendiği için bu yeni uzay doğrusal ayrılabilirdir.

Benzer bir yaklaşım kullanarak çıktının 1 olduğu her girdi birleşimi için o koşulu VE kapısıyla denetleyen bir saklı birim tanımlayabiliriz. Ardından gelen çıktı kat-

manı bunları VEYA ile birleştirir. Bu biçimde her mantıksal işleve karşılık gelen bir çok katmanlı algılayıcı yazabiliriz, ama böyle yazılan algılayıcıların çok fazla (d girdi için en kötü durumda 2^d tane) saklı birim kullanması gerekebilir ki böylesi bir ağ genelleme yapmayacağı için istenmez.

Girdi ve çıktı sürekli olduğunda da tek bir saklı katman yeterlidir; çok katmanlı algılayıcının *evrensel yaklaşıklayıcı* olduğunu göstermek iki saklı katman için kolaydır. Her girdi örneğine karşılık gelen alanın sınırları birinci saklı katmandaki saklı birimlerce çevresinde tanımlanabilir; ikinci saklı katmandaki saklı birim bunları VE ile birleştirerek alanı tanımlar. Ardından bu saklı birimden çıktıya giden ağırlık o alan için istenen çıktıyı belirler. Böylece işlevin, *parçalı sabit yaklaşımlaması* gerçeklenir; bu yaklaşıklama, Taylor açılımında sabit dışındaki tüm terimleri göz ardı eder. Saklı birim sayısı artırılıp girdi uzayında daha sık bölmelerle yaklaşıklamanın hassasiyeti istendiği kadar artırılabilir. Burada saklı birim sayısıyla ilgili bir sınır verilmemiştir ve bu bize yalnızca bir çözüm olduğunu gösterir. *Tek* saklı katmanlı (ama istendiği kadar çok saklı birim kullanabilen) algılayıcının girdinin herhangi bir doğrusal olmayan işlevini yaklaşıklayabildiği de gösterilmiştir (Hornik, Stinchcombe, ve White 1989).

11.7 Hata Geri Yayma Algoritması

Çok katmanlı algılayıcıyı eğitmek algılayıcı eğitmek gibidir; tek fark, saklı birimlerdeki doğrusal olmayan taban işlevleri yüzünden çıktının girdinin doğrusal olmayan bir işlevi olmasıdır. Saklı birimlerini girdi olarak düşünürsek ikinci katman bir algılayıcıdır ve z_h girdileri verildiğinde v_{ij} ağırlıklarını nasıl güncelleyeceğimizi biliyoruz. Birinci katmandaki w_{hj} ağırlıkları için eğimi hesaplarken zincir kuralını kullanırız:

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}$$

Sanki hata y çıktısından girdiye, geriye doğru yayıldığı için bu algoritmaya *hata geri yayma* adı verilmiştir (Rumelhart, Hinton, ve Williams 1986a).

11.7.1 Doğrusal Olmayan Bağlanım

Önce (tek çıktılı) doğrusal olmayan bağlanımı görelim:

$$y^t = \sum_{h=1}^{H} v_h z_h^t + v_0 \tag{11.13}$$

z_h değerleri denklem 11.11 kullanılarak hesaplanmıştır. Bağlanımda tüm örneklem üzerindeki toplam hatayı hatırlayalım:

$$E(\mathbf{W}, v|\mathcal{X}) = \frac{1}{2} \sum_t (r^t - y^t)^2 \tag{11.14}$$

Saklı birimler girdi olarak alındığında ikinci katman bir algılayıcıdır ve en küçük kareler yöntemiyle ikinci katman ağırlıklarını güncelleyebiliriz:

$$\Delta v_h = \eta \sum_t (r^t - y^t) z_h^t \tag{11.15}$$

Birinci katman çıktıları saklı birimler olan algılayıcılardan oluşur, ama saklı birimler için istenen çıktı değerleri bilinmediğinden doğrudan en küçük kareler yöntemini kullanamayız. Zincir kuralı burada devreye girer:

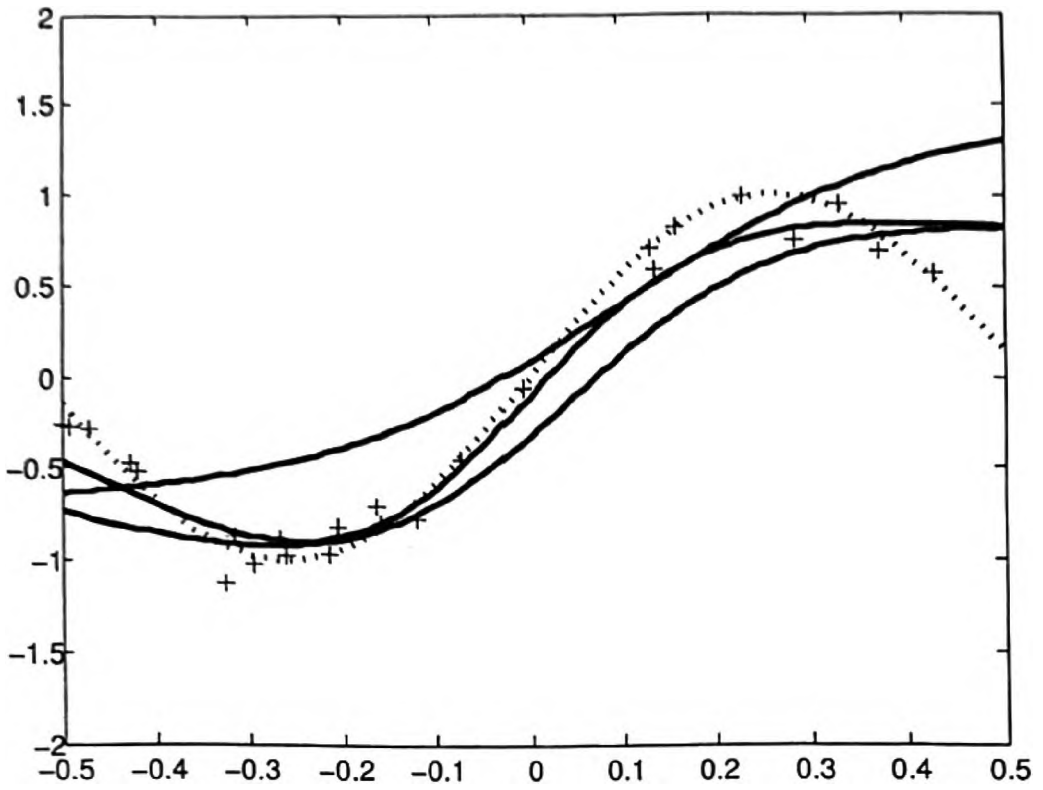
$$\begin{aligned}
\Delta w_{hj} &= -\eta \frac{\partial E}{\partial w_{hj}} \\
&= -\eta \sum_t \frac{\partial E^t}{\partial y^t} \frac{\partial y^t}{\partial z_h^t} \frac{\partial z_h^t}{\partial w_{hj}} \\
&= -\eta \sum_t \underbrace{-(r^t - y^t)}_{\partial E^t/\partial y^t} \underbrace{v_h}_{\partial y^t/\partial z_h^t} \underbrace{z_h^t (1 - z_h^t) x_j^t}_{\partial z_h^t/\partial w_{hj}} \\
&= \eta \sum_t (r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t
\end{aligned} \tag{11.16}$$

İlk iki terimin çarpımı olan $(r^t - y^t)v_h$ değeri h saklı biriminin hata terimiymiş gibi bir rol oynar. Zincir kuralıyla hata saklı birime doğru *geriye* taşınmıştır. $(r^t - y^t)$ terimi çıktıdaki hatadır ve saklı birimin "sorumluluğunu" gösteren v_h ağırlığıyla çarpılır. Üçüncü terim olan $z_h(1 - z_h)$ terimi S işlevinin türevi, x_j^t ise ağırlıklı toplamın w_{hj} ağırlığına göre türevidir. Burada bir noktaya dikkat etmek gerekiyor: Birinci katman ağırlığındaki güncelleme değeri olan Δw_{hj}, ikinci katman ağırlığı olan v_h değerini kullanıyor. Bu yüzden önce her iki katmandaki güncellemeleri (Δ değerlerini) hesaplamalı ve ardından birinci katman ağırlığına Δ eklerken ikinci katman ağırlığının *o anki* değerini kullanmalı ve ardından ikinci katman ağırlıklarını (Δ değerleriyle) güncellemeliyiz.

w_{hj} ve v_h ağırlıklarını rastgele küçük sayılardan, örneğin, $[-0.01, 0.01]$ aralığından başlatırız ki S işlevinin uçlarında kalmayalım ve öğrenme olabilsin. Tüm ağırlıklarda aynı η çarpanını kullanabilmek için girdiler eğer çok farklı ölçeklerdeyse önce sıfır ortalamalı ve birim standart sapmalı olacak biçimde normalleştirilirler.

Buradaki öğrenme denklemlerinde, her örnekte, her parametre için güncellemenin yönü ve büyüklüğü hesaplanır. *Toptan öğrenmede*, bu güncellemeleri tüm örnekler için toplar ve güncellemeyi sonda bir kez yaparız; yukarıdaki öğrenme denklemleri de bu biçimdedir.

Çevrimiçi öğrenmeyle ağırlıklar her örnekten sonra da güncellenebilir, ki buna rastgele eğim inişi diyoruz. Öğrenme kümesi üzerinden bir tam geçiş yapıp bütün örnekler için birer kez güncelleme yapmaya *dönem* diyoruz. Örnekleri birer birer öğrenirken η öğrenme çarpanının toptan öğrenmeye göre daha küçük seçilmesi ve örneklerin rastgele sırada görülmesi gerekir. Öğrenme kümesinde birbirine benzeyen



Şekil 11.8: x^t öğrenme örnekleri $(-0.5, 0.5)$ aralığından tekdüze çekilmiş ve '+' ile gösterilmiştir; $y^t = f(x^t) + \mathcal{N}(0, 0.1)$ olarak oluşturulmuştur. $f(x) = \sin(6x)$ kesikli çizgiyle gösterilmiştir. İki saklı birimli çok katmanlı algılayıcının 100, 200, ve 300 dönemden sonra çıktısı çizilmiştir.

örnekler olacağından çevrimiçi öğrenme daha hızlı yakınsar; rastgeleliğin gürültü ekleme gibi rastgele bir etkisi vardır ve yerel olarak en küçük noktalardan kaçmayı sağlar.

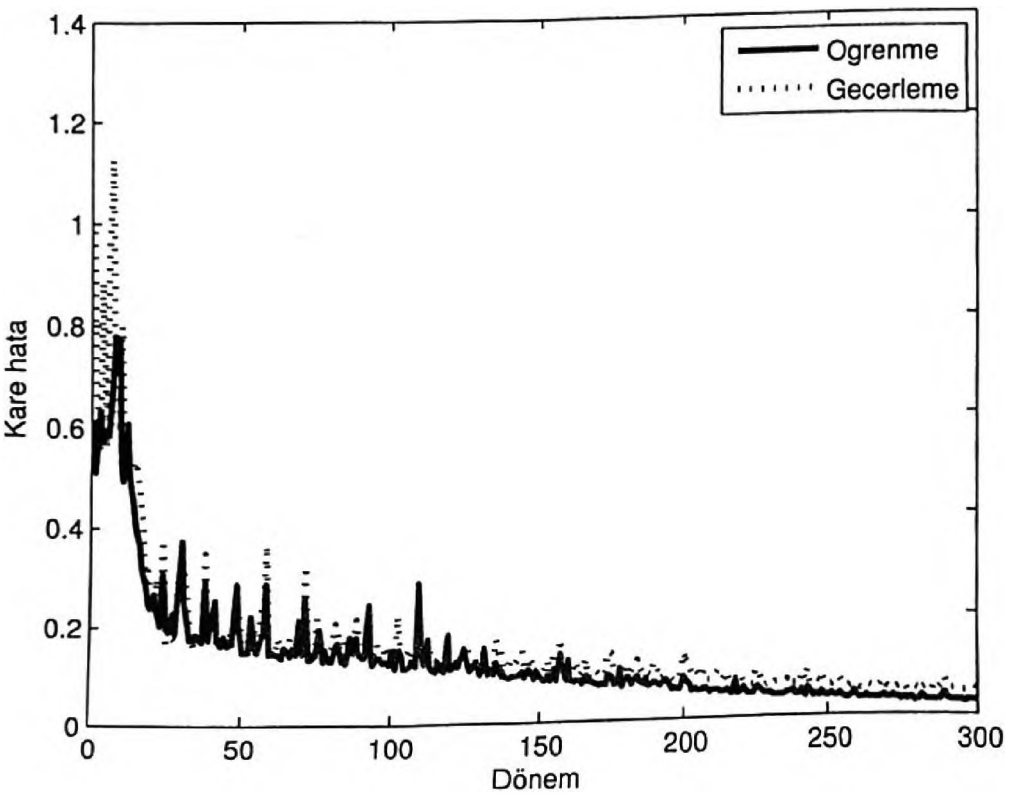
Bağlanımda çok katmanlı algılayıcı kullanmaya bir örnek şekil 11.8'de verilmiştir. Öğrenme sürdükçe çıktı yaklaşıklanan işleve daha çok yaklaşır ve hata azalır (bkz. şekil 11.9). Şekil 11.10'da çok katmanlı algılayıcı çıktısının saklı birim çıktıları üzerinden nasıl hesaplandığını görüyoruz.

Aynı girdilere bağlı birden çok bağlanım yapılmak istendiğinde birden çok çıktı birimi gerekir:

$$y_i^t = \sum_{h=1}^{H} v_{ih} z_h^t + v_{i0} \tag{11.17}$$

Tüm çıktılardaki kare hatalar toplanır:

$$E(\mathbf{W}, \mathbf{V}|\mathcal{X}) = \frac{1}{2} \sum_t \sum_i (r_i^t - y_i^t)^2 \tag{11.18}$$



Şekil 11.9: Öğrenme ve geçerleme kümeleri üzerindeki ortalama kare hata öğrenme dönem sayısına göre gösterilmiştir.

Toptan güncelleme kuralları aynı biçimde eğim inişle türetilir:

$$\Delta v_{ih} = \eta \sum_t (r_i^t - y_i^t) z_h^t \tag{11.19}$$

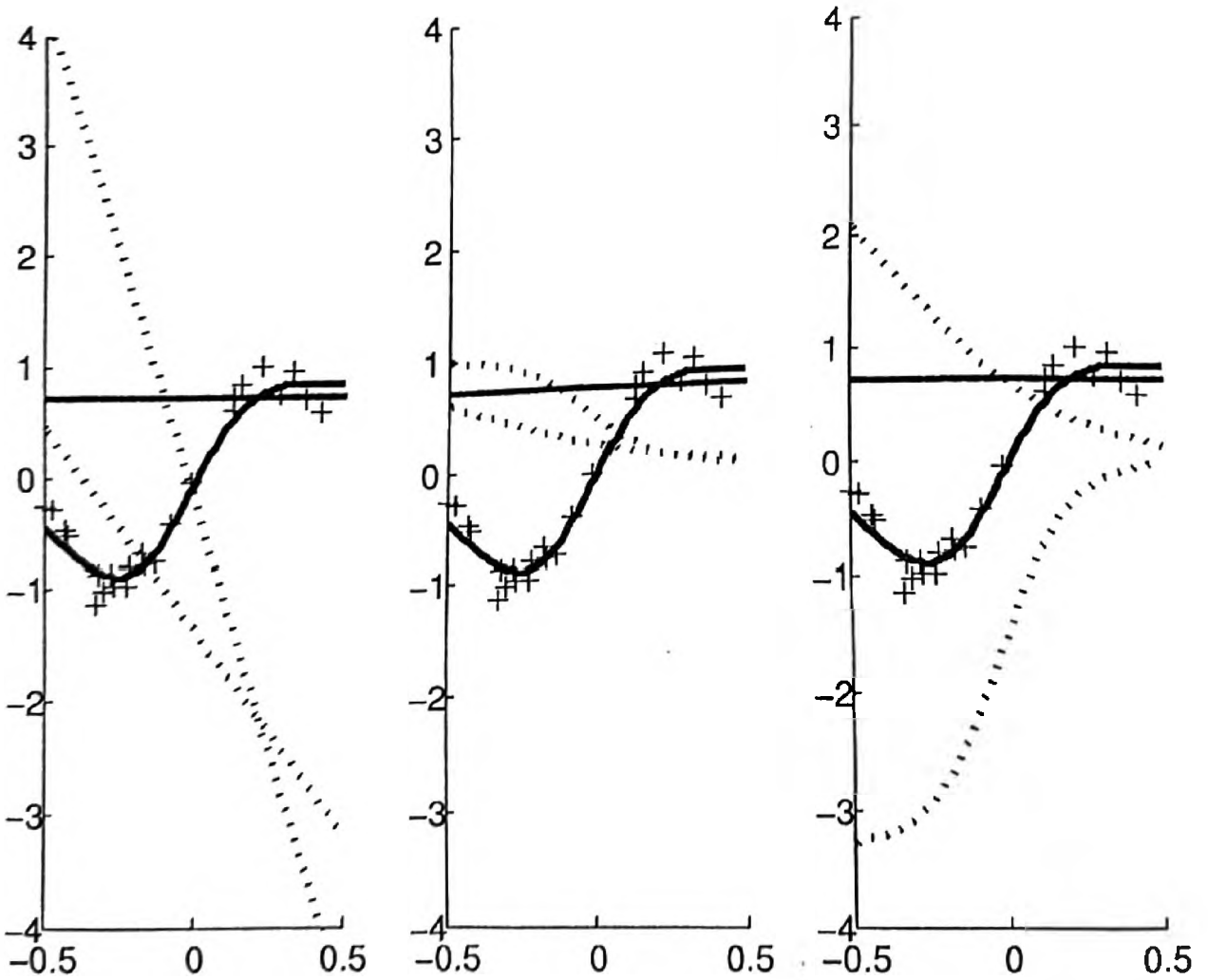
$$\Delta w_{hj} = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) v_{ih} \right] z_h^t (1 - z_h^t) x_j^t \tag{11.20}$$

$\sum_i (r_i^t - y_i^t) v_{ih}$ terimi h saklı birimine tüm çıktılar üzerinden toplam geriye taşınan hatadır. Örnek program, şekil 11.11'de verilmiştir. Tüm çıktılar aynı saklı birimleri paylaştığı için aynı saklı gösterimi kullanır; bu durumda çıktıların birbiriyle ilgili bağlanımlar yaptığını varsaymış oluruz. Bir başka olasılık ise farklı çıktıların aynı girdilere bağlı ama ayrık saklı birim kümeleri kullanmasıdır.

11.7.2 İki Sınıf Sınıflandırma

İki sınıf olduğunda tek çıktı yeter:

$$y^t = \text{sigmoid}\left(\sum_{h=1}^{H} v_h z_h^t + v_0 \right) \tag{11.21}$$



Şekil 11.10: (Left) Birinci katmandaki ağırlıkların tanımladığı doğrular, (middle) saklı birim çıktıları, ve (right) saklı birim çıktılarıyla ikinci katman ağırlıklarının çarpımı. Biri hafif yana kaydırılmış ve eksi ağırlıkla çarpılmış iki S işlevli saklı birimin çıktıları eklendiğinde bir tümsek tanımlar. Daha çok saklı birimle daha iyi bir yakınsama elde edilebilir (bkz. şekil 11.12).

v_{ih} ve w_{hj} ağırlıklarını rand$(-0.01, 0.01)$ aralığından başlat
Yinele
  Tüm$(\boldsymbol{x}^t, r^t) \in \mathcal{X}$ için rastgele sırada
    Tüm $h = 1, \ldots, H$
      $z_h \leftarrow \text{sigmoid}(\boldsymbol{w}_h^T \boldsymbol{x}^t)$
    Tüm $i = 1, \ldots, K$
      $y_i = \boldsymbol{v}_i^T \boldsymbol{z}$
    Tüm $i = 1, \ldots, K$
      $\Delta \boldsymbol{v}_i = \eta(r_i^t - y_i^t)\boldsymbol{z}$
    Tüm $h = 1, \ldots, H$
      $\Delta \boldsymbol{w}_h = \eta(\sum_i (r_i^t - y_i^t) v_{ih}) z_h (1 - z_h) \boldsymbol{x}^t$
    Tüm $i = 1, \ldots, K$
      $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \Delta \boldsymbol{v}_i$
    Tüm $h = 1, \ldots, H$
      $\boldsymbol{w}_h \leftarrow \boldsymbol{w}_h + \Delta \boldsymbol{w}_h$
Yakınsamaya dek

Şekil 11.11: K çıktılı çok katmanlı algılayıcıda bağlanım için hata geri yayma algoritması. Bu program, S işlevli tek çıktı ile iki sınıf, ve eşiksiz en büyük işleviyle $K > 2$ sınıf sınıflandırma için de kullanılabilir.

Bu değer $P(C_1|\boldsymbol{x}^t)$ sonsal olasılığı için bir kestirimdir; benzer biçimde $\hat{P}(C_2|\boldsymbol{x}^t) \equiv 1 - y^t$ olur. 10.7. bölümden bu durumda kullanılan hatanın çapraz düzensizlik olduğunu hatırlayalım:

$$E(\mathbf{W}, \boldsymbol{v}|\mathcal{X}) = -\sum_t r^t \log y^t + (1 - r^t) \log(1 - y^t) \tag{11.22}$$

Eğim iniş için güncelleme kurallarını türetebiliriz:

$$\Delta v_h = \eta \sum_t (r^t - y^t) z_h^t \tag{11.23}$$

$$\Delta w_{hj} = \eta \sum_t (r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t \tag{11.24}$$

Tek katmanlı algılayıcıda olduğu gibi bağlanım ve sınıflandırma için güncelleme kuralları aynıdır (ama değerleri aynı olmak zorunda değildir).

11.7.3 $K > 2$ Sınıf Sınıflandırma

$K > 2$ sınıf olduğunda K çıktı vardır:

$$o_i^t = \sum_{h=1}^{H} v_{ih} z_h^t + v_{i0} \tag{11.25}$$

Eşiksiz en büyük işlevini kullanarak sınıflar arasındaki bağımlılığı tanımlarız:

$$y_i^t = \frac{\exp o_i^t}{\sum_k \exp o_k^t} \tag{11.26}$$

Burada y_i ile $P(C_i|x^t)$ sonsalını kestirdiğimizi hatırlayalım. Hata işlevi

$$E(\mathbf{W}, \mathbf{V}|\mathcal{X}) = -\sum_t \sum_i r_i^t \log y_i^t \tag{11.27}$$

üzereinden eğim inişle güncelleme kurallarını türetebiliriz:

$$\Delta v_{ih} = \eta \sum_t (r_i^t - y_i^t) z_h^t \tag{11.28}$$

$$\Delta w_{hj} = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) v_{ih} \right] z_h^t (1 - z_h^t) x_j^t \tag{11.29}$$

Richard ve Lippmann (1991), yeterince karmaşık bir ağ ve büyük bir öğrenme kümesiyle, uygun eğitilmiş çok katmanlı bir algılayıcının sonsal ağırlıkları kestirebildiğini göstermiştir.

11.7.4 Birden Çok Saklı Katman

Daha önce gördüğümüz gibi her biri kendi ağırlıkları olan ve S işlevi kullanan birden çok saklı katmanımız olabilir. Örneğin bağlanım için iki saklı katmanımız olsun:

$$z_{1h} = \text{sigmoid}(\mathbf{w}_{1h}^T \mathbf{x}) = \text{sigmoid}\left(\sum_{j=1}^{d} w_{1hj} x_j + w_{1h0} \right), \quad h = 1, \ldots, H_1$$

$$z_{2l} = \text{sigmoid}(\mathbf{w}_{2l}^T \mathbf{z}_1) = \text{sigmoid}\left(\sum_{h=0}^{H_1} w_{2lh} z_{1h} + w_{2l0} \right), \quad l = 1, \ldots, H_2$$

$$y = \mathbf{v}^T \mathbf{z}_2 = \sum_{l=1}^{H_2} v_l z_{2l} + v_0$$

Burada, $\mathbf{w}_{1h}$ ve $\mathbf{w}_{2l}$, sırasıyla birinci ve ikinci katman ağırlıklarını, z_{1h} ve z_{2h}, sırasıyla birinci ve ikinci saklı katman birimlerini, ve $\mathbf{v}$, üçüncü katman ağırlıklarını gösterir. Böyle bir ağı aynı biçimde eğitebiliriz, tek fark birinci katman ağırlıkları için hatayı zincir kuralıyla bir katman daha geriye yaymamızın gerekmesidir (alıştırma 5).

11.8 Öğrenme Yordamları

11.8.1 Yakınsamayı İyileştirme

Eğim inişin birçok güzel yanı vardır. Basittir. Yereldir; bir bağlantı ağırlığındaki değişiklik yalnızca öncesindeki ve sonrasındaki birimlere ve (geriye yayılmış) hataya

bağlıdır. Çevrimiçi öğrenme kullanıldığında öğrenme kümesinin saklanması gerekmez ve öğrenilecek problem değiştiğinde model kendini yavaş yavaş uyarlayabilir. Bu nedenlerden dolayı donanım gerçeklemesi de zor değildir (ve yapılmıştır). Ama eğim inişin yakınsaması yavaş olabilir ve öğrenme zamanı önemli olduğunda daha gelişmiş yordamlar kullanmak gerekebilir (Battiti 1992). Bishop (1995), çok katmanlı algılayıcıların eğitilmesinde eşlenik eğim ve ikinci dereceden yöntemlerin nasıl kullanıldığını tartışır. Ama sıkça kullanılan iki basit yöntemle eğim inişin başarımı artırılıp gerçek uygulamalarda kullanılması sağlanabilir.

Momentum

w_i, çok katmanlı algılayıcının ağırlıklarından herhangi biri olsun. Her güncellemede hesaplanan ardışık Δw_i^t değerleri farklı olup hatada salınımlara neden olabilir ve yakınsamayı yavaşlatabilir. t zaman dizin değişkeni, toptan öğrenmede dönem sayısını, çevrimiçi öğrenmedeyse yineleme sayısını göstersin. Buradaki düşünce, önceki güncelleme değerlerini de kullanarak kayan ortalama alan bir *momentum* terimi hesaplamak ve öğrenmeye eklemektir:

$$\Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1} \tag{11.30}$$

Monentum çarpanı α genelde 0.5 ve 1.0 arasında bir değer alınır. Bu yaklaşım özellikle çevrimiçi öğrenmede, ortalama alarak öğrenme sırasında ağırlık yöneyinin izlediği gezingeyi düzleştirdiği için yararlıdır. Çekincemiz önceki Δw_i^{t-1} değerlerini saklamak için ek bellek gerektirmesidir.

Uyarlanabilir Öğrenme Hızı

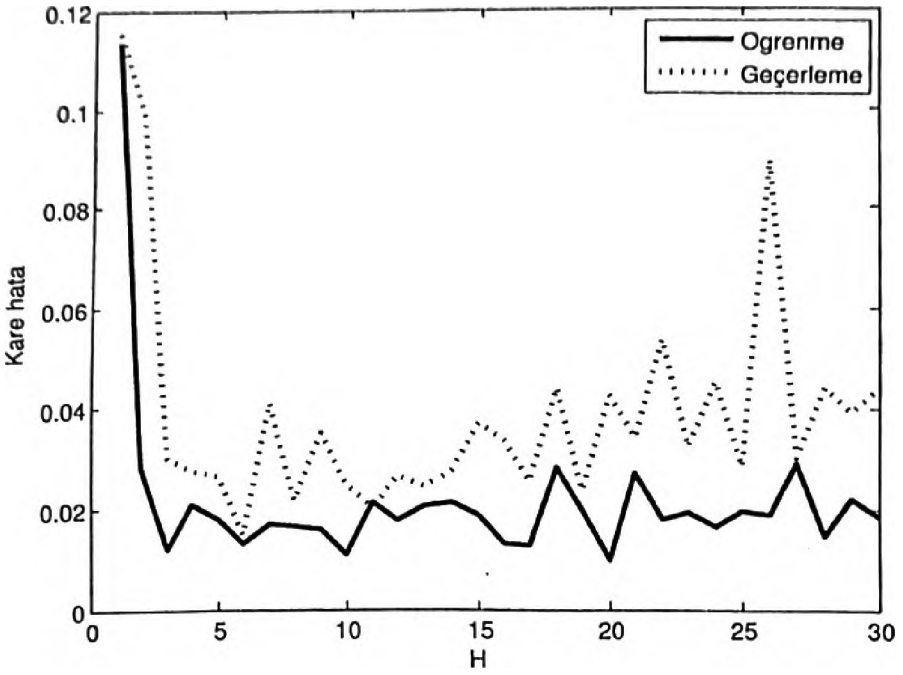
Eğim inişte η öğrenme çarpanı değişikliğin büyüklüğünü belirler. Genelde 0.0 ve 1.0 arasında, çoğunlukla 0.2 çevresinde alınır. Bu çarpan hızlı yakınsama için öğrenme sırasında uyarlanabilir, öyle ki öğrenme sürerken büyüktür ve yavaşladıkça küçülür:

$$\Delta \eta = \begin{cases} +a & \text{eğer } E^{t+\tau} < E^t \\ -b\eta & \text{değilse} \end{cases} \tag{11.31}$$

Öğrenme kümesindeki hata azaldığında η sabit bir değerle artırılıyor ve hata artarsa geometrik azaltılıyor. Bir dönemden sonrakina E salınabileceğinden E^t kestirimi olarak önceki birkaç dönem hata değerinin ortalamasını kullanmak daha iyi olabilir.

11.8.2 Fazla Öğrenme

d tane girdisi, H tane saklı birimi ve K tane çıktısı olan bir çok katmanlı algılayıcının birinci katmanında $H(d+1)$ tane, ikinci katmanında $K(H+1)$ tane bağlantısı vardır; bu yüzden hem zaman, hem de bellek karmaşıklığı $\mathcal{O}(H \cdot (K+d))$ olur. Öğrenme dönem sayısını e ile gösterirsek öğrenme zaman karmaşıklığı $\mathcal{O}(e \cdot H \cdot (K+d))$ olur.

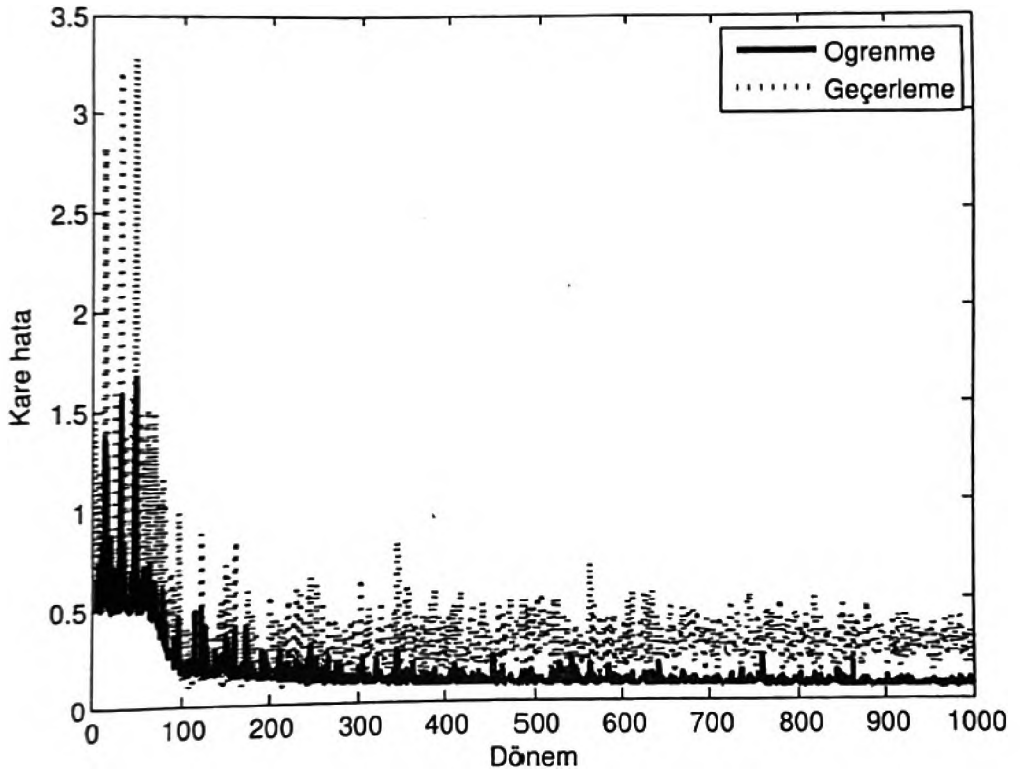


Şekil 11.12: Karmaşıklık (saklı birim sayısı) arttıkça öğrenme hatası değişmezken geçerleme hatası artabilir, buna "aşırı öğrenme" denir.

Bir uygulamada d ve K değerleri önceden belirlidir; karmaşıklığı uyarlamak için kullanabileceğimiz tek üstparametre H ile gösterdiğimiz saklı birim sayısıdır. Önceki bölümlerde, fazla karmaşık bir modelin öğrenme kümesindeki gürültüyü öğrenip geçerleme kümesinde iyi genelleme yapamadığını görmüştük. Örneğin çokterimli bağlanımda gürültü olduğunda ya da küçük örneklemlerde, çokterimlinin derecesini artırmak daha kötü genellemeye neden oluyordu. Benzer biçimde çok katmanlı algılayıcının saklı birim sayısı fazla olduğunda genelleme kötüleşir (bkz. şekil 11.12). Yanlılık-değişinti ikilemi, başka istatistiksel kestiriciler gibi çok katmanlı algılayıcı için de geçerlidir (Geman, Bienenstock, ve Doursat 1992).

Benzer bir davranış öğrenme uzun sürdülürse de görülür: Daha çok öğrenme dönemi geçtikçe öğrenme kümesindeki hata azalsa da geçerleme kümesindeki hata bir noktadan sonra artabilir (bkz. şekil 11.13). Başlangıçta tüm ağırlıklar sıfıra yakındır ve etkisizdir. Öğrenme sürdükçe önemli ağırlıklar sıfırdan uzaklaşır ve önemli olmaya başlar. Ama öğrenme uzun süre devam ederse tüm ağırlıklar sıfırdan uzaklaşıp devreye girecektir. Sanki öğrenme sürdükçe kestiriciye parametre eklenecek, karmaşıklık artacak, algılayıcı aşırı öğrenecek, ve genelleme kötüleşecektir. Öğrenme sırasında *erken durma*, bu biçimde *aşırı eğitmeyi* engelleyecektir. Öğrenmenin nerede durdurulacağı ve en iyi saklı birim sayısı çapraz geçerlemeyle, yani algılayıcıyı öğrenme sırasında görülmemiş geçerleme kümesi üzerinde deneyerek belirlenir.

Saklı birimlerdeki doğrusal olmayan etkilenim işlevi yüzünden hata işlevinin en küçük değer aldığı birçok nokta vardır ve eğim iniş bunların en yakınına yakınsar. Beklenen hatayı ölçmek için aynı ağı farklı ilk ağırlık değerlerinden başlayıp birçok



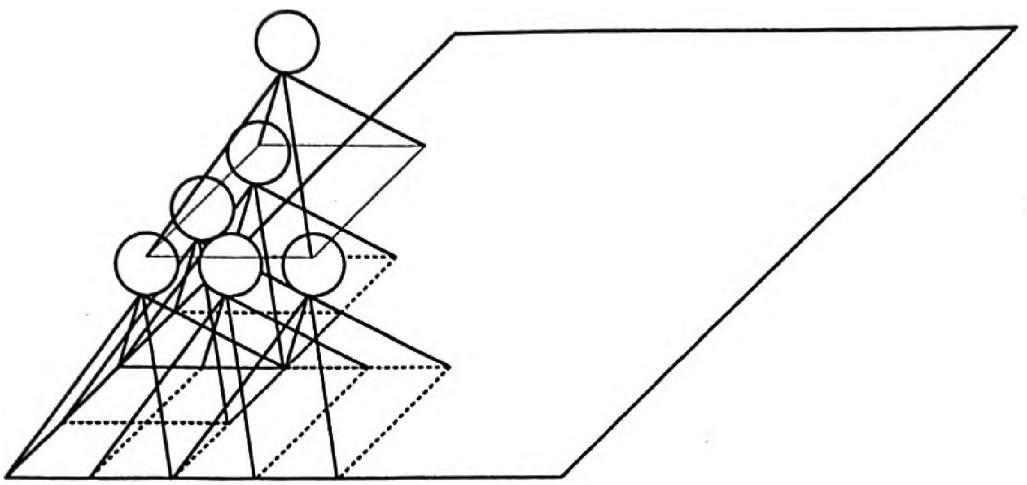
Şekil 11.13: Öğrenme sürdükçe geçerleme hatası artıp algılayıcı ezberlemeye başlayabilir; buna "aşırı eğitme" denir ve bir tür aşırı öğrenmedir.

kez eğitmek ve ortalama hatayı kullanmak gerekir.

11.8.3 Ağın Yapılandırılması

Bazı uygulamalarda girdinin yerel bir yapısı olabilir. Örneğin görüntü tanımada komşu noktalar ilintilidir, kenar ya da köşe gibi yerel öznitelikler vardır, ve nesneler, örneğin elle yazılmış bir rakam, böylesi temel özniteliklerin bir bileşimidir. Benzer biçimde konuşma tanımada, zamanda yerellik vardır ve zaman içinde yakın girdiler temel konuşma öznitelikleri oluşturabilir. Bu temel öznitelikleri birleştirerek daha uzun ses dizileri, örneğin sesbirimleri, tanımlanabilir. Böyle bir durumda çok katmanlı algılayıcıyı tasarlarken saklı birimler tüm girdilere bağlanmaz çünkü bütün girdilerin ilintili olmadığı bilinir. Bunun yerine saklı birimler girdi uzayında bir pencere tanımlar ve yalnızca yerel bir küme girdiye bağlanır. Bu bağlantı sayısını ve dolayısıyla serbest parametre sayısını azaltır (Le Cun ve ötekiler 1989).

Bunu ardışık katmanlarda yineleyebiliriz. Her katmandaki birim altındaki katmandaki yerel bir kümeye bağlı olduğunda katman katman yükseldikçe çıktıya gelene dek gittikçe daha karmaşık özniteliklere, girdinin daha büyük bir bölümünde bakarız (bkz. şekil 11.14). Örneğin ilk girdiler imge noktaları olabilir; sonra onları birleştirerek ilk saklı katman farklı açılardaki kenarları, ikinci saklı katman kenarların birleşimiyle köşeleri, kavisleri, ve daha yukarı katmanlar, dikdörtgenleri, çemberleri,



Şekil 11.14: Yapılandırılmış çok katmanlı algılayıcıya bir örnek. Her birim, altındaki yerel bir küme birime bağlıdır ve yerel bir özniteliğe tepki verir; örneğin görüntü tanımada bu, bir kenar ya da köşe olabilir. Kolay anlaşılabilmesi için her alanda yalnızca tek bir saklı birim gösterilmiştir; normalde farklı yerel öznitelikleri bulabilmek için her noktada birden çok birim kullanılır.

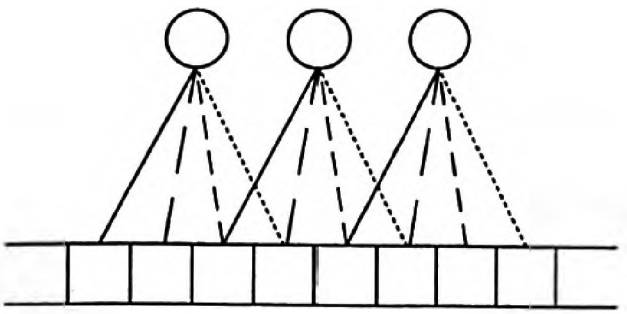
ve onları da birleştirip, örneğin bir yüz tanıma uygulamasında, gözleri, ağzı bulmayı öğrenebilir. Buna *öznitelik ağacı* denir; yükseldikçe sınıflara gelene dek öznitelikler daha karmaşık, soyut ve daha az sayıdadır.

Böyle bir uygulamada, *ağırlık paylaşımıyla* parametre sayısı daha da azalabilir. Yine görüntü tanımayı örnek olarak alırsak, örneğin 45 derece açılı bir kenar resmin farklı yerlerinde olabilir. Bu yüzden girdinin farklı yerlerinde farklı öznitelikler için farklı saklı birimler tanımlamak yerine aynı saklı birimin farklı yerlerde kopyalarını tanımlarız (bkz. şekil 11.15). Öğrenme sırasında girdileri farklı olduğu için eğim yöneyleri farklı olacaktır ama bu eğim yöneylerinin ortalamasını alıp tek bir güncelleme yaparız, ve böylece aynı parametre farklı bağlantıların paylaştığı tek bir ağırlık değeri tanımlar. Üstelik ağırlıktaki güncelleme farklı girdiler üzerinden hesaplandığı için sanki öğrenme kümesi büyümüş gibi olur.

11.8.4 İpucu

Girdide yerel bir düzen olduğunu bilmek, çokkatmanlı ağ için bir yapı tanımlamayı ve ağırlık paylaşımıyla parametre sayısını azaltabilmeyi sağlar. Tüm birimlerin birbirine bağlı olduğu ağ yapısızdır ve eğitilmesi daha zordur. Uygulamayla ilgili bir önsel bilgimiz varsa ağ yapısına yansıtılması öğrenmeyi kolaylaştırır. Böyle önsel bilgilere *ipucu* (Abu-Mostafa 1995) denir ve öğrenme örneklerinden bağımsız olarak öğrenmek istediğimiz işlevin özelliklerine karşılık gelir.

Görüntü tanımada değişmezlik ipuçları vardır: Nesnenin kimliği taşındığında, döndürüldüğünde, ya da ölçeklendiğinde değişmez (bkz. şekil 11.16). İpuçları öğrenmeyi kolaylaştıran ek bilgidir ve özellikle öğrenme kümesi küçük olduğunda çok faydalıdır. İpuçları farklı biçimlerde kullanılabilir:



Şekil 11.15: Ağırlık paylaşımında farklı birimlerin farklı girdilere bağlantıları aynı ağırlık değerini paylaşır (burada farklı çizgi türleriyle gösterilmiştir). Yalnızca tek bir birim gösterilmiştir; bir uygulamada farklı özniteliklere bakan farklı birimler (ve kopyaları) olacaktır.

1. İpucuyla *sanal örnekler* oluşturabiliriz. Örneğin, ölçeklendiğinde nesne sınıfının değişmediğini biliyorsak bir öğrenme örneğini farklı büyüklüklere ölçekleyerek ek öğrenme örnekleri oluşturabiliriz. Böylece öğrenme modelini değiştirmeden öğrenme kümesini ipucunu öğrenebilecek biçimde büyütmüş oluruz. Ama bazen değişmezlikler yalnızca öğrenme örnek sayısını artırarak öğrenilemeyebilir.

2. Değişmezlik bir önişlemeyle kazandırılabilir. Örneğin, harf görüntüleri önişlenip aynı büyüklüğü getirildikten sonra tanıma yapılır. Yapılabiliyorsa bu en kolay çözümdür.

3. İpucu ağ yapısına yansıtılabilir. 11.8.3. bölümde gördüğümüz yerel bağlantılar ve ağırlık paylaşımı küçük taşınma ve döndürmelere değişmezlik sağlar ve buna bir örnektir.

4. İpucu hata işlevine yansıtılabilir. Uygulama açısından $\boldsymbol{x}$ ve $\boldsymbol{x}'$ örneklerinin aynı olduğunu varsayalım ($\boldsymbol{x}'$, $\boldsymbol{x}$ için "sanal örnek" olsun), öyle ki $f(\boldsymbol{x})$ yakınsamak istediğimiz işlevse $f(\boldsymbol{x}) = f(\boldsymbol{x}')$ sağlansın. $g(\boldsymbol{x}|\theta)$ ile yakınsayan modelimizi gösterelim; model, örneğin ağırlıkları θ ile gösterilen çok katmanlı bir algılayıcı olabilir. Bu durumda olası tüm $(\boldsymbol{x}, \boldsymbol{x}')$ çiftleri için bir ceza işlevi

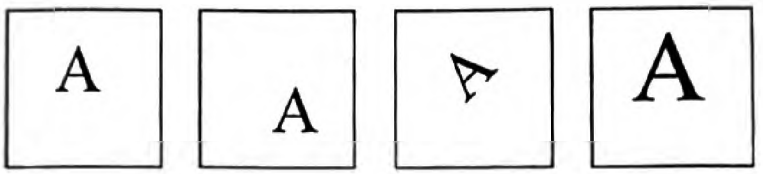
$$E_h = [g(\boldsymbol{x}|\theta) - g(\boldsymbol{x}'|\theta)]^2$$

tanımlayıp normalde kullandığımız hata işlevine ek bir ceza terimi olarak ekleyebiliriz:

$$E' = E + \lambda_h \cdot E_h$$

Bu ceza terimi öngörülerimizin ipucuna uymadığı örnekleri cezalandıracaktır; λ_h burada bu cezanın büyüklüğünü gösterir (Abu-Mostafa 1995).

Bir başkası yaklaşıklama ipucudur: Örneğin x girdisi için kesin $f(x)$ değerini değil ama düşmesi gereken $[a_x, b_x]$ aralığını biliyor olalım. Bu durumda yine



Şekil 11.16: Nesnenin kimliği taşındığında, döndürüldüğünde, ya da ölçeklendiğinde değişmez. Bu her zaman doğru olmayabilir: 'b' döndürüldüğünde 'q' olur. Bu tür değişmezlikler öğrenmeyi kolaylaştırmak için kullanılabilecek ipuçlarına örnektir.

ipucundan gelen bir ceza işlevi yazabiliriz:

$$E_h = \begin{cases} 0 & \text{eğer } g(x|\theta) \in [a_x, b_x] \\ (g(x) - a_x)^2 & \text{eğer } g(x|\theta) < a_x \\ (g(x) - b_x)^2 & \text{eğer } g(x|\theta) > b_x \end{cases}$$

Bu, destek yöney bağlanımında (bölüm 13.10) kullanılan ve küçük hataları hoşgören hata işlevine benzer.

Bir başka örnek uygulamada ipucunun tanımladığı dönüşüm, örneğin, belirli bir açıda döndürme, bir işlev olarak tanımlanır ve hata işlevi bu dönüşümün tanımladığı gezinge üzerindeki oynamalar hatayı etkilemeyecek biçimde yazılır (Simard ve ötekiler 1992).

11.9 Ağ Büyüklüğünün Uyarlanması

Önceki bölümlerde ağ büyük ve dolayısıyla çok serbest parametreli olduğunda iyi genelleme yapmadığını gördük. En iyi ağ büyüklüğünü bulurken en sık kullanılan yaklaşım farklı yapıları öğrenme kümesinde eğitip geçerleme kümesinde en iyi olanı seçmektir. Bir başkasıysa bu tür *yapısal uyarlamayı* öğrenme algoritmasının içine koymaktır. Bu iki farklı biçimde yapılabilir:

1. *Azalımlı* yaklaşımda büyük bir ağla başlar, sonra gerekmeyen birim ve/ya da bağlantıları budarız.

2. *Artımlı* yaklaşımda küçük bir ağla başlar, başarımı artıracak biçimde yeni birim ve/ya da bağlantılar ekleriz.

Azalımlı yaklaşıma bir örnek gereksiz bağlantıların atıldığı *ağırlık sönümüdür*. Gerçekte bir birim ya da bağlantının gereksiz olduğunu anlamanın yolu bir onunla, bir onsuz eğitim yapıp geçerleme hatasında fark olup olmadığına bakmaktır. Ama olası tüm birim/bağlantı bileşimleri için bunu yapmak çok pahalı olacaktır.

Ağırlığı 0 olan bir bağlantı kullanılmadığı düşüncesiyle her ağırlığa sıfıra doğru sönme eğilimi ekleriz; böylece hatayı azaltmak için özellikle sıfırdan uzaklaştırılmazsa

ağırlık 0 olur ve karşılık gelen bağlantı atılabilir. Ağdaki herhangi bir w_i ağırlığı için güncelleme kuralı

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} - \lambda w_i \tag{11.32}$$

genişletilmiş hata işlevi

$$E' = E + \frac{\lambda}{2} \sum_i w_i^2 \tag{11.33}$$

üzerinde eğim inişe karşılık gelir. Eklenmiş terim sıfırdan farklı ağırlıkları λ oranında cezalandırır.

Basit ağların daha iyi genelleme yaptığı bir ipucu bilgisidir ve böyle bir ipucu, karmaşıklığı cezalandıran bir terimi hataya ekleyerek tanımlanabilir. Ama dikkat: Basit ağların karmaşıklardan her zaman daha iyi olduğunu söylemiyoruz; söylediğimiz, ikisi de aynı öğrenme hatasına sahip iki ağ olduğunda basit, yani daha az parametreli olanın geçerleme kümesinde daha iyi genelleme yapma olasılığının daha yüksek olduğudur.

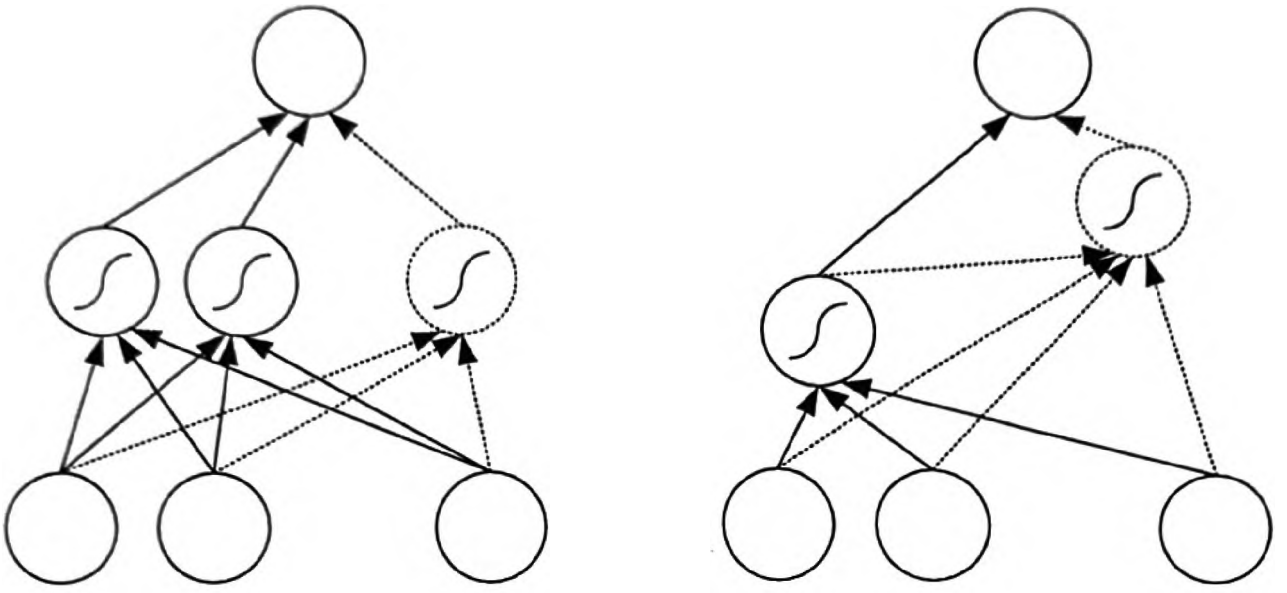
11.32. denklemdeki ikinci terimin etkisi her ağırlığı sıfıra çeken bir yay gibidir. Sıfıra yakın bir değerden başlayıp her ağırlık, eğer hata eğimi büyük olup güncelleştirmeye neden olmazsa ikinci terimle zaman içinde sıfıra doğru söner. λ, öğrenme kümesindeki hata ve sıfırdan farklı parametrelerin neden olduğu ceza terimlerinin görece etkilerini ve dolayısıyla sıfıra sönümün hızını belirler: λ büyük olduğunda ne ne olursa olsun ağırlıklar sıfıra gider; küçük bir λ olduğunda sıfırdan farklı ağırlıkların cezası fazla olmaz. En iyi değer çapraz geçerlemeyle ayarlanır.

Büyük bir ağdan başlayıp gerekmeyen bağlantı ve birimleri *budamak* yerine küçük bir ağla başlayıp gerektikçe yeni birim ve ilgili bağlantılar da eklenebilir (bkz. şekil 11.17). *Devingen düğüm yaratma* (Ash 1989) algoritmasında tek saklı katmanında tek birim olan çok katmanlı bir algılayıcıyla başlarız; bu ağ yakınsadığında hata hâlâ yüksekse bir saklı birim daha ekleriz. Her eklemede yeni eklenmiş birime gelen ve ondan çıkan ağırlıklar rastgele değerlerle başlatılır, öteki ağırlıklarse sıfırlanmaz ve önceki değerlerinden devam eder.

Ardışık ilinti (Fahlman ve Lebiere 1990) algoritmasında her eklenen birim yeni bir saklı katmandaki yeni bir birimdir. Her saklı katmandaki birim önceki tüm saklı birimlere ve girdiye bağlıdır. Önceki bağlantılar dondurulur ve eğitilmez; yalnızca yeni eklenen birime gelen ve ondan çıkan bağlantıların ağırlıkları eğitilir.

Devingen düğüm yaratma var olan saklı katmana birim ekler, ama yeni saklı katman eklemez. Ardışık ilintiyse tek birimli yeni saklı katmanlar ekler. İyi bir artımlı algoritmanın ikisini de yapabilmesi ve yeni katman eklemeyle olan katmana birim ekleme arasında seçim yapabilmesi gerekir. Bu açık bir araştırma konusudur.

Artımlı ve azalımlı algoritmaların ilginç yanı öğrenme sırasında yalnızca parametreleri değil yapıyı da değiştirebilmeleridir. Çok katmanlı ağın yapısının tanımladığı bir uzay düşünürsek birim ya da katman ekleme/çıkarma işlemleri bu uzayda yer değiştirmeye karşılık gelir (Aran ve ötekiler 2009). Artımlı/azalımlı algoritmalar bu uzayda arama yapar ve farklı işlemleri deneyip bir iyilik kıstasına göre kabul ya da ret eder. Bu kıstasın, başarıyla karmaşıklığın bir birleşimi olmasını bek-



Şekil 11.17: Artımlı algoritmalara iki örnek. (a) Devingen düğüm yaratma algoritmasında olan katmana yeni birim eklenir. (b) Ardışık ilinti algoritmasında her birim önceki tüm katmanlara bağlı yeni bir katman tanımlar. Kesikli çizgiler yeni eklenmiş birim/bağlantıları gösterir. Anlaşılabilirlik için ek girdiler gösterilmemiştir.

leriz. Bir benzetme yaparsak bu, çokterimli bağlanımda yüksek dereceli terimlerin öğrenme sırasında eklenip çıkarılabilmesidir. Bu tür model seçim işlemlerinin öğrenme sırasında kullanıcıyı işe karıştırmadan yapılabilmesi yeğlenir.

11.10 Öğrenmeye Bayesçi Bakış

Yapay sinir ağlarının eğitiminde Bayesçi yaklaşım w_i bağlantı ağırlıklarını $p(w_i)$ önsel dağılımından çekilmiş rastsal değişkenler olarak görür ve sonsal olasılığı hesaplar:

$$p(\boldsymbol{w}|\mathcal{X}) = \frac{p(\mathcal{X}|\boldsymbol{w})p(\boldsymbol{w})}{p(\mathcal{X})} \tag{11.34}$$

Burada $\boldsymbol{w}$ yöneyi ağdaki tüm ağırlıkları içerir. Sonsal olasılığın tepe noktası en büyük sonsal (EBS) kestirimimiz olur:

$$\hat{\boldsymbol{w}}_{EBS} = \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{w}|\mathcal{X}) \tag{11.35}$$

Denklem 11.34'ün logaritmasını alalım:

$$\log p(\boldsymbol{w}|\mathcal{X}) = \log p(\mathcal{X}|\boldsymbol{w}) + \log p(\boldsymbol{w}) + C$$

Sağdaki ilk terim log olabilirlik, ikinci de önselin log değeridir. Ağırlıklar bağımsız ve önsel olasılık $\mathcal{N}(0, 1/2\lambda)$ alınırsa,

$$p(w) = \prod_i p(w_i) \text{ ve } p(w_i) = c \cdot \exp\left[-\frac{w_i^2}{2(1/2\lambda)}\right] \tag{11.36}$$

EBS kestiricisi genişletilmiş hatayı en küçük yapan değerdir:

$$E' = E + \lambda \|w\|^2 \tag{11.37}$$

Burada E ile gösterdiğimiz uygulamaya bağlı olarak sınıflandırma ya da bağlanım hatasıdır (eksi log olabilirlik). Bu genişletilmiş hatanın ağırlık sönümündeki hatayla aynı olduğuna dikkat edin (denklem 11.33). Yüksek bir λ değeri parametrelerde küçük bir değişkenlik varsayar ve onları sıfıra yakın tutmak için daha büyük bir güç uygular; λ küçük olduğunda parametrelerin değişkenliği daha fazla olabilir. Gereksiz parametrelerin bu biçimde elenmesine istatistikte *sırt bağlanımı* denir.

Burada yaptığımız bir tür *düzenlileştirmedir*:

$$\text{maliyet} = \text{verideki-hata} + \lambda \cdot \text{karmaşıklık} \tag{11.38}$$

Çok katmanlı algılayıcıların eğitiminde Bayesçi kestirim (MacKay 1992a, b) içinde tartışılır. Bayesçi yöntemleri daha geniş olarak 14. bölümde göreceğiz.

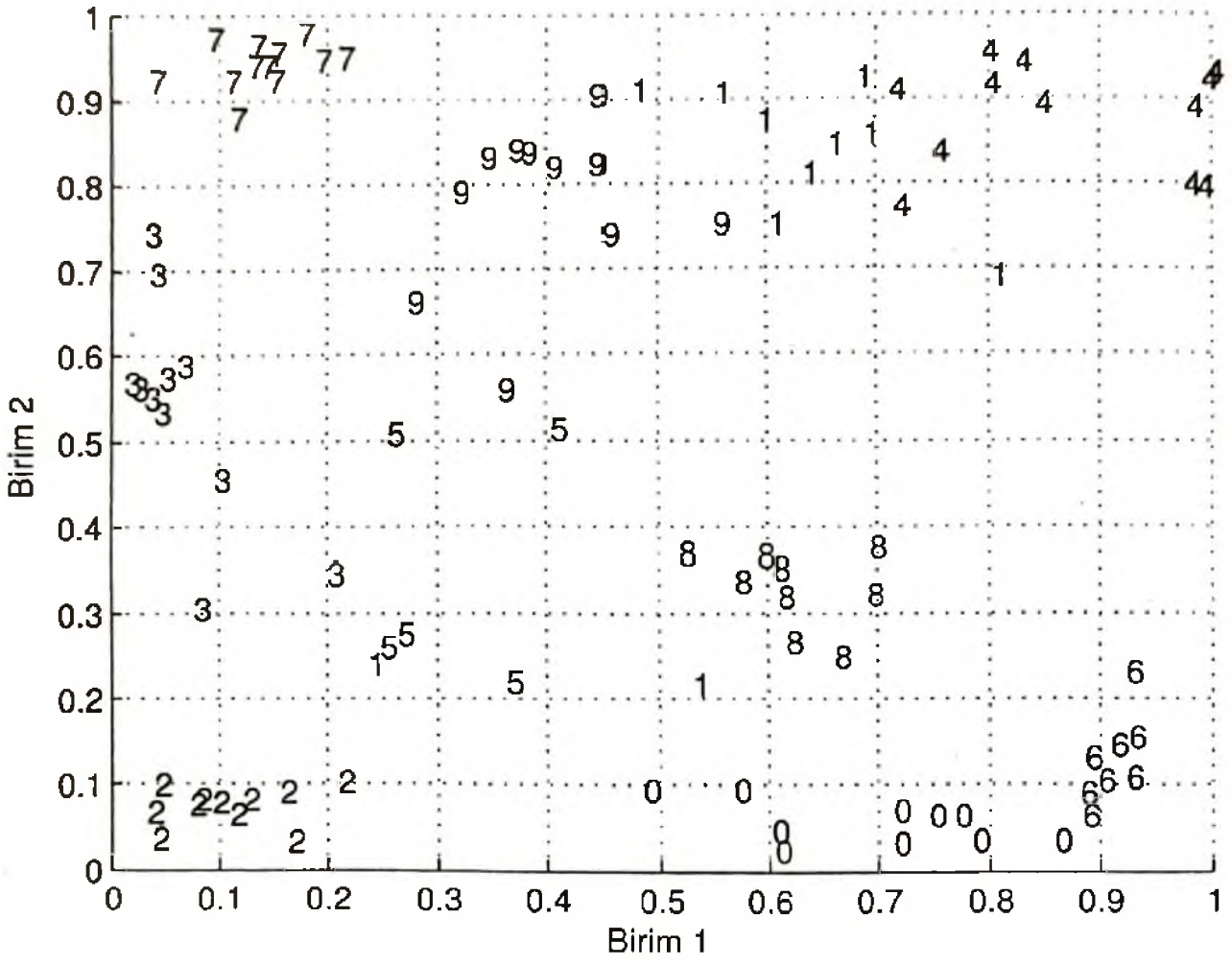
Gerçek uygulamalarda, öğrenmeden sonra çok katmanlı algılayıcı ağırlıklarının 0 çevresinde normal dağıldığı gözlenmiştir, ki bu, ağırlık sönümü algoritmasını destekleyen bir gözlemdir. Ama bu her zaman geçerli olmayabilir. Nowlan ve Hinton (1992), *eşiksiz ağırlık paylaşımı* algoritmasında ağırlıkların öncel dağılımını bir Gauss karışımı olarak modellemiştir. Böylece ağırlıklar birden çok öbekten gelebilir; bu öbeklerin merkezinin 0 olması gerekmez ve değişintileri ayarlanabilir. Bu, 11.36. denklemin önselini bir $M \geq 2$ Gauss karışımı yapar:

$$p(w_i) = \sum_{j=1}^{M} \alpha_j p_j(w_i) \tag{11.39}$$

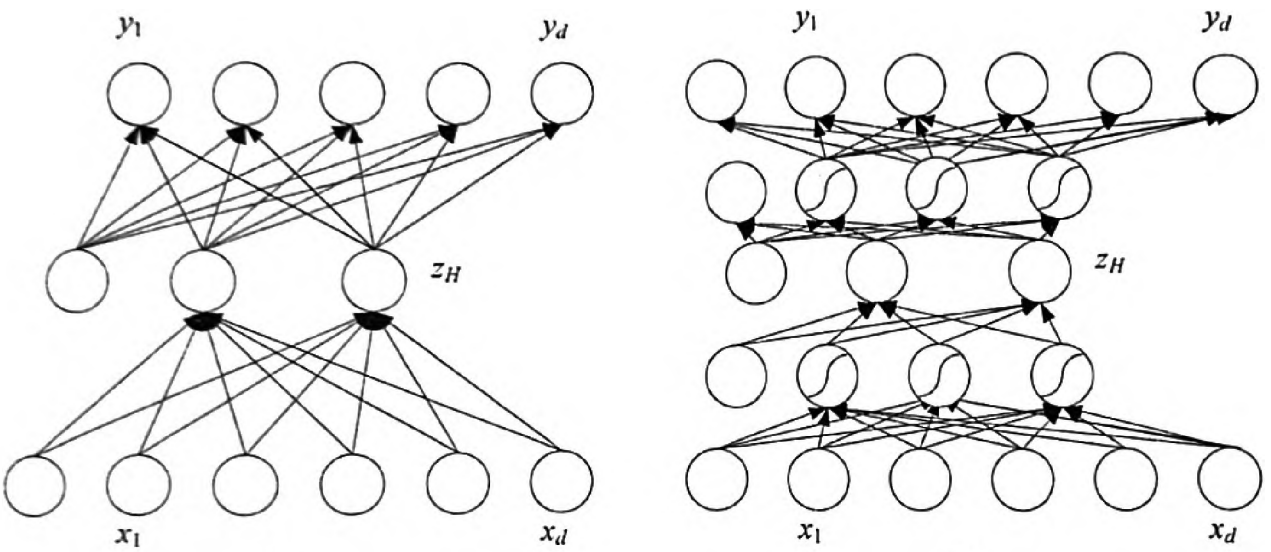
Burada, α_j ile bileşen ağırlıklarını, $p_j(w_i) \sim \mathcal{N}(m_j, s_j^2)$ ile Gauss bileşenlerini gösteriyoruz. M değeri kullanıcı tarafından tanımlanır ve $\{\alpha_j, m_j, s_j\}_j$ parametreleri veriden, bağlantı ağırlıklarıyla birlikte öğrenilir. Böyle bir önsel dağılım kullanıp onu hataya eklediğimizde öğrenme sonunda ağırlıklar bir yandan hatayı azaltırken bir yandan da log önsel değerini artıracak biçimde öbekler oluşturur.

11.11 Boyut Azaltma

Çok katmanlı bir algılayıcıda, saklı birim sayısı girdiden azsa ilk katman boyut azaltır. Bu azaltmanın biçimi ve saklı birimlerin tanımladığı yeni uzay, algılayıcının ne için eğitildiğine bağlıdır. Eğer sınıflandırma içinse yeni uzay sınıflandırma hatasını azaltacak biçimde öğrenilir (bkz. şekil 11.18).



Şekil 11.18: Optdigits verisiyle iki saklı birimli çok katmanlı algılayıcı eğittikten sonra verinin bu iki saklı birimin tanımladığı uzaydaki izdüşümü. Yalnız ilk yüz örneğin sınıfı gösterilmiştir. 64 girdi, iki saklı, on çıktı birimli bu modelin başarısı yüzde 80'dir. S biçimli işlev yüzünden saklı birim çıktıları 0 ve 1 arasındadır ve sınıflar köşelere toplanmıştır. Bu izdüşüm çizimi, 6. bölümdeki öteki boyut azaltma yöntemlerinin sonuçlarıyla kıyaslanabilir.



Şekil 11.19: Özilişkilendirici ağında girdi sayısı kadar çıktı vardır ve istenen çıktı yöneyiyle girdi yöneyi aynıdır. Saklı birim sayısı girdi sayısından az olduğunda böyle bir çok katmanlı ağ, girdiyi saklı birimlerinde en iyi biçimde gizyazarak boyut azaltır. (a) Doğrusal ağda birinci katman gizyazar, ikinciyse gizçözer işlev görür. (b) Gizyazar ve gizçözer saklı birimli çok katmanlı algılayıcılar olduğundan tüm ağ doğrusal olmayan bir biçimde boyut azaltabilir.

Çok katmanlı algılayıcının ne yaptığını ağırlıkları inceleyerek anlayabiliriz. İki yöney eşit olduğunda iç çarpımları en büyük değerini alacağından her saklı birimi, ona gelen ağırlıklarla tanımlanan bir şablon eşleyici gibi düşünebilir ve bu şablonlara bakarak bilgi çıkarabiliriz. Eğer girdiler normalleştirilmişse ağırlıkları bize görece önemlerini belirtir. Böyle bir yorumlama kolay olmasa da ama çok katmanlı algılayıcının ne yaptığı konusunda sezgi kazandırıp kara kutunun içine göz atmamızı sağlayabilir.

Özilişkilendirici (Cottrell, Munro, and Zipser 1987) denilen çok katmanlı ağ yapısında girdiyle çıktı sayısı eşittir ve istenen çıktı girdiyle aynıdır (bkz. şekil 11.19). Girdiyi çıktıda yeniden oluşturabilmek için çok katmanlı ağ saklı birimlerinde girdinin en iyi gösterimini bulmak zorundadır. Eğer saklı birim sayısı girdi sayısından azsa ilk katman boyut azaltır. Öğrenme bittikten sonra girdiden saklı katmana giden ilk katmanın gizyazar bir görevi vardır ve saklı birimlerin değeri bir gizyazı gösterimi oluşturur. Saklı katmandan çıktıya giden ikinci katman gizyazılmış bu gösterimden girdiyi yeniden oluşturarak gizçözer bir görev üstlenir.

Tek saklı katmanlı bir ağın temel bileşen çözümlemesiyle (bölüm 6.3) aynı işi yaptığı gösterilmiştir (Bourlard and Kamp 1988). Fark şudur: Saklı birim ağırlıkları özdeğerlerine göre sıralanmış özyöneyler değildir, ama en önemli H özyöneyle aynı uzayı tanımlar. Eğer gizyazar ve gizçözerler tek katmanlı yerine kendi saklı birimleri olan çok katmanlı ağlar olursa gizyazar doğrusal olmayan bir biçimde boyut azaltır (Hinton ve Salakhutdinov 2006).

Çok katmanlı algılayıcı boyut azaltmada çok boyutlu ölçekleme için de kullanılabilir (bölüm 6.5). Mao ve Jain (1995), çok katmanlı bir ağın *Sammon eşlemesi* öğrenmesinde kullanılabileceğini göstermiştir. 6.29. denklemde Sammon gerilimini tanımlamıştık:

$$E(\theta|\mathcal{X}) = \sum_{r,s} \left[\frac{\|g(x^r|\theta) - g(x^s|\theta)\| - \|x^r - x^s\|}{\|x^r - x^s\|} \right]^2 \tag{11.40}$$

d tane girdisi, H tane saklı birimi ve $k < d$ tane çıktısı olan bir çok katmanlı algılayıcı $g(x|\theta) \in \Re^k$ modelini gerçeklemek için kullanıldığında d boyutlu girdiyi k boyuta indirir; burada θ, ağ ağırlıklarına karşılık gelir. $\mathcal{X} = \{x^t\}_t$ öğrenme kümesi verildiğinde Sammon gerilimini hata işlevi olarak kullanan bir eğim inişle çok katmanlı ağı, yani $g(x|\theta)$ modelini öğrendiğimizde k boyutlu gösterimler arasındaki uzaklıklar ilk uzaydaki uzaklıklara olabildiğince yakın olur.

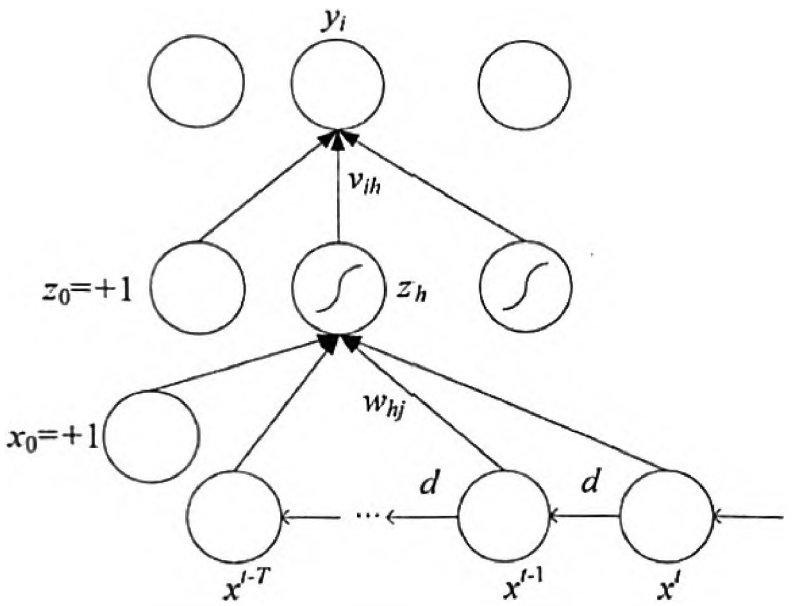
11.12 Zamanı Öğrenmek

Şimdiye dek çok boyutlu girdinin aynı anda, hepsi birlikte gözlendiği uygulamaları gördük. Bazı uygulamalarda girdi zaman içinde sıralıdır ve bu zaman sırasını öğrenmemiz gerekir. Başka uygulamalarda çıktı zaman içinde değişebilir. Örnek uygulamalara bakalım:

- *Dizi tanımada* bir diziyi sınıflardan birine atarız. Örneğin konuşma tanımada girdi sesbirimlerinin bir dizisidir ve sınıflar farklı sözcüklerdir. Girdi zaman içinde değişen bir sıradır ama çıktı anlıktır.
- *Dizi üretiminde* dizinin bir bölümünü gördükten sonra kalanını öngörmemiz istenir. Zaman dizisi öngörüsünde örneğin girdi için çıktı zaman içinde değişir.
- *Zamansal ilişkilendirici* en genel durumdur ve bir girdi dizisine bağlı olarak bir çıktı dizisi oluşturulur. Girdi ve çıktı dizileri farklı olabilir ve hem girdi, hem de çıktı dizisi zaman içinde değişir.

11.12.1 Zaman Gecikmeli Sinir Ağı

Zaman içindeki bir diziyi tanımanın en kolay yolu bekletip uzay içindeki bir diziye çevirmektir. Bu yapıldığında şu ana dek gördüğümüz herhangi bir yöntem sınıflandırma için kullanılabilir. *Zaman gecikmeli sinir ağında* (Waibel ve ötekiler 1989) önceki girdiler geciktirilip en son girdiyle aynı anda bir yöney olarak verilir (bkz. şekil 11.20). Hata geri yayma kullanılarak ağırlıklar güncellenir. Zaman içinde yerel öznitelikleri öğrenmek için yapısal bağlantılar ve zamanda taşımaya karşı değişmezlik için ağırlık paylaşımı kullanılabilir. Bu yapının en önemli kısıtı geçmişi saklayan pencerenin büyüklüğünün sabit olması ve baştan tanımlanmasının gerekmesidir.



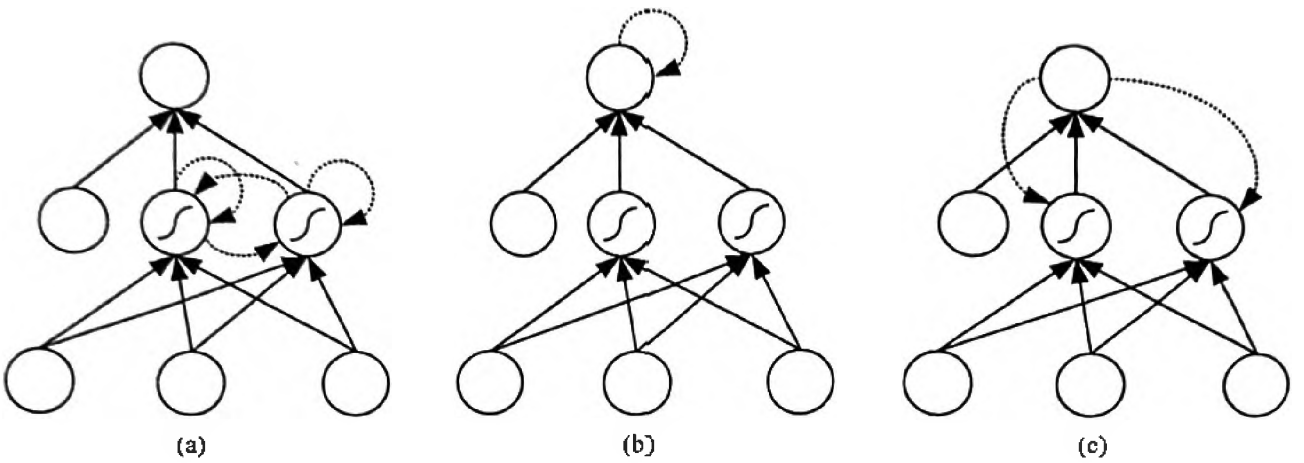
Şekil 11.20: Zaman gecikmeli sinir ağında T boyundaki bir pencere içindeki girdi zaman içinde geciktirilip (d) tüm T değer aynı anda bir algılayıcıya girdi olarak verilir.

11.12.2 Özyineli Ağlar

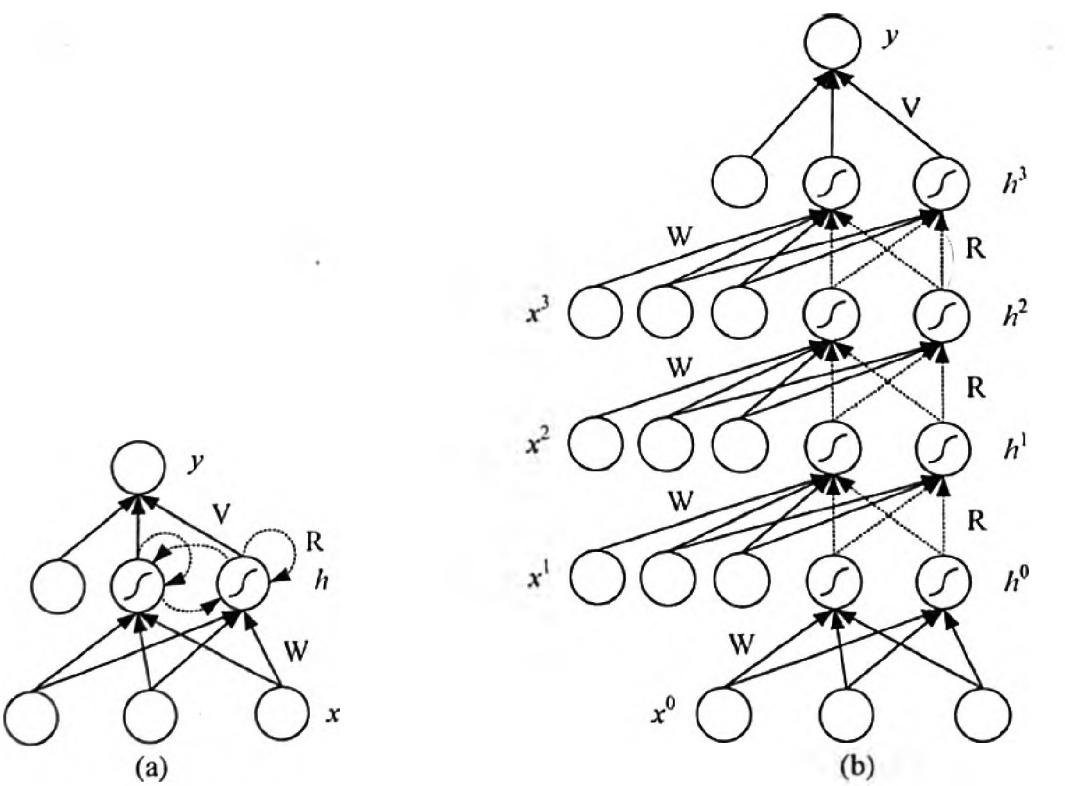
Özyineli ağda, ileriye doğru bağlantılar yanında, birimlerin aynı ya da önceki katmandaki öteki birimlere bağlantıları olabilir. Bu özyineleme kısa soluklu bir bellek gibi davranır ve ağın yakın geçmişte gördüğü veriyi hatırlamasını sağlar.

En sık, çok katmanlı bir ağa kısıtlı sayıda özyineli bağlantı eklenen kısmen özyineli bir yapı kullanılır (bkz. şekil 11.21). Böylece çok katmanlı ağın doğrusal olmayan kestiriciliğiyle özyineli bağlantıların zamansal gösterim becerisi birleştirilir ve her üç tür uygulama da gerçeklenebilir. Geriye doğru giden özyineli bağlantılarda saklı birimler kullanmak da olasıdır, bunlara *bağlam birimleri* denir. Ne yazık ki belli bir uygulama için hangi özyineli ağ yapısının en iyi olduğu konusunda fazla kuramsal çalışma yapılmamıştır.

Eğer diziler çok uzun değilse *zaman içinde açılarak* herhangi bir özyineli ağ eşdeğer özyinesiz, ileriye doğru yayılan bir ağa dönüştürülebilir (bkz. şekil 11.22). Farklı zamanlar için birim ve bağlantıların kopyaları oluşturulur. Böylece oluşan ağı hata geri yayma ile eğittikten sonra ağırlık paylaşımında yaptığımız gibi aynı ağırlık için farklı zamanlardaki güncellemelerin ortalamasını alır ve tek bir güncelleme yaparız. Buna *Zamanda hata geri yayma* denir (Rumelhart, Hinton, ve Willams 1986b). Bu yaklaşımdaki sorun dizinin uzun olması durumunda bellek gereksinimin yüksek olmasıdır. *Gerçek zamanlı özyineli öğrenme* (Williams ve Zipser 1989), özyineli ağı açmadan eğitebilmeyi sağlar ve farklı uzunluklardaki dizileri kullanabilir.



Şekil 11.21: Kısmen özyineli ağ örnekleri. Özyineli bağlantılar kesikli çizgilerle gösterilmiştir: (a) Saklı katmanda, (b) çıktı katmanında, ve (c) çıktıdan saklı katmana doğru özyinelilik. Bunların birleşimleri de kullanılabilir.



Şekil 11.22: Zamanda hata geri yayma: (a) özyineli ağ, ve (b) dört zaman adımı için açılmış eşdeğer ağ.

11.13 Kısa Kısa

Yapay sinir ağları konusundaki araştırma sayısal bilgisayar kadar eskidir. McCulloch ve Pitts (1943), yapay sinir hücresi için ilk matematiksel modeli önermiştir. Rosenblatt (1962), algılayıcı ağını ve bir öğrenme algoritmasını 1962'de önerdi. Ardından, Minsky ve Papert (1969) tek katmanlı algılayıcının sınırlarını, örneğin dışlamalı VEYA'yı öğrenemediğini gösterince o zaman henüz saklı katmanlı ağları eğitmek için bir algoritma olmadığından yapay sinir ağları konusundaki araştırmalar birkaç yer dışında durdu. Yapay sinir ağlarının yeniden doğuşu Hopfield'in makalesiyle oldu (1982). Ardından iki ciltlik PDP kitabı geldi (Rumelhart, McClelland, ve PDP Araştırmacıları 1986). Hata geri yayma hemen hemen aynı zamanlarda farklı yerlerde bağımsız olarak geliştirildi ve bu sayede tek katmanlı algılayıcının sınırları ortadan kalktı.

1980'li yılların ortalarından başlayarak yapay sinir ağı yapılarına fizik, istatistik, psikoloji, bilişsel bilim, sinirbilim, dilbilim, bilgisayar bilimleri, elektronik, ve robotik gibi çok farklı alanlarda bir ilgi doğdu. Belki yapay sinir ağlarının en büyük katkısı farklı alanları, özellikle istatistik ve mühendisliği bir araya getirebilmesidir; bugün yapay öğrenme dediğimiz alan bu birliktelikten ortaya çıktı.

Bu konulardaki araştırma günümüzde daha olgun ve amacı hem daha alçak gönüllü, hem de daha iyi tanımlıdır. Hata geri yayma önerildiğinde eleştirilerden biri biyolojik olarak inandırıcı olmadığıydı! Her ne kadar "yapay sinir ağı" deyimi hâlâ kullanılıyorsa da artık bunları, örneğin çok katmanlı algılayıcıları, dağılımdan bağımsız kestiriciler istatistiksel modeller olarak düşünüyor ve öyle inceliyoruz.

Örneğin, çok katmanlı ağa benzeyen istatistiksel bir modelde (Friedman ve Stuetzle 1981)

$$y = \sum_{h=1}^{H} \phi_h(w_h^T x)$$

her "saklı birimin" ayrı bir etkilenim işlevi $\phi_h(\cdot)$ olabilir; çok katmanlı ağdaki gibi hepsinin S işlevini kullanması gerekmez. 12. bölümde bu işlevin Gauss olduğu dairesel taban işlev ağlarını göreceğiz.

Yapay sinir ağlarına özel ders kitapları yazılmıştır: (Hertz, Krogh, ve Palmer 1991), bunların ilkidir ve hâlâ okunabilir. (Bishop 1995) örüntü tanımada kullanımına yoğunlaşmıştır ve öğrenme için farklı iyileme yöntemlerinden bahseder. (Ripley 1996) konuya daha istatistiksel bir açıdan bakar.

Yapay sinir ağları, örneğin çok katmanlı algılayıcılar, birçok uygulamada başarıyla kullanılmaktadır. Denetim ve robotik dışında konuşma ve görüntü tanımada uygulamaları vardır, ama özellikle iki tanesi bahsetmeye değerdir: Tesauro'nun TD-Gammon programının (Tesauro 1994) pekiştirmeli öğrenme (bölüm 18) kullanarak eğittiği çok katmanlı algılayıcı tavlayı usta derecesinde oynamaktadır. Pomerleau'nun ALVINN adını verdiği yapay sinir ağı modeli, bir şoförü beş dakika izleyip öğrendikten sonra bir minibüsü saatte 30 kilometre hızında kullanabilmektedir (Pomerleau 1991).

11.14 Alıştırmalar

1. Girdinin DEĞİL'ini hesaplayan algılayıcıyı gösterin.
2. İki girdisinin VE-DEĞİL'ini hesaplayan algılayıcıyı gösterin.
3. Üç girdisinin eşliğini hesaplayan algılayıcıyı gösterin.
4. Çok katmanlı algılayıcıda saklı birimler S yerine tanh etkilenim işlevi kullandığında güncelleme kuralları nasıl olur? $\tanh' = (1 - \tanh^2)$ olarak yazılabilir.
5. İki saklı katmanlı çok katmanlı algılayıcı için güncelleme kurallarını türetin.
6. Öyle bir tek saklı katmanlı ağ düşünün ki ek olarak girdi katmanından çıktıya da doğrudan bağlantılar olsun. Böyle bir yapı ne zaman yararlı olur? Nasıl eğitilebilir?
7. Eşlik işlevi kaydırmaya değişmezdir, örneğin, "0101" ve "1010" aynı eşliklidir. Bu ipucunu çok katmanlı bir ağı eğitmede nasıl kullanabilirsiniz?
8. Ardışık ilinti yapısında eski bağlantıları dondurmanın ne faydası vardır?
9. Sammon eşlemesi için kullanılan ve Sammon gerilimini (denklem 11.40) azaltan çok katmanlı ağ için güncelleme kurallarını türetin.
10. 11.6. bölümde iki saklı katmanlı bir ağın parçalı sabit yaklaşıklama gerçeklediğini gördük. Son katmandaki ağ sabit yerine girdinin doğrusal bir işlevi olduğunda böyle bir yapının parçalı doğrusal yaklaşıklama gerçekleyeceğini gösterin.
11. Eşiksiz ağırlık paylaşımı için güncelleme kurallarını türetin.
12. Özilişkilendirici ağında saklı birim sayısını nasıl belirleyebiliriz?
13. Ağ yapısını öğrenmeyi yapı uzayında bir arama olarak gördüğümüzde uygun işlemler neler olabilir? İyilik nasıl ölçülebilir? Hangi arama yordamları uygundur? Böyle bir tanımlamada devingen düğüm yaratma ve ardışık ilinti algoritmalarının özel durumlar olduğunu gösterin.
14. Şekil 11.22'de gösterilmiş açılmış ağ için güncelleme kurallarını türetin.

11.15 Kaynaklar

Abu-Mostafa, Y. 1995. "Hints." *Neural Computation* 7: 639–671.

Aran, O., O. T. Yıldız, E. Alpaydın. 2009. "An Incremental Framework Based on Cross-Validation for Estimating the Architecture of a Multilayer Perceptron." *International Journal of Pattern Recognition and Artificial Intelligence* 23: 159–190.

Ash, T. 1989. "Dynamic Node Creation in Backpropagation Networks." *Connection Science* 1: 365–375.

Battiti, R. 1992. "First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method." *Neural Computation* 4: 141–166.

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition.* Oxford: Oxford University Press.

Bourlard, H., Y. Kamp. 1988. "Auto-Association by Multilayer Perceptrons and Singular Value Decomposition." *Biological Cybernetics* 59: 291–294.

Cottrell, G. W., P. Munro, D. Zipser. 1987. "Learning Internal Representations from Gray-Scale Images: An Example of Extensional Programming." *Ninth Annual Conference of the Cognitive Science Society*, 462–473. Hillsdale, NJ: Erlbaum.

Durbin, R., D. E. Rumelhart. 1989. "Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks." *Neural Computation* 1: 133–142.

Fahlman, S. E., C. Lebiere. 1990. "The Cascade Correlation Architecture." *Advances in Neural Information Processing Systems 2*, ed. D. S. Touretzky, 524–532. San Francisco: Morgan Kaufmann.

Friedman, J. H., and W. Stuetzle. 1981. "Projection Pursuit Regression." *Journal of the American Statistical Association* 76: 817–823.

Geman, S., E. Bienenstock, R. Doursat. 1992. "Neural Networks and the Bias/Variance Dilemma." *Neural Computation* 4: 1–58.

Hertz, J., A. Krogh, R. G. Palmer. 1991. *Introduction to the Theory of Neural Computation*. Reading, MA: Addison Wesley.

Hinton, G. E., R. R. Salakhutdinov. 2006. "Reducing the dimensionality of data with neural networks." *Science* 313: 504–507.

Hopfield, J. J. 1982. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities." *Proceedings of the National Academy of Sciences USA* 79: 2554–2558.

Hornik, K., M. Stinchcombe, H. White. 1989. "Multilayer Feedforward Networks Are Universal Approximators." *Neural Networks* 2: 359–366.

Le Cun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel. 1989. "Backpropagation Applied to Handwritten Zipcode Recognition." *Neural Computation* 1: 541–551.

MacKay, D. J. C. 1992a. "Bayesian Interpolation." *Neural Computation* 4: 415–447.

MacKay, D. J. C. 1992b. "A Practical Bayesian Framework for Backpropagation Networks" *Neural Computation* 4: 448–472.

Mao, J., A. K. Jain. 1995. "Artificial Neural Networks for Feature Extraction and Multivariate Data Projection." *IEEE Transactions on Neural Networks* 6: 296–317.

Marr, D. 1982. *Vision*. New York: Freeman.

McCulloch, W. S., W. Pitts. 1943. "A Logical Calculus of the Ideas Immenent in Nervous Activity." *Bulletin of Mathematical Biophysics* 5: 115–133.

Minsky, M. L., S. A. Papert. 1969. *Perceptrons*. Cambridge, MA: MIT Press. (Expanded ed. 1990.)

Nowlan, S. J., G. E. Hinton. 1992. "Simplifying Neural Networks by Soft Weight Sharing." *Neural Computation* 4: 473–493.

Pomerleau, D. A. 1991. "Efficient Training of Artificial Neural Networks for Autonomous Navigation." *Neural Computation* 3: 88–97.

Posner, M. I., ed. 1989. *Foundations of Cognitive Science*. Cambridge, MA: MIT Press.

Richard, M. D., R. P. Lippmann. 1991. "Neural Network Classifiers Estimate Bayesian *a Posteriori* Probabilities." *Neural Computation* 3: 461–483.

Ripley, B. D. 1996. *Pattern Recognition and Neural Networks.* Cambridge, UK: Cambridge University Press.

Rosenblatt, F. 1962. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* New York: Spartan.

Rumelhart, D. E., G. E. Hinton, R. J. Williams. 1986a. "Learning Representations by Backpropagating Errors." *Nature* 323: 533–536.

Rumelhart, D. E., G. E. Hinton, R. J. Williams. 1986b. "Learning Internal Representations by Error Propagation." *Parallel Distributed Processing,* ed. D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, 318–362. Cambridge, MA: MIT Press.

Rumelhart, D. E., J. L. McClelland, and the PDP Research Group, eds. 1986. *Parallel Distributed Processing.* Cambridge, MA: MIT Press.

Simard, P., B. Victorri, Y, Le Cun, J. Denker. 1992. "Tangent Prop: A Formalism for Specifying Selected Invariances in an Adaptive Network." *Advances in Neural Information Processing Systems 4,* ed. J. E. Moody, S. J. Hanson, R. P. Lippman, 895–903. San Francisco: Morgan Kaufmann.

Tesauro, G. 1994. "TD-Gammon, A Self-Teaching Backgammon Program, Achieves Master-Level Play." *Neural Computation* 6: 215–219.

Thagard, P. 2005. *Mind: Introduction to Cognitive Science.* 2nd ed. Cambridge, MA: MIT Press.

Waibel, A., T. Hanazawa, G. Hinton, K. Shikano, K. Lang. 1989. "Phoneme Recognition Using Time-Delay Neural Networks." *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37: 328–339.

Williams, R. J., D. Zipser. 1989. "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks." *Neural Computation* 1: 270–280.

Bölüm 12

Yerel Modeller

Saklı birimleri girdi uzayının yerel bir alanı içindeki örneklere tepki veren çok katmanlı yapay sinir ağ yapılarını tartışacağız. Saklı katmandan sonraki ikinci katmanda bu yerel alanlara bağlı olarak sınıflandırma ya da bağlanım gerçeklenebilir. Bu bölümde hem böylesi yerel alanları bulmayı, hem de oralardan sorumlu yerel modelleri öğrenmeyi konuşacağız.

12.1 Giriş

İşlev yaklaşıklamanın bir yolu da girdi uzayını yerel bölgelere ayırıp her bölgeye ayrı bir model oturtmaktır. Bölüm 7'de gördüğümüz öbekleme yöntemleri girdi örneklerini bir araya getirerek girdi dağılımını modeller. Çevrimiçi öbekleme için yarışmacı yapay sinir ağları da kullanılabilir. Bu bölümde, k merkezli öbekleme algoritmasının çevrimiçi sürümünü ve buna dayanan iki yapay sinir ağı yöntemini, artırımlı çevrimiçi öbekleme ve özdüzenlemeli haritaları tartışacağız.

Girdilerin yerelleştirilmesi arkasından gözetimli öğrenme yapılabilir. Eğer yerel bölgelere oturtulan model sabitse bu yönteme dairesel tabanlı işlev ağı, eğer girdinin doğrusal bir işleviyse uzmanların birleşimi diyoruz. Böylesi modellerle sınıflandırma ve bağlanımın nasıl gerçeklendiğini, ve 11. bölümde gördüğümüz S saklı birimli ağlardan farklarını tartışacağız.

12.2 Yarışmacı Öğrenme

Bölüm 7'de, girdilerin k tane Gauss kaynağının birinden geldiğini varsayan dağılıma yarı bağlı karışım modelini kullandık. Bu bölümde yine aynı şekilde veride k öbek olduğu varsayımını koruyacağız, ama yaklaşımımız olasılıksal olmayacak ve kaynakların dağılımı için olasılıksal bir model beklemeyeceğiz. Bir başka fark da bu bölümde tartışacağımız öğrenme yöntemlerinin çevrimiçi olmasıdır. Öğrenme sırasında elimizde tüm öğrenme kümesi olmayabilir; bu durumda örnekler birer birer geldikçe

model parametrelerini ayarlamamız gerekecek. Sanki bu öbekler girdiyi betimlemek için kendi aralarında yarışma yaparlarmış gibi *yarışmacı öğrenme* deyimini kullanacağız. Sanki öbekler yarışıyor, biri kazanıyor ve yalnızca kazananın parametreleri güncelleniyor, diye düşünüldüğünden böylesi yaklaşıma bazen "Kazanan hepsini alır," denir.

Bu yöntemler kendi başlarına, 7. bölümde tartıştığımız toptan yöntemlerden farklı olarak çevrimiçi öbekleme için kullanılabilirler. Çevrimiçi yöntemlerin bazı üstünlükleri vardır: (1) Tüm öğrenme kümesini saklamak için bellek ayırmak gerekmez; (2) Her örnekte yapılan güncellemenin donanım ya da yazılım gerçeklemesi daha basittir; (3) Girdi dağılımı zaman içinde değiştiğinde model kendisini bu değişikliğe kolayca uyarlayabilir. Toptan bir öğrenme yöntemi kullanıyor olsak yeni bir öğrenme kümesi toplamamız ve toptan yöntemi sıfırdan tekrar çalıştırmamız gerekir.

Bölüm 12.3'ten başlayarak böylesi bir öbekleme ardından gözetimli sınıflandırma ya da bağlanımın nasıl gerçeklenebileceğiniz konuşacağız. Bu iki aşamalı öğrenmeyi iki katmanlı bir ağ olarak da düşünebiliriz; öyle ki, birinci aşama (ya da katman), girdi dağılımını modelleyip sorumlu yerel modeli bulurken, ikincisi de sorumlu yerel model olarak çıktıyı üretir.

12.2.1 k Merkezli Çevrimiçi Öbekleme

Denklem 7.3'te geriçatma hatasını tanımlamıştık:

$$E(\{\boldsymbol{m}_i\}_{i=1}^k|\mathcal{X}) = \frac{1}{2}\sum_t\sum_i b_i^t\|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2 \tag{12.1}$$

Burada, $\mathcal{X} = \{\boldsymbol{x}^t\}_t$ öğrenme kümesini ve $\boldsymbol{m}_i, i = 1,\ldots,k$, öbek merkezlerini göstermektedir. Eğer $\boldsymbol{m}_i$ merkezi, $\boldsymbol{x}^t$ girdisine Euclid uzaklığı olarak en yakınsa girdiden o sorumludur; bunu göstermek için b_i^t, 1 değerini, değilse 0 değerini alır:

$$b_i^t = \begin{cases} 1 & \text{eğer } \|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_l \|\boldsymbol{x}^t - \boldsymbol{m}_l\| \\ 0 & \text{değilse} \end{cases} \tag{12.2}$$

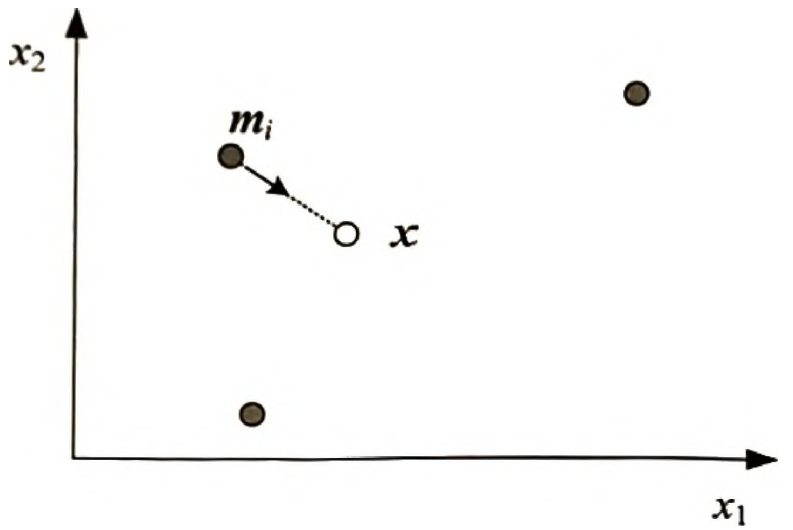
Sanki bütün $\boldsymbol{m}_l, l = 1,\ldots,k$ öbek merkezleri girdi için yarışmakta ve en yakın olduğu için yarışmayı $\boldsymbol{m}_i$ kazanmaktadır.

Toptan k merkez algoritmasında, merkezler sorumlu (en yakın) oldukları örneklerin ortalama değerini alacak biçimde güncellenirler:

$$\boldsymbol{m}_i = \frac{\sum_t b_i^t \boldsymbol{x}^t}{\sum_t b_i^t} \tag{12.3}$$

Kazananlar denklem 12.2'yle seçilir ve bu güncelleme, denklem 12.1'de tanımlanan hatayı küçültür. b_i^t değerlerinin hesaplanması ve ardından $\boldsymbol{m}_i$ merkezlerinin güncellenmesinden oluşan bu iki adım yakınsama olana dek tekrarlanır.

Eğer elimizde zaten toplu bir öğrenme kümesi varsa *çevrimiçi k merkezli öbekleme* algoritmasıyla eğim inişi kullanılarak örnekleri rastgele teker teker alıp (daha önceki



Şekil 12.1: Gölgeli çemberler merkezleri, boş çember girdiyi göstermektedir. k merkezli öbeklemenin çevrimiçi sürümü en yakın merkezi girdiye doğru, $(\boldsymbol{x} - \boldsymbol{m}_i)$ yönünde, η çarpanı oranında hareket ettirir.

güncellemeleri unutmamak için her biri için küçük) bir güncelleme yaparak öğrenme gerçeklenebilir. Geriçatma hatasını tek bir örnek için tanımlayalım:

$$E^t(\{\boldsymbol{m}_i\}_{i=1}^k | \boldsymbol{x}^t) = \frac{1}{2}\sum_i b_i^t \|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2 = \frac{1}{2}\sum_i \sum_{j=1}^d b_i^t (x_j^t - m_{ij})^2 \tag{12.4}$$

Burada b_i^t değeri, denklem 12.2'deki gibi tanımlanmıştır. Bu hata işlevi üzerinde eğim inişi yaparak $\boldsymbol{x}^t$ girdisi için bir güncelleme kuralı çıkarabiliriz:

$$\Delta m_{ij} = -\eta \frac{\partial E^t}{\partial m_{ij}} = \eta b_i^t (x_j^t - m_{ij}) \tag{12.5}$$

Böylesi bir güncelleme, en yakın (b_i^t değeri 1 olan) merkezi girdiye η çarpanı oranında yaklaştırır. Öteki merkezlerin, $b_l^t, l \neq i$, değerleri 0 olduğu için güncellenmezler (bkz. şekil 12.1). İstersek denklem 12.5'teki güncellemeyi tüm t üzerinden toplayarak toptan bir algoritma da oluşturabiliriz. Bir momentum terimi de eklenebilir. Yakınsamayı sağlamak için η yavaşça sıfıra yaklaştırılır, ama bu, *kararlılık-yoğrukluk ikilemi* durumunu ortaya çıkarır: η değeri sıfıra yaklaştıkça model kararlı olur ama güncelleme değerleri çok küçüleceği için farklı yeni örneklere uyarlanabilirlik azalır; η değerini yüksek tutarsak $\boldsymbol{m}_i$ değerleri bir oraya bir buraya dalgalanabilir.

k merkezli çevrimiçi öbekleme algoritmasının örnek programı şekil 12.2'de verilmiştir. Bu, şekil 7.3'teki toptan algoritmanın çevrimiçi sürümüdür.

Böylesi bir yarışma şekil 12.3'teki gibi tek katmanı özyineli bir yapay sinir ağı ile de gerçeklenebilir. Girdi katmanı, $\boldsymbol{x}$ girdisini gösterir (ek girdi olmadan). Çıktı birimleri ise b_i değerlerine karşılık gelir ve birer algılayıcıyla hesaplanırlar:

$$b_i = \boldsymbol{m}_i^T \boldsymbol{x} \tag{12.6}$$

Merkez $\boldsymbol{m}_i, i = 1, \ldots, k$, değerlerini başlat; örneğin, k rastgele $\boldsymbol{x}^t$ değeriyle
Tekrar
  Tüm $\boldsymbol{x}^t \in \mathcal{X}$ için, rastgele sırayla
    $i \leftarrow \arg\min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\|$
    $\boldsymbol{m}_i \leftarrow \boldsymbol{m}_i + \eta(\boldsymbol{x}^t - \boldsymbol{m}_i)$
$\boldsymbol{m}_i$ değerleri yakınsayana dek

Şekil 12.2: Çevrimiçi k merkezli öbekleme algoritması. Toptan sürümü şekil 7.3'te verilmiştir.

b_i değerlerinin en büyüğünü 1, ötekileri 0 yapmak için gereken *yanal ketleme* özyineli bir sinir ağıyla gerçeklenebilir. Şekil 12.3'te görüldüğü gibi, her birim kendisine artı, öteki birimlere eksi ağırlıklarla bağlanır. Uygun bir doğrusal olmayan etkilenim işlevi ve uygun artı ve eksi özyineli ağırlıklarla, böylesi bir ağ, yinelemelerin ardından, en büyüğün 1 ötekilerin 0 değerini aldığı duruma yakınsayacaktır (Grossberg 1980; Feldman ve Ballard 1982).

Denklem 12.6'da kullanılan iç çarpım, bir benzerlik ölçütüdür ve bölüm 5.5'de (denklem 5.26) gördüğümüz gibi, eğer $\boldsymbol{m}_i$ yöneyleri yaklaşık aynı büyüklükte olurlarsa, girdiye Euclid uzaklığı ($\|\boldsymbol{m}_i - \boldsymbol{x}\|$) en küçük olan merkezin girdiyle iç çarpımı ($\boldsymbol{m}_i^T \boldsymbol{x}$) da en büyük olacaktır.

Bu kısımda ve ileride öteki yarışmacı yöntemlerden bahsederken Euclid uzaklığının hep girdilerin aynı değişkenlikte ve ilintisiz olmaları durumunu varsaydığını unutmamamız gerekir. Eğer bu koşul sağlanmıyorsa, ya uzaklık ölçütü uygun şekilde değiştirilmeli (örneğin, Euclid yerine Mahalanobis uzaklığı kullanarak), ya da öncesinde bir normalleştirme yapılmalıdır (örneğin, temel bileşen çözümlemesi kullanarak).

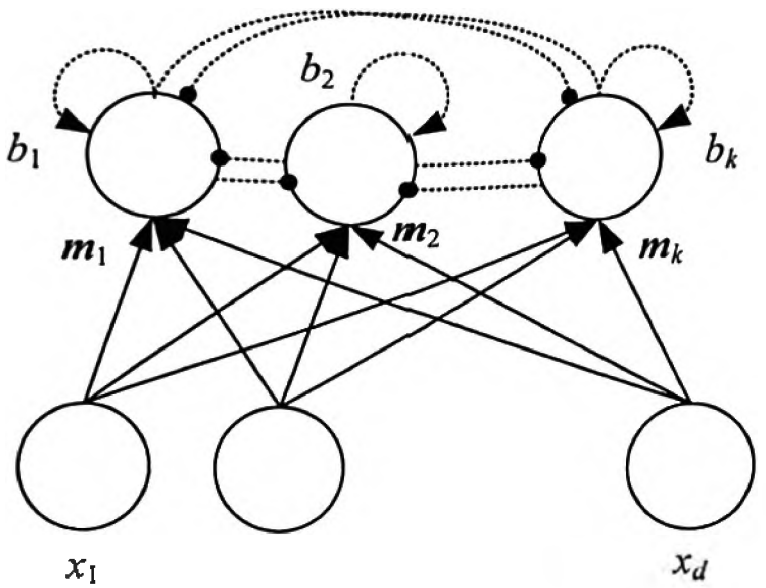
Denklem 12.5'i şu biçimde de yazabiliriz:

$$\Delta m_{ij}^t = \eta b_i^t x_j^t - \eta b_i^t m_{ij} \tag{12.7}$$

m_{ij} değerinin x_j girdisinden b_i çıktısına giden bağlantının ağırlığı olduğunu hatırlayalım. Birinci terimdeki gibi bir güncelleme

$$\Delta m_{ij}^t = \eta b_i^t x_j^t \tag{12.8}$$

Hebbci öğrenmedir ve güncellemeyi bağlantının önündeki ve sonundaki birimlerin çarpımı cinsinden tanımlar. Bir sinir bağlantısı, bağlantının iki ucundaki sinir hücreleri aynı anda etkin olduklarında, yani aralarında bir ilinti olduğu anlaşılınca, daha önemli olur. Ama yalnız Hebbci öğrenme kullanıldığında ağırlıkların değeri sürekli artacaktır ve güncellenmeyen ağırlıkların değerlerini azaltacak ikinci bir terim eklemek gerekir. Bunun için, ya her örnekten sonra yeniden $\|\boldsymbol{m}_i\| = 1$ olacak şekilde normalleştirme yapılır, ya da bir ağırlık sönüm terimi (Oja 1982) eklenir, ki denklem 12.7'deki ikinci terim bu biçimde yorumlanabilir. Hertz, Krogh, ve Palmer (1991), yarışmacı ağları ve Hebbci öğrenmeyi daha ayrıntılı anlatır ve böylesi ağların, örneğin



Şekil 12.3: "Kazanan hepsini alır," ilkesini gerçekleyen yarışmacı yapay sinir ağı yapısında çıktılarında özyineli yanal ağırlıklar bulunan k tane algılayıcı vardır. Kesikli çizgiler bu yanal bağlantıları gösterir; bunların sonunda ok olanları artı, çember olanları eksi ağırlıklıdır. Her çıktı kendi değerini artırırken ötekileri ketlemeye çalışır. Bu bağlantılara verilecek uygun ağırlık değerleriyle en yüksek değeri olan ötekileri sıfırlar ve böylece girdiye en yakın olan merkezin b_i değeri 1, ötekilerin, $b_l, l \neq i$, ise 0 olur.

temel bileşen çözümlemesini gerçekleyebileceğini gösterir. Mao ve Jain (1995), temel bileşen çözümlemesi ve doğrusal ayırtaç çözümlemesi için çevrimiçi algoritmaları tartışır.

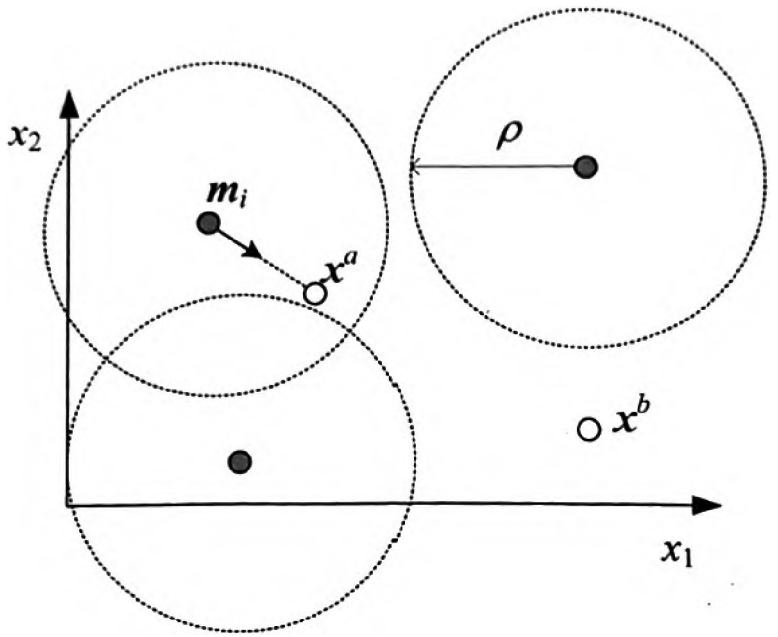
Bölüm 7'de gördüğümüz gibi, bir sorun işe yaramayan merkezlerin oluşabilmesidir. Yarışmacı ağlarda bu hiçbir zaman yarışmayı kazanamayan merkezler demektir. Çünkü büyük olasılıkla bunlar girdilerden çok uzak yerlerden başlatılmışlardır. Bunu engellemek için farklı yaklaşımlar vardır:

1. m_i merkezleri rastgele seçilmiş örnekleri ilk değer olarak alabilirler, böylece girdi uzayının veri bulunan bölümlerinden başlarlar.

2. Artımlı, birimlerin modele gerektikçe eklendiği bir algoritma kullanabiliriz; böylece yeni merkez her zaman gerektiği yerden başlatılır. Buna bir örnek olarak bölüm 12.2.2'de artımlı bir çevrimiçi öbekleme yönteminden bahsedeceğiz.

3. Güncelleme yaptığımızda en yakın merkezle birlikte bazı öbür merkezleri de girdiye yaklaştırırsak onlar da girdi uzayının veri olan bölümlerine yaklaşacak ve bir zaman sonra kendi başlarına yarışmayı kazanabileceklerdir. Buna bir örnek olarak bölüm 12.2.3'te özörgütlemeli haritaları göreceğiz.

4. Bir başka olasılık ise merkezlere bir *vicdan* parametresi eklemektir (DeSieno



Şekil 12.4: x^a örneğiyle en yakın merkez arasındaki uzaklık, ρ tetiklik parametresinden daha düşük olduğu için güncelleme k merkez algoritmasında olduğu gibi yapılır. x^b örneğiyse hiçbir merkeze yeterince yakın olmadığından o noktada yeni bir merkez oluşturulur.

1988): Yarışmayı sık kazanan merkez kendini suçlu hisseder ve sonraki girdilerde başkalarının kazanmasına izin verir.

12.2.2 Artımlı Çevrimiçi Öbekleme

Öbeklemeye başlamadan önce öbek sayısı olan k değerinin belirlenmiş olması gerekir. Bir başka olasılıksa tek bir merkezle başlayıp gerektikçe yeni merkezlerin eklendiği *artımlı* yaklaşımdır (Carpenter ve Grossberg 1988). Bu algoritmada girdi verildiğinde önce tüm çıktılar kendi değerini hesaplar ve girdiye en benzer olan seçilir. Eğer denklem 12.6'daki gibi iç çarpım kullanılıyorsa bu değeri en büyük olan, Euclid uzaklığı kullanılıyorsa da değeri en düşük olan birimdir.

Euclid uzaklığını kullandığımızı varsayalım. Bu durumda eğer en yakın merkezin girdi örneğine olan uzaklığı *tetiklik* parametresi dediğimiz bir eşik değerinden küçükse güncelleme k merkezli çevrimiçi öbeklemedeki gibi yapılır, yani en yakın merkez girdiye doğru biraz çekilir. Eğer en yakın merkezin uzaklığı tetiklik değerinden büyükse, yani en yakın merkez girdiye pek yakın değilse, o merkezin değeriyle oynamak yerine girdi noktasında yeni bir merkez oluşturulur. Bu şekilde her merkez, yarıçapı tetiklik değerine eşit olan bir üstünküreyi kapsar ve hiçbir merkez tarafından kapsanmayan bir girdi için yeni bir merkez oluşturulur (bkz. şekil 12.4).

Tetiklik değerini ρ ile gösterirsek güncelleme aşağıdaki gibidir:

$$b_i = \|m_i - x^t\| = \min_{l=1}^{k} \|m_l - x^t\| \tag{12.9}$$

$$\begin{cases} m_{k+1} \leftarrow x^t & \text{eğer } b_i > \rho \\ \Delta m_i = \eta(x^t - m_i) & \text{değilse} \end{cases}$$

Uzaklık için bir eşik değeri belirlemek, girdi başına geriçatma hatası için bir eşik değeri belirlemeye denktir ve eğer uzaklık ölçütü Euclid, hata denklem 12.4'teki gibi tanımlıysa, girdi başına izin verilen en büyük geriçatma hatası tetiklik değerinin karesine eşittir.

12.2.3 Özörgütlemeli Haritalar

İşe yaramayan birimlerin önüne geçmenin bir yolu, yalnızca kazananı değil, öteki birimlerden bazılarını da güncelleyip girdiye doğru çekmektir. Kohonen (1990, 1995) tarafından önerilen *özörgütlemeli harita* algoritmasında, i birim dizini bir *komşuluk* tanımlar ve en yakın merkez m_i olduğunda, ona ek olarak komşuları da güncellenir. Örneğin, komşuluk büyüklüğü 2 ise, $m_{i-2}, m_{i-1}, m_{i+1}, m_{i+2}$ birimleri de, dizin değeri olarak uzaklaştıkça biraz daha az olacak biçimde güncellenir. En yakın merkezin dizin değerini i ise merkezlerin güncellenmesi şöyle yapılır:

$$\Delta m_l = \eta\, e(l, i)(x^t - m_l) \tag{12.10}$$

Burada $e(l, i)$ *komşuluk işlevidir*: $l = i$ ise $e(l, i) = 1$ ve $|l - i|$ arttıkça $e(l, i)$'nin değeri, örneğin bir Gauss eğrisi $\mathcal{N}(i, \sigma)$ biçiminde, azalır:

$$e(l, i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(l-i)^2}{2\sigma^2}\right] \tag{12.11}$$

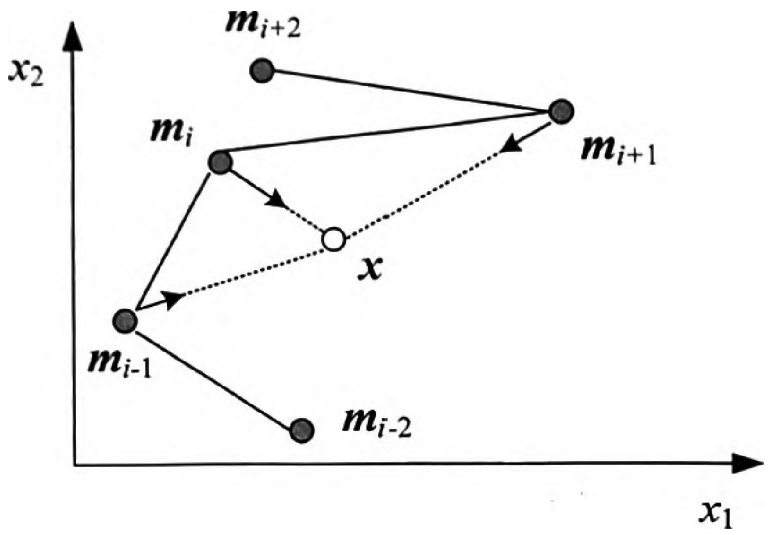
Yakınsama için öğrenme sırasında komşuluk işlevinin genişliği yavaşça azaltılır; örneğin Gauss eğrisi kullanılıyorsa bu σ değerinin azaltılmasına karşılık gelir. Öyle ki en sonda asıl k merkezli öbeklemedeki gibi yalnızca en yakın birim güncellenir.

Komşu birimler de girdiye doğru çekildiğinden işe yaramayan birimler engellenmiş olur, çünkü ilk başta komşularından yardım alarak girdi olan bölgelere yaklaşıp ardından onlar da kendi başlarına yarışmayı kazanmaya ve en yakın olmaya başlarlar (bkz. şekil 12.5).

Merkezler rastgele olarak uzayın çok farklı yerlerinden başlasalar bile, komşular birlikte güncellendikleri için, aynı girdilere doğru birlikte çekildikleri için zaman içinde birbirlerine benzeyecekler ve komşu dizinli merkezler girdi uzayında da komşu olacaklardır.

Çoğu uygulamada, birimler iki boyutlu bir *harita* oluşturacak şekilde tanımlanır. Böyle bir durumda her birimin iki dizini vardır, örneğin, $m_{i,j}$, ve komşulukta her iki dizin boyutu da kullanılır. Eğer $m_{i,j}$ en yakın merkez ise, merkezler iki boyutlu komşuluğa göre güncellenir:

$$\Delta m_{k,l} = \eta e(k, l, i, j)(x^t - m_{k,l}) \tag{12.12}$$



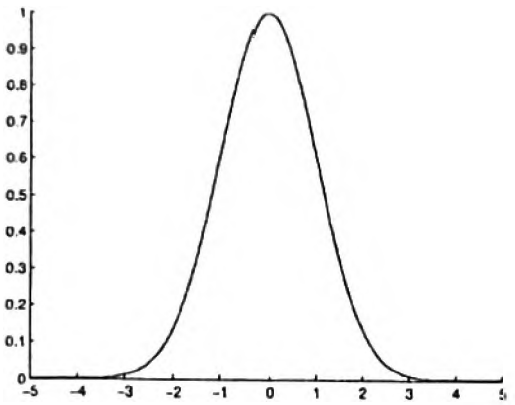
Şekil 12.5: Özörgütlemeli haritalar oluştururken yalnızca en yakın merkez değil, onun dizin cinsinden komşuları da girdiye doğru çekilir. Bu örnekte komşuluk büyüklüğü 1 olarak alınmıştır, ve m_i ile birlikte onun birinci en yakın komşuları da güncellenir. Dikkat edilmesi gereken nokta şudur: m_{i+1} ile m_i başta girdi uzayında çok uzak yerlerden başlasalar da, ikisi birlikte güncellendiklerinden zamanla birbirlerine benzeyecekler ve girdi uzayında komşu olacaklardır.

Burada komşuluk işlevi $e(k, l, i, j)$ iki boyutludur ve (k, l) ile (i, j) arasındaki uzaklığı ölçer. Bunun için örneğin, iki boyutlu bir Gauss eğrisi kullanılabilir. Yakınsama olduğunda d boyutlu girdinin iki boyutlu bir *yerbetimsel haritası* oluşmuş olur. Girdi uzayında örneklerin yoğun olduğu bölge için haritada çok merkez ayrılmış olmasını ve girdi olmayan bölgeler için merkez ayrılmamasını bekleriz. Bu bağlamda baktığımızda haritayı d boyutlu uzayın iki (i, j) boyutlarında doğrusal olmayan bir çokboyutlu ölçeklemesi olarak yorumlayabiliriz. Benzer biçimde bir boyutlu bir harita merkezleri girdi uzayındaki en yoğun bölgelerden geçen bir *temel eğri* tanımlar.

12.3 Dairesel Tabanlı İşlev Ağları

11. bölümde gördüğümüz, saklı birimleri S işlevi ve iç çarpım kullanan çok katmanlı algılayıcılarda her saklı birim bir üstündüzlem tanımlar ve doğrusal olmayan S işleviyle girdinin bu üstündüzleme olan uzaklığına bağlı olarak 0 ile 1 arasında bir değer alır. Her üstündüzlem, girdi uzayını iki parçaya böler ve çoğu zaman bir girdi için saklı birimlerin çoğu sıfırdan farklı bir değer alır. Buna *dağıtık gösterim* diyoruz, çünkü böyle bir gösterimde her girdi, birçok saklı birimin değerlerinin birleşimiyle tanımlanır.

Bir başka olasılık olarak *yerel gösterimde*, her girdi için yalnızca bir ya da çok az sayıda birim etkindir (sıfırdan farklı değer alır). Sanki bu *yerel birimler* girdi uzayını kendi aralarında paylaşıyor ve yalnızca belirli girdilere tepki veriyorlardır.



Şekil 12.6: Dairesel tabanlı işlev ağında kullanılan etkilenim işlevi tek boyutta çan eğrisi biçimindedir. Burada $m = 0$ ve $s = 1$ alınmıştır. Bu eğri, Gauss dağılımı biçimindedir ama integralinin 1 olması gerekmez. Hızlı hesaplama için $(m - 3s, m + 3s)$ dışında 0 değeri aldığı kabul edilebilir.

Girdi uzayında bir birimin sıfırdan farklı çıktı verdiği bölgeye onun *algı alanı* diyoruz. Farklı bölgelerine böylesi yerel birimler yerleştirilerek girdi uzayı bu birimler tarafından kaplanır.

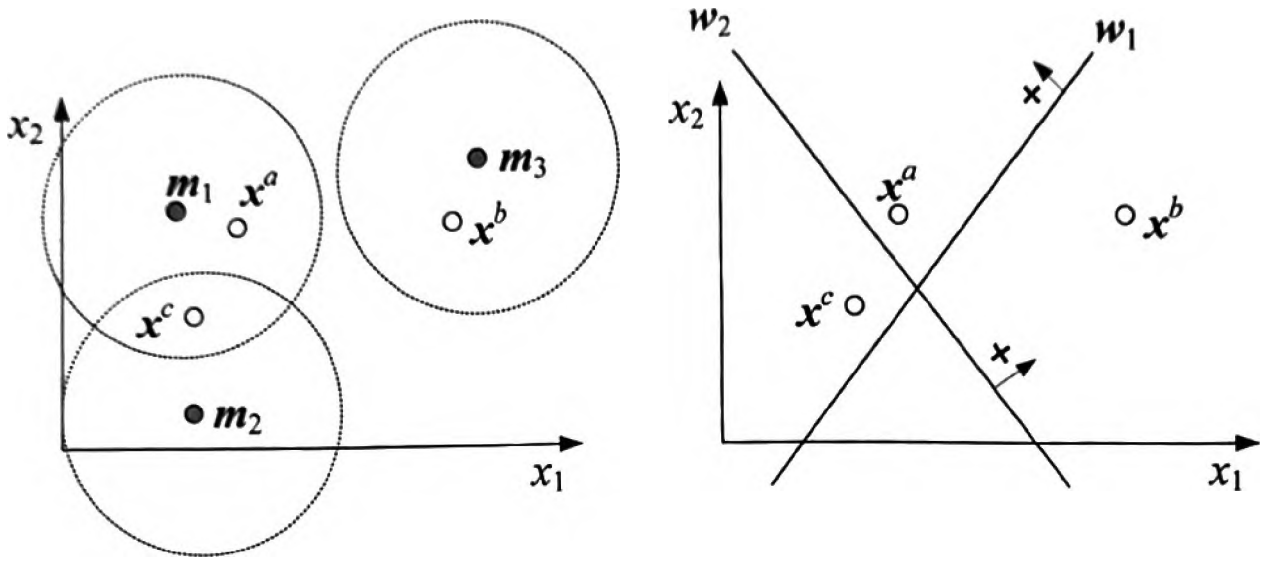
Beyin kabuğunun farklı kısımlarında böyle yerel tepki özellikleri olan sinir hücreleri bulunur. Örneğin, görme bölgesindeki hücreler, hem retina üzerindeki konum, hem de açı olarak yerel dürtülere seçiçi olarak tepki verir. Böylesi yerel ayarlanmış hücreler tepki verdikleri değişkenlere bağlı olarak konumlandıkları yerbetimsel haritalar oluşturur.

Yerellik kavramı, $\boldsymbol{x}$ girdi yöneyiyle h biriminin konumunu gösteren $\boldsymbol{m}_h$ yöneyi arasında bir uzaklık ölçütü tanımını gerektirir. Çoklukla bu Euclid uzaklığı, $\|\boldsymbol{x} - \boldsymbol{m}_h\|$, olarak alınır. Tepki işlevi de $\boldsymbol{x} = \boldsymbol{m}_h$ olduğunda en büyük değeri alır ve uzaklık arttıkça (benzerlik azaldıkça) azalır. En yaygın olarak Gauss eğrisi kullanılır (bkz. şekil 12.6):

$$p_h^t = \exp\left[-\frac{\|\boldsymbol{x}^t - \boldsymbol{m}_h\|^2}{2s_h^2}\right] \tag{12.13}$$

Tabii ki bu bir dağılım değildir, ama çan eğrisi şeklinde olduğu için yine aynı Gauss adı kullanılır. $\boldsymbol{m}_j$ ve s_j j yerel biriminin sırasıyla merkezini ve yayılımını belirler ve bu dairesel bir taban işlevine karşılık gelir. Karmaşıklığı artırarak farklı boyutlarda farklı yayılımlara ve ilintili girdilere izin veren Mahalanobis uzaklığı kullanan eliptik bir taban işlevi de gerçeklenebilir (bkz. alıştırma 2).

Verideki örneklerin öbekler oluşturduğunu varsayıyoruz. Her h öbeği, yalnızca öbeğine düşen örneklerde sıfırdan farklı bir p_h^t değer alır. $\boldsymbol{m}_h$ ile gösterdiğimiz öbek merkezlerini bölüm 12.2'deki çevrimiçi öbekleme yöntemlerinden herhangi biriyle hesaplayabiliriz. Yayılımları bulmak için çok basit sezgisel bir yöntem şudur: Her merkez için sorumlu olduğu girdilerden en uzağını bulur ve yayılımı ona olan uzaklığın yarısı olarak alırız. Ya da beklenti büyütme gibi istatistiksel bir öbekleme algoritması (bölüm 7) kullanıp merkezlerle (ortalamalar) birlikte yayılımları da (değişintileri ve



(p_1, p_2, p_3)

x^a: (1.0, 0.0, 0.0)
x^b: (0.0, 0.0, 1.0)
x^c: (1.0, 1.0, 0.0)

(h_1, h_2)

x^a: (1.0, 1.0)
x^b: (0.0, 1.0)
x^c: (1.0, 0.0)

Şekil 12.7: Solda yerel ve sağda dağıtık gösterimler arasındaki fark. Buradaki değerler 0/1 alınmıştır. Daha çok bilgi veren bir gösterim için uzaklığa bağlı olarak (0, 1) aralığında değişen değerler de kullanılabilir. Yerel gösterimde Gauss eğrisi merkeze olan Euclid uzaklığını, dağıtık gösterimde de S eğrisi üstündüzlem ağırlıklarıyla iç çarpım değerini kullanır.

istersek eş değişintileri) hesaplayabiliriz.

Saklı $p_h^t, h = 1, \ldots, H$ birimleri, H boyutlu yeni bir uzay tanımlar ve $\boldsymbol{x}^t$ girdisi için yeni bir gösterim oluşturur. Girdiyi göstermek için b_h^t (denklem 12.2) değerlerini de kullanabilirdik ama b_h^t değerleri 0/1 olduğu için bilgi kaybına neden olur; p_h^t değerleri girdinin uzaklığını 0 ile 1 arasındaki bir değere çevirir. Uzaklık artarken p_h^t değerinin ne kadar hızlı sıfıra yaklaşacağı s_h parametresine bağlıdır. Şekil 12.7'de böylesi bir *yerel gösterime* örnek verilmekte ve S işlevli saklı birimli çok katmanlı algılayıcılarda kullanılan *dağıtık gösterimle* karşılaştırılmaktadır. Gauss eğrileri yerel olduğundan genelde dağıtık gösterim için gerekenden daha fazla saklı birim kullanmak gerekir, özellikle de girdinin boyut sayısı fazlaysa.

Gözetimli öğrenmede bu yerel gösterimi girdi olarak kullanabiliriz. Bir algılayıcı kullanırsak çıktı, H tane yerel değerin ağırlıklı toplamı olarak yazılır:

$$y^t = \sum_{h=1}^{H} w_h p_h^t + w_0 \tag{12.14}$$

Bu yapıya *dairesel tabanlı işlev* (DTİ) ağı diyoruz (Broomhead ve Lowe 1988; Moody ve Darken 1989). Genelde, yerel birimli ağlarda birden fazla saklı katman kullanılmaz, H, çok katmanlı algılayıcılarda olduğu gibi saklı birim sayısıdır ve karmaşıklık parametresidir, öncesinde öbekleme yapıyorsak öbek sayısı olan k değerine eşittir.

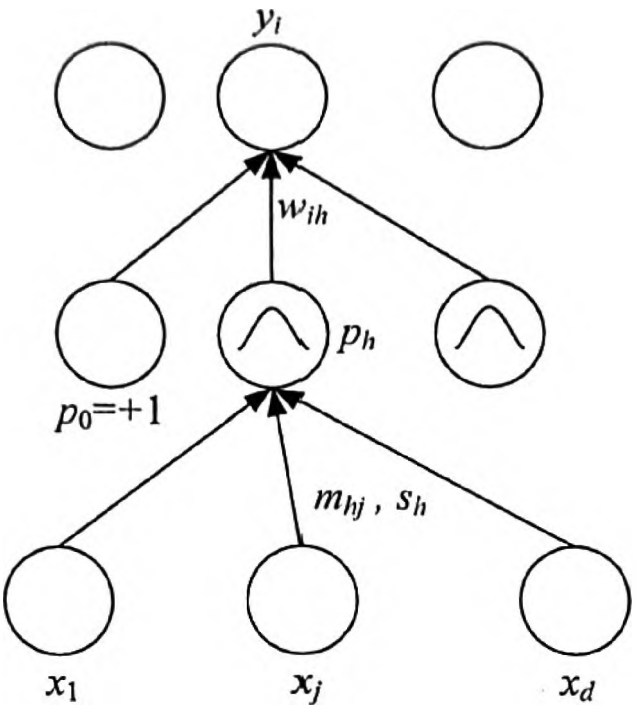
b_h yerine p_h değerlerini kullanmak yeğlenir, çünkü b_h değerleri 0/1 olduğundan eğer denklem 12.14'te p_h yerine b_h kullansaydık yerel birimlerin alan sınırlarında süreksizlikler oluşturan parçalı sabit bir yaklaşıklama gerçeklemiş olurduk. p_h ise sürekli olduğu için bir alandan öbürüne geçerken ağırlıklı toplam sayesinde daha düzleştirilmiş bir yaklaşıklama oluşturuyoruz. Böylesi bir ağ yapısının evrensel bir yaklaşıklama sağlayacağını, yani yeterince saklı birimle herhangi bir işlevi istenen kesinlikte yaklaşıklayabileceğini görmek kolaydır. Taylor açılımını düşünüp girdi uzayını istediğimiz kesinliği sağlayacak şekilde küçük alanlara bölüp her bir küçük alan için yerel bir saklı birim tanımlar ve çıktı ağırlığı olan w_h değerini de orası için istediğimiz değer olarak atayabiliriz.

Bu yapı, 8. bölümde gördüğümüz dağılımdan bağımsız kestiricilere, örneğin Parzen pencerelerine, çok benzerdir ve bu bağlamda p_h değerlerini çekirdek işlevleri gibi görebiliriz. Aralarındaki fark, her öğrenme örneğine bir çekirdek yerleştirmek yerine öbekleme yapıp daha az çekirdek kullanabilmemizdir. Saklı birim sayısı olan H karmaşıklık parametresidir ve basitlikle başarım arasındaki ödünleşimi belirler. Daha çok birimle öğrenme verisini daha iyi yaklaşıklayabiliriz, ama elimizde karmaşık bir model olur ve aşırı öğrenme riski oluşur. Bu tür durumlarda hep yaptığımız gibi en iyi değeri çapraz geçerlemeyle belirleriz.

m_h ve s_h değerleri belirlendiğinde p_h değerleri de belirlenmiş olur ve bu durumda w_h ağırlıklarını toptan ya da çevrimiçi olarak öğrenebiliriz. Bağlanım uygulamasında elimizdeki (ikinci katman) algılayıcı bu p_h değerlerinin girdi olduğu doğrusal bir modeldir ve w_h değerleri çözümsel bir şekilde bulunabilir (bölüm 4.6). Sınıflamada yinelemeli bir yordam gerekir ve bunun nasıl yapılacağını 10. bölümde uzun uzun konuştuğumuz için burada yinelemeyeceğiz.

Sonuç olarak iki aşamalı bir öğrenmeden bahsediyoruz. Birincisinde gözetimsiz bir öbekleme algoritmasıyla merkezleri buluyor ve üstüne gözetimli bir katman ekliyoruz. Bu yüzden bu yaklaşıma bazen *melez öğrenme* denir. İstersek tabii ki tüm parametreleri ve bu arada, m_h ve s_h değerlerini de gözetimli öğrenebiliriz. Denklem 12.13'teki Gauss eğrisi türevi alınabilir olduğundan çok katmanlı algılayıcılardaki geri yayma algoritmasıyla ilk katmandaki bu parametreleri de her yinelemede hatayı biraz azaltacak biçimde güncelleyebiliriz. p_h değerlerini saklı birimlerin çıktıları, Gauss eğrisini etkilenim işlevi, m_h ve s_h değerlerini ilk katman ve w_h değerlerini ikinci katman ağırlıkları olarak düşünürsek bu yapı bir saklı katmanlı bir algılayıcı ağı olarak gösterilebilir (bkz. şekil 12.8).

Ama buna geçmeden önce iki katmanlı bir ağı eğitmenin daha yavaş olduğunu hatırlayalım. Melez öğrenme katmanları birer birer eğittiği için daha hızlıdır. Hatta *çapa* yönteminde ilk katman için hiç öğrenme yapılmaz ve merkezler olarak rastgele rastgele seçilmiş öğrenme örnekleri kullanılır; eğer birim sayısı az tutulmazsa bu yeterli başarım gösterecektir.



Şekil 12.8: Dairesel tabanlı işlev ağı. p_h, çan eğrisi biçimindeki Gauss etkilenim işlevini kullanan saklı birimlerdir. m_h ve s_h birinci katmanda, w_i ikinci katmandaki ağırlıklardır.

Öte yandan melez öğrenmek tüm parametreleri gözetimli öğrenmek kadar başarılı olmayabilir. Girdi dağılımının tekdüze olduğunu varsayalım, bu durumda öbekleme sonrasında birimler girdi uzayında eşit aralıklı yerleştirilir. Ama öğrenmek istediğimiz işlev uzayın yalnız bir parçasında değişiyor öteki taraflarda sabit kalıyorsa, birimleri işlevin hızlı değiştiği yerlere koymak hatayı daha çok azaltır; tümüyle gözetimli öğrenme de zaten bu biçimde davranacaktır, çünkü merkezlerde yapılacak güncellemeler yalnızca girdilere değil çıktılara ve çıktı üzerinde tanımlanan hataya da bağlıdır.

Gelin şimdi, tüm parametrelerin nasıl birlikte gözetimli olarak eğitilebileceğini konuşalım. Yaklaşımımız çok katmanlı algılayıcılardaki hata geri yaymayla aynı. Önce çok çıktılı bağlanımı görelim. Azaltmak istediğimiz toptan hata karelerin farklarının toplamıdır:

$$E(\{m_h, s_h, w_{ih}\}_{i,h}|\mathcal{X}) = \frac{1}{2}\sum_t \sum_i (r_i^t - y_i^t)^2 \tag{12.15}$$

ve çıktıyı saklı birimler değerlerinin ağırlıklı toplamı olarak yazarız:

$$y_i^t = \sum_{h=1}^{H} w_{ih} p_h^t + w_{i0} \tag{12.16}$$

Eğim inişi kullanarak ikinci katman ağırlıklarını güncelleyebiliriz:

$$\Delta w_{ih} = \eta \sum_t (r_i^t - y_i^t) p_h^t \tag{12.17}$$

Burada yazdığımız p_h değerlerinin girdi olduğu bildiğimiz algılayıcı güncelleme kuralıdır. Yerel oldukları için p_h değerleri fazla örtüşmez ve genelde herhangi bir girdi için bir ya da az sayıda p_h sıfırdan farklı değer alır ve yalnızca onların w_h değerleri güncellenir. Bu yüzden yerel birimli ağlar, dağıtık gösterim kullanan çok katmanlı algılayıcılardan daha hızlı öğrenir.

Benzer biçimde, hata geri yayma (türevde zincir kuralı) kullanarak ilk katman parametreleri olan merkez ve yayılımları da güncelleyebiliriz:

$$\Delta m_{hj} = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) w_{ih} \right] p_h^t \frac{(x_j^t - m_{hj})}{s_h^2} \tag{12.18}$$

$$\Delta s_h = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) w_{ih} \right] p_h^t \frac{\|x^t - m_h\|^2}{s_h^3} \tag{12.19}$$

Denklem 12.18'le denklem 12.5'i karşılaştıralım: Öncelikle burada, b_h yerine p_h değerini kullanıyor, yani yalnızca en yakın olanı değil, merkez ve yayılımlarına bağlı olarak tüm birimleri güncelliyoruz (ama tabii ki çoğu uzakta olduklarından p_h değerleri sıfıra çok yakın olacak ve değerleri değişmeyecek). İkinci olarak buradaki güncelleme gözetimli ve geriye taşınmış hatayı da içeriyor. Bu da demek ki güncelleme yalnızca girdinin merkeze olan uzaklığına değil, $(r_i^t - y_i^t)$ çıktı hatasına, bu birimin çıktıya olan etkisini gösteren w_{ih} değerine ve birimin p_h etkilenim derecesine, ve girdiye olan $(x - m_h)$ uzaklığına bağlı olacak.

12.18. ve 12.19. denklemleri yazılımda gerçeklerken ek komutlar kullanarak eğim inişi sırasında s_h değerlerinin çok küçük ya da çok büyük değerler almasını ve m_h değerlerinin geçerli girdi aralığından çıkmasını engellemek gerektiğini aklımızda tutalım.

Sınıflandırmada çıktılar sınıf olasılıkların karşılık geldiğinden eşiksiz en büyük işleviyle normalleştirme gerekir:

$$y_i^t = \frac{\exp\left[\sum_h w_{ih} p_h^t + w_{i0}\right]}{\sum_k \exp\left[\sum_h w_{kh} p_h^t + w_{k0}\right]} \tag{12.20}$$

Azaltmak istediğimiz hata toplam çapraz düzensizliktir:

$$E(\{m_h, s_h, w_{ih}\}_{i,h} | \mathcal{X}) = -\sum_t \sum_i r_i^t \log y_i^t \tag{12.21}$$

Güncelleme kuralları eğim inişiyle benzer biçimde elde edilebilir (alıştırma 3).

Şimdi denklem 12.14'e bir kez daha bakalım: Bir girdi için p_h değeri sıfırdan farklıysa çıktıya etkisi w_h kadardır, yani etkisi sabittir. Yerel gösterimde Gauss

eğrileri fazla örtüşmez ve yalnızca birkaç tanesinin p_h değeri sıfırdan farklı olur ve çıktıyı belirleyen onlardır. w_0 değeri sabittir, etkin olan ($p_h > 0$) birimlerin w_h çıktılarına eklenir; eğer tüm p_h değerleri 0 olursa (tüm merkezlerden çok uzakta bir girdi için) çıktı w_0 olur. Bu bağlamda w_0 değerini y için bir ön değer olarak görebiliriz; eğer hiçbir birim etkin olmazsa çıktı bu değeri alacaktır. O zaman bu ön değeri sabit almak yerine daha gelişmiş bir modelle, bir "kuralla" hesaplamak düşünülebilir.

Örneğin, çıktıyı bir genel kuralla yerel "istisnaların" toplamı olarak yazabiliriz:

$$y^t = \underbrace{\sum_{h=1}^{H} w_h p_h^t}_{istisnalar} + \underbrace{v^T x^t + v_0}_{kural} \tag{12.22}$$

Buradaki kural doğrusaldır ve sabit olmak yerine girdiye bağlıdır: $v^T x^t + v_0$. Gauss birimleri sıfırdan farklı bir p_h değeri alıp etkin olduklarında yerel "istisnalar" tanımlar ve kuralın üstüne eklenerek çıktıyı değiştirirler. Böylesi bir modelde kural ve istisnalar birlikte gözetimli eğitilebilir (alıştırma 4). Bu yaklaşıma benzeyen ardışık sınıflandırıcılar modelini bölüm 17.11'de, biri kuralı, öbürü istisnaları öğrenen iki modelin birleşimi olarak tartışacağız.

12.4 Kural Tabanlı Bilginin Tanımlanması

Eğer bir uygulamayla ilgili *önsel bilgimiz* varsa ve kullanabilirsek öğrenme çok daha kolay olacaktır. Örneğin önsel bilgi girdiyle çıktı arasındaki ilişkiyi tanımlayan bir kural kümesi tanımlayabilir. Endüstriyel ve tıbbi uygulamalarda uzmanların önceden bildiği kurallar rastgele parametrelerle başlamak yerine modele önceden tanımlanabilir. Ya da eğitilmiş bir modelden böylesi kurallar çıkararak modelin öğrendiği bilgi kolay anlaşılabilir bir biçime sokulabilir.

Önsel bilgiyi belirtmenin bir getirisi de öğrenme kümesinde olmayan örnekler için öngörü yapmayı sağlayabilmesidir. Birçok uygulamada öğrenme çevrimiçiyken modelin ilk baştan beri akla yatkın çıktılar üretebilmesi beklenir ve henüz fazla örnek görülmediği durumda model bu önsel bilgiyi kullanabilir.

Bazı uygulamalarda önsel bilgiyle başlarız ve bu bilgiyi deneyimle geliştirir, arıtır, farklı koşullara uyarlarız. Önsel bilgimiz iyi olursa daha yüksek bir başarıma çok daha hızlı ulaşırız.

Önsel bilginin modele tanımlanması ya da eğitilmiş modelden bilgi çıkarılması, yerel gösterim kullandığı için dairesel tabanlı işlev ağlarında çok daha kolaydır. Bu da *kural çıkarımını* kolaylaştırır (Tresp, Hollatz, ve Ahmad 1997). Örnek bir kural alalım:

$$\text{EĞER } ((x_1 \approx a) \text{ VE } (x_2 \approx b)) \text{ VEYA } (x_3 \approx c) \text{ İSE } y = 0.1 \tag{12.23}$$

Burada, $x_1 \approx a$, "x_1 yaklaşık olarak a değerine eşit" demektir. Dairesel tabanlı işlev

ağında bu kural iki Gauss birimiyle gösterilebilir:

$$p_1 = \exp\left[-\frac{(x_1-a)^2}{2s_1^2}\right] \cdot \exp\left[-\frac{(x_2-b)^2}{2s_2^2}\right] \text{ ve } w_1 = 0.1$$

$$p_2 = \exp\left[-\frac{(x_3-c)^2}{2s_3^2}\right] \text{ ve } w_2 = 0.1$$

"Yaklaşık olarak eşit" deyişini, merkezi ideal değer ve yayılımı da bu idealden kabul edilebilir sapma olan bir Gauss eğrisiyle gösteriyoruz. VE işlevi için iki tane tek boyutlu Gauss işlevini çarparak elde ettiğimiz iki boyutlu bir Gauss işlevini kullanıyoruz. Bu durumda ilk terim, iki boyutlu, merkezi (a, b) ve yayılımları s_1 ve s_2 olan bir Gauss birimiyle hesaplanabilir. VEYA kapısının terimleri için ayrı birimler kullanıyoruz ve sonraki toplama bunları birleştiriyor.

Etiketlenmiş gözetimli verimiz varsa bu ilk değerlerden sonra küçük bir η değeriyle öğrenme yaparak bu parametreleri daha da iyiye çekmeye çalışabiliriz.

Burada konuştuğumuz gösterim bulanık mantığa çok benzemektedir ve denklem 12.23 bir *bulanık kural* olarak da görülebilir. Yaklaşık eşitlik ölçen Gauss eğrisi, bir tür *bulanık üyelik işlevidir* (Berthold 1999; Cherkassky ve Mulier 1998).

12.5 Normalleştirilmiş Taban İşlevleri

Denklem 12.14'de bir girdi tüm merkezlerden uzaksa tüm p_h değerleri 0 olabilir. Bazı uygulamalarda, yerel birimler toplamları 1 olacak biçimde bir *normalleştirme* adımından geçer ve böylece herhangi bir girdi için en azından bir birimin etkin ve sorumlu (çıktısının sıfırdan farklı) olması sağlanır:

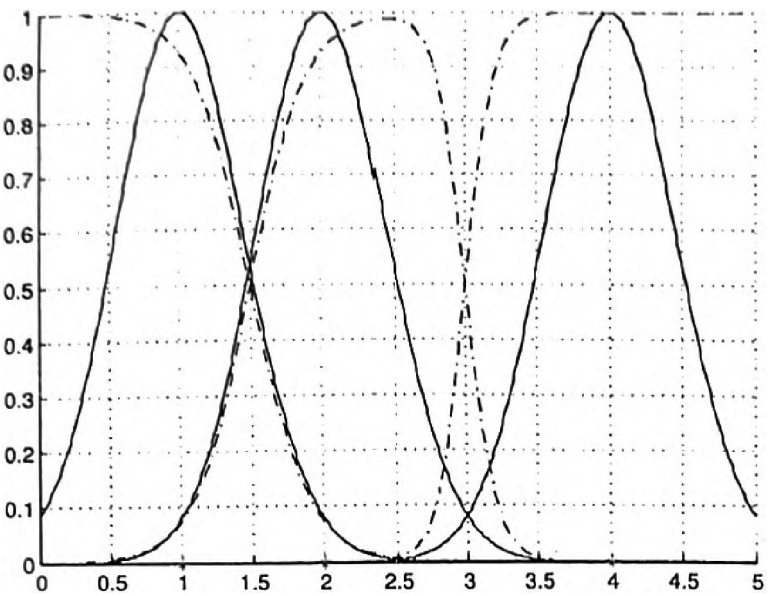
$$g_h^t = \frac{p_h^t}{\sum_{l=1}^{H} p_l^t} = \frac{\exp[-\|\mathbf{x}^t - \mathbf{m}_h\|^2/2s_h^2]}{\sum_l \exp[-\|\mathbf{x}^t - \mathbf{m}_l\|^2/2s_l^2]} \tag{12.24}$$

Şekil 12.9'da bir örnek verilmiştir. p_h değerini bir $p(\mathbf{x}|h)$ koşullu olasılığı olarak yazarsak g_h değeri de $p(h|\mathbf{x})$ sonsal olasılığına karşılık gelir, yani $\mathbf{x}$ verildiğinde h biriminin sorumlu olma sonsal olasılığıdır. Sanki birimler kendi aralarında uzayı bölüşmüşlerdir ve girdi birimlerden birini seçerek sınıflandırılır. g_h değeri girdiye göre sorumlu sınıfı (saklı birimi) seçen bir sınıflandırıcı olarak görülebilir; sınıflandırma uzaklığa bağlı olarak yapılır ve önsel olasılıklarının eşit olduğu Gauss dağılımına bağlı bir sınıflandırıcı kullanılmaktadır (Bölüm 5).

Çıktı yine birimlerin değerlerinin ağırlıklı toplamıdır:

$$y_i^t = \sum_{h=1}^{H} w_{ih} g_h^t \tag{12.25}$$

Tek farkı artık bir ek girdiye ve w_0 değerine gereksinim olmamasıdır; bunun nedeni de herhangi bir girdi için en az bir g_h değerinin sıfırdan farklı olmasıdır. p_h yerine g_h kullanmak ek parametre eklemez, yalnızca birimleri birbirleriyle ilişkilendirir:



Şekil 12.9: Normalleştirmeden önce (-) ve sonra (- -). Merkezleri '*' ile gösterilen üç Gauss birimi görüyoruz. Normalleştirme sonrasında bir birimin sıfırdan farklı değer aldığı yerler yalnız onun değil, öteki merkezlerin de nerede olduğuna bağlıdır. Yayılımlar küçük olduğunda şeçişler sert olur; yayılımlar genişledikçe birimler daha fazla örtüşür ve geçişler daha yumuşak olur.

p_h yalnızca kendi m_h ve s_h parametrelerine bağlıyken g_h değeri normalleştirme yüzünden bütün birimlerin merkez ve yayılımlarına bağlıdır.

Örnek olarak bağlanım için eğim inişi kullanan güncellemeyi yazalım:

$$\Delta w_{ih} = \eta \sum_t (r_i^t - y_i^t) g_h^t \tag{12.26}$$

$$\Delta m_{hj} = \eta \sum_t \sum_i (r_i^t - y_i^t)(w_{ih} - y_i^t) g_h^t \frac{(x_j^t - m_{hj})}{s_h^2} \tag{12.27}$$

Benzer biçimde s_h değerlerini de güncelleyebiliriz. Sınıflandırma için benzer güncelleme kuralları da kolaylıkla çıkarılabilir.

Buradaki güncelleme kurallarını denklem 12.17'de verilmiş normalleştirilmemiş Gauss birimleri için geçerli kurallarla karşılaştıralım. Öncelikle p_h yerine g_h değerinin kullanılıyor olması bir girdideki güncellemenin tüm birimlerin parametrelerine bağlı olduğunu gösteriyor. Denklem 12.27'yle denklem 12.18'i karşılaştırdığımızda, w_{ih} yerine $(w_{ih} - y_i^t)$ teriminin kullanılması da yine normalleştirmenin etkisidir: "Sorumlu" birim kendi çıktı değeri w_{ih} ile çıktı olan y_i^t değeri arasındaki farkı, sorumluluğu (g_h) oranında azaltmaya çalışıyor.

12.6 Yarışmacı Taban İşlevleri

Şimdiye dek gördüğümüz DTİ ağlarında çıktıyı yerel birimlerinin katkılarının ağırlıklı toplamı olarak hesapladık. Birimler yerel olmasına rağmen önemli olan en sondaki ağırlıklı toplamdır ve istenen çıktıya yaklaştırmak istediğimiz bu toplam değerdir. Örneğin bağlanımda denklem 12.15'e karşılık gelen olasılıksal modeli yazalım:

$$p(r^t|x^t) = \prod_i \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(r_i^t - y_i^t)^2}{2\sigma^2}\right] \tag{12.28}$$

Burada y_i^t, denklem 12.16 (normalleştirilmemiş) ya da denklem 12.25'deki (normalleştirilmiş) biçimde tanımlıdır. Her iki durumda da elimizdeki *işbirlikçi* bir modeldir, çünkü birimler son y_i^t çıktısını birlikte oluşturur. *yarışmacı taban işlevlerindeyse* çıktının bir karışım modelinden çekildiği varsayılır:

$$p(r^t|x^t) = \sum_{h=1}^{H} p(h|x^t)p(r^t|h, x^t) \tag{12.29}$$

$p(h|x^t)$ karışım oranları, ve $p(r^t|h, x^t)$ ise çıktıyı belirleyen karışım bileşenleridir. Her ikisi de x girdisine bağlıdır.

Karışım oranları girdiye uzaklığa bağlıdır:

$$p(h|x) = \frac{p(x|h)p(h)}{\sum_l p(x|l)p(l)} \tag{12.30}$$

$$g_h^t = \frac{a_h \exp[-\|x^t - m_h\|^2/2s_h^2]}{\sum_l a_l \exp[-\|x^t - m_l\|^2/2s_l^2]} \tag{12.31}$$

Genelde a_h değerlerini eşit kabul edip görmezden geliriz. Önce bileşenlerin Gauss olduğu bağlanımdan başlayalım. Denklem 12.28'de gürültü, ağırlıklı toplama ekleniyordu; buradaysa bir bileşen belirlenir ve gürültü onun y_{ih}^t çıktısına eklenir.

Denklem 12.29'daki karışım modeline göre log olabilirlik işlevini tanımlayalım:

$$\mathcal{L}(\{m_h, s_h, w_{ih}\}_{i,h}|\mathcal{X}) = \sum_t \log \sum_h g_h^t \exp\left[-\frac{1}{2}\sum_i (r_i^t - y_{ih}^t)^2\right] \tag{12.32}$$

Burada $y_{ih}^t = w_{ih}$, h bileşeninin i çıktısı için tanımladığı sabit değerdir, ve gerçekte girdiye bağlı değildir.(Bölüm 12.8.2'de yerel kestirimin girdinin doğrusal bir işlevi olduğu yarışmacı uzman karışımı yaklaşımından bahsedeceğiz.) $g_h^t = 1$ ise doğru çıktıyı üretmekten h bileşeni sorumludur ve istenen çıktıyla kendi kestirimi arasındaki farkın karesini $(\sum_i (r_i^t - y_{ih}^t)^2)$ en küçüğe indirmek zorundadır.

Log olabilirliği büyütmek için eğim çıkış algoritmasını kullanabiliriz:

$$\Delta w_{ih} = \eta \sum_t (r_i^t - y_{ih}^t) f_h^t \tag{12.33}$$

Artık bir birimin (bileşenin) etkisi yalnızca merkezlere ve yayılımlara değil birimlerin çıktı kestirimlerine de bağlıdır:

$$f_h^t = \frac{g_h^t \exp[-\frac{1}{2}\sum_i (r_i^t - y_{ih}^t)^2]}{\sum_l g_l^t \exp[-\frac{1}{2}\sum_i (r_i^t - y_{il}^t)^2]} \tag{12.34}$$

$$p(h|r,x) = \frac{p(h|x)p(r|h,x)}{\sum_l p(l|x)p(r|l,x)} \tag{12.35}$$

$g_h^t \equiv p(h|x^t)$, girdi verildiğinde h bileşeninin sonsal olasılığıdır ve bileşenlerin merkez ve yayılımlarına bağlıdır. $f_h^t \equiv p(h|r,x^t)$ ise girdi ve istenen çıktı verildiğinde hesaplanan sonsal olasılıktır ve sorumlu bileşenin seçilmesinde artık yalnızca girdinin değil hatanın da önemli olduğunu gösterir.

Benzer biçimde merkezleri de güncelleyebiliriz:

$$\Delta m_{hj} = \eta \sum_t (f_h^t - g_h^t) \frac{(x_j^t - m_{hj})}{s_h^2} \tag{12.36}$$

g_h girdiye bağlı sonsal olasılıkken f_h istenen çıktıya da bağlıdır. Aralarındaki fark, merkezler için hatayı verir. Δs_h de benzer şekilde türetilebilir. Yardımlaşmacı durumda birimleri yerelleştiren bir güç yoktur. Hatayı azaltmak için merkez ve yayılımlar herhangi değerleri alabilir, hatta bu yüzden yayılım değerleri çok artıp Gauss eğrileri düzleşebilir. Yarışmacı durumdaysa olabilirliği artırmak için birimlerin merkezleri arasındaki uzaklık artar ve yayılımları azalır, bu da birimlerin yerelleşmesi demektir.

Sınıflandırmada her bileşen bir katlıterimli bir dağılımdır ve log olabilirlik buna uygun olarak yazılır:

$$\mathcal{L}(\{m_h, s_h, w_{ih}\}_{i,h}|\mathcal{X}) = \sum_t \log \sum_h g_h^t \prod_i (y_{ih}^t)^{r_i^t} \tag{12.37}$$

$$= \sum_t \log \sum_h g_h^t \exp\left[\sum_i r_i^t \log y_{ih}^t\right] \tag{12.38}$$

Sınıf sonsal olasılığı bileşen çıktıları eşiksiz en büyük işleviyle normalleştirilerek hesaplanır:

$$y_{ih}^t = \frac{\exp w_{ih}}{\sum_k \exp w_{kh}} \tag{12.39}$$

Eğim çıkış kullanarak w_{ih}, m_h, ve s_h için güncelleme kuralları türetilebilir. Bu durumda girdi ve istenen çıktıya bağlı birim sonsal olasılığı bağlanımdakinden biraz farklıdır:

$$f_h^t = \frac{g_h^t \exp[\sum_i r_i^t \log y_{ih}^t]}{\sum_l g_l^t \exp[\sum_i r_i^t \log y_{il}^t]} \tag{12.40}$$

Bölüm 7'de, veriye Gauss karışımları oturtmak için beklenti büyütme algoritmasından bahsetmiştik. Bu algoritma gözetimli öğrenme için genellenebilir ve f_h^t

değerlerinin hesaplanması beklenti adımına karşılık gelir. Bölüm 7'de gözetimsiz durumda $P(h|\boldsymbol{x}^t)$ değerini kullanıyorduk, buradaysa yerine $f_h^t \equiv p(\boldsymbol{r}|h, \boldsymbol{x}^t)$ değerini kullanıyoruz. Bağlanımda, büyütme adımında parametreleri bu değerlere bağlı olarak güncelliyoruz:

$$\boldsymbol{m}_h = \frac{\sum_t f_h^t \boldsymbol{x}^t}{\sum_t f_h^t} \tag{12.41}$$

$$\mathbf{S}_h = \frac{\sum_t f_h^t (\boldsymbol{x}^t - \boldsymbol{m}_h)(\boldsymbol{x}^t - \boldsymbol{m}_h)^T}{\sum_t f_h^t} \tag{12.42}$$

$$w_{ih} = \frac{\sum_t f_h^t r_i^t}{\sum_t f_h^t} \tag{12.43}$$

Gördüğümüz gibi w_{ih} değeri girdilerin ağırlıklı bir ortalaması oluyor ve ağırlıklar bu sonsal olasılıklarca veriliyor. Sınıflandırma durumunda büyütme adımı çözümsel değildir ve yerine yinelemeli, örneğin eğim çıkış kullanan bir öğrenme yapmamız gerekir (Jordan ve Jacobs 1994).

12.7 Öğrenen Yöney Nicemleme

Bu kısımda çevrimiçi öbeklemenin gözetimli bir sürümünden bahsedeceğiz. Elimizde sınıf başına H birim olsun ve bu birimlere sınıflarının rastgele seçilmiş örneklerini ilk değer olarak verelim. Sonra çevrimiçi bir biçimde, her girdi için ona Euclid uzaklığı olarak en yakın $\boldsymbol{m}_i$ birimini bulup güncelleyelim:

$$\begin{cases} \Delta \boldsymbol{m}_i = \eta(\boldsymbol{x}^t - \boldsymbol{m}_i) & \text{eğer } \boldsymbol{x}^t \text{ ve } \boldsymbol{m}_i \text{ aynı sınıf etiketini taşıyorsa} \\ \Delta \boldsymbol{m}_i = -\eta(\boldsymbol{x}^t - \boldsymbol{m}_i) & \text{farklı sınıflardanlarsa} \end{cases} \tag{12.44}$$

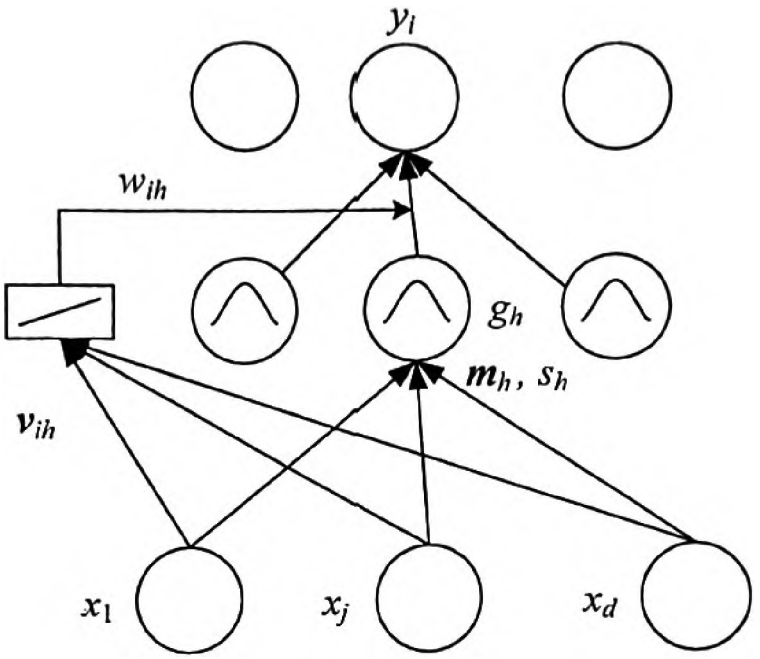
Eğer girdiyle ona en yakın merkez aynı sınıftansa girdiyi daha az hatayla gösterebilmek için merkez girdiye doğru çekilir. Yanlış sınıftansa merkez girdiden uzağa itilir; böylece eğer yeterince itilirse, yinelemelerden sonra başka (doğru sınıftan) bir merkezin o girdiye en yakın birim olacağı umulur. Bu algoritmaya *öğrenen yöney nicemleme* algoritması denir (Kohonen 1990, 1995).

Buradaki güncelleme kuralı, denklem 12.36'daki gibi gözetimlidir; bir merkez için yapılan güncelleme hem girdiye, hem de çıktıya ve doğru sınıfa bağlıdır.

12.8 Uzmanların Karışımı

DTİ'de her yerel alanda sabit bir çıktı vardır. Herhangi bir girdi için g_h değerlerinden birinin 1, ötekilerin 0 olduğu durumda parçalı sabit bir yaklaşıklama yaparız; i çıktısı için h biriminin öngörüsü w_{ih} ağırlığınca belirlenir. Taylor açılımında sabit terimin yanında doğrusal terimi de kullanıp daha iyi bir yaklaşıklama yapabileceğimizi biliyoruz:

$$f(x) = f(a) + (x - a)f'(a) + \cdots \tag{12.45}$$



Şekil 12.10: Uzmanların karışımı ikinci katman ağırlıkları girdiye bağlı hesaplanan dairesel tabanlı bir işlev ağı olarak görülebilir. Kolay anlaşılabilisin diye yalnızca bir doğrusal bağlantı çizilmiştir.

Eğer x, a noktasına yakınsa, yani $f(x)$ işlevi, a yöresinde düzse sabit yakınsama başarılı olur. Değilse girdi uzayını çok fazla parçalayıp çok küçük alanlara bölmek gerekir ki bu da özellikle girdi boyut sayısının az olmadığı durumda yakamıza yapışan çok boyutluluğun laneti yüzünden gereken birim sayısını çok artıracak, hem de aşırı öğrenmeye neden olacaktır.

Taylor açılımındaki bir sonraki doğrusal terimi de işin içine katıp *doğrusal parçalı yaklaşıklama* yapmaya *uzmanların karışımı* (Jacobs ve ötekiler 1991) diyoruz.

Çıktımız yine yerel modellerin ağırlıklı toplamıdır:

$$y_i^t = \sum_{h=1}^{H} w_{ih} g_h^t \tag{12.46}$$

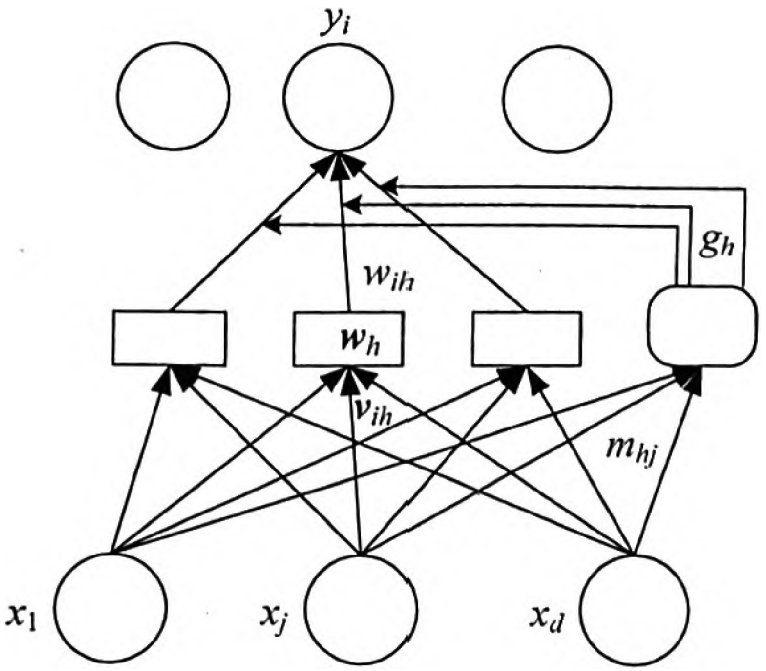
Denklem 12.25'nin aynısı gibi görünse de burada w_{ih} değeri sabit değil girdinin doğrusal bir işlevidir:

$$w_{ih}^t = v_{ih}^T x^t \tag{12.47}$$

v_{ih} bu doğrusal modeli belirleyen ağırlık yöneyidir ve ek girdi de içerir. Birim etkilenimleri yine normalleştirilmiş Gauss eğrileri olarak alınabilir:

$$g_h^t = \frac{\exp[-\|x^t - m_h\|^2/2s_h^2]}{\sum_l \exp[-\|x^t - m_l\|^2/2s_l^2]} \tag{12.48}$$

Bu yapıyı, ikinci katman ağırlıkları sabit olmak yerine girdinin doğrusal işlevleri olan bir DTİ ağı olarak görebiliriz (bkz. şekil 12.10). Bir başka açıdan da (Jacobs ve



Şekil 12.11: Uzmanların karışımı birden çok modelin birleştirilmesi olarak da görülebilir. w_h değerleri doğrusal uzman modellerin çıktılarıdır ve geçit ağı, her uzmana girdiye bağlı olarak bir g_h ağırlığı atar. Bu açıdan bakıldığında ne uzmanların, ne de geçit ağının doğrusal olması kısıtı yoktur.

ötekiler, 1991), w_h ile gösterdiğimiz her biri girdiye göre bir öngörü yapan doğrusal *uzmanların* çıktılarıdır. g_h değerleri bir *geçit* ağının çıktılarıdır ve girdiye bağlı olarak uzmanlara (toplamları 1 olacak biçimde) ağırlıklar atar (bkz. şekil 12.11).

Bu biçimde yorumladığımızda geçit için herhangi bir sınıflandırıcı modeli kullanılabileceğini görürüz. x yüksek boyutlu olduğunda yerel Gauss birimleri kullanmak fazla uzman gerektirebilir ve Jacobs ve diğerleri (1991) doğrusal sınıflandırıcı kullanmayı önermişlerdir:

$$g_h^t = \frac{\exp[\boldsymbol{m}_h^T \boldsymbol{x}^t]}{\sum_l \exp[\boldsymbol{m}_l^T \boldsymbol{x}^t]} \tag{12.49}$$

Burada, $\boldsymbol{m}_h$ değerleri artık merkezler değil üstündüzlem ağırlıklarıdır ve ek girdi de içerirler (ama yayılım parametresi gerektirmezler). Böylesi bir geçit ağı, uzmanların uzmanlık alanlarını girdi uzayında doğrusal olarak ayırır. Bölüm 17'de daha ayrıntılı olarak tartışacağımız gibi, uzmanların karışımı modellerin birleştirilmesi için genel bir yapıdır ve ne uzmanların, ne de geçit ağının doğrusal olma zorunluluğu vardır; örneğin bunlar için çok katmanlı algılayıcılar da kullanılabilir (alıştırma 6).

Uzmanların karışımı ve akan çizgi yumuşatıcısına (bölüm 8.6.3) benzeyen bir yapı da Bottou ve Vapnik tarafından önerilmiştir (1992). Bu algoritmada başta hiç öğrenme yapılmaz. Her deneme örneği verildiğinde öğrenme kümesinden ona en yakın örnekler bulunur (en yakın k komşu algoritmasındaki gibi ama daha büyük bir k değeriyle) ve öğrenme kümesinin bu yerel altkümesiyle doğrusal bir model eğitilir.

Bu modelle deneme örneği için çıktı hesaplanır ve sonra bu model silinip bir sonraki deneme örneğine geçilir. Elle yazılmış sayıların sınıflandırılmasında bu yapı, çok katmanlı algılayıcılar, en yakın k komşu ve Parzen pencerelerinden daha başarılıdır; belki tek çekince, her deneme örneği için sıfırdan bir model eğitmenin gerekmesidir.

12.8.1 Yardımlaşan Uzmanlar

Uzmanlar yardımlaştıklarında y_i^t çıktısı denklem 12.46'yla tanımlanır ve istediğimiz bu çıktının istenen r_i^t değerine olabildiğince yakın olmasıdır. Bağlanımda hata hep olduğu gibi farkların karelerinin toplamıdır:

$$E(\{m_h, s_h, w_{ih}\}_{i,h}|\mathcal{X}) = \frac{1}{2}\sum_t \sum_i (r_i^t - y_i^t)^2 \tag{12.50}$$

Eğim inişi kullanarak ikinci katman (uzman) ağırlıklarını güncelleyebiliriz:

$$\Delta v_{ih} = \eta \sum_t (r_i^t - y_i^t) g_h^t x^t \tag{12.51}$$

Denklem 12.26'yla karşılaştırdığımızda tek farkın bu yeni güncellemenin girdiyi de kullanması olduğunu görürüz.

Geçit için doğrusal modeller kullanırsak (denklem 12.49), üstündüzlem parametrelerinin eğim inen güncellemeleri girdiyle çarpılarak bulunur:

$$\Delta m_{hj} = \eta \sum_t \sum_i (r_i^t - y_i^t)(w_{ih}^t - y_i^t) g_h^t x_j^t \tag{12.52}$$

Eğer dairesel tabanlı geçit kullanırsak (denklem 12.48), yalnızca en son $\partial p_h / \partial m_{hj}$ terimi değişir ve girdiye olan uzaklık kullanılır.

Sınıflandırmada çıktıda her zamanki gibi eşiksiz en büyük işlevi vardır:

$$y_i = \frac{\exp\left[\sum_h w_{ih} g_h^t\right]}{\sum_k \exp\left[\sum_h w_{kh} g_h^t\right]} \tag{12.53}$$

ve yine uzman katkıları doğrusaldır: $w_{ih} = v_{ih}^T x$. Çapraz düzensizliği azaltacak güncelleme kuralları bağlanımdaki gibi eğim inişle elde edilebilir (alıştırma 7).

12.8.2 Yarışmacı Uzmanlar

Yine önce model için log olabilirlik yazalım:

$$\mathcal{L}(\{m_h, s_h, w_{ih}\}_{i,h}|\mathcal{X}) = \sum_t \log \sum_h g_h^t \exp\left[-\frac{1}{2}\sum_i (r_i^t - y_{ih}^t)^2\right] \tag{12.54}$$

ve yine fark burada uzman çıktılarının girdiye bağlı doğrusal olmasıdır: $y_{ih}^t = w_{ih}^t = \boldsymbol{v}_{ih}\boldsymbol{x}^t$. Eğim çıkış kullanarak güncelleme kurallarını yazalım:

$$\Delta \boldsymbol{v}_{ih} = \eta \sum_t (r_i^t - y_{ih}^t) f_h^t \boldsymbol{x}^t \tag{12.55}$$

$$\Delta m_h = \eta \sum_t (f_h^t - g_h^t) \boldsymbol{x}^t \tag{12.56}$$

Burada denklem 12.49'deki doğrusal geçit modelini varsaydık.
Sınıflandırmada log olabilirliği yazalım:

$$\mathcal{L}(\{m_h, s_h, w_{ih}\}_{i,h}|\mathcal{X}) = \sum_t \log \sum_h g_h^t \prod_i (y_{ih}^t)^{r_i^t} \tag{12.57}$$

$$= \sum_t \log \sum_h g_h^t \exp\left[\sum_i r_i^t \log y_{ih}^t\right] \tag{12.58}$$

Burada uzman çıktıları yine doğrusaldır, ama sınıflandırma olduğu için olasılıklara karşılık gelmesi gerekir ve bunun için normalleştirilir:

$$y_{ih}^t = \frac{\exp w_{ih}^t}{\sum_k \exp w_{kh}^t} = \frac{\exp[\boldsymbol{v}_{ih}\boldsymbol{x}^t]}{\sum_k \exp[\boldsymbol{v}_{kh}\boldsymbol{x}^t]} \tag{12.59}$$

Jordan ve Jacobs (1994), beklenti büyütme algoritmasını yarışmacı yerel doğrusal modeller için genelleştirmiştir. Alpaydın ve Jordan (1996), yardımlaşan ve yarışmacı uzman karışımlarını sınıflandırma uygulamalarında karşılaştırmıştır. Yardımlaşan yaklaşım daha başarılıdır ama yarışmacı yaklaşım daha hızlı öğrenir. Bunun nedeni yardımlaşan uzmanların daha fazla örtüşmesi ve böylece daha düz bir yaklaşıklama gerçekleştirebilmesidir; yarışmacı yaklaşımdaysa her girdi için genelde yalnızca bir uzman etkin olduğundan öğrenme de daha hızlıdır.

12.9 Uzmanların Karışım Ağacı

Şekil 12.11'de uzmanları ve bunların arasından seçim yapan bir geçit ağını görüyoruz. Eğer her uzman yerine özçağrılı olarak böyle bir yapı koysak elimizde *uzmanların karışım ağacı* dediğimiz yapı oluşur (Jordan ve Jacobs 1994). Bu yapı, 9. bölümde gördüğümüz karar ağaçlarına benzer; her geçit ağı, dallar arasında seçim yapan bir karar düğümü gibidir. Geçit ağı doğrusal olduğunda bu çok değişkenli bir karar ağacını andırır (kısım 9.6). Buradaki fark, geçit ağının karar ağacındaki bir karar düğümünden farklı olarak 0/1 diye karar vermemesi, ve dallardan yalnızca birini seçmek yerine tüm dallara, ama farklı ağırlıklarla gidip hepsinin çıktısını ağırlıklı olarak birleştirmesidir. Yapraklar, girdinin doğrusal işlevleridir ve değerleri, aşağıdan köke doğru ağaç boyunca ağırlıklı toplanarak aktarılır. Kök, bütün yaprakların birleşiminden oluşan en son çıktıyı verir. Bu bağlamda elimizdeki yapı, kökten bir yaprağa giden bir yol seçen kesikli karar ağacından farklı olarak kökten tüm

yapraklara giden tüm yolları ve tüm yaprak çıktılarını birleştiren *eşiksiz bir karar ağacıdır*.

Derinliği, uzmanları, ve geçit modeli bir kez tanımlandıktan sonra tüm ağaçı etiketlenmiş bir veri kümesiyle eğitebiliriz. Jordan ve Jacobs (1994), böylesi bir yapı için eğim iniş ya da beklenti büyütme kullanan öğrenme kurallarını türetmiştir.

12.10 Kısa kısa

Dairesel tabanlı işlev ağı, basit işlem birimleri tarafından gerçeklenen bir yapay sinir ağı olarak görülebilir. Birinci ve ikinci katmanlarının farklı olması yüzünden çok katmanlı algılayıcılardan farklıdır. Omohundro (1987), yerel modellerin nasıl yapay sinir ağları olarak gerçeklenebileceğini ve ilgili yerel birimlerin hızla bulunabilmesi için kullanılabilecek ağaç biçimli veri yapılarını tartışır. Specht (1991), Parzen pencerelerine eş bir yapay sinir ağı yapısı önermiştir.

Platt (1991), gerektikçe yeni birimlerin eklendiği artımlı bir DTİ ağını, Fritzke de (1995) benzer olarak özörgütlemeli haritaların artımlı bir sürümünü önermiştir.

Lee (1991), en yakın k komşu, çok katmanlı algılayıcı ve DTİ ağını, elle yazılmış sayıların tanınması uygulamasında karşılaştırmış ve üç algoritmanın da yaklaşık aynı oranda hata yaptığını gözlemlemiştir. DTİ ağları çok katmanlı algılayıcılardan daha hızlı öğrenir ama daha fazla saklı birim kullanır. Her iki algoritma da en yakın k komşu algoritmasından daha hızlıdır ve daha az bellek kullanır. Zaman, bellek, ve işlem karmaşıklığı gibi kıstaslar gerçek uygulamalarda bazen hatadaki küçük farklardan daha önemli olabilir.

Kohonen'in önerdiği özörgütlemeli haritalar bir çok uygulamada başarıyla kullanılmış ve gözetimli öğrenmeden önce girdi dağılımını modellemek için en başarılı yapay sinir ağı modellerinden olmuştur. Bir başka ilginç ve başarılı uygulaması da gezgin satıcı uygulamasındadır (Angeniol, Vaubois, ve Le Texier 1988). k merkezli öbeklemeyle Gauss karışımlarında beklenti büyütme (bölüm 7) arasındaki ilişki, özörgütlemeli haritalarla *üretici yerbetimsel eşleme* (Bishop, Svensén, ve Williams 1998) arasında da vardır; özörgütlemeli haritaların olasılıksal bir sürümü olarak Gauss karışımları üzerinde verinin log olabilirliği artırılırken karışım içindeki bileşenler daha düşük boyutlu bir altuzayda yerleşecek biçimde kısıtlanmıştır.

DTİ ağında merkezler ve yayılımlar bir kez verildikten sonra (örneğin çapa yöntemindeki gibi veriden rastgele seçilerek), ikinci katman bildiğimiz bir doğrusal modeldir. Bu model, Gauss çekirdekleri kullanan destek yöney makinelerine eşittir ve bölüm 13'de göreceğimiz gibi bu yaklaşımda öğrenme, öğrenme kümesinin, destek yöneyleri adını verdiğimiz bir alt kümesini seçmeye karşılık gelir. Saklanmış öğrenme örneklerinden ara değerleme yaptığımız Gauss süreçleri (bölüm 14) de yine bunlara benzerdir.

12.11 Alıştırmalar

1. Dışlamalı VEYA işlevini gerçekleyen DTİ ağını gösterin.

2. Denklem 12.13'teki gibi dairesel yerine elips biçiminde birimler kullanan DTİ ağını tanımlayın.
3. Sınıflandırmaya uygun DTİ ağı için güncelleme kurallarını türetin (bkz. denklemler 12.20 ve 12.21).
4. Denklem 12.22'deki modelin nasıl eğitilebileceğini gösterin.
5. Uzmanların karışımı ve dairesel tabanlı işlev ağını bellek karmaşıklığına göre karşılaştırın.
6. Uzmanların ve geçit ağının çok katmanlı algılayıcı olduğu uzmanların karışımı modelini yazın, ve hem bağlanım, hem de sınıflandırma için güncelleme kurallarını türetin.
7. Yardımlaşan uzmanlar için sınıflandırmada kullanılacak güncelleme kurallarını türetin.
8. Yarışmacı uzmanlar için sınıflandırmada kullanılacak güncelleme kurallarını türetin.
9. İki katlı bir uzmanların karışım ağacını yazın ve hem bağlanım, hem de sınıflandırma için güncelleme kurallarını türetin.
10. Uzmanların karışımında farklı uzmanlar girdi uzayının farklı alanlarında yoğunlaşır. Bu alanlarda yerel boyut azaltmanın nasıl yapılabileceğini tartışın.

12.12 Kaynaklar

Alpaydın, E., M. I. Jordan. 1996. "Local Linear Perceptrons for Classification." *IEEE Transactions on Neural Networks* 7: 788–792.

Angeniol, B., G. Vaubois, Y. Le Texier. 1988. "Self Organizing Feature Maps and the Travelling Salesman Problem." *Neural Networks* 1: 289–293.

Berthold, M. 1999. "Fuzzy Logic." *Intelligent Data Analysis: An Introduction*, ed. M. Berthold and D. J. Hand, 269–298. Berlin: Springer.

Bishop, C. M., M. Svensén, C. K. I. Williams. 1998. "GTM: The Generative Topographic Mapping." *Neural Computation* 10: 215–234.

Bottou, L., V. Vapnik. 1992. "Local Learning Algorithms." *Neural Computation* 4: 888–900.

Broomhead, D. S., D. Lowe. 1988. "Multivariable Functional Interpolation and Adaptive Networks." *Complex Systems* 2: 321–355.

Carpenter, G. A., S. Grossberg. 1988. "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network." *IEEE Computer* 21(3): 77–88.

Cherkassky, V., F. Mulier. 1998. *Learning from Data: Concepts, Theory, and Methods.* New York: Wiley.

DeSieno, D. 1988. "Adding a Conscience Mechanism to Competitive Learning." *IEEE International Conference on Neural Networks*, 117–124. Piscataway, NJ: IEEE Press.

Feldman, J. A., D. H. Ballard. 1982. "Connectionist Models and their Properties." *Cognitive Science* 6: 205–254.

Fritzke, B. 1995. "Growing Cell Structures: A Self Organizing Network for Unsupervised and Supervised Training." *Neural Networks* 7: 1441–1460.

Grossberg, S. 1980. "How does the Brain Build a Cognitive Code?" *Psychological Review* 87: 1–51.

Hertz, J., A. Krogh, R. G. Palmer. 1991. *Introduction to the Theory of Neural Computation.* Reading, MA: Addison Wesley.

Jacobs, R. A., M. I. Jordan, S. J. Nowlan, G. E. Hinton. 1991. "Adaptive Mixtures of Local Experts." *Neural Computation* 3: 79–87.

Jordan, M. I., R. A. Jacobs. 1994. "Hierarchical Mixtures of Experts and the EM Algorithm." *Neural Computation* 6: 181–214.

Kohonen, T. 1990. "The Self-Organizing Map." *Proceedings of the IEEE* 78: 1464–1480.

Kohonen, T. 1995. *Self-Organizing Maps.* Berlin: Springer.

Lee, Y. 1991. "Handwritten Digit Recognition Using k-Nearest Neighbor, Radial Basis Function, and Backpropagation Neural Networks." *Neural Computation* 3: 440–449.

Mao, J., A. K. Jain. 1995. "Artificial Neural Networks for Feature Extraction and Multivariate Data Projection." *IEEE Transactions on Neural Networks* 6: 296–317.

Moody, J., C. Darken. 1989. "Fast Learning in Networks of Locally-Tuned Processing Units." *Neural Computation* 1: 281–294.

Oja, E. 1982. "A Simplified Neuron Model as a Principal Component Analyzer." *Journal of Mathematical Biology* 15: 267–273.

Omohundro, S. M. 1987. "Efficient Algorithms with Neural Network Behavior." *Complex Systems* 1: 273–347.

Platt, J. 1991. "A Resource Allocating Network for Function Interpolation." *Neural Computation* 3: 213–225.

Specht, D. F. 1991. "A General Regression Neural Network." *IEEE Transactions on Neural Networks* 2: 568–576.

Tresp, V., J. Hollatz, S. Ahmad. 1997. "Representing Probabilistic Rules with Networks of Gaussian Basis Functions." *Machine Learning* 27: 173–200.

Bölüm 13

Çekirdek Makineleri

Çekirdek makineleri kenar payını ölçüt olarak kullanır. Model parametreleri öğrenme örneklerinin bir altkümesinin etkilerinin ağırlıklı toplamı olarak yazılır ve bu etkiler uygulamaya özgü bir benzerlik çekirdeğince tanımlanır. Bu bölümde, bu biçimde tanımlanan çekirdek sınıflandırma, bağlanım, aykırı gözlem bulma, ve boyut azaltma yöntemlerini tartışacağız.

13.1 Giriş

Bu bölümde doğrusal sınıflandırma ve bağlanım için farklı bir yaklaşım göreceğiz. Doğrusal model için bile farklı yöntemler görmek bizi şaşırtmasın. Her öğrenme algoritması farklı varsayımlar yapar, iyilemede farklı bir ölçüt kullanır ve dolayısıyla farklı parametreler bulabilir.

Şimdi göreceğimiz ve *destek yöney makinesi* (DYM) denilen ve sonradan *çekirdek makinesi* adıyla genellenen model son yıllarda birkaç nedenden dolayı yaygın olarak kullanılmaktadır:

1. Ayırtaç tabanlı bir yöntemdir ve Vapnik'in savunduğu (1995), "İnsan daha basit bir işi çözmek için ön aşama olarak daha karmaşık bir iş çözmemelidir," düşüncesine dayanır. Örneğin sınıflandırmada ayırtacı öğrenmek için $p(\boldsymbol{x}|C_i)$ sınıf olasılıklarını ya da $P(C_i|\boldsymbol{x})$ sonsal olasılıklarını hassas olarak kestirmek gerekmez; sınırları, yani, $P(C_i|\boldsymbol{x}) = P(C_j|\boldsymbol{x})$ koşulunu sağlayan $\boldsymbol{x}$ değerlerini, bilmek yeter. Benzer biçimde aykırı örnekleri bulmak için $p(\boldsymbol{x})$ olasılığını kestirmeye gerek yoktur; belli (küçük) bir $\theta \in (0, 1)$ eşik değeri için $p(\boldsymbol{x}) < \theta$ koşulunu sağlayan $\boldsymbol{x}$ değerlerinin sınırını bilmek yeter.

2. Öğrenme bittikten sonra doğrusal modelin parametresi olan ağırlık yöneyi, öğrenme kümesinin *destek yöneyleri* denilen bir altkümesi kullanılarak yazılır. Sınıflandırmada bunlar sınıra en yakın örneklerdir; bunları bilmek bilgi verir çünkü bunlar iki sınıfı ayıran sınırı tanımşayan, sınıra en yakın muğlak ya

da hatalı örneklerdir. Kaç tane olduklarına bakarak genelleme hatasını kestirebiliriz. Az sonra göreceğimiz gibi modeli örnekler üzerinden yazabilmek "çekirdekleştirmeye" de izin verir.

3. Çıktı, destek yöneylerininin etkilerinin toplamı olarak yazılır ve bu etki, örnekler arasında uygulamaya özgü bir benzerlik tanımlayan *çekirdek işlevince* belirlenir. Önceki bölümlerde girdiyi doğrusal bir çözümün tanımlanabildiği yeni bir uzaya eşleyen doğrusal olmayan taban işlevlerinden bahsetmiştik; çekirdek işlevi de aynı düşünceye dayanır.

4. Çoğu öğrenme algoritmasında girdiler yöneyler olarak gösterilir ve ardından benzerlik için (çok katmanlı algılayıcılardaki gibi) iç çarpım ya da (dairesel tabanlı işlev ağlarındaki gibi) Euclid uzaklığı kullanılır. Çekirdek işlevi bunun ötesine gitmemizi sağlar. Örneğin, iki çizge G_1 ve G_2 için $K(G_1, G_2)$, bu iki çizgenin benzerliğini ölçen bir ölçüt, örneğin, her iki çizgede de olan ortak yol sayısı olduğunda bu benzerliği çizgeleri yöneyler olarak göstermeye gerek kalmadan hesaplayıp kullanabiliriz.

5. Çekirdek tabanlı algoritmalar dışbükey iyileme problemleri olarak tanımlanır ve en iyi olan tek bir çözüm vardır. Bu yüzden eğim inişteki gibi öğrenme hızı, ilk değer, durma noktası belirleme gibi sorunlarımız yoktur. Bu tabii ki model seçiminde üstparametremiz yok demek değildir; her yöntemin modeli veriye uydurmak için böyle üstparametreleri vardır.

Tartışmamıza sınıflandırmayla başlayıp ardından bağlanım, aykırı örnek (yenilik) bulma, ve boyut azaltmayı göreceğiz. Tüm bu farklı uygulamalarda "kenar payı" dediğimiz ayrılabilirlik ölçütünü artırırken çözümün karmaşıklığını azaltan bir ikinci derece program şablonu göreceğiz. Çözümü bize destek yöneylerini verecek. Her çekirdek işlevi dayandığı benzerlik ölçütüne göre örnekler için bir uzay tanımlar ve bir çekirdek işlevinin iyiliği tanımladığı uzayda ne kadar iyi bir ayrım sağlandığına bağlıdır.

13.2 En İyi Ayıran Üstündüzlem

Yine iki sınıfla başlayalım ve $-1/+1$ bu iki sınıfın etiketlerini göstersin: $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}$ örnekleminde $\boldsymbol{x}^t \in C_1$ ise $r^t = +1$ ve $\boldsymbol{x}^t \in C_2$ ise $r^t = -1$ değerini alsın. $\boldsymbol{w}$ ve w_0 parametrelerinin saplaması gereken koşulları tanımlayalım:

$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{x}^t + w_0 &\geq +1 \quad \text{eğer} \quad r^t = +1 \\ \boldsymbol{w}^T\boldsymbol{x}^t + w_0 &\leq -1 \quad \text{eğer} \quad r^t = -1 \end{aligned}$$

Bu koşulları tek satırda özetleyebiliriz:

$$r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) \geq +1 \tag{13.1}$$

Koşulun

$$r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) \geq 0$$

olmaması önemlidir. Örneklerin üstündüzlemin yalnızca doğru yanında değil, daha iyi genelleme için ona belli bir uzaklıkta kalmasını istiyoruz. Sınırın her iki yanında en yakın örneklerin sınıra uzaklığına *kenar payı* diyoruz ve iyi genelleme için bu kenar payını olabildiğince büyük yapmaya çalışıyoruz.

En en başta, 2.1. bölümde veriye bir dikdörtgen oturturken kenar payı kavramından bahsetmiş ve dikdörtgeni S ve G denencelerinin ortasına koyarak pay bırakmanın daha iyi olacağını savunmuştuk. Böylece eğer gürültü bir örneği hafifçe kıpırdatsa da hâlâ sınırın doğru yanında olma olasılığı yüksek olacaktır.

en iyi ayıran üstündüzlemi, kenar payını en büyük yapan üstündüzlem olarak tanımlıyoruz.

10.3. bölümden, $\boldsymbol{x}^t$ örneğinin ayırtaca uzaklığını hatırlayalım:

$$\frac{|\boldsymbol{w}^T\boldsymbol{x}^t + w_0|}{\|\boldsymbol{w}\|}$$

$r^t \in \{-1, +1\}$ olduğunda bunu her iki yandaki örnekler için yazalım:

$$\frac{r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0)}{\|\boldsymbol{w}\|}$$

Bu değerin en az ρ olmasını istediğimizi düşünelim:

$$\frac{r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0)}{\|\boldsymbol{w}\|} \geq \rho, \forall t \tag{13.2}$$

ρ değerini olabildiğince büyütmek isteriz ama bunu $\boldsymbol{w}$ yöneyini ölçekleyerek yapmak istemeyiz. Tek sonuç olması için $\rho\|\boldsymbol{w}\| = 1$ olarak sabitleyip kenar payını olabildiğince büyütmek için $\|\boldsymbol{w}\|$ değerini (yöneyin boyunu) olabildiğince küçültmek istiyoruz (Cortes ve Vapnik 1995; Vapnik 1995):

$$\min \frac{1}{2}\|\boldsymbol{w}\|^2 \text{ kısıt: } r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) \geq +1, \forall t \tag{13.3}$$

Bu yazdığımız, karmaşıklığı d değerine bağlı ikinci dereceden bir iyileme problemidir ve çözümü bize en iyi $\boldsymbol{w}$ ve w_0 değerlerini verir. Böylece üstündüzlemin her iki yanındaki örnekler sınıra en az $1/\|\boldsymbol{w}\|$ uzaklığında ve kenar payı en az $2/\|\boldsymbol{w}\|$ olacaktır.

10.2. bölümde, eğer sınıflar doğrusal ayrılamıyorsa doğrusal olmayan bir ayırtaç yerine doğrusal olmayan taban işlevleriyle yeni bir uzaya geçebileceğimizi görmüştük. Bu yeni uzayın boyut sayısı çoğu zaman ilk uzayınkinden daha fazla olacağından karmaşıklığı boyut sayısına bağlı olmayan bir öğrenme yöntemi bizim için daha ilginç olacaktır.

En iyi üstündüzlemi bulurken iyileme problemini karmaşıklığı d yerine örnek sayısı olan N değerine bağlı bir biçimde yazabiliriz. Böyle bir yazımın bir faydası

da, bölüm 13.5'te göreceğimiz gibi, taban işlevlerinden çekirdek işlevlerine geçmemizi sağlaması olacaktır.

Bu yeni tanımlama için önce denklem 13.3'ü α^t Lagrange çarpanları kullanarak kısıtlanmamış biçimde yazıyoruz:

$$\begin{aligned} L_p &= \frac{1}{2}\|w\|^2 - \sum_{t=1}^{N} \alpha^t [r^t(w^T x^t + w_0) - 1] \\ &= \frac{1}{2}\|w\|^2 - \sum_t \alpha^t r^t (w^T x^t + w_0) + \sum_t \alpha^t \end{aligned} \tag{13.4}$$

Bu işlevi, w, w_0 değerlerine göre en küçük ve $\alpha^t \geq 0$ değişkenlerine göre en büyük yapan noktayı arıyoruz. Eğer noktası bize çözümü verir.

Bu dışbükey, ikinci dereceden bir iyileme problemidir çünkü hem ana terim, hem de doğrusal kısıtlar dışbükeydir. Bu yüzden yerine Karush-Kuhn-Tucker koşullarını kullanarak eşlek problemi çözebiliriz. Eşlek, L_p problemini α^t değişkenlerine göre büyütmektir ve kısıtlar, L_p işlevinin w ve w_0 parametrelerine göre eğiminin 0 ve $\alpha^t \geq 0$ olmasıdır:

$$\frac{\partial L_p}{\partial w} = 0 \quad \Rightarrow \quad w = \sum_t \alpha^t r^t x^t \tag{13.5}$$

$$\frac{\partial L_p}{\partial w_0} = 0 \quad \Rightarrow \quad \sum_t \alpha^t r^t = 0 \tag{13.6}$$

Bunları denklem 13.4'te yerine koyalım:

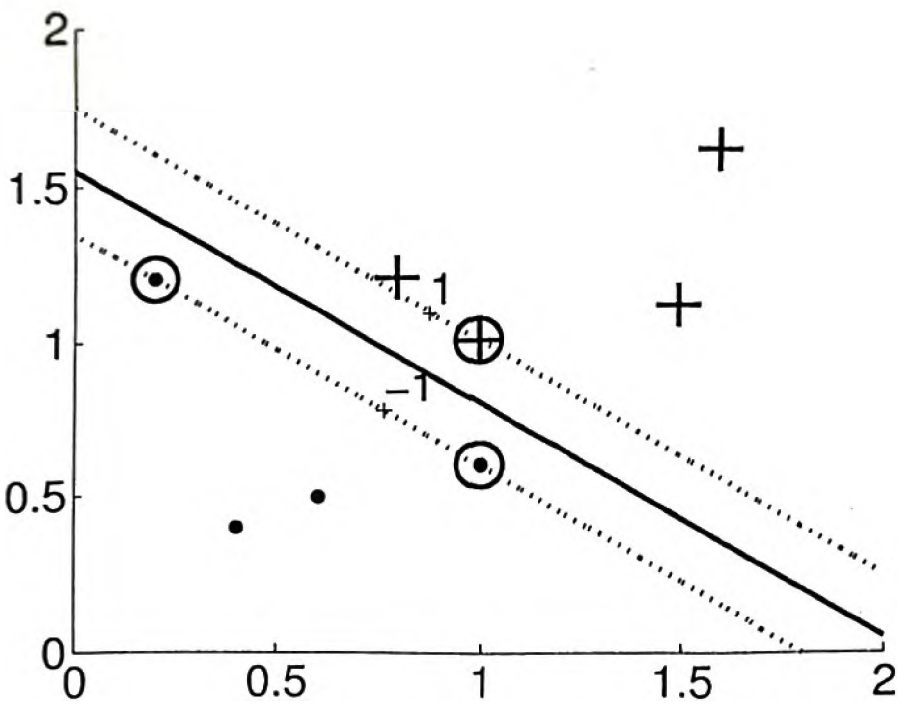
$$\begin{aligned} L_d &= \frac{1}{2}(w^T w) - w^T \sum_t \alpha^t r^t x^t - w_0 \sum_t \alpha^t r^t + \sum_t \alpha^t \\ &= -\frac{1}{2}(w^T w) + \sum_t \alpha^t \\ &= -\frac{1}{2}\sum_t \sum_s \alpha^t \alpha^s r^t r^s (x^t)^T x^s + \sum_t \alpha^t \end{aligned} \tag{13.7}$$

Bu işlevi α^t değişkenlerine göre büyütürken kısıtlarımızı yazalım:

$$\sum_t \alpha^t r^t = 0, \text{ ve } \alpha^t \geq 0, \forall t$$

Bu ikinci dereceden iyilemeyle çözülür. Eşleğin karmaşıklığı örneklem boyu olan N değerine bağlıdır, girdi boyutu olan d değerine bağlı değildir. Hesap karmaşıklığı $\mathcal{O}(N^3)$, bellek karmaşıklığı da $\mathcal{O}(N^2)$ olur.

Çözüp α^t değerlerini bulduğumuzda, N tane α^t değerinin çoğunun 0 olduğunu ve yalnızca küçük bir yüzdesi için $\alpha^t > 0$ olduğunu görürüz. $\alpha^t > 0$ olan x^t örneklerine



Şekil 13.1: Sınıfların artı ve noktayla gösterildiği iki sınıflı bir veride kalın çizgiyle sınır, kesikli çizgilerle de kenar payı gösterilmiştir. Çember içine alınmış örnekler destek yöneyleridir.

destek yöneyleri denir ve denklem 13.5'te gördüğümüz gibi $\boldsymbol{w}$, destek yöneyi olarak seçilen öğrenme örneklerinin ağırlıklı toplamı olarak yazılır. Böylesi $\boldsymbol{x}^t$ örnekleri

$$r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) = 1$$

koşulunu sağlar ve kenarın ucundadır. Bu bilgiyi herhangi bir destek yöneyiyle w_0 değerini hesaplamakta kullanabiliriz:

$$w_0 = r^t - \boldsymbol{w}^T\boldsymbol{x}^t \tag{13.8}$$

Daha güvenilir bir kestirim için bunu tüm destek yöneyleri için yapıp ortalama alırız. Böyle bulunmuş ayırtaca *destek yöney makinesi* (DYM) denir (bkz. şekil 13.1).

α^t değerlerinin çoğu 0 olur, bu örnekler için $r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) > 1$ sağlanır ve doğru sınıflandırılırlar. Bu $\boldsymbol{x}^t$ örnekleri ayırtaçtan yeterince uzak olan örneklerdir ve üstündüzlemi etkilemezler. Destek yöneyi olmayanlar bilgi taşımaz; bunların herhangi bir altkümesini öğrenme kümesinden çıkarsak bile çözüm değişmez. Bu bakış açısından baktığımızda DYM, yalnızca sınıra yakın ve ayırtacı tanımlayan örneklerin saklandığı seyrek en yakın komşu sınıflandırıcısına (bölüm 8.5) benzer.

Ayırtaç tabalı bir yöntem olduğu için DYM yalnızca sınıra yakın örneklerle ilgilenir ve içerideki örnekleri gözardı eder. Böyle olduğuna göre, önce daha basit bir sınıflandırıcı kullanarak sınıf içlerindeki örnekleri bulup öğrenme kümesinden atabilir ve böylece eğitilecek destek yöney makinesinin karmaşıklığını azaltabiliriz (alıştırma 1).

Deneme sırasında yeni bir örnek sınıflandırılırken kenar payına bakılmaz. $g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$ ayırtaç değerini hesaplar ve işaretine göre karar veririz:

$$\text{Seç } C_1 \text{ eğer } g(\boldsymbol{x}) > 0 \text{ ve } C_2 \text{ değilse}$$

13.3 Doğrusal Ayrılamayan Sınıflar: Eşiksiz Kenarlı Üstündüzlem

Eğer veri doğrusal ayrılamıyorsa bahsettiğimiz algoritma çalışmaz. Sınıflar doğrusal ayrılamıyorsa ikisini hatasız ayıran üstündüzlem yoktur, bu durumda en az hata yapanı ararız. Bunun için kenardan sapmayı gösteren $\xi^t \geq 0$ *artık değişkenlerini* tanımlıyoruz. İki tür sapma olabilir: Bir örnek sınırın yanlış yanında olup hatalı sınıflandırılabilir. Ya da doğru yandadır ama yeterince uzak değildir ve kenar payının içine düşer. Denklem 13.1'e artık değişkenini ekleyip yeniden yazalım:

$$r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) \geq 1 - \xi^t \tag{13.9}$$

$\xi^t = 0$ ise $\boldsymbol{x}^t$ ile ilgili bir sorun yoktur. $0 < \xi^t < 1$ ise $\boldsymbol{x}^t$ doğru sınıflandırılıyordur ama kenar payının içindedir. $\xi^t \geq 1$ ise $\boldsymbol{x}^t$ yanlış taraftadır ve yanlış sınıflandırılmıştır (bkz. şekil 13.2). $\#\{\xi^t > 1\}$, yanlış sınıflandırılan örnek sayısı ve

$$\sum_t \xi^t$$

doğrusal ayrılamayan örnek sayısını verir. Bu *eşiksiz hata* ölçütünü bir ceza terimi olarak problemimize ekleriz:

$$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_t \xi^t \tag{13.10}$$

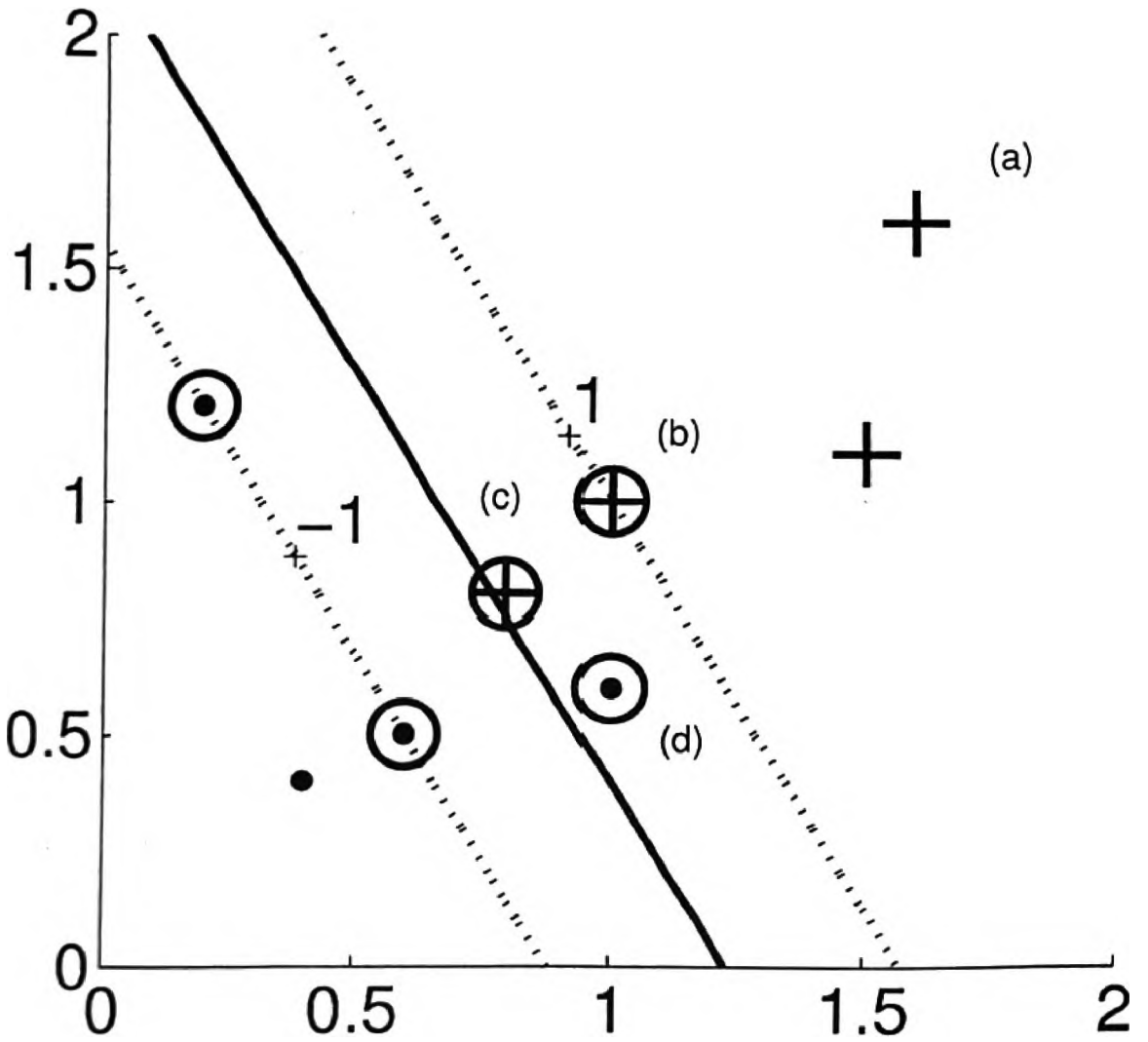
Kısıtlar, denklem 13.9 ile tanımlanmıştır. C, herhangi bir düzgünleştirme uygulamasında olduğu gibi karmaşıklıkla hatanın görece önemini söyler. Karmaşıklığı burada ağırlık yöneyinin L_2 normu olarak ölçüyoruz (çok katmanlı algılayıcılardaki ağırlık sönümü de benzer bir ceza terimi kullanır; bkz. bölüm 11.9). Hata ise doğrusal ayrılamayan örnek sayısıdır; yalnızca hatalı sınıflandırılan örneklere değil iyi genelleme için (doğru sınıflandırılacak bile olsalar) kenara düşen örneklere de bakıyoruz.

Yeni kısıtları ekleyip denklem 13.4'ü yeniden yazalım:

$$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_t \xi^t - \sum_t \alpha^t[r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) - 1 + \xi^t] - \sum_t \mu^t\xi^t \tag{13.11}$$

Burada, μ_t ile gösterdiğimiz ξ^t değerlerinin eksi olmamasını sağlayan yeni Lagrange parametreleridir. Türev alıp sıfıra eşitleyelim:

$$\frac{\partial L_p}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_t \alpha^t r^t \boldsymbol{x}^t = 0 \Rightarrow \boldsymbol{w} = \sum_t \alpha^t r^t \boldsymbol{x}^t \tag{13.12}$$



Şekil 13.2: Bir örneği sınıflandırırken olası dört durum vardır: (a) örnek ayırtacın doğru yanında ve kenardan uzak; $r^t g(\boldsymbol{x}^t) > 1$, $\xi^t = 0$. (b) $\xi^t = 0$; örnek doğru yanda ve kenarın üstünde. (c) $\xi^t = 1 - g(\boldsymbol{x}^t), 0 < \xi < 1$; örnek doğru yanda ama kenarın içinde, ayırtaca yeterince uzak değil. (d) $\xi^t = 1 + g(\boldsymbol{x}^t) > 1$; örnek ayırtacın yanlış yanında ve yanlış sınıflandırılmış. (a) dışındaki tüm örnekler destek yöneyidir. Eşlek değişken olarak yazarsak (a) $\alpha^t = 0$; (b) $\alpha^t < C$; (c) ve (d) $\alpha^t = C$.

$$\frac{\partial L_p}{\partial w_0} = \sum_t \alpha^t r^t = 0 \tag{13.13}$$

$$\frac{\partial L_p}{\partial \xi^t} = C - \alpha^t - \mu^t = 0 \tag{13.14}$$

$\mu^t \geq 0$ olduğu için son satır $0 \leq \alpha^t \leq C$ koşulunu gerektirir. Bunları denklem 13.11'e koyduğumuzda α^t değişkenlerine göre büyüteceğimiz eşleği elde ederiz:

$$L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s (\boldsymbol{x}^t)^T \boldsymbol{x}^s \tag{13.15}$$

Kısıtlar:

$$\sum_t \alpha^t r^t = 0 \text{ ve } 0 \leq \alpha^t \leq C, \forall t$$

Bunu çözdüğümüzde doğrusal ayrılabilen durumda olduğu gibi sınırın doğru yanında ve yeterince uzak olan örnekler için $\alpha^t = 0$ olur (bkz. şekil 13.2). Destek yöneyleri $\alpha^t > 0$ olan örneklerdir ve denklem 13.12'dekş gibi $\boldsymbol{w}$ ağırlık yöneyini tanımlarlar. Bunların içinden $\alpha^t < C$ olanlar kenarın üstündedir ve w_0 değerini hesaplamakta kullanılırlar bunlar için $\xi^t = 0$ olur ve $r^t(\boldsymbol{w}^T \boldsymbol{x}^t + w_0) = 1$ eşitliğini sağlarlar. Burada da bunu birçok farklı örnek için yapıp ortalama almak yeğdir. Kenar payının içine düşen ya da yanlış sınıflandırılan örnekler için $\alpha^t = C$ olur.

Destek yöneyi olarak sakladığımız örnekler öğrenme kümesi içinde olmasalardı sınıflandırırken sorun yaşayacağımız örneklerdir; bu örnekler ya yanlış sınıflandırılır, ya da doğru sınıflandırılsalar da sınıra çok yakındırlar ve yeterince güvenle karar vermeyiz. Destek yöney sayısının beklenen hata için bir üst sınır oluşturduğu söylenebilir. Gerçekten de Vapnik (1995) beklenen hata oranı için böyle bir sınır olduğunu göstermiştir:

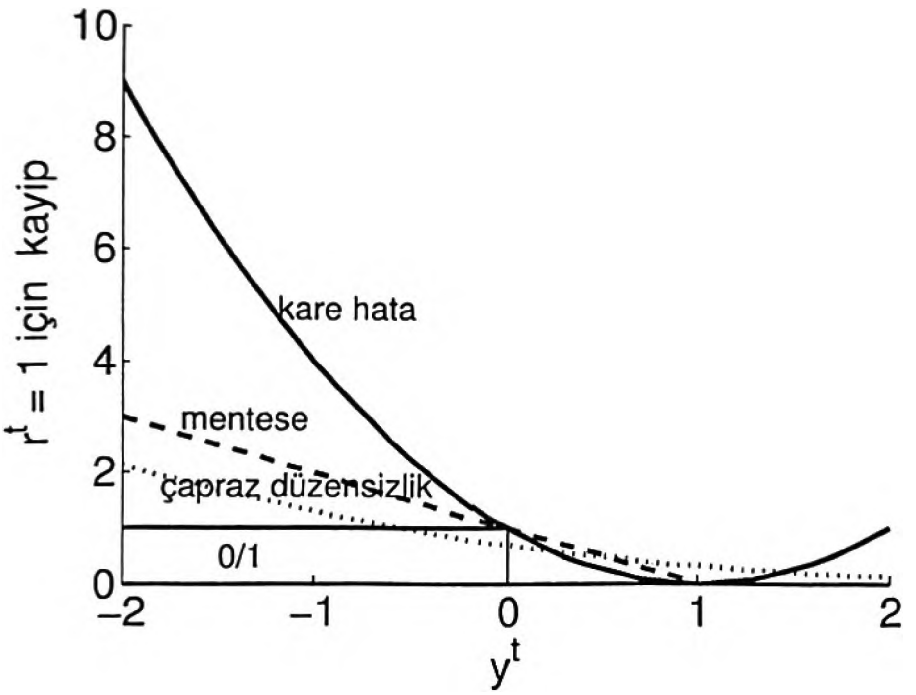
$$E_N[P(\text{hata})] \leq \frac{E_N[\#\text{ destek yöneyi}]}{N}$$

ki burada $E_N[\cdot]$, N büyüklüğündeki öğrenme kümeleri üzerinde beklenen değeri gösterir. Bunun ilginç yanı hatanın girdi boyutuna değil destek yöneyi sayısına bağlı olmasıdır.

Eğer örnek yanlış taraftaysa, ya da kenar birden azsa denklem 13.9' göre hatalıdır. Buna *menteşe kayıp işlevi* denir. $y^t = \boldsymbol{w}^T \boldsymbol{x}^t + w_0$ çıktısı ve r^t istenen değeri için menteşe kayıp işlevini yazalım:

$$L_m(y^t, r^t) = \begin{cases} 0 & \text{eğer } y^t r^t \geq 1 \\ 1 - y^t r^t & \text{değilse} \end{cases} \tag{13.16}$$

Şekil 13.3'te menteşe kayıp işlevini, 0/1 kayıp, kare hata, ve çapraz düzensizlikle görsel olarak karşılaştırıyoruz. 0/1 kayıp işlevinden farklı olarak menteşe kayıp işlevinin kenar payındaki örnekleri de (doğru olmalarına rağmen) cezalandırdığını ve cezanın yanlış yanda sınırdan uzaklaştıkça doğrusal olarak arttığını görüyoruz. Kare hata bu ikinci açıdan farklıdır ve dolayısıyla aykırı örneklere menteşe hata kadar gürbüz değildir. S biçimli sınıflandırma (bölüm 10.7) ve doğrusal algılayıcının



Şekil 13.3: $r^t = 1$ için farklı kayıp işlevlerinin karşılaştırılması: 0/1 yitimi, $y^t = 1$ ise 0, değilse 1 olur. Menteşe (m) kayıp, $y^t > 1$ ise 0, değilse $1 - y^t$ olur. Kare hata (k), $(1 - y^t)^2$ ve çapraz düzensizlik (d) $\log(1/(1 + \exp(-y^t)))$ olarak tanımlıdır.

(bölüm 11.3) azalttığı çapraz düzensizliğin menteşe kayıp işlevine oldukça benzediğini görüyoruz.

Denklem 13.10'daki C parametresi çapraz geçerlemeyle uyarlanır. Kenar payını artırmakla hatayı azaltmak arasındaki görece ağırlığı belirler: Çok büyük olursa ayrılamayan örneklerin cezası fazla olur, çok destek yöneyi saklamak gerekir ve aşırı öğrenme olabilir. Çok küçük olursa öğrenme kümesinde bile başarılı olmayan fazla basit çözümler bulabiliriz. Genelde, $[10^{-6}, 10^{-5}, \ldots, 10^{+5}, 10^{+6}]$ içinde log ölçeğinde denemeler yapıp geçerleme kümesinde en yüksek başarım göstereni seçeriz.

13.4 ν-DYM

Eşiksiz kenarlı üstündüzleme eşdeğer, C yerine $\nu \in [0, 1]$ parametresi kullanan bir tanımlama da vardır. Amaç işlevi

$$\min \frac{1}{2}\|w\|^2 - \nu\rho + \frac{1}{N}\sum_t \xi^t \tag{13.17}$$

ve kısıtları

$$r^t(w^T x^t + w_0) \geq \rho - \xi^t,\ \xi^t \geq 0,\ \rho \geq 0 \tag{13.18}$$

biçiminde yazılır. Yeni parametre ρ, kenar payını ölçekler: kenar payı, $2\rho/\|w\|$ değerine eşittir. ν değerinin destek yöney oranı için bir alt sınır, eşiksiz kenar payı hatasını

$(\sum_t \#\{\xi^t > 0\})$ oluşturan örneklerin oranı içinse bir üst sınır olduğu gösterilmiştir. Eşlek

$$L_d = -\frac{1}{2}\sum_t \sum_s \alpha^t \alpha^s r^t r^s (\boldsymbol{x}^t)^T \boldsymbol{x}^s \tag{13.19}$$

ve kısıtlarını yazalım:

$$\sum_t \alpha^t r^t = 0,\ 0 \leq \alpha^t \leq \frac{1}{N},\ \sum_t \alpha^t \leq \nu$$

Denklem 13.19'u denklem 13.15'le karşılaştırırsak $\sum_t \alpha^t$ teriminin artık amaç işlevinde olmadığını ve bir kısıta dönüştüğünü görüyoruz. ν ile oynayarak destek yöneylerinin oranını belirleyebiliriz ve bu yüzden bu parametrenin C parametresine göre daha anlaşılır olduğu söylenir.

13.5 Çekirdeklemek

Bölüm 10.2'de, eğer problem doğrusal değilse, doğrusal olmayan bir model oturtmak yerine doğrusal olmayan taban işlevleriyle yeni bir uzaya gidip orada doğrusal bir model öğrenebileceğimizi konuştuk. Yeni uzaydaki doğrusal model ilk uzayda doğrusal olmayan bir modele karşılık gelir. Bu yaklaşımı hem sınıflandırma, hem de bağlanım için, herhangi bir doğrusal modelle kullanabiliriz. Destek yöney makineleri özel durumunda bu yaklaşım şimdi göreceğimiz gibi işimizi oldukça kolaylaştırır.

Yeni boyutlarımızın taban işlevleriyle tanımlandığını varsayalım:

$$\boldsymbol{z} = \boldsymbol{\phi}(\boldsymbol{x}) \text{ öyle ki } z_j = \phi_j(\boldsymbol{x}), j = 1, \ldots, k$$

ve böylece d boyutlu $\boldsymbol{x}$ uzayından k boyutlu $\boldsymbol{z}$ uzayına geçip ayırtacı bu yeni uzayda yazalım:

$$\begin{aligned} g(\boldsymbol{z}) &= \boldsymbol{w}^T \boldsymbol{z} \\ g(\boldsymbol{x}) &= \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \\ &= \sum_{j=1}^{k} w_j \phi_j(\boldsymbol{x}) \end{aligned} \tag{13.20}$$

Ayrı bir w_0 değeri kullanmayıp $z_1 = \phi_1(\boldsymbol{x}) \equiv 1$ olarak ek bir yeni boyut olduğunu varsayıyoruz. Genelde k değeri, d ya da N değerinden çok daha büyüktür, ve bu yüzden karmaşıklığı N değerine bağlı olan eşlek tanımı kullanmak, karmaşıklığı k değerine bağlı olan asal tanımı kullanmaya yeğlenir. Bu yeni uzayda doğrusal ayrılabilirlik güvencemiz olmadığı için daha genel olan eşiksiz kenarlı üstündüzlem yaklaşımını kullanalım.

İyileme problemi yine aynıdır:

$$L_p = \frac{1}{2}\|w\|^2 + C\sum_t \xi^t \tag{13.21}$$

Tek fark, kısıtların yeni uzayda tanımlı olmasıdır:

$$r^t w^T \phi(x^t) \geq 1 - \xi^t \tag{13.22}$$

Lagrange terimlerini ekleyelim:

$$L_p = \frac{1}{2}\|w\|^2 + C\sum_t \xi^t - \sum_t \alpha^t \left[r^t w^T \phi(x^t) - 1 + \xi^t\right] - \sum_t \mu^t \xi^t \tag{13.23}$$

Yine parametrelere göre türev alıp sıfıra eşitleyelim:

$$\frac{\partial L_p}{\partial w} = w = \sum_t \alpha^t r^t \phi(x^t) \tag{13.24}$$

$$\frac{\partial L_p}{\partial \xi^t} = C - \alpha^t - \mu^t = 0 \tag{13.25}$$

Bu durumda eşleği

$$L_d = \sum_t \alpha^t - \frac{1}{2}\sum_t \sum_s \alpha^t \alpha^s r^t r^s \phi(x^t)^T \phi(x^s) \tag{13.26}$$

ve kısıtları

$$\sum_t \alpha^t r^t = 0 \text{ ve } 0 \leq \alpha^t \leq C, \forall t$$

yazalım.

Çekirdek makinelerinde ana düşünce, taban işlevlerinde $\phi(x^t)^T \phi(x^s)$ iç çarpımını ilk girdiler üzerinde bir $K(x^t, x^s)$ *çekirdek işlevi* olarak yazmaktır. Yani, x^t ve x^s girdilerini önce z uzayına eşleyip orada iç çarpmak yerine çekirdek işlevini doğrudan ilk uzaydaki gösterimlere uyguluyoruz.

$$L_d = \sum_t \alpha^t - \frac{1}{2}\sum_t \sum_s \alpha^t \alpha^s r^t r^s K(x^t, x^s) \tag{13.27}$$

Çekirdek işlevi ayırtaçta da kendini gösteriyor:

$$\begin{aligned} g(x) &= w^T \phi(x) = \sum_t \alpha^t r^t \phi(x^t)^T \phi(x) \\ &= \sum_t \alpha^t r^t K(x^t, x) \end{aligned} \tag{13.28}$$

Demek ki eğer elimizde çekirdek işlevi zaten varsa yeni uzaya eşlememize hiç gerek yoktur. Gerçekte her geçerli çekirdek için karşılık gelen bir taban işlevi vardır ama $K(x^t, x)$ değerini hesaplayıp kullanmak, $\phi(x^t)$ ve $\phi(x)$ değerlerini hesaplayıp kullanmaktan çok daha kolay olabilir. Az sonra göreceğimiz gibi birçok algoritma bu biçimde *çekirdeklenebilir*, ve "çekirdek makineleri" adı, bu tür algoritmaları kapsayan genel ad olarak kullanılır.

$\mathbf{K}_{ts} = K(\boldsymbol{x}^t, \boldsymbol{x}^s)$ çekirdek değerlerinden oluşan $\mathbf{K}$ çekirdek dizeyine *Gram dizeyi* de denir; bakışımlı ve yarı kesin artı olması gerekir. Son yıllarda veri kümelerini $\boldsymbol{x}^t$ ya da $\phi(\boldsymbol{x}^t)$ yerine doğrudan $\mathbf{K}$ çekirdek dizeyleri olarak saklamak yaygın bir uygulama olmaya başlamıştır. Özellikle biyoinformatik ya da doğal dil işleme gibi $\boldsymbol{x}$ (ya da $\phi(\boldsymbol{x})$) gösteriminin yüzler ya da binlerce boyutlu olduğu durumlarda $N \times N$ dizeyi saklamak ya da indirmek çok daha kolaydır (Vert, Tsuda, ve Schölkopf 2004). Ancak bu durumda yalnızca elimizdeki örnekleri kullanabiliriz ve bu küme dışından bir örnek için öngörü yapamayız.

13.6 Yöney Çekirdekleri

Sıkça kullanılan genel amaçlı çekirdekler vardır:

- *Çokterimli çekirdek*: Derece q için

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = (\boldsymbol{x}^T \boldsymbol{x}^t + 1)^q \tag{13.29}$$

  olarak yazılır ve q değerinin üstparametre olarak önceden belirlenmesi gerekir. Örneğin, $q = 2$ ve $d = 2$ için

$$\begin{aligned} K(\boldsymbol{x}, \boldsymbol{y}) &= (\boldsymbol{x}^T \boldsymbol{y} + 1)^2 \\ &= (x_1 y_1 + x_2 y_2 + 1)^2 \\ &= 1 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 x_2 y_1 y_2 + x_1^2 y_1^2 + x_2^2 y_2^2 \end{aligned}$$

  çekirdeği

$$\phi(\boldsymbol{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2]^T$$

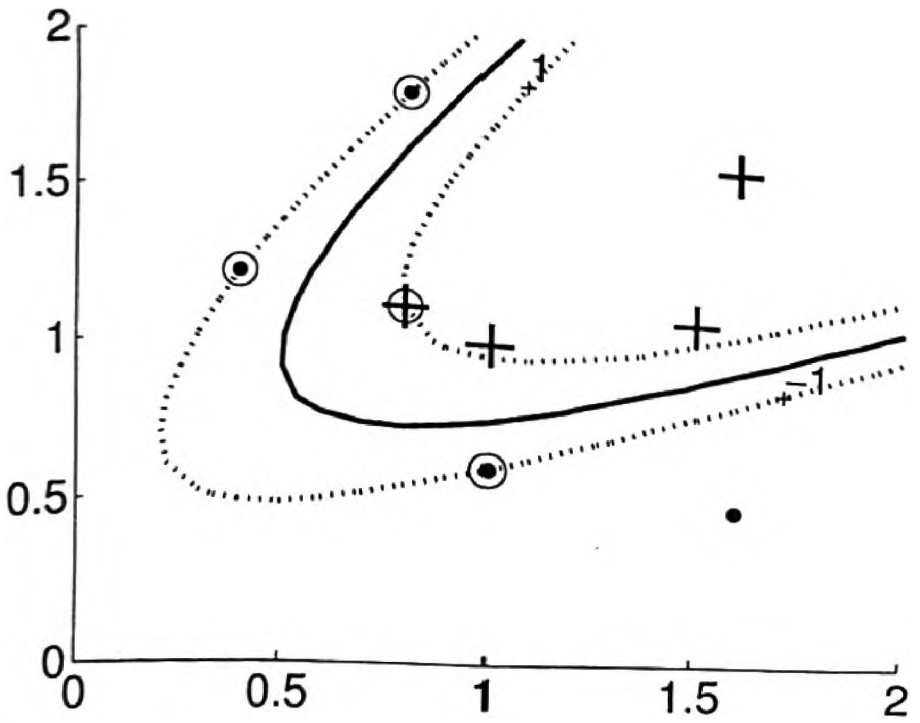
  taban işlevinin iç çarpımına karşılık gelir (Cherkassky ve Mulier 1998). Örnek, şekil 13.4'te gösterilmiştir. $q = 1$ olduğundaki *doğrusal çekirdek* ilk gösterime eşittir.

- *Dairesel tabanlı (Gauss) işlev çekirdeği*:

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = \exp\left[-\frac{\|\boldsymbol{x}^t - \boldsymbol{x}\|^2}{2s^2}\right] \tag{13.30}$$

  Parzen penceresi (bölüm 8) gibi küresel bir çekirdek tanımlar; $\boldsymbol{x}^t$ merkezi, önceden sabitlenen s de yarıçapı belirler. Bu yaklaşım, 12. bölümde gördüğümüz dairesel tabanlı işlev ağlarına da benzer.

  Bir örnek, şekil 13.5'te gösterilmiştir. Orada gördüğümüz gibi yarıçap büyük olduğunda ayırtaç daha düz olur; en iyi değer çapraz geçerlemeyle bulunur. Çapraz geçerlemeyle iki üstparametreyi birlikte ayarlarken (örneğin burada C ve s^2), olası tüm değer çiftleri için iki boyutta ızgara biçiminde (etkensel) bir



Şekil 13.4: İkinci dereceden çokterimli çekirdek için ayırtaç ve kenarlar gösterilmiştir. Çember içine alınmış örnekler destek yöneyleridir.

arama yapmamız gerekir; etkenleri bu biçimde ayarlamanın nasıl yapıldığını 19.2. bölümde daha ayrıntılı göreceğiz.

Euclid yerine S eşdeğişinti dizeyine bağlı Mahalanobis uzaklığı kullanan bir çekirdek de kullanabiliriz:

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = \exp\left[-\frac{1}{2}(\boldsymbol{x}^t - \boldsymbol{x})^T \mathbf{S}^{-1}(\boldsymbol{x}^t - \boldsymbol{x})\right] \tag{13.31}$$

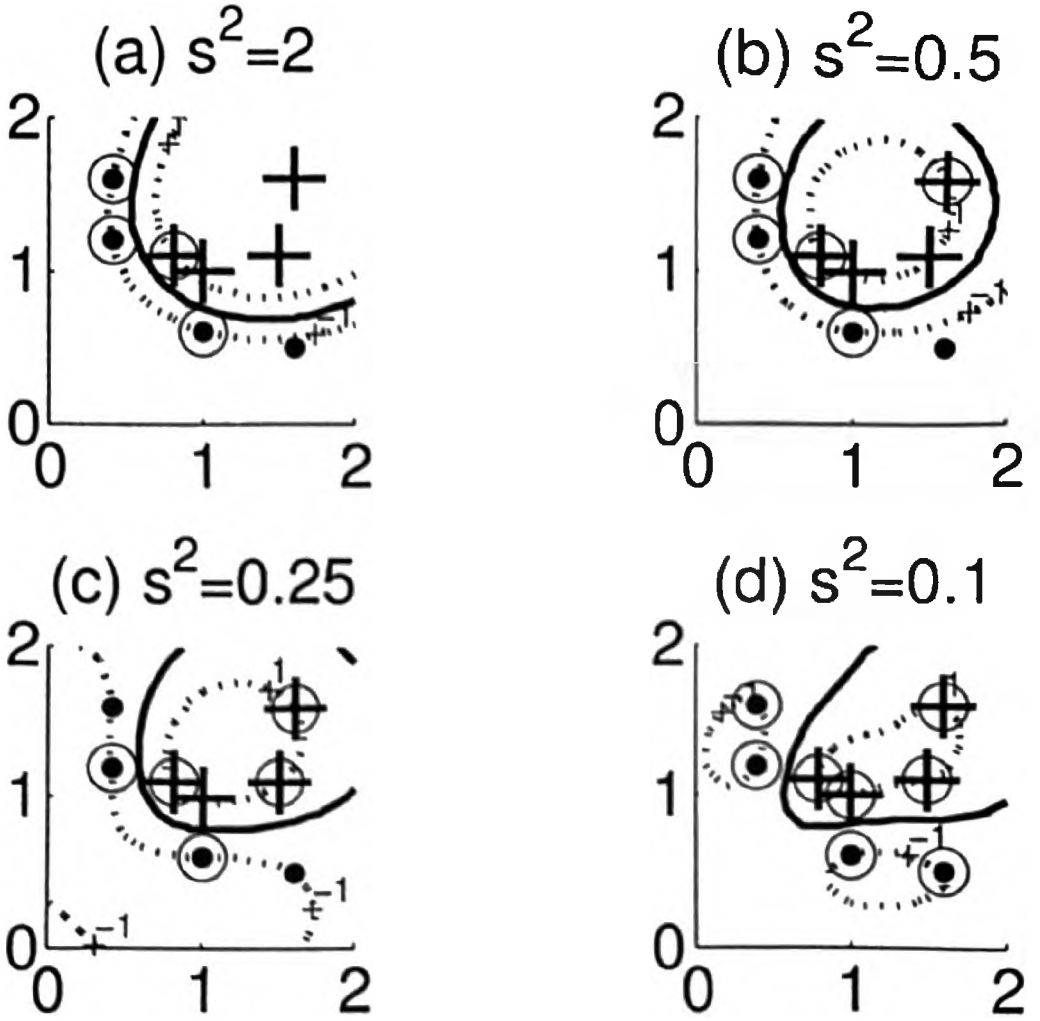
Ya da daha genel olarak, bir $\mathcal{D}(\boldsymbol{x}^t, \boldsymbol{x})$ uzaklık ölçütü kullanan bir çekirdek işlevi tanımlayabiliriz:

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = \exp\left[-\frac{\mathcal{D}(\boldsymbol{x}^t, \boldsymbol{x})}{2s^2}\right] \tag{13.32}$$

- *S biçimli çekirdek*:

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = \tanh(2\boldsymbol{x}^T\boldsymbol{x}^t + 1) \tag{13.33}$$

tanh(·), sigmoid(·) gibi S biçimlidir ama −1 ve +1 arasında bir değer alır. Bu yapı, 11. bölümde gördüğümüz çok katmanlı algılayıcılara benzer.



Şekil 13.5: Gauss çekirdeğiyle farklı s^2 yarıçaplarıyla bulunan sınır ve kenarlar gösterilmiştir. Yarıçap büyüdükçe sınırlar yumuşar.

13.7 Çekirdek Tanımlama

Uygulamaya bağlı çekirdekler de tanımlanabilir. Çekirdeği benzerlik ölçütü gibi düşünebiliriz; $\boldsymbol{x}$ ve $\boldsymbol{y}$, uygulamaya bağlı olarak daha "benzer" oldukça $K(\boldsymbol{x}, \boldsymbol{y})$ daha yüksek bir değer alacaktır. Bu biçimde uygulamayla ilgili önsel bilgimizi uygun tanımlanmış çekirdeklerle modele yansıtabiliriz; buna "çekirdek mühendisliği" denir. Bu açıdan baktığımızda önsel bilgiyle çekirdek tanımlamak bir tür "ipucudur" (bölüm 11.8.4).

Karakter dizileri, ağaçlar, çizgeler, vs gibi çok farklı yapılar için veri yapısında ve benzerliği nasıl ölçtüğümüze göre farklı çekirdekler tanımlanmış ve kullanılmaktadır (Vert, Tsuda, ve Schölkopf 2004).

Örneğin iki belgede ortak geçen sözcük sayısı bir çekirdek olabilir. D_1 ve D_2, iki belge olsun ve *sözcük torbası* gösterimini kullandığımızı varsayalım: Önceden uygulamaya göre M sözcük seçelim ve $\boldsymbol{\phi}(D_1)$, M boyutlu ikili yöney olsun, öyle ki, i. sözcük belgede geçiyorsa i. boyutu 1, geçmiyorsa 0 değerini alsın. Bu durumda, $\boldsymbol{\phi}(D_1)^T\boldsymbol{\phi}(D_2)$, ortak sözcükleri sayar. Burada doğrudan $K(D_1, D_2)$ çekirdeğini yazıp kullanmanın üstünlüğü, herhangi bir sözcüğü (çok sık geçen ve bilgi vermeyen, "bir," "ve," vs, gibi sözcükleri gözardı edip) kullanabilmemiz ve sözcük torbası gösterimini oluşturmaya hiç gerek olmamasıdır; böylece M istediğimiz kadar büyük olabilir.

Bazen, örneğin biyoinformatik uygulamalarında, iki örnek arasında bir *benzerlik değeri* hesaplayabiliriz ama oluşan dizey, yarı kesin artı olmayabilir. İki (gen) dizisi için olası bir çekirdek, *düzeltme uzaklığını*, yani dizinlerden birinin ötekine kaç tane ekleme, çıkarma, yerine koymayla dönüştürülebildiğini kullanabilir; bu işleme *hizalama* denir. Böyle bir durumda bir olasılık, M şablon örnek belirlemek ve bir örneğin bu şablonlara benzerliklerini M boyutlu bir yöney olarak almaktır. $\boldsymbol{m}_i, i = 1, \ldots, M$, şablonları ve $s(\boldsymbol{x}^t, \boldsymbol{m}_i)$, $\boldsymbol{x}^t$ ve $\boldsymbol{m}_i$ arasındaki benzerlik değerini göstersin. Bu durumda, taban işlev yöneyi

$$\boldsymbol{\phi}(\boldsymbol{x}^t) = [s(\boldsymbol{x}^t, \boldsymbol{m}_1), s(\boldsymbol{x}^t, \boldsymbol{m}_2), \ldots, s(\boldsymbol{x}^t, \boldsymbol{m}_M)]^T$$

olur. *Gözlemsel çekirdek gösterimi* bu yöneylerin iç çarpımıdır

$$K(\boldsymbol{x}^t, \boldsymbol{x}^s) = \boldsymbol{\phi}(\boldsymbol{x}^t)^T\boldsymbol{\phi}(\boldsymbol{x}^s)$$

ve geçerli bir çekirdektir.

Bazen benzerlik ölçütü ikili değer alır; örneğin iki protein, ya etkileşir, ya da etkileşmez. Bu bilgiden hareketle herhangi iki örnek için benzerlik hesaplayabilmek isteriz. Böyle bir durumda bir yaklaşım, düğümleri örnekler olan bir çizge tanımlamak ve benzer olduğunu bildiğimiz örneklerin düğümlerini bir ayrıtla bağlamaktır. Sonra, iki düğüm bağlı olmasa da aralarındaki en kısa yol kısaysa, ya da aralarında farklı birçok yol varsa, karşılık gelen örneklerin "benzer" olduğu söylenebilir. Bu sayede yerel etkileşimlerden küresel bir benzerlik ölçütü oluşturulur; bu bir açıdan eşölçümsel öznitelik eşleme algoritmasındaki (bölüm 6.7) yerölçümsel uzaklığa benzer. Buna *yayınım çekirdeği* denir.

Bir $p(\boldsymbol{x})$ olasılık dağılımı verildiğinde

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = p(\boldsymbol{x}^t)p(\boldsymbol{x})$$

geçerli bir çekirdektir. Örneğin $p(\boldsymbol{x})$, $\boldsymbol{x}$ örneğinin görülme olasılığını gösteren üretici bir model olabilir. Örneğin, $\boldsymbol{x}$ bir dizi ve $p(\boldsymbol{x})$ ise saklı bir Markov modeli (bölüm 15) olabilir. Bu durumda, $\boldsymbol{x}^t$ ve $\boldsymbol{x}$ örneklerinin ikisinin de aynı modelden üretilmiş olma olasılığı yüksekse $K(\boldsymbol{x}^t, \boldsymbol{x})$ çekirdeği yüksek bir değer alacaktır. Bu üretici modele parametreler eklenip $p(\boldsymbol{x}|\theta)$ olarak yazılabilir ve θ parametresi veriden öğrenilebilir; buna *Fisher çekirdeği* denir (Jaakkola ve Haussler 1998).

13.8 Çekirdek Birleştirme

Basit çekirdekleri birleştirerek yeni çekirdekler oluşturabiliriz. İki geçerli $K_1(\boldsymbol{x}, \boldsymbol{y})$ ve $K_2(\boldsymbol{x}, \boldsymbol{y})$ çekirdeği ve c sabiti için

$$K(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} cK_1(\boldsymbol{x}, \boldsymbol{y}) \\ K_1(\boldsymbol{x}, \boldsymbol{y}) + K_2(\boldsymbol{x}, \boldsymbol{y}) \\ K_1(\boldsymbol{x}, \boldsymbol{y}) \cdot K_2(\boldsymbol{x}, \boldsymbol{y}) \end{cases} \tag{13.34}$$

geçerli çekirdeklerdir.

Farklı çekirdekler, $\boldsymbol{x}$ yöneyinin içerdiği özniteliklerin farklı altkümelerini kullanabilir. Bu açıdan baktığımızda çekirdek birleştirmek farklı kaynaklardan gelen bilgiyi birleştirmek için bir yol olarak görülebilir; her kaynak kendi gösterimine uygun farklı bir çekirdek kullanır ve bu çekirdeklerin değerleri birleştirilir. A ve B iki farklı gösterim yöneyi olsun:

$$\begin{aligned} K_A(\boldsymbol{x}_A, \boldsymbol{y}_A) + K_B(\boldsymbol{x}_B, \boldsymbol{y}_B) &= \boldsymbol{\phi}_A(\boldsymbol{x}_A)^T \boldsymbol{\phi}_A(\boldsymbol{y}_A) + \boldsymbol{\phi}_B(\boldsymbol{x}_B)^T \boldsymbol{\phi}_B(\boldsymbol{y}_B) \\ &= \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{y}) \\ &= K(\boldsymbol{x}, \boldsymbol{y}) \end{aligned} \tag{13.35}$$

Burada, $\boldsymbol{x} = [\boldsymbol{x}_A, \boldsymbol{x}_B]$, iki gösterim yöneyinin uç uca eklendiği yeni (ve daha uzun) yöneydir. Bu da gösteriyor ki çekirdekleri toplamak uç uca eklenmiş yöneylerde iç çarpım yapmaya eşittir. Bunu istediğimiz sayıda çekirdek için genelleyebiliriz:

$$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{m} K_i(\boldsymbol{x}, \boldsymbol{y}) \tag{13.36}$$

ve sınıflandırıcı çıktılarını birleştirmeye benzer (bölüm 17.4) biçimde çekirdekleri birleştirir; bu da bizi tek bir çekirdek seçme zorunluluğundan kurtarır. Ağırlıklı bir toplam almak ve bu ağırlıkları öğrenmek de önerilmiştir (Lanckriet ve ötekiler 2004; Sonnenburg ve ötekiler 2006):

$$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{m} \eta_i K_i(\boldsymbol{x}, \boldsymbol{y}) \tag{13.37}$$

$\eta_i \geq 0$ kısıtı vardır, üstüne $\sum_i \eta_i = 1$ kısıtı da olabilir; buna sırasıyla dışbükey ya da konik birleşim denir. Bu yaklaşıma, yani bir çekirdek yerine çekirdeklerin ağırlıklı toplamını kullanmaya genel olarak *çok çekirdekli öğrenme* denir. Denklem 13.27'deki tek çekirdekli amaç işlevi

$$L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s \sum_i \eta_i K_i(\boldsymbol{x}^t, \boldsymbol{x}^s) \tag{13.38}$$

olarak yazılır ve çözüm bize hem destek yöney makinesi parametreleri olan α^t değerlerini, hem de çekirdek ağırlıkları olan η_i değerlerini verir. Çekirdeklerin ağırlıklı toplamı ayırtaçta da ortaya çıkar:

$$g(\boldsymbol{x}) = \sum_t \alpha^t r^t \sum_i \eta_i K_i(\boldsymbol{x}^t, \boldsymbol{x}) \tag{13.39}$$

Öğrenme bittikten sonraki η_i değerinin $K_i(\boldsymbol{x}^t, \boldsymbol{x})$ çekirdeğinin sınıflandırmada ne kadar önemli olduğunu gösterdiği söylenebilir. Ağırlıkları $\boldsymbol{x}$ girdisinin bir işlevi olarak yazıp çekirdekleri yerelleştirebiliriz; bu uzmanların karışımındaki geçit işlevine benzer (bölüm 17.8)

$$g(\boldsymbol{x}) = \sum_t \alpha^t r^t \sum_i \eta_i(\boldsymbol{x}|\theta_i) K_i(\boldsymbol{x}^t, \boldsymbol{x}) \tag{13.40}$$

ve geçit parametreleri olan θ_i değerleri, destek yöney makineleriyle birlikte öğrenilebilir (Gönen ve Alpaydın 2008).

Farklı kaynaklardan farklı kiplerde bilgi, örneğin konuşma tanımada ses ve dudak hareketi, geldiğinde genelde yapılan bunları ayrı sınıflandırıcılara verip kararlarını birleştirmektir; bunu 17. bölümde ayrıntılı olarak göreceğiz. Farklı çekirdekleri birleştiren tek bir sınıflandırıcı eğitmek farklı kaynaklardan gelen bilgiyi birleştirmek için bir başka yol sunar (Noble 2004). Yerel yaklaşım, girdiye bağlı olarak farklı kaynaklar arasında, yani farklı benzerlik ölçütleri arasında seçim yapmamızı sağlar.

13.9 Çok Sınıflı Çekirdek Makinesi

$K > 2$ sınıf varsa en kolayı K tane iki sınıf problemi tanımlayarak her sınıfı öteki tüm sınıflardan ayıran K ayrı $g_i(\boldsymbol{x}), i = 1, \ldots, K$ destek yöney makinesi eğitmektir. Yani, $g_i(\boldsymbol{x})$ ayırtacını eğitirken C_i sınıfından örnekler $+1$, tüm öteki $C_k, k \neq i$ sınıflarından örneklerse -1 olarak etiketlenir. Deneme sırasında tüm $g_i(\boldsymbol{x})$ değerleri hesaplanır ve en büyük olan seçilir.

Platt (1999), tek (2 sınıflı) bir DYM çıktısını S biçimli bir işlevli doğrusal modelle sonsal olasılığa çevirmeyi önermiştir. Benzer biçimde, $K > 2$ tane DYM'nin çıktısı eşiksiz en büyük işlevli doğrusal bir modelle çapraz düzensizliği azaltacak biçimde eğitilip sonsal olasılıklar kestirilebilir (Mayoraz ve Alpaydın 1999):

$$y_i(\boldsymbol{x}) = \sum_{j=1}^{K} v_{ij} f_j(\boldsymbol{x}) + v_{i0} \tag{13.41}$$

Burada, $f_j(\boldsymbol{x})$ ile DYM çıktılarını, y_i ile de sonsal olasılıkları gösteriyoruz. v_{ij} ağırlıkları çapraz düzensizliği azaltacak biçimde eğitilir. Ezberlemeyi engellemek için, yığılmış genellemede (bölüm 17.9) yaptığımız gibi, v_{ij} ağırlıklarını öğrenmekte kullanılan veriyle taban $f_j(\boldsymbol{x})$ DYM ayırtaçlarını eğittiğimiz verinin ayrık olması önerilir.

Her biri bir sınıfı öteki sınıflardan ayıran K tane iki sınıflı DYM eğitmek yerine, başka sınıflandırıcılarda da yapabileceğimiz gibi, $K(K-1)/2$ tane *ikili* sınıflandırıcı da eğitebiliriz (bkz. bölüm 10.4). Her $g_{ij}(\boldsymbol{x})$ ayırtacı, C_i sınıfından örnekler için $+1$, C_j sınıfından örnekler için de -1 çıktı verecek biçimde eğitilir; öteki sınıflardan örnekler kullanılmaz. Sınıfları ikişer ikişer ayırmanın daha kolay olmasını bekleriz, üstelik veri daha az olacağı için çözüm de daha hızlı olacaktır; çekincemiz, $\mathcal{O}(K)$ yerine $\mathcal{O}(K^2)$ tane ayırtaç gerekmesidir.

Genel durumda, hem bir sınıfı tüm ötekilerden, hem de ikili ayırmak *hata düzelten çıktı gizyazıları* yaklaşımının özel durumlarıdır ve çok sınıflı bir problemi bir küme ikili problem olarak yeniden tanımlamanın farklı yollarıdır (Dietterich ve Bakiri 1995) (bkz. bölüm 17.6). DYM, iki sınıflı tanımlandığı için bu çatıya çok iyi uyar (Allwein, Schapire, ve Singer 2000); artımlı olarak her aşamada en çok karışan iki sınıf için ayrı bir ayırtacın eklendiği ve gizyazı dizeyinin öğrenme sırasında iyileştirildiği yaklaşımlar da önerilmişir (Mayoraz ve Alpaydın 1999).

Bir başka olasılık, tüm sınıfları içeren *çok sınıflı* tek bir iyileme problemi yazmaktır (Weston ve Watkins 1998):

$$\min \frac{1}{2}\sum_{i=1}^{K}\|\boldsymbol{w}_i\|^2 + C\sum_i\sum_t \xi_i^t \tag{13.42}$$

Kısıtlar:

$$\boldsymbol{w}_{z^t}\boldsymbol{x}^t + w_{z^t 0} \geq \boldsymbol{w}_i\boldsymbol{x}^t + w_{i0} + 2 - \xi_i^t, \forall i \neq z^t \text{ ve } \xi_i^t \geq 0$$

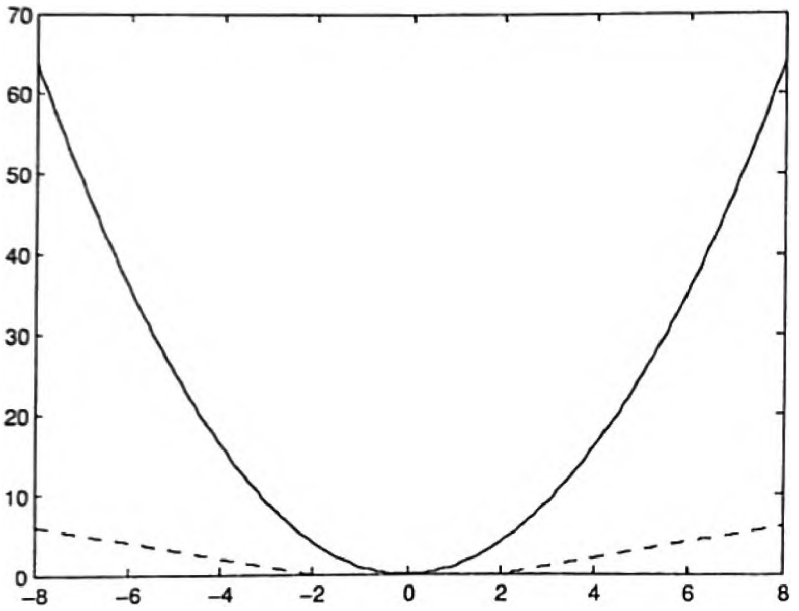
Burada z^t ile $\boldsymbol{x}^t$ örneğinin sınıf değerini gösteriyoruz. Düzgünleştirme terimi tüm üstündüzlem yöneylerinin büyüklüklerini birlikte azaltır. Kısıta göre her örnek için doğru sınıfın ayırtacıyla farklı bir sınıfın ayırtaç değerleri arasında en az 2 birimlik kenar payı gerekir: Doğru sınıfın ayırtacının en az $+1$, farklı sınıfın ayırtacının değerinin ise en fazla -1 olmalıdır; artık değişken aradaki farkın ikiden az olması durumunda eşiksiz hata tanımlar.

Bu çok iyi bir yaklaşım olmasına rağmen çoğunlukla bir sınıfı öteki sınıflardan ayıran yaklaşım kullanılır, çünkü $K \cdot N$ değişkenli tek bir problem çözmek yerine K tane N değişkenli ayrık problem çözer.

13.10 Bağlanım için Çekirdek Makinesi

Gelim şimdi de destek yöney makinelerinin bağlanımda nasıl kullanıldığını görelim. Sınıflandırmaya benzer biçimde bağlanımda da kenar payı ve artık değişkenler tanımlayıp düzgünlük ve hatayı birleştiren bir düzgünleştirici kullanıyoruz. Yine önce doğrusal modelle başlayacak ve sonra çekirdek işlevleriyle modeli genelleştireceğiz:

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$$



Şekil 13.6: Karesel (sürekli) ve ϵ duyarlı (kesikli) hata işlevleri. ϵ duyarlı işlev küçük hataları gözardı eder ve doğrusal olduğu için aykırı gözlemlere karşı daha gürbüzdür.

Bağlanımda hep hata olarak farkın karesini

$$e_2(r^t, f(\boldsymbol{x}^t)) = [r^t - f(\boldsymbol{x}^t)]^2$$

kullanıyorduk ama destek yöney bağlanımında ϵ duyarlı kayıp işlevini kullanıyoruz:

$$e_\epsilon(r^t, f(\boldsymbol{x}^t)) = \begin{cases} 0 & \text{eğer } |r^t - f(\boldsymbol{x}^t)| < \epsilon \\ |r^t - f(\boldsymbol{x}^t)| - \epsilon & \text{değilse} \end{cases} \tag{13.43}$$

Bu biçimde, hem ϵ değerine kadar olan hataları gözardı ediyoruz, hem de daha büyük hataların etkisi karesel değil doğrusal oluyor. Böylece gürültüye dayanıklı, *gürbüz* (bkz. şekil 13.6) bir bağlanım elde ediyoruz. Menteşe kayıp işlevinde olduğu gibi hata oluşturmayan bir alan tanımlamak sonucun seyrek olmasını sağlıyor.

Eşiksiz kenarlı üstündüzlemde yaptığımız gibi ϵ alanından (hatalı) sapmalar için artık değişkenler tanımlıyoruz (Vapnik 1995)

$$\min \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_t(\xi_+^t + \xi_-^t) \tag{13.44}$$

Sapmalar iki yönde de olabileceğinden artı ve eksi sapmalar için ayrı artık değişkenlerimiz var ve kısıtlarımız da iki yönlü:

$$\begin{aligned} r^t - (\boldsymbol{w}^T\boldsymbol{x} + w_0) &\leq \epsilon + \xi_+^t \\ (\boldsymbol{w}^T\boldsymbol{x} + w_0) - r^t &\leq \epsilon + \xi_-^t \\ \xi_+^t, \xi_-^t &\geq 0 \end{aligned}$$

Bu hata işlevini sırt sırta ters oturmuş, bir artı, öbürü eksi sapmalara bakan iki menteşe kayıp işlevi olarak da görebiliriz. Bu tanımlama, denklem 13.43'teki ϵ duyarlı hata işlevine karşılık gelir. Lagrange terimlerini ekleyelim:

$$\begin{aligned} L_p &= \frac{1}{2}\|w\|^2 + C\sum_t(\xi_+^t + \xi_-^t) \\ &\quad - \sum_t \alpha_+^t \left[\epsilon + \xi_+^t - r^t + (w^T x + w_0)\right] \\ &\quad - \sum_t \alpha_-^t \left[\epsilon + \xi_-^t + (w^T x + w_0) - r^t\right] \\ &\quad - \sum_t (\mu_+^t \xi_+^t + \mu_-^t \xi_-^t) \end{aligned} \tag{13.45}$$

Kısmı türevleri alıp sıfıra eşitleyelim:

$$\frac{\partial L_p}{\partial w} = w - \sum_t (\alpha_+^t - \alpha_-^t)x^t = 0 \Rightarrow w = \sum_t (\alpha_+^t - \alpha_-^t)x^t \tag{13.46}$$

$$\frac{\partial L_p}{\partial w_0} = \sum_t (\alpha_+^t - \alpha_-^t)x^t = 0 \tag{13.47}$$

$$\frac{\partial L_p}{\partial \xi_+^t} = C - \alpha_+^t - \mu_+^t = 0 \tag{13.48}$$

$$\frac{\partial L_p}{\partial \xi_-^t} = C - \alpha_-^t - \mu_-^t = 0 \tag{13.49}$$

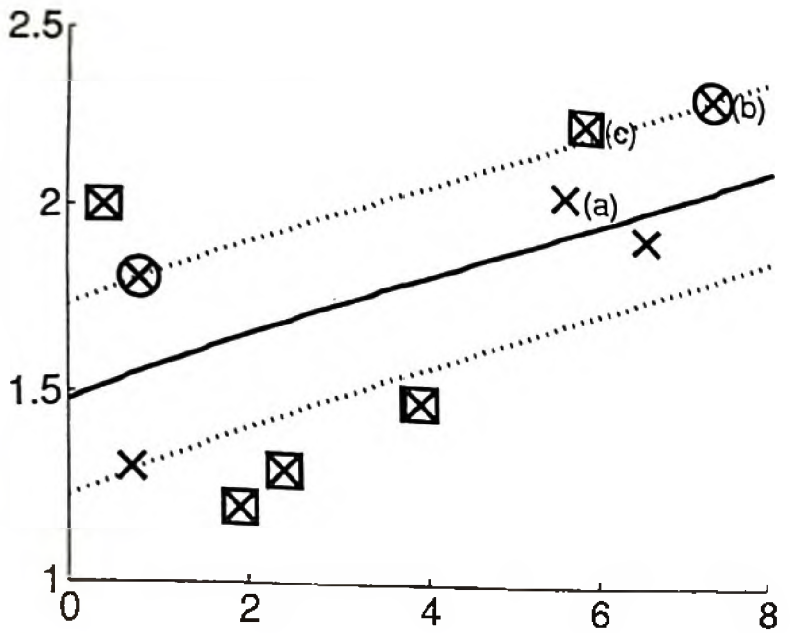
Eşleği yazalım:

$$\begin{aligned} L_d &= -\frac{1}{2}\sum_t\sum_s (\alpha_+^t - \alpha_-^t)(\alpha_+^s - \alpha_-^s)(x^t)^T x^s \\ &\quad -\epsilon\sum_t (\alpha_+^t + \alpha_-^t) - \sum_t r^t(\alpha_+^t - \alpha_-^t) \end{aligned} \tag{13.50}$$

Kısıtlar:

$$0 \le \alpha_+^t \le C\ ,\ 0 \le \alpha_-^t \le C\ ,\ \sum_t (\alpha_+^t - \alpha_-^t) = 0$$

Çözdüğümüzde ϵ alanında düşen örnekler için $\alpha_+^t = \alpha_-^t = 0$ olur; bunlar istenilen duyarlılıkta kestirilebilen örneklerdir (bkz. şekil 13.7). Destek yöneyleri için $\alpha_+^t > 0$ ya da $\alpha_-^t > 0$ sağlanır ve iki türlüdürler. ϵ alanının sınırında olabilirler (α_+^t ya da α_-^t, 0 ile C arasındadır) ve bunları kullanarak w_0 değerini hesaplarız. Örneğin, $\alpha_+^t > 0$ için $r^t = x^T x^t + w_0 + \epsilon$ sağlanması gerekir. ϵ alanının dışında kalan örnekler ikinci tür destek yöneyidir; bunlar için yeterince iyi bir kestirim yapılamamıştır ($\alpha_+^t = C$) (bkz. şekil 13.7).



Şekil 13.7: Çarpıyla gösterilmiş veri noktalarına oturtulmuş bağlanım doğrusu ve ϵ alanı gösterilmiştir ($C = 10, \epsilon = 0.25$). Olası üç durum vardır: (a) örnek ϵ alanı içindedir; (b) örnek sınırdadır (çember içine alınmış örnekler); (c) örnek alanın dışındadır ve artık değişken(lerin ikisinden biri) sıfırdan büyük değer alır; $\xi_+^t > 0$ (kare içine alınmış örnekler). (b) ve (c) destek yöneyleridir. Eşlek değişkenler olarak bakarsak (a) $\alpha_+^t = 0, \alpha_-^t = 0$; (b) $\alpha_+^t < C$, ve (c) $\alpha_+^t = C$.

Denklem 13.46'yı kullanarak bağlanım doğrusunu destek yöneylerinin ağırlıklı toplamı olarak yazabiliriz:

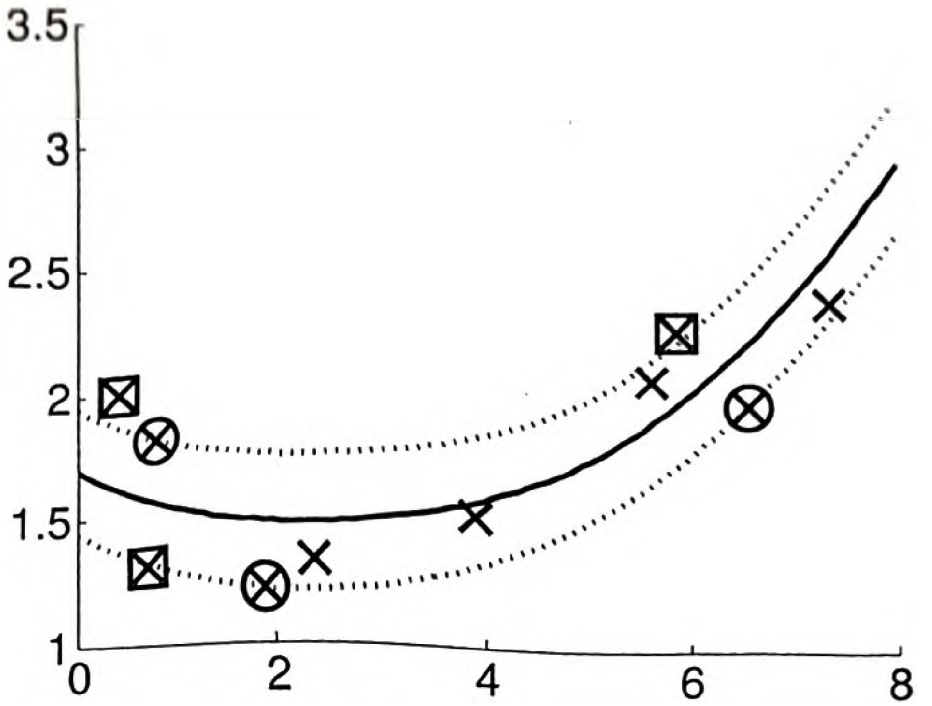
$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 = \sum_t (\alpha_+^t - \alpha_-^t)(\boldsymbol{x}^t)^T\boldsymbol{x} + w_0 \tag{13.51}$$

Bağlanımda da denklem 13.50'deki $(\boldsymbol{x}^t)^T\boldsymbol{x}^s$ iç çarpımı yerine bir $K(\boldsymbol{x}^t, \boldsymbol{x}^s)$ çekirdeği ve $(\boldsymbol{x}^t)^T\boldsymbol{x}$ yerine $K(\boldsymbol{x}^t, \boldsymbol{x})$ kullanılarak doğrusal olmayan bağlanım gerçeklenebilir. Çokterimli bir çekirdek kullanmak çokterimli bağlanıma benzer (şekil 13.8). Gauss çekirdeğiyse (şekil 13.9) dağılımdan bağımsız düzleştirici modeller (bölüm 8.6) gibi çalışır; tek fark, çözümün seyrek olması ve öğrenme kümesinin tamamının değil bir altkümesinin kullanılmasıdır.

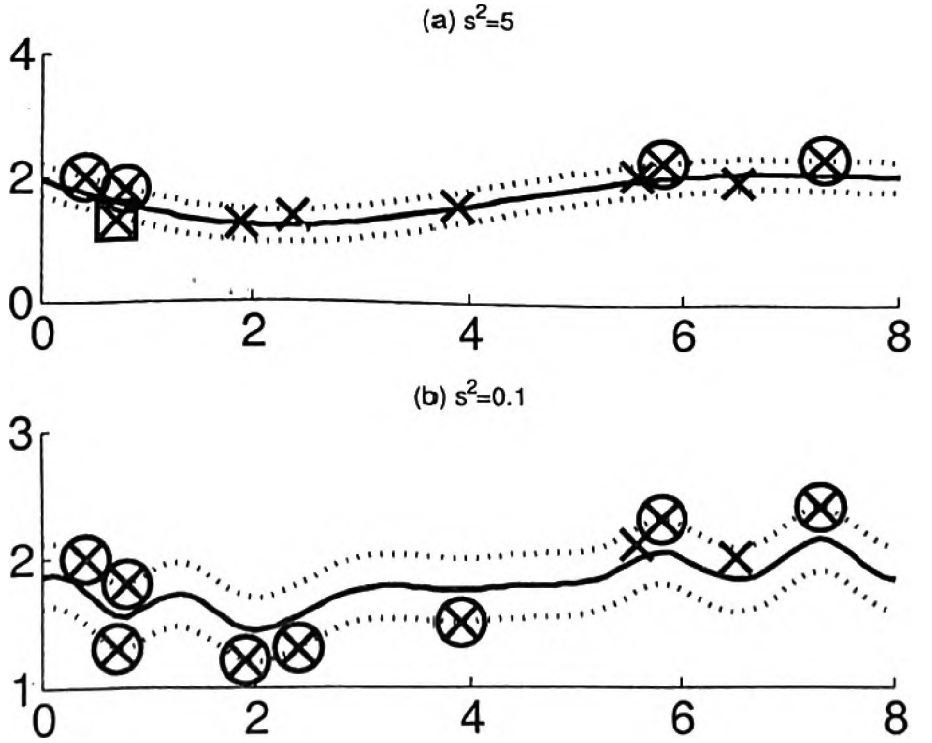
Bağlanım için de eşdeğer bir ν-DYM tanımlaması yapılmıştır (Schölkopf ve ötekiler 2000); ϵ değeri yerine destek yöney oranını belirleyen ν sabitlenir; yine ek bir C parametresi de vardır.

13.11 Tek Sınıflı Çekirdek Makinesi

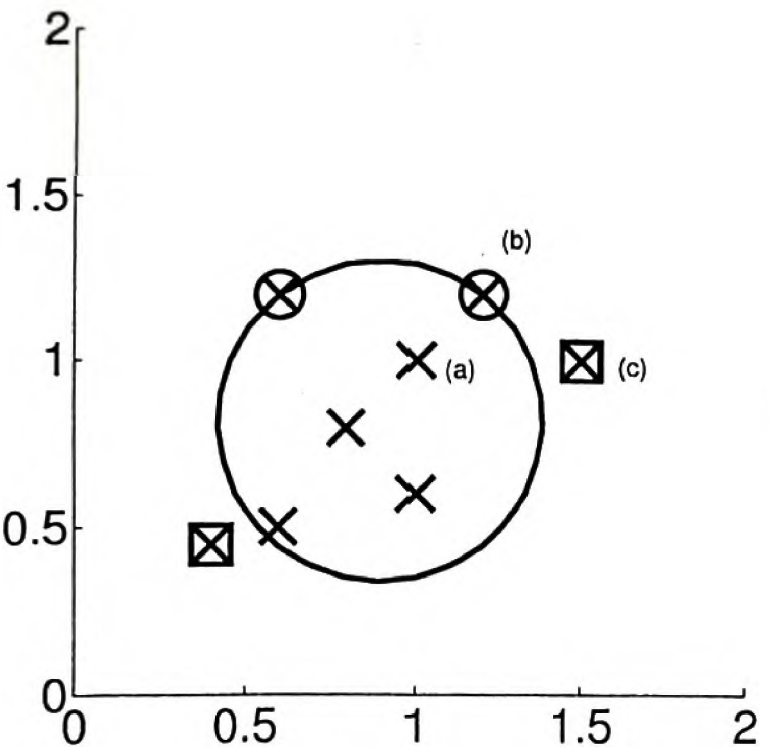
Destek yöney makineleri önce sınıflandırma için önerilmiş, sonra bağlanım doğrusu çevresindeki sapmalar için artık değişkenleri tanımlanarak bağlanıma genellenmiştir. Şimdi de düşük olasılıklı alanların bulunmasında nasıl kullanılabileceğini göreceğiz;



Şekil 13.8: İkinci dereceden bir çokterimli çekirdekle yapılan bağlanım ve ϵ alanı gösterilmiştir ($C = 10, \epsilon = 0.25$). Çembere alınmış örnekler sınırdaki destek yöneylerini, kare içine alınmışlarsa dışarıda kalan destek yöneylerini gösterir.



Şekil 13.9: Farklı yarıçap değerleri için Gauss çekirdeğiyle yapılan bağlanım ve ϵ alanı gösterilmiştir ($C = 10, \epsilon = 0.25$). Çembere alınmış örnekler sınırdaki destek yöneylerini, kare içine alınmışlarsa dışarıda kalan destek yöneylerini gösterir.



Şekil 13.10: Tek sınıflı destek yöney makinesi örneklerin olabildiğince büyük oranını içeren ve olabildiğince düzgün (doğrusal çekirdek için bu yarıçapa karşılık geliyor) bir sınır arar. Olası üç durum vardır: (a) örnek içeridedir; (b) örnek sınırdadır ($\xi^t = 0$) ve böylesi örnekler R yarıçap değerini belirler. (c) örnek dışarıdadır ve aykırılık gösterir: $\xi^t > 0$. (b) ve (c) destek yöneyleridir. Eşlek değişken olarak bakarsak (a) $\alpha^t = 0$; (b) $0 < \alpha^t < C$; ve (c) $\alpha^t = C$.

bu bir tür gözetimsiz öğrenmedir. Yaptığımız olasılık kestirimi değil, (sınıflandırmaya benzer biçimde) düşük olasılıklı (nadir görülen) alanı daha yüksek olasılıklı alandan ayıran bir sınır öğrenmek (Tax ve Duin 1999). Böyle bir sınır, *yeniliklerin* ya da *aykırılıkların sezimi* için kullanılabilir. Buna *tek sınıf sınıflandırma* da denir.

Merkezi $\boldsymbol{a}$ ve yarıçapı R ile tanımlanan öyle bir küre tanımlamak istiyoruz ki olasılık dağılımının (öğrenme kümesinin) olabildiğince büyük bir oranını içersin. Aynı zamanda basit bir çözüm için bu kürenin hacminin (yarıçapının) olabildiğince küçük olmasını istiyoruz. Kürenin dışına düşen örnekler için artık değişkenler tanımlıyoruz; örnekleri içermek istediğimiz için sınırın içinde kalmanın bir cezası yok. Düzgünlük ölçütümüz ise yarıçapına oranlı:

$$\min R^2 + C \sum_t \xi^t \tag{13.52}$$

Kısıtlarımız, kürenin dışına düşmekle ilgili:

$$\|\boldsymbol{x}^t - \boldsymbol{a}\|^2 \leq R^2 + \xi^t \text{ ve } \xi^t \geq 0, \forall t$$

Kısıtları Lagrange terimleri olarak ekleyelim ($\|\boldsymbol{x}^t - \boldsymbol{a}\|^2 = (\boldsymbol{x}^t - \boldsymbol{a})^T(\boldsymbol{x}^t - \boldsymbol{a})$):

$$L_p = R^2 + C\sum_t \xi^t - \sum_t \alpha^t \left(R^2 + \xi^t - \left[(\boldsymbol{x}^t)^T\boldsymbol{x}^t - 2\boldsymbol{a}^T\boldsymbol{x}^t + \boldsymbol{a}^T\boldsymbol{a}\right]\right) - \sum_t \gamma^t\xi^t \quad (13.53)$$

Burada, $\alpha^t \geq 0$ ve $\gamma_t \geq 0$, Lagrange çarpanlarıdır. Türevleri alıp sıfıra eşitleyelim:

$$\frac{\partial L}{\partial R} = 2R - 2R\sum_t \alpha^t = 0 \Rightarrow \sum_t \alpha^t = 1 \quad (13.54)$$

$$\frac{\partial L}{\partial \boldsymbol{a}} = \sum_t \alpha^t(2\boldsymbol{x}^t - 2\boldsymbol{a}) = 0 \Rightarrow \boldsymbol{a} = \sum_t \alpha^t\boldsymbol{x}^t \quad (13.55)$$

$$\frac{\partial L}{\partial \xi^t} = C - \alpha^t - \gamma^t = 0 \quad (13.56)$$

$\gamma^t \geq 0$ olduğu için son kısıtı $0 \leq \alpha^t \leq C$ olarak da yazabiliriz. Bunları denklem 13.53'e koyduğumuzda α^t değerlerine göre büyüteceğimiz eşleği oluştururuz:

$$L_d = \sum_t \alpha^t(\boldsymbol{x}^t)^T\boldsymbol{x}^t - \sum_t\sum_s \alpha^t\alpha^s(\boldsymbol{x}^t)^T\boldsymbol{x}^s \quad (13.57)$$

Kısıtlar:

$$0 \leq \alpha^t \leq C \text{ ve } \sum_t \alpha^t = 1$$

Çözüm yine seyrek olur, çünkü örneklerin çoğu için $\alpha^t = 0$ olur; bunlar küre içine düşüen yüksek olasılıklı, sık rastlanan örneklerdir (şekil 13.10). İki tür destek yöneyi için $\alpha^t > 0$ olur: Bazı örnekler için $0 < \alpha^t < C$ olur; bunlar sınırın üstündedir: $\|\boldsymbol{x}^t - \boldsymbol{a}\|^2 = R^2$ ($\xi^t = 0$) ve bunları kullanarak R değerini hesaplarız. $\alpha^t = C$ ($\xi^t > 0$) olan örnekler ise sınırın dışındadır; bunların olasılığı düşüktür ve yeni ya da aykırıdırlar. Denklem 13.55'te merkez olan $\boldsymbol{a}$ yöneyinin destek yöneylerinin ağırlıklı toplamı olarak yazıldığını görüyoruz.

Yeni bir $\boldsymbol{x}$ deneme örneği için eğer

$$\|\boldsymbol{x} - \boldsymbol{a}\|^2 > R^2$$

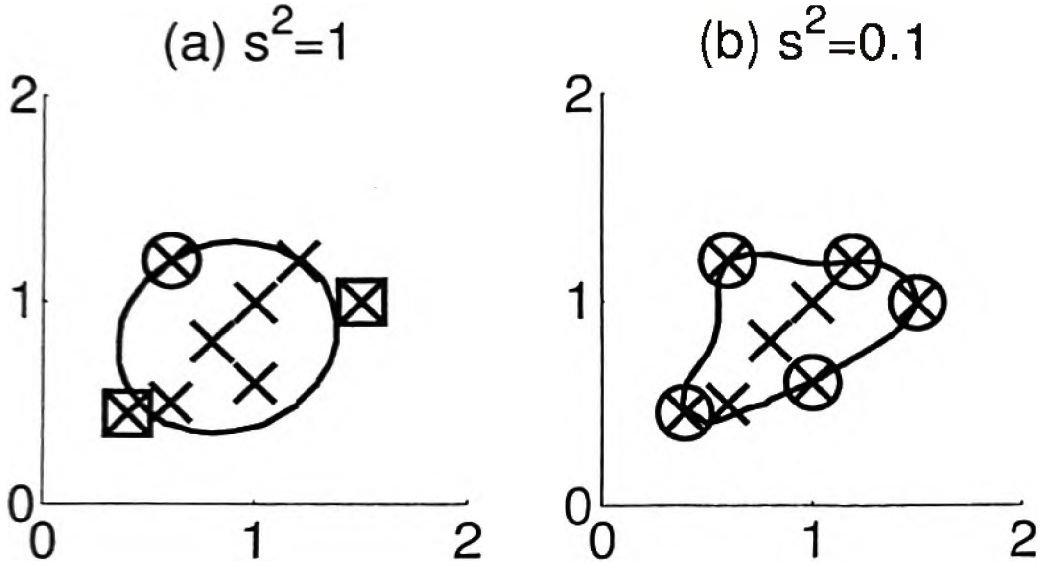
ya da

$$\boldsymbol{x}^t\boldsymbol{x} - 2\boldsymbol{a}^T\boldsymbol{x} + \boldsymbol{a}^T\boldsymbol{a} > R^2$$

sağlanırsa bu örneğin olasılığının düşük olduğunu ve aykırı olduğunu söyleyebiliriz.

Çekirdek işlevi kullanarak küreden farklı sınırlar tanımlayabiliriz. İç çarpım yerine çekirdek işlevi kullanalım (aynı kısıtlarla):

$$L_d = \sum_t \alpha^t K(\boldsymbol{x}^t, \boldsymbol{x}^t) - \sum_t\sum_s \alpha^t\alpha^s K(\boldsymbol{x}^t, \boldsymbol{x}^s) \quad (13.58)$$



Şekil 13.11: Farklı yarıçaplarda Gauss çekirdekleri için tek sınıflı destek yöney makinesi.

Örneğin ikinci dereceden çokterimli bir çekirdekle ikinci dereceden sınırlar tanımyabiliriz Gauss çekirdeğiyle (denklem 13.30) tanımlanan alan yerel kürelerin birleşiminden oluşur. Eğer $\boldsymbol{x}$

$$K(\boldsymbol{x},\boldsymbol{x}) - 2\sum_t \alpha^t K(\boldsymbol{x},\boldsymbol{x}^t) + \sum_t \sum_s \alpha^t \alpha^s K(\boldsymbol{x}^t,\boldsymbol{x}^s) > R^2$$

koşulunu sağlarsa aykırı bir örnektir. Burada, üçüncü terim $\boldsymbol{x}$ girdisine bağlı değildir ve sabittir (bunu $\boldsymbol{x}$ sınırda olduğunda eşitlik olarak alıp R değerini hesaplamakta kullanırız). Gauss çekirdeğinde $K(\boldsymbol{x},\boldsymbol{x}) = 1$ olduğundan koşul, belli bir R_c sabiti için

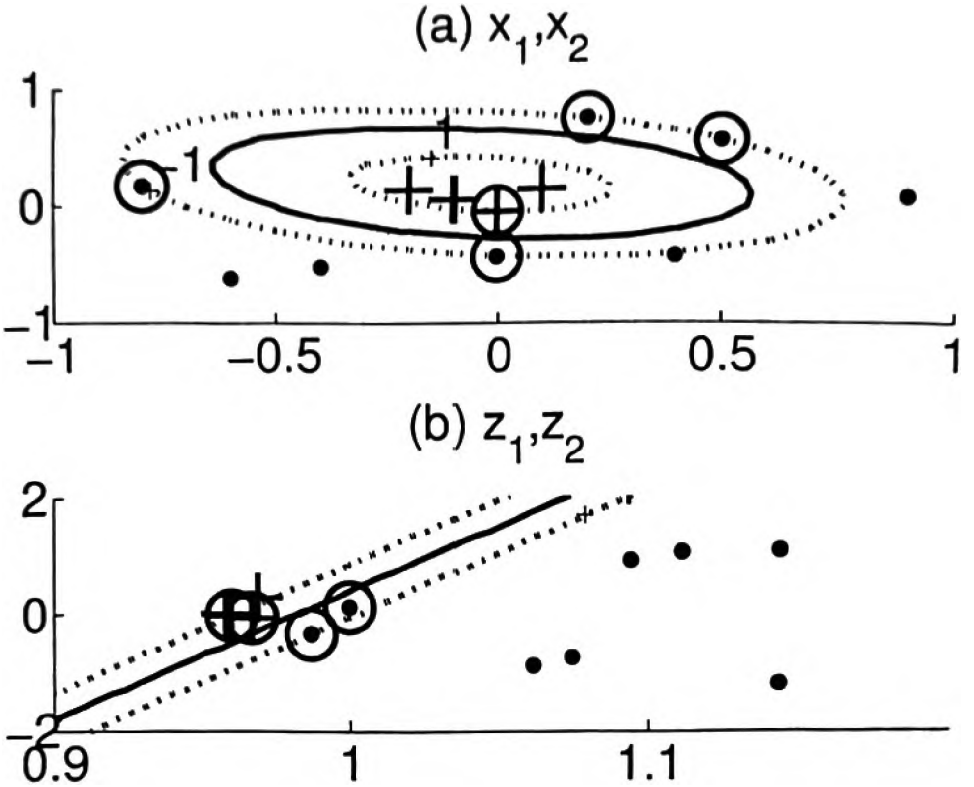
$$\sum_t \alpha^t K_G(\boldsymbol{x},\boldsymbol{x}^t) < R_c$$

olarak basitleşir. Bu yazdığımız dağılımdan bağımsız çekirdek yoğunluk kestiricisine (bölüm 8.2.2) benzer, farkı seyrekliğidir; bu durumda R_c değeri, bir olasılık eşiğine karşılık gelir (bkz. şekil 13.11).

Tek sınıflı durum için de eşdeğer bir ν-DYM yaklaşımı vardır ve düzgünlük için sınıflandırma ya da bağlanımdaki gibi $(1/2)\|\boldsymbol{w}\|^2$ terimini kullanır (Schölkopf ve ötekiler 2001).

13.12 Çekirdekli Boyut Azaltma

Bölüm 6.3'de temel bileşen çözümlemesinin $\boldsymbol{\Sigma}$ eşdeğişinti dizeyinin en yüksek özdeğerli özyöneylerine izdüşüm alarak boyut azalttığını görmüştük. Veriyi merkezi 0 olacak biçimde taşırsak ($E[\boldsymbol{x}] = 0$) $\boldsymbol{\Sigma} = \mathbf{X}^T\mathbf{X}$ olur. Çekirdekli durumda, $\boldsymbol{x}$ yerine $\boldsymbol{\phi}(\boldsymbol{x})$



Şekil 13.12: (a) Girdi uzayında ikinci dereceden çekirdek kullanmak yerine, (b) ikinci dereceden çekirdek dizeyinde temel bileşen çözümlemesiyle iki boyuta inip doğrusal sınıflandırıcı kullanabiliriz; bu iki boyut (beş tane içinden) değişkenliğin yüzden 80'ini açıklıyor.

uzayında çalışırız ve bu uzayın boyut sayısı olan d değeri örneklem büyüklüğü olan N değerinden fazla olabileceğinden, $d \times d$ boyundaki $\mathbf{X}^T\mathbf{X}$ dizeyi yerine $N \times N$ boyundaki $\mathbf{X}\mathbf{X}^T$ dizeyini kullanmayı yeğleriz. İzdüşüm veri dizeyini $\mathbf{\Phi} = \phi(\mathbf{X})$ ile gösterirsek bu durumda $\mathbf{\Phi}^T\mathbf{\Phi}$, yani $\mathbf{K}$ çekirdek dizeyinin özyöneylerini kullanırız.

Çekirdekli temel bileşen çözümlemesi, çekirdek dizeyinin özdeğer ve özyöneylerini kullanır; bu da $\phi(x)$ uzayında doğrusal boyut azaltmaya karşılık gelir. c_i ve λ_i özyöney ve özdeğerleri gösterirken k boyutlu yeni izdüşüm değerleri

$$z_j^t = \sqrt{\lambda_i} c_j^t, j = 1, \ldots, k, \; t = 1, \ldots, N$$

olarak hesaplanabilir.

Şekil 13.12'deki örnekte önce ikinci dereceden bir çekirdek kullanıyor, ardından temel bileşen çözümlemesiyle boyut sayısını beşten ikiye indirip bu yeni uzayda doğrusal bir DYM kullanıyoruz. Genelde (örneğin, Gauss çekirdeği kullanıldığında), özdeğerler her zaman azalmayabilir ve bu biçimde boyut azaltma kullanılamayabilir.

Başka bir açıdan bakarsak burada yaptığımız çekirdek değerlerini benzerlik olarak alıp çokboyutlu ölçekleme (bölüm 6.5) yapmaktır. Örneğin $k = 2$ alıp çekirdek dizeyinin belirlediği uzayda veri görselleştirilebilir ve böylece çekirdeğin verideki benzerliği nasıl tanımladığı anlaşılabilir. Benzer biçimde doğrusal ayırtaç çözümlemesi

de (bölüm 6.6) çekirdek işlevi kullanacak biçimde tanımlanabilir (Müller ve ötekiler 2001).

6. bölümde doğrusal olmayan boyut azaltma için eşölçümsel öznitelik eşleme ve yerel doğrusal gömme adlı iki yöntem gördük. Denklem 6.47'deki hata dizey değerlerini girdi çiftleri için çekirdek değerleri olarak düşündüğümüzde yerel doğrusal gömmenin belli bir çekirdek için bir tür çekirdekli boyut azaltma yaptığı söylenebilir. Aynı açıklama, çekirdeği çizge üzerindeki yerölçümsel uzaklıkların bir işlevi olarak tanımladığımızda eşölçümsel öznitelik eşleme algoritması için de geçerlidir.

13.13 Kısa Kısa

Doğrusal olmayan taban işlevleriyle yeni bir uzaya geçerek doğrusal modelleri genellemek çok eski bir düşünceyse de destek yöney makinelerinin yeniliği bunu parametreleri veri örneklerinin bir altkümesi olarak yazılabilen (buna *eşlek gösterim* diyoruz) bir öğrenme algoritmasına bütünleştirebilmesidir. Bu sayede taban işlevlerini hiç hesaplamadan sonucu hesaplayabiliyoruz ve karmaşıklık, girdi boyut sayısına değil öğrenme kümesi büyüklüğüne bağlı oluyor; aynı düşünce, çekirdek işlevi yerine eşdeğişinti işlevi adının kullanıldığı Gauss süreçleri için de geçerlidir (bölüm 14.4).

Çözümün seyrekliği, en yakın k komşu ya da Parzen penceresi gibi dağılımdan bağımsız kestirim yöntemleri ya da Gauss süreçlerine göre üstünlüğünü gösteriyor. Ayrıca çekirdek işlevlerini kullanabilmek sayesinde veri örneklerinin yöneyler olarak gösterilmesi gerekmiyor. İyileme probleminin tek çözümü olduğu için yapay sinir ağlarındaki gibi yinelemeli bir öğrenmeye de gerek kalmıyor. Tüm bu nedenler sayesinde destek yöney makineleri bu son yıllarda en çok kullanılan öğrenme algoritması olmuştur, ve birçok alanda, örneğin biyoinformatikte (Schölkopf, Tsuda, ve Vert 2004) ve doğal dil işlemede uygulamaya bağlı çekirdekler geliştirme konusunda çalışmalar sürmektedir (Shawe-Taylor ve Cristianini 2004).

Çekirdek dizeyi verinin farklı bir gösterimidir; artık bir örneği (nesne ya da olayı) bir öznitelikler yöneyi olarak değil başka örneklere ne kadar benzer, ya da onlardan ne kadar farklı olduğuna göre tanımlıyoruz. Bu fark, belli bir uzayda temel bileşen çözümlemesi yaparak boyut azaltmayla, çokboyutlu ölçeklemenin yaptığı gibi (nasıl hesaplandığı bizi ilgilendirmeyen) uzaklıklar dizeyi üzerinde boyut azaltma arasındaki farka benziyor.

(Vapnik 1995, 1998) ve (Schölkopf ve Smola 2002), destek yöney makineleriyle ilgili ek bilgi için okunması gereken kitaplardır. (Cherkassky ve Mulier 1998)'deki DYM bölümü iyi bir giriştir. (Burges 1998) ve (Smola ve Schölkopf 1998), sırasıyla sınıflandırma ve bağlanım için iki iyi kaynaktır. http://www.kernel-machines.org adresinde önemli birçok kaynak bulunabilir; DYM için SVMlight (Joachims 2004) ve LIBSVM (Chang ve Lin 2008) gibi açık ve bedava yazılımlar araştırmacıların kullanımınaa açıktır.

13.14 Alıştırmalar

1. Destek yöneyi olma olasılığı çok az olan örnekleri bulup öğrenme kümesinden atmak için bir süzme algoritması önerin.
2. Denklem 13.31'de S dizeyini nasıl kestirebiliriz?
3. Gözlemsel çekirdek gösteriminde şablonları nasıl belirleyebiliriz?
4. Denklem 13.40'daki yerel çoklu çekirdek yönteminde $\eta_i(x|\theta_i)$ için uygun bir model önerin ve nasıl eğitilebileceğini gösterin.
5. Çekirdekli bağlanımda, eğer varsa, ϵ ve gürültünün değişintisi arasında nasıl bir ilişki vardır?
6. Çekirdekli bağlanımda, farklı ϵ değerlerinin yanlılık ve değişinti üzerinde nasıl bir etkisi vardır?
7. Tek sınıflı DYM modelini sınıflandırmada nasıl kullanabiliriz?
8. Şekil 13.12'deki gibi örnek bir veride Gauss çekirdekli temel bileşen çözümlemesi nasıl bir sonuç verir?
9. Her nesne için iki ayrı gösterim ve her gösterim için farklı bir çekirdek tanımlı olsun. Bu durumda çekirdekli temel bileşen çözümlemesiyle ortak bir uzayda nasıl boyut azaltabiliriz?

13.15 Kaynaklar

Allwein, E. L., R. E. Schapire, Y. Singer. 2000. "Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers." *Journal of Machine Learning Research* 1: 113–141.

Burges, C. J. C. 1998. "A Tutorial on Support Vector Machines for Pattern Recognition." *Data Mining and Knowledge Discovery* 2: 121–167.

Chang, C.-C., C.-J. Lin. 2008. *LIBSVM: A Library for Support Vector Machines.* `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

Cherkassky, V., F. Mulier. 1998. *Learning from Data: Concepts, Theory, and Methods.* New York: Wiley.

Cortes, C., V. Vapnik. 1995. "Support Vector Networks." *Machine Learning* 20: 273–297.

Dietterich, T. G., G. Bakiri. 1995. "Solving Multiclass Learning Problems via Error-Correcting Output Codes." *Journal of Artificial Intelligence Research* 2: 263–286.

Gönen, M., E. Alpaydın. 2008. "Localized Multiple Kernel Learning." *25th International Conference on Machine Learning*, ed. A. McCallum, S. Roweis, 352–359. Madison, WI: Omnipress.

Jaakkola, T., D. Haussler. 1999. "Exploiting Generative Models in Discriminative Classifiers." *Advances in Neural Information Processing Systems 11*, ed. M. J. Kearns, S. A. Solla, D. A. Cohn, 487-493. Cambridge, MA: MIT Press.

Joachims, T. 2004. *SVMlight*, `http://svmlight.joachims.org`.

Lanckriet, G. R. G, N. Cristianini, P. Bartlett, L. El Ghaoui, M. I. Jordan. 2004. "Learning the Kernel Matrix with Semidefinite Programming." *Journal of Machine Learning Research* 5: 27–72.

Mayoraz, E., E. Alpaydın. 1999. "Support Vector Machines for Multiclass Classification." *Foundations and Tools for Neural Modeling, Proceedings of IWANN'99, LNCS 1606*, ed. J. Mira, J. V. Sanchez-Andres, 833–842. Berlin: Springer.

Müller, K. R., S. Mika, G. Rätsch, K. Tsuda, B. Schölkopf. 2001. "An Introduction to Kernel-Based Learning Algorithms." *IEEE Transactions on Neural Networks* 12: 181–201.

Noble, W. S. 2004. "Support Vector Machine Applications in Computational Biology." *Kernel Methods in Computational Biology*, ed. B. Schölkopf, K. Tsuda, J.-P. Vert, 71–92. Cambridge, MA: MIT Press.

Platt, J. 1999. "Probabilities for Support Vector Machines." *Advances in Large Margin Classifiers*, ed. A. J. Smola, P. Bartlett, B. Schölkopf, D. Schuurmans, 61–74. Cambridge, MA: MIT Press.

Schölkopf, B., J. Platt, J. Shawe-Taylor, A. J. Smola, R. C. Williamson. 2001. "Estimating the Support of a High-Dimensional Distribution." *Neural Computation* 13: 1443–1471.

Schölkopf, B., A. J. Smola. 2002. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* Cambridge, MA: MIT Press.

Schölkopf, B., A. J. Smola, R. C. Williamson, P. L. Bartlett. 2000. "New Support Vector Algorithms." *Neural Computation* 12: 1207–1245.

Schölkopf, B., K. Tsuda, J.-P. Vert, eds. 2004. *Kernel Methods in Computational Biology.* Cambridge, MA: MIT Press.

Shawe-Taylor, J., N. Cristianini. 2004. *Kernel Methods for Pattern Analysis.* Cambridge, UK: Cambridge University Press.

Smola, A., B. Schölkopf. 1998. *A Tutorial on Support Vector Regression*, NeuroCOLT TR-1998-030, Royal Holloway College, University of London, UK.

Sonnenburg, S., G. Rätsch, C. Schäfer, B. Schölkopf. 2006. "Large Scale Multiple Kernel Learning." *Journal of Machine Learning Research* 7: 1531–1565.

Tax, D. M. J., R. P. W. Duin. 1999. "Support Vector Domain Description." *Pattern Recognition Letters* 20: 1191–1199.

Vapnik, V. 1995. *The Nature of Statistical Learning Theory.* New York: Springer.

Vapnik, V. 1998. *Statistical Learning Theory.* New York: Wiley.

Vert, J.-P., K. Tsuda, B. Schölkopf. 2004. "A Primer on Kernel Methods." *Kernel Methods in Computational Biology*, ed. B. Schölkopf, K. Tsuda, J.-P. Vert, 35–70. Cambridge, MA: MIT Press.

Weston, J., C. Watkins. 1998. "Multiclass Support Vector Machines." *Technical Report CSD-TR-98-04*, Department of Computer Science, Royal Holloway, University of London.

Bölüm 14

Bayesçi Kestirim

Bayesçi yaklaşımda parametreleri öznel bir dağılımdan çekilmiş rastsal değişkenler olarak görürüz. Bölüm 4.4'te kaldığımız yerden tartışmaya ardından üç Bayesçi uygulama göreceğiz: bir dağılımın parametrelerinin kestirimi, bir modelin parametrelerinin kestirimi, ve Gauss süreçleri.

Giriş

ni parametrelerle ilgili öznel bir bilgi olduğunda kullanılır. Örneğin ılamının kestirirken örnekleme bakmadan önce, ikiye yakın, 1 ve 3 nın biliyor olabiliriz. Böylesi öznel bilgiler, özellikle örneklem küçükse lıtacağımız, verinin bize söylediği, yani örneklemden hesapladığımız nsel bilgiyi birleştirmektir.

ın 12'de tartıştığımız en büyük olabilirlik kestirimi yaklaşımında parametre-ılmayan sabitler olarak görüldü. Bayesçi kestirimde, bölüm 4.4'te tartışmaya ığımız gibi, parametreleri rastsal değişkenler olarak görüyoruz. Böylece pa-,metrelerle ilgili öznel bilgiyi *öznel bir olasılık dağılımı* olarak gösterebiliyoruz. .rneğin, μ değerinin büyük olasılıkla 1 ile 3 arasında olduğunu biliyorsak $p(\mu)$ dağılımını büyük bölümü $[1,3]$ aralığını kapsayacak biçimde tanımlarız.

Bayes kuralı, öznel olasılık ve olabilirliği birleştirip *sonsal olasılık dağılımını* hesaplamamızı sağlar:

$$p(\theta|\mathcal{X}) = \frac{p(\theta)p(\mathcal{X}|\theta)}{p(\mathcal{X})} \tag{14.1}$$

$p(\theta)$ öznel dağılım örnekleme bakmadan *önce* θ parametrelerinin değerlerinin olasılıklarını verir. $p(\mathcal{X}|\theta)$ *örneklem olabilirliğiyse* dağılımın parametreleri θ değerini aldığında $\mathcal{X}$ örnekleminin çekilme olasılığını söyler. Örneğin, elimizdeki örneklerin hepsi 0 ile 10 arasındaysa böyle bir örneklemi görme olasılığı $\mu = 7$ için yüksek, $\mu = 4$ için daha düşük, $\mu = 1$ içinse çok daha düşüktür. Paydadaki $p(\mathcal{X})$ değeri normalleştiricidir ve *sonsal* $p(\theta|\mathcal{X})$ dağılımının integralinin 1 olmasını sağlar. Buna son-

Bölüm 14

Bayesçi Kestirim

Bayesçi yaklaşımda parametreleri önsel bir dağılımdan çekilmiş rastsal değişkenler olarak görürüz. Bölüm 4.4'te kaldığımız yerden tartışmayı sürdürüp üç Bayesçi uygulama göreceğiz: bir dağılımın parametrelerinin kestirimi, bir modelin parametrelerinin kestirimi, ve Gauss süreçleri.

14.1 Giriş

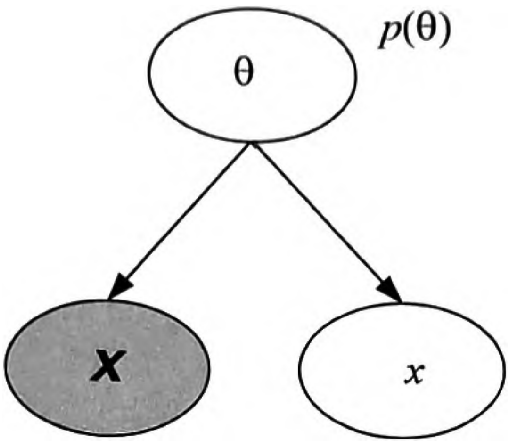
Bayesçi kestirimi parametrelerle ilgili önsel bir bilgi olduğunda kullanılır. Örneğin bir dağılımın ortalamasını kestirirken örnekleme bakmadan önce, ikiye yakın, 1 ve 3 arasında olduğunu biliyor olabiliriz. Böylesi önsel bilgiler, özellikle örneklem küçükse çok önemlidir. İstediğimiz, verinin bize söylediği, yani örneklemden hesapladığımız değerle bu önsel bilgiyi birleştirmektir.

Bölüm 4.2'de tartıştığımız en büyük olabilirlik kestirimi yaklaşımında parametreleri bilinmeyen sabitler olarak gördük. Bayesçi kestirimde, bölüm 4.4'te tartışmaya başladığımız gibi, parametreleri rastsal değişkenler olarak görüyoruz. Böylece parametrelerle ilgili önsel bilgiyi *önsel bir olasılık dağılımı* olarak gösterebiliyoruz. Örneğin, μ değerinin büyük olasılıkla 1 ile 3 arasında olduğunu biliyorsak $p(\mu)$ dağılımını büyük bölümü $[1, 3]$ aralığına düşecek biçimde tanımlarız.

Bayes kuralı, önsel olasılık ve olabilirliği birleştirip *sonsal olasılık dağılımını* hesaplamamızı sağlar:

$$p(\theta|\mathcal{X}) = \frac{p(\theta)p(\mathcal{X}|\theta)}{p(\mathcal{X})} \tag{14.1}$$

$p(\theta)$ önsel dağılımı örnekleme bakmadan *önce* θ parametrelerinin değerlerinin olasılıklarını verir. $p(\mathcal{X}|\theta)$ *örneklem olabilirliğiyse* dağılımın parametreleri θ değerini aldığında $\mathcal{X}$ örnekleminin çekilme olasılığını söyler. Örneğin, elimizdeki örneklerin hepsi 5 ile 10 arasındaysa böyle bir örneklemi görme olasılığı $\mu = 7$ için yüksek, $\mu = 3$ için daha düşük, $\mu = 1$ içinse çok daha düşüktür. Paydadaki $p(\mathcal{X})$ değeri normalleştiricidir ve *sonsal* $p(\theta|\mathcal{X})$ dağılımının integralinin 1 olmasını sağlar. Buna son-



Şekil 14.1: Üretici çizgesel model. Oklar örnekleme yönündedir; önce θ değerini $p(\theta)$ dağılımından çeker, ardından $p(x^t|\theta)$ dağılımından veriyi örnekleriz. Yeni örnek x ve geçmiş örneklem $\mathcal{X}$, θ bilindiğinde birbirinden bağımsızdır: bu, bağımsız ve özdeşçe dağılım varsayımıdır. θ değerini bilmediğimizde bağımlı olurlar: verilmiş (gölgeli düğüm) $\mathcal{X}$ örneklemini kullanarak Bayes kuralıyla oku ters yöne çevirip $p(\theta|\mathcal{X})$ dağılımını hesaplar, ve sonra bunu x düğümünü kestirmek için kullanırız.

sal dağılım denilmesinin nedeni, θ parametrelerinin değerleri için örneklemi gördükten *sonraki* olasılıkları belirtmesidir. Bayes kuralı, önsel dağılımı veriyle birleştirip sonsal dağılımu oluşturur. Ardından öngörü yaparken bu sonsal dağılımı kullanırız.

Örneğin elimizde, bilinmeyen θ parametreli bir dağılımdan çekilmiş $\mathcal{X}$ örneklemi olsun ve ardından çektiğimiz bir sonraki x örneğinin alabileceği değerlerin olasılığını hesaplamak isteyelim. Bunu şekil 14.1'deki *üretici* çizgesel modelle (bölüm 16) görselleştirebiliriz. Önce $p(\theta)$ dağılımından bir θ değeri çekip ardından bunu kullanarak $\mathcal{X}$ örneklemini ve yeni x örneğini örnekliyoruz. Birleşik dağılımı koşullu olasılıklar kullanarak açalım:

$$p(x, \mathcal{X}, \theta) = p(\theta)p(\mathcal{X}|\theta)p(x|\theta)$$

Geçmiş örneklem $\mathcal{X}$ verildiğinde yeni x örneğinin olasılığını hesaplayabiliriz:

$$\begin{aligned} p(x|\mathcal{X}) &= \frac{p(x, \mathcal{X})}{p(\mathcal{X})} = \frac{\int_\theta p(x, \mathcal{X}, \theta)d\theta}{p(\mathcal{X})} = \frac{\int_\theta p(\theta)p(\mathcal{X}|\theta)p(x|\theta)d\theta}{p(\mathcal{X})} \\ &= \int_\theta p(\theta|\mathcal{X})p(x|\theta)d\theta \end{aligned} \tag{14.2}$$

$p(\theta|\mathcal{X})$ dağılımını hesaplarken Bayes kuralı, okun yönünü çevirmemizi ve tanı koymamızı sağlar. Ardından çıkarsadığımız θ dağılımını yeni x için bir öngörü dağılımı olarak kullanırız.

Sonuç kestirim, olası tüm θ değerlerine bağlı yapılan kestirimlerin bu θ değerlerini görmenin ($\mathcal{X}$ örneklemine koşullu) olasılıklarıyla ağırlıklı toplamıdır; θ kesikliyse integral yerine kesikli toplam alırız.

Eğer sonsal dağılımın integralini almak kolay değilse bölüm 4.4'te gördüğümüz gibi *En büyük sonsal kestiriciyi* (EBS) kullanabiliriz:

$$\theta_{EBS} = \arg\max_{\theta} p(\theta|\mathcal{X}) \text{ ve } p_{EBS}(x|\mathcal{X}) = p(x|\theta_{EBS})$$

En büyük sonsal kestirici, sonsal dağılımın kendi tepe noktası çevresinde çok dar ağıldığını varsayar. Önsel $p(\theta)$ olasılığı tüm θ için tekdüzeyse sonsal $p(\theta|\mathcal{X})$ ve olabilirlik $p(\mathcal{X}|\theta)$ dağılımlarının tepeleri aynı yerde olur ve en büyük sonsal kestiriciyle en büyük olabilirlik kestiricisi aynı değeri döndürür. Bu da, en büyük olabilirlik kestiricisi kullanmanın θ değerleri arasında önsel bir fark olmadığını varsaymaya eşit olduğunu gösterir.

Gelin, Bayesçi kestirimin farklı dağılım ve uygulamalarda nasıl kullanıldığını görelim.

14.2 Bir Dağılımın Parametrelerinin Kestirimi

14.2.1 Kesikli Değişkenler

Her örnek, K farklı değerden birini alan katlıterimli bir dağılım (section 4.2.2) olsun. t örneği i değerini aldığında $x_i^t = 1$ ve $\forall j \neq i$ için $x_j^t = 0$ olsun. Katlıterimli dağılımda parametreler durum olasılıkları olan $\boldsymbol{q} = [q_1, q_2, \ldots, q_k]^T$ değerleridir ve $q_i \geq 0, \forall i, \sum_i q_i = 1$ koşullarını sağlarlar.

Örneklem olabilirliğini yazalım:

$$p(\mathcal{X}|\boldsymbol{q}) = \prod_{t=1}^{N}\prod_{i=1}^{K} q_i^{x_i^t}$$

Önsel, *Dirichlet dağılımıdır*:

$$\text{Dirichlet}(\boldsymbol{q}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_1)\cdots\Gamma(\alpha_K)}\prod_{i=1}^{K} q_i^{\alpha_i - 1}$$

Burada, $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_K]^T$ ($\alpha_0 = \sum_i \alpha_i$) ile gösterdiğimiz *üstparametreler* önsel dağılımın parametrelidir. $\Gamma(x)$ ile gösterilen *Gamma işlevidir*:

$$\Gamma(x) \equiv \int_0^\infty u^{x-1}e^{-u}du$$

Örneğin, x^t farklı haber belgeleri ve durumlar, spor, politika, sanat, vs. gibi K farklı belge türüne karşılık gelebilir. Bu durumda, q_i olasılıkları farklı haber türlerinde belge oranlarını belirler ve önseller bu oranlarla ilgili önsel bilgi tanımlamamızı sağlar; örneğin, sanat haberinden daha fazla spor haberi görmeyi bekliyor olabiliriz.

Önsel ve olabilirlik verildiğinde sonsal olasılığı yazabiliriz:

$$\begin{aligned} p(q|\mathcal{X}) &\propto p(\mathcal{X}|q)p(q|\alpha) \\ &\propto \prod_i q_i^{\alpha_i+N_i-1} \end{aligned} \tag{14.3}$$

$N_i = \sum_{t=1}^{N} x_i^t$. Sonsal dağılımın önselle aynı biçimde olduğunu görüyoruz; böyle önsellere *eşlenik önsel* denir. Hem önsel, hem de olabilirlik, q_i parametrelerinin üstlerinin çarpımları olarak yazılır ve bunları birleştirip sonsalı oluştururuz:

$$\begin{aligned} p(q|\mathcal{X}) &= \frac{\Gamma(\alpha_0+N)}{\Gamma(\alpha_1+N_1)\cdots\Gamma(\alpha_K+N_K)}\prod_{i=1}^{K} q_i^{\alpha_i+N_i-1} \\ &= \text{Dirichlet}(q|\alpha+n) \end{aligned} \tag{14.4}$$

öyle ki $n = [N_1,\ldots,N_K]^T$ ve $\sum_i N_i = N$.

Denklem 14.3'e bakıp α_i üstparametreleriyle ilgili yorum yapabiliriz (Bishop 2006). n_i değerlerinin N büyüklüğündeki örneklemde i. türdeki örneklerin sayısı olması gibi α_i değerlerini, α_0 tane örnek içeren hayali bir örneklemdeki i. tür örneklerin sayısı olarak düşünebiliriz. Önseli tanımlarken öznel olarak, "α_0 örnekli bir örneklemde α_i tanesinin i. türden olmasını beklerim," demiş oluyoruz. Oranlar aynı olsa bile daha büyük α_0 öznel oranlarımıza daha fazla güven (daha dar tepeli bir dağılım) gösterir: 10 örneğin 6 tanesi yerine 100 örneğin 60 tanesi dediğimde söylediğim daha kesindir. Sonsal dağılım başka bir Dirichlet oluşturur ve parametreleri, önsel ve olabilirlik tarafından verilen hayali ve gözlenen sayıların toplamıyla belirlenir.

Eşlenikliğin bir güzel yanı, örnekleri sırayla gördüğümüz sıralı bir uygulamada önceki örnekler üzerinden hesaplanan sonsalın bir sonraki örnek için önsel olabilmesidir.

Değişken ikili olduğu özel durumda $x^t \in \{0,1\}$ değişkenleri Bernoulli dağılır:

$$p(\mathcal{X}|q) = \prod_t q^{x^t}(1-q)^{1-x^t}$$

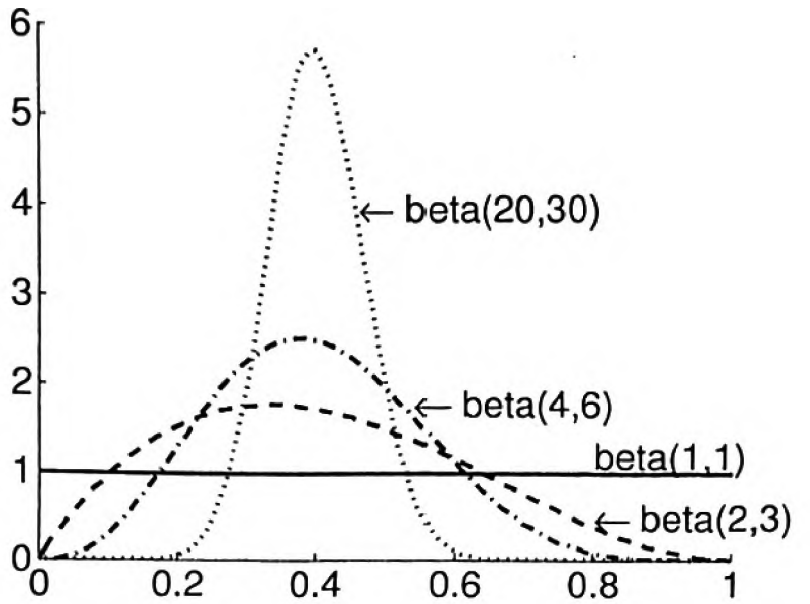
ve Dirichlet önseli, *beta dağılımı* olarak basitleşir:

$$\text{beta}(q|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}q^{\alpha-1}(1-q)^{\beta-1}$$

Örneğin, x^t örnekleri elektronik posta iletileri olup 0 ve 1 değerleri sırasıyla t. iletinin geçerli mi, istenmeyen mi olduğunu gösterebilir. Bu durumda q parametresi üzerinde tanımlanan önsel olasılık, istenmeyen eposta ileti oranıyla ilgili önsel bilgimizi belirtmemizi sağlar: "Ortalama olarak bana gelen iletilerin $\alpha/(\alpha+\beta)$ oranının istenmeyen türde olmasını beklerim."

Beta da eşlenik bir önsel olduğu için sonsal yine beta dağılır:

$$p(q|A,N,\alpha,\beta) \propto q^{A+\alpha-1}(1-p)^{N-A+\beta-1}$$



Şekil 14.2: Farklı (α, β) değerleri için beta dağılımları.

öyle ki $A = \sum_t x^t$. Burada yine hayali ve gözlenen örneklerin birleştirildiğini görüyoruz. $\alpha = \beta = 1$ olduğunda önsel tekdüzedir ve sonsal dağılımla olabilirlik aynı biçimdedir. Bu iki sayaç değeri, önsel için α ve β, ve sonsal için $\alpha + A$ ve $\beta + N - A$, artıp aralarındaki fark açıldıkça daha dik tepeli, daha küçük değişintili bir sonsal oluşur (bkz. şekil 14.2). Daha fazla veri gördükçe (hayali ya da gerçek) değişinti azalır.

14.2.2 Sürekli Değişkenler

Örneklerin $p(x) \sim \mathcal{N}(\mu, \sigma^2)$ dağıldığı ve parametrelerin μ ve σ^2 olduğu durumu bölüm 4.4'te kısaca görmüştük. Örneklem olabilirliğini yazalım:

$$p(\mathcal{X}|\mu, \sigma^2) = \prod_t \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x^t - \mu)^2}{2\sigma^2}\right] \tag{14.5}$$

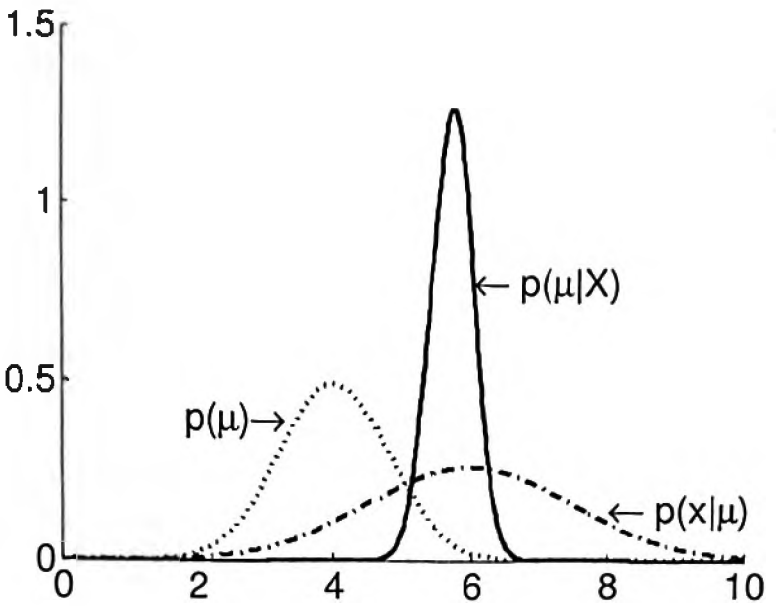
μ için eşlenik önsel Gauss dağılımıdır: $p(\mu) \sim \mathcal{N}(\mu_0^2, \sigma_0^2)$. Sonsal olasılığı yazalım:

$$\begin{aligned} p(\mu|\mathcal{X}) &\propto p(\mu)p(\mathcal{X}|\mu) \\ &\sim \mathcal{N}(\mu_N, \sigma_N^2) \end{aligned}$$

öyle ki

$$\mu_N = \frac{\sigma^2}{N\sigma_0^2 + \sigma^2}\mu_0 + \frac{N\sigma_0^2}{N\sigma_0^2 + \sigma^2}m \tag{14.6}$$

$$\frac{1}{\sigma_N^2} = \frac{1}{\sigma_0^2} + \frac{N}{\sigma^2} \tag{14.7}$$



Şekil 14.3: $p(x) \sim \mathcal{N}(6, 1.5^2)$ dağılımdan 20 örnek çekilmiştir. Önsel $p(\mu) \sim \mathcal{N}(4, 0.8^2)$ olduğunda sonsal $p(\mu|\mathcal{X}) \sim \mathcal{N}(5.7, 0.3^2)$ olur.

Burada, $m = \sum_t x^t/N$, örneklem ortalamasıdır.

Sonsal dağılımın ortalaması (Bayesçi kestirici değeri) olan μ_N, önsel ortalama (μ_0) ve örneklem ortalamasının (m) ağırlıklı ortalamasıdır ve ağırlıklar değişintileriyle ters orantılıdır (örnek için bkz. şekil 14.3). Her iki çarpan da 0 ve 1 arasında ve toplamları 1 olduğu için μ_N değeri her zaman μ_0 ve m arasındadır. Örneklem büyüklüğü N ya da önselin değişintisi σ_0^2 büyük olduğunda Bayesçi kestirim örneklem bilgisine daha fazla ağırlık verir ve m değerine daha yakın olur. σ_0^2 küçük olduğunda, yani önsel bilinmezlik küçük olduğunda ya da örneklem küçük olduğunda, önsel öngörümüz olan μ_0 değerinin etkisi daha çok olur.

σ_0 ya da σ küçüldükçe ya da N büyüdükçe, σ_N küçülür. σ_N, hem σ_0, hem de $\sigma/\sqrt{N}$ değerinden daha küçüktür; sonsalın değişintisi hem önselin, hem de örneklem kestiricisinin değişintisinden küçüktür. Her iki değeri birleştirmek sonsalı ikisinin birer birer olduğundan daha iyi bir kestirici yapar.

Değişintiyi Bayesçi yaklaşımla kestirirken değişintinin tersi olan *kesinlik* değeriyle çalışırız: $\lambda \equiv 1/\sigma^2$. Örneklem olabilirliğini buna göre yazalım:

$$\begin{aligned} p(\mathcal{X}|\lambda) &= \prod_t \frac{\lambda^{1/2}}{\sqrt{2\pi}} \exp\left[-\frac{\lambda}{2}(x^t-\mu)^2\right] \\ &= \lambda^{N/2}(2\pi)^{-N/2} \exp\left[-\frac{\lambda}{2}\sum_t (x^t-\mu)^2\right] \end{aligned} \tag{14.8}$$

Kesinlik için eşlenik önsel *Gamma dağılımıdır*:

$$p(\lambda) \sim \text{Gamma}(a_0, b_0) = \frac{1}{\Gamma(a_0)} b_0^{a_0} \lambda^{a_0-1} \exp(-b_0\lambda)$$

Sonsalı yazalım:

$$\begin{aligned} p(\lambda|\mathcal{X}) &\propto p(\mathcal{X}|\lambda)p(\lambda) \\ &\sim \text{Gamma}(a_N, b_N) \end{aligned}$$

öyle ki

$$\begin{aligned} a_N &= a_0 + N/2 \\ b_N &= b_0 + \frac{N}{2}s^2 \end{aligned} \tag{14.9}$$

Burada, $s^2 = \sum_t (x^t - \mu)^2/N$, örneklem değişintisidir. Bir kez daha sonsal kestirimin önselin söylediği ve örneklemden hesaplanan değerlerinin ağırlıklı toplamı olduğunu görüyoruz.

14.3 Bir İşlevin Parametrelerinin Bayesçi Kestirimi

Şimdi de bir dağılım yerine girdiye bağlı bağlanım ya da sınıflandırma için bir işlevin parametrelerinin Bayesçi kestirilmesini tartışacağız. Yaklaşımımız yine bu parametreleri önsel bir dağılımdan gelen rastsal değişkenler olarak alıp Bayes kuralıyla bir sonsal dağılım hesaplamak olacak. Ardından tüm integrali alabilir, integrali çözümsel hesaplayamadığımızda yaklaşıklayabilir, ya da en büyük sonsal kestiriciyi kullanabiliriz.

14.3.1 Bağlanım

Doğrusal bağlanım modeliyle bağlayalım:

$$r = \boldsymbol{w}^T\boldsymbol{x} + \epsilon \text{ öyle ki } \epsilon \sim \mathcal{N}(0, \beta^{-1}) \tag{14.10}$$

ve β eklenen gürültünün kesinliğini gösterir.

Parametreler $\boldsymbol{w}$ ağırlıklarıdır. $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}_{t=1}^N$ örneklemini ($\boldsymbol{x} \in \Re^d$ ve $r^t \in \Re$), girdi ve istenen çıktı yöneylerinden oluşan bir $\mathcal{X} = [\mathbf{X}, \boldsymbol{r}]$ dizeyi olarak yazalım. Denklem 14.10'u kullanalım:

$$p(r^t|\boldsymbol{x}^t, \boldsymbol{w}, \beta) \sim \mathcal{N}(\boldsymbol{w}^T\boldsymbol{x}, \beta^{-1})$$

Daha önce, bölüm 4.6'da log olabilirliği yazmıştık, hatırlayalım:

$$\begin{aligned} \mathcal{L}(\mathcal{X}|\boldsymbol{w}) \equiv \log p(\mathcal{X}|\boldsymbol{w}) &= \log p(\boldsymbol{r}, \mathbf{X}|\boldsymbol{w}) \\ &= \log p(\boldsymbol{r}|\mathbf{X}, \boldsymbol{w}) + \log p(\mathbf{X}) \end{aligned}$$

İkinci terim sabit ve parametrelerden bağımsızdır. İlk terimi açalım:

$$\begin{aligned} \mathcal{L}(\boldsymbol{r}|\mathbf{X}, \boldsymbol{w}, \beta) &= \log \prod_t p(r^t|\boldsymbol{x}^t, \boldsymbol{w}, \beta) \\ &= -N\log(\sqrt{2\pi}) + N\log\beta - \frac{\beta}{2}\sum_t (r^t - \boldsymbol{w}^T\boldsymbol{x}^t)^2 \end{aligned} \tag{14.11}$$

En büyük olabilirlik kestiriminde bu değeri en büyük yapan, ya da buna eşdeğer olarak toplam kare hatayı en küçük yapan w yöneyini bulmak isteriz. Toplam kare hatayı yöney-dizey gösteriminde yazalım:

$$E = (r - \mathbf{X}w)^T(r - \mathbf{X}w)$$

w parametrelerine göre türev alıp sıfıra eşitlediğimizde en büyük olabilirlik (EBO) kestiricisini buluruz (bunu daha önce bölüm 5.8'de de görmüştük):

$$w_{EBO} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T r \tag{14.12}$$

Parametreleri bulduğumuza göre şimdi öngörü yapabiliriz. Yeni x' girdisi için öngörümüzü yazalım:

$$r' = w_{EBO}^T x' \tag{14.13}$$

Doğrusal olmayan bir $g(x|w)$ modeli, örneğin, w yöneyininin bağlantı ağırlıkları olduğu bir çok katmanlı algılayıcı, için toplam hatayı

$$E(\mathcal{X}|w) = \left[r^t - g(x^t|w)\right]^2$$

küçültmede eğim iniş kullanırız ve çözüm w_{EKK} yöneyine *en küçük kareler* kestiricisi denir. Bu durumda öngörü bu parametre değeriyle hesaplanır:

$$r' = g(x'|w_{EKK})$$

Bayesçi yaklaşımda parametreler için bir *Gauss önsel dağılımı* kullanırız. Örneğin:

$$p(w) \sim \mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbf{I})$$

Bu eşlenik bir önseldir ve sonsal yine bir Gauss dağılımı olur:

$$p(w|\mathcal{X}) \sim \mathcal{N}(\mu_N, \Sigma_N)$$

öyle ki

$$\mu_N = \beta\Sigma_N\mathbf{X}^T r \tag{14.14}$$

$$\Sigma_N = (\alpha\mathbf{I} + \beta\mathbf{X}^T\mathbf{X})^{-1} \tag{14.15}$$

Öngörüyü hesaplarken tüm sonsal üzerinden toplam alırız:

$$r' = \int w^T x' p(w|\mathcal{X})dw$$

En büyük sonsal (ya da Bayesçi) kestirici

$$w_{EBS} = \mu_N = \beta(\alpha\mathbf{I} + \beta\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T r \tag{14.16}$$

kullanırsak dağılımı tek bir noktaya, yani ortalamaya, indirgemiş oluruz:

$$r' = \boldsymbol{w}_{EBS}^T \boldsymbol{x}'$$

Değişintisini de hesaplayabiliriz:

$$\text{Var}(r') = \beta^{-1} + (\boldsymbol{x}')^T \boldsymbol{\Sigma}_N \boldsymbol{x}' \tag{14.17}$$

Denklem 14.16'yı denklem 14.12'deki en büyük olabilirlik kestiricisiyle karşılaştırırsak bir tür düzenlileştirme yaptığını görebiliriz: Tersi alınan dizeyi koşullandırmak için köşegenine α sabiti ekliyoruz.

$p(\boldsymbol{w}) \sim \mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbf{I})$ önsel dağılımı, ağırlıkların 0 çevresinde ve α ile ters orantılı bir yayılımla dağıldığını söyler. α sıfıra giderken önsel tekdüzeleşir ve en büyük sonsal kestirici, en büyük olabilirlik kestiricisine yakınsar.

Şekil 14.4'te, α değerini artırdığımızda parametreleri sıfıra yaklaşmaya zorladığımızı ve sonsal dağılımın sıfıra yaklaşıp büzüldüğünü görüyoruz. β değerini artırmak daha yüksek değişintili gürültü varsaydığından sonsalın değişintisi de artar.

Sonsal dağılımın logaritmasını yazalım:

$$\begin{aligned}
\log p(\boldsymbol{w}|\mathbf{X}, \boldsymbol{r}) &\propto \log p(\mathbf{X}, \boldsymbol{r}|\boldsymbol{w}) + \log p(\boldsymbol{w}) \\
&\propto \log p(\boldsymbol{r}|\mathbf{X}, \boldsymbol{w}) + \log p(\boldsymbol{w}) \\
&= -\frac{\beta}{2}\sum_t (r^t - \boldsymbol{w}^T\boldsymbol{x}^t)^2 - \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + c
\end{aligned}$$

Bu işlevi en büyük yapan değer en büyük sonsal kestiricimizdir. Genel durumda, $g(\boldsymbol{x}|\boldsymbol{w})$ modeli için genişletilmiş hata işlevi yazabiliriz

$$E_{ridge}(\boldsymbol{w}|\mathcal{X}) = \sum_t [r^t - g(\boldsymbol{x}^t|\boldsymbol{w})]^2 + \lambda \sum_i w_i^2$$

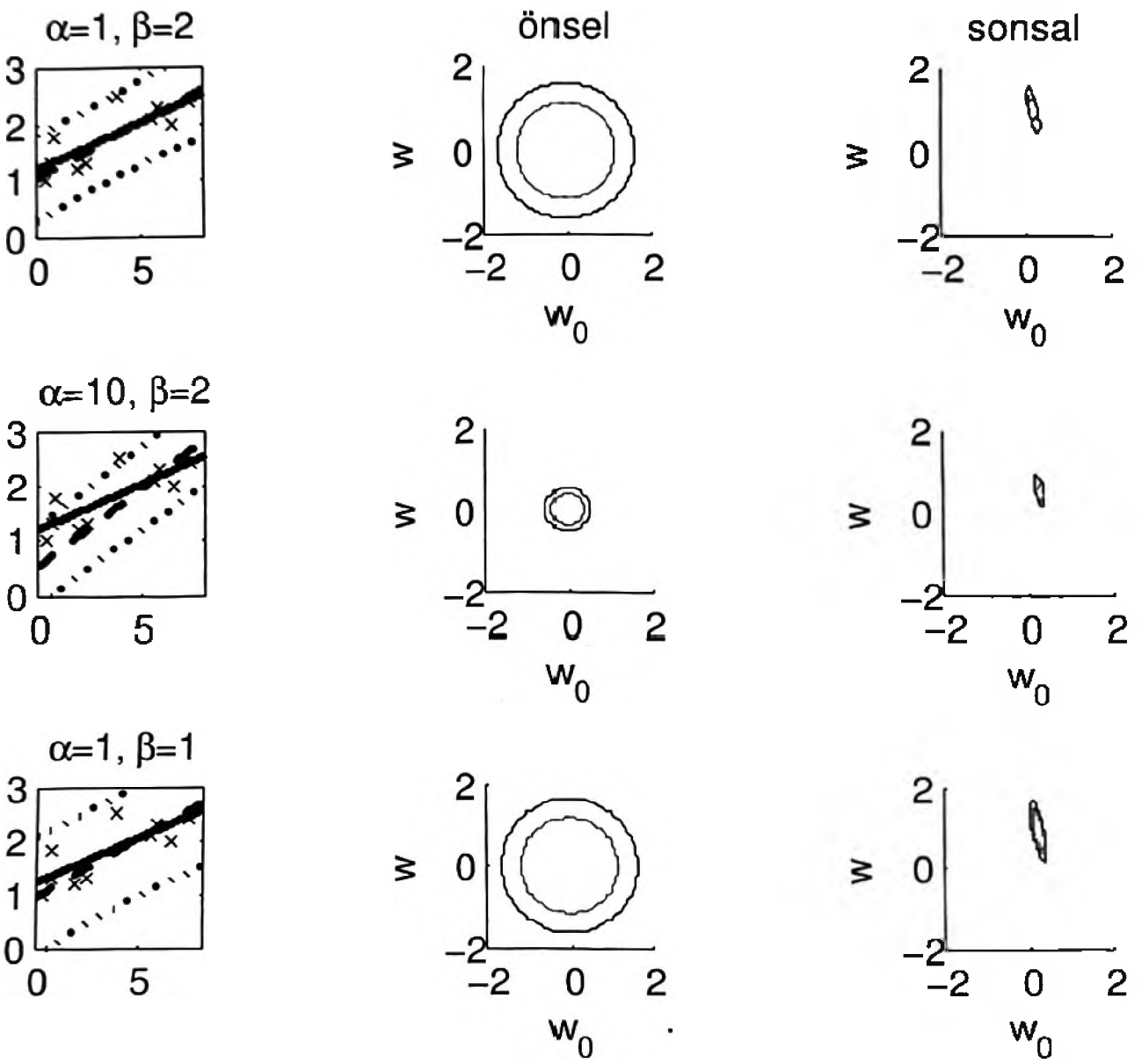
öyle ki $\lambda \equiv \alpha/\beta$. Buna, *parametre büzüşmesi* ya da istatistikte *sırt bağlanımı* denir. Bölüm 4.8'de bunu *düzenlileştirme*, bölüm 11.9'da yapay sinir ağları bağlamında *ağırlık sönümü* diye adlandırdık. İlk terim olabilirliğin eksi logaritmasıdır; ikinciyse w_i değerlerinin sıfırdan uzaklaşmasını (önselin α parametresine bağlı olarak) cezalandırır.

Her ne kadar bu yaklaşım $\sum_i w_i^2$ değerini azaltsa da w_i değerlerini birer birer sıfıra çekmeyebilir, yani, hangi x_i girdilerinin gereksiz olduğunu anlayıp boyut azaltma için uygun değildir. Bu amaç için, L_2 yerine L_1 büyüklüğüne dayanan *Laplace önseli* kullanılır (Figueiredo 2003):

$$p(\boldsymbol{w}|\alpha) = \prod_i \frac{\alpha}{2}\exp(-\alpha|w_i|) = \left(\frac{\alpha}{2}\right)^d \exp\left(-\alpha\sum_i |w_i|\right)$$

Sonsal dağılım artık Gauss değildir ve en büyük sonsal kestiricisi

$$E_{lasso}(\boldsymbol{w}|\mathcal{X}) = \sum_t (r^t - \boldsymbol{w}^T\boldsymbol{x}^t)^2 + 2\sigma^2\alpha\sum_i |w_i|$$



Şekil 14.4: Farklı α ve β değerleri için Bayesçi doğrusal bağlanım. Solda: çarpılar veri noktalarını ve doğru, en büyük olabilirlik çözümünü gösterir. En büyük sonsal çözümü ve bir standart sapma hata çubuğu kesikli çizgiyle gösterilmiştir. Ortada: 0 merkezli ve $1/\alpha$ değişintili önsel dağılım. Sağda: merkezi en büyük sonsal çözümü olan sonsal dağılım. α arttığında önselin değişintisi azalır ve doğru sabit 0 doğrusuna yaklaşır. β azaldığında daha fazla gürültü varsayılır ve sonsal dağılımın değişintisi artar.

hatasını azaltan değerdir; burada σ^2 gürültünün değişintisidir (ve yerine bir değer kestirip koyarız). Bu yaklaşıma *lasso* denir (Tibshirani 1996). L_1 büyüklüğünün neden seyrekliği sağladığını anlamak için iki ağırlıklı $[w_1, w_2]^T$ bir örnek görelim: $\|[1,0^T\|_2 = \|[1/\sqrt{2}, 1/\sqrt{2}]^T\|_2 = 1$, ama $\|[1,0]^T\|_1 = 1 < \|[1/\sqrt{2}, 1/\sqrt{2}]^T\|_1 = \sqrt{2}$. Bu yüzden L_1, her iki ağırlığı da küçük yapmak yerine w_2 değerini 0 yapıp büyük bir w_1 değeri kullanır (Figueiredo 2003).

14.3.2 Taban ve Çekirdek İşlevlerinin Kullanımı

Denklem 14.14'teki Bayesçi kestirimi öngörüde kullanalım:

$$\begin{aligned} r' &= (\boldsymbol{x}')^T \boldsymbol{w} \\ &= \beta(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N \mathbf{X}^T \boldsymbol{r} \\ &= \sum_t \beta(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N \boldsymbol{x}^t r^t \end{aligned}$$

Bu biçimde parametreleri ve ardından öngörüyü öğrenme örnekleri üzerinden yazmaya *eşlek gösterim* denir. Parametreyi öğrenme verisi (ya da destek yöney makinelerindeki (bölüm 13) gibi bir altkümesi) kullanarak yazdığımızda öngörü değeri, yeni girdiyle geçmiş verinin bir işlevi olur. Bunu bir çekirdek işlevi tanımlayıp kullanarak yeniden yazalım:

$$r' = \sum_t K(\boldsymbol{x}', \boldsymbol{x}^t) r^t \tag{14.18}$$

öyle ki

$$K(\boldsymbol{x}', \boldsymbol{x}^t) = \beta(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N \boldsymbol{x}^t \tag{14.19}$$

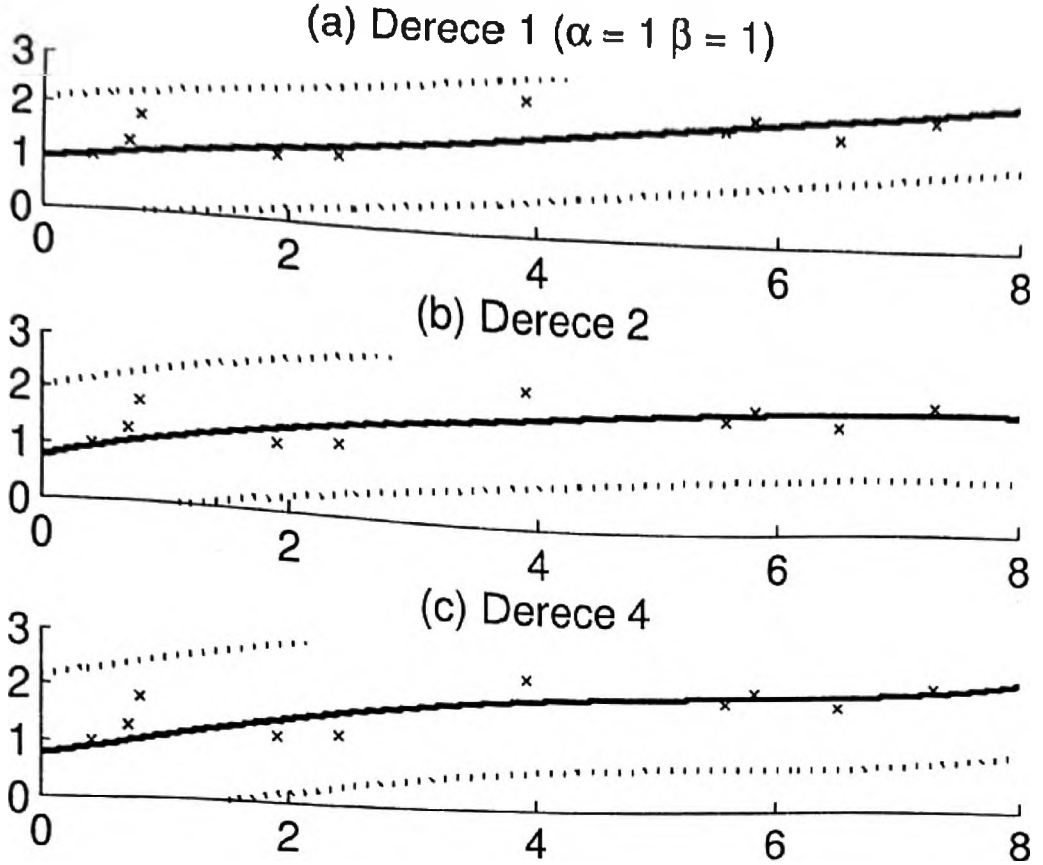
Denklem 14.19'daki doğrusal çekirdeği, doğrusal olmayan bir $\boldsymbol{\phi}(\boldsymbol{x})$ *taban işleviyle* yeni bir uzaya geçip orada doğrusal bir model kullanarak genelleyebildiğimizi biliyoruz. Böyle bir durumda d boyutlu $\boldsymbol{x}$ yerine k taban işlevi için k boyutlu $\boldsymbol{\phi}(\boldsymbol{x})$ ile çalışacağız ve elimizde $N \times d$ veri dizeyi $\mathbf{X}$ yerine taban işlev çıktılarından oluşan $N \times k$ büyüklüğünde $\boldsymbol{\Phi}$ dizeyi olacak.

Bu genel durumda yeni bir örnek için öngörü yapalım:

$$\begin{aligned} r' &= \boldsymbol{\phi}(\boldsymbol{x}')^T \boldsymbol{w} \text{ öyle ki } \boldsymbol{w} = \beta \boldsymbol{\Sigma}_N^{\phi} \boldsymbol{\Phi}^T \boldsymbol{r} \text{ ve } \boldsymbol{\Sigma}_N^{\phi} = \left(\alpha \mathbf{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}\right)^{-1} \\ &= \beta \boldsymbol{\phi}(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N^{\phi} \boldsymbol{\Phi}^T \boldsymbol{r} \\ &= \sum_t \beta \boldsymbol{\phi}(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N^{\phi} \boldsymbol{\phi}(\boldsymbol{x}^t) r^t \\ &= \sum_t K(\boldsymbol{x}', \boldsymbol{x}^t) r^t \end{aligned} \tag{14.20}$$

öyle ki

$$K(\boldsymbol{x}', \boldsymbol{x}^t) = \beta \boldsymbol{\phi}(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N^{\phi} \boldsymbol{\phi}(\boldsymbol{x}^t) \tag{14.21}$$



Şekil 14.5: Farklı çekirdek ve bir standart sapma hata çubuğuyla Bayesçi bağlanım. (a) doğrusal: $\phi(x) = [1, x]^T$, (b) ikinci derece: $\phi(x) = [1, x, x^2]^T$, ve (c) dördüncü derece: $\phi(x) = [1, x, x^2, x^3, x^4]^T$.

Yeni girdi için kestirim değeri, öğrenme örneklerinin etkilerinin ağırlıklı bir toplamıdır ve bu etki değerleri, $K(x', x^t)$ *çekirdek işlevi* tarafından tanımlanır; bunu daha önce bölüm 8'deki dağılımdan bağımsız çekirdek düzleştiricilerinde ve bölüm 13'teki çekirdek makinelerinde de görmüştük.

Öngörümüzün güven aralığını belirten hata çubuğunu girdiye bağlı olarak hesaplayabiliriz:

$$\text{Var}(r') = \beta^{-1} + \phi(x')^T \Sigma_N^{\phi} \phi(x')$$

Şekil 14.5'te doğrusal, ikinci, ve dördüncü derece çekirdeklerle örnek bağlanım gösterilmiştir.

Geleneksel bağlanımda x yerine $\phi(x)$ uzayında çalışabildiğimiz gibi burada, Bayesçi bağlanımda da $\phi(x)$ uzayında çalışıp parametreleri o uzayda tanımlayabiliriz. Bu bölümde az sonra, $\phi(x)$ taban işlevlerini tanımlamaya ya da hesaplamaya gerek olmadan doğrudan $K(x, x^t)$ çekirdekleriyle öngörü yapabildiğimiz Gauss süreçlerini göreceğiz.

14.3.3 Bayesçi Sınıflandırma

İki sınıflı bir uygulamada tek çıktımız vardır. Doğrusal bir model varsayalım:

$$P(C_1|\boldsymbol{x}^t) = y^t = \text{sigmoid}(\boldsymbol{w}^T\boldsymbol{x}^t)$$

Bernoulli örneklemin olabilirliğini yazalım:

$$\mathcal{L}(\boldsymbol{r}|\mathbf{X}) = \sum_t r^t \log y_t + (1-r^t)\log(1-y^t)$$

En büyük olabirlik kestiriminde, bunu en büyük yapan değeri, ya da eşdeğer olarak logaritmasının eksisini, yani çapraz düzensizliği, en küçük yapan değeri, örneğin, eğim inişle buluruz. Buna bölüm 10.7'de S biçimli sınıflandırma demiştik. Şimdi bunu nasıl Bayesçi yapacağımızı göreceğiz.

Bayesçi yaklaşımda önselin Gauss olduğunu varsayalım:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{m}_0, \mathbf{S}_0) \tag{14.22}$$

ve sonsalın logaritmasını yazalım:

$$\begin{aligned}\log p(\boldsymbol{w}|\boldsymbol{r}, \mathbf{X}) &\propto \log p(\boldsymbol{w}) + \log p(\boldsymbol{r}|\boldsymbol{w}, \mathbf{X}) \\ &= -\frac{1}{2}(\boldsymbol{w}-\boldsymbol{m}_0)^T\mathbf{S}_0^{-1}(\boldsymbol{w}-\boldsymbol{m}_0) \\ &\quad + \sum_t r^t \log y_t + (1-r^t)\log(1-y^t) + c \end{aligned} \tag{14.23}$$

Sonsal dağılım artık Gauss değildir ve integralini kolayca alamayız, ama *Laplace yaklaşıklama* kullanabiliriz (MacKay 2003). Bir $f(x)$ dağılımını yaklaşıklamak istediğimizi düşünelim. Laplace yaklaşıklamada x_0 tepe noktasını bulup orayı merkez alan bir $q(x)$ Gauss dağılımı oturtur, ardından örneğin integral almak istiyorsak bu yaklaşık Gauss dağılımının integralini alırız. Gauss dağılımının değişintisini bulurken $f(x)$ dağılımının $x = x_0$'daki Taylor açılımını kullanırız:

$$\log f(x) = \log f(x_0) - \frac{1}{2}a(x-x_0)^2 + \cdots$$

öyle ki

$$a \equiv -\frac{d}{dx^2}\log f(x)\Big|_{x=x_0}$$

Tepe noktasında birinci türev 0 olacağı için açılımdaki doğrusal terim yok olur. exp işlevinden geçirelim:

$$f(x) = f(x_0)\exp\left[-\frac{a}{2}(x-x_0)^2\right]$$

$f(x)$ dağılımını normalleştirirken Gauss dağılımını düşünelim:

$$\int \frac{1}{\sqrt{2\pi}(1/\sqrt{a})}\exp\left[-\frac{a}{2}(x-x_0)^2\right] = 1 \Rightarrow \int \exp\left[-\frac{a}{2}(x-x_0)^2\right] = \sqrt{a/2\pi}$$

ve bu yüzden

$$q(x) = \sqrt{a/2\pi}\exp\left[-\frac{a}{2}(x-x_0)^2\right] \sim \mathcal{N}(x_0, 1/a)$$

Çok boyutlu $\boldsymbol{x} \in \Re^d$ durumu için genelleyelim:

$$\log f(\boldsymbol{x}) = \log f(\boldsymbol{x}_0) - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_0)^T\mathbf{A}(\boldsymbol{x}-\boldsymbol{x}_0) + \cdots$$

Burada $\mathbf{A}$, ikinci türev (Hess) dizeyidir:

$$\mathbf{A} = -\nabla\nabla \log f(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_0}$$

Laplace yaklaşıklamayı türetelim:

$$f(\boldsymbol{x}) = \frac{|\mathbf{A}|^{1/2}}{(2\pi)^{d/2}}\exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_0)^T\mathbf{A}(\boldsymbol{x}-\boldsymbol{x}_0)\right] \sim \mathcal{N}_d(\boldsymbol{x}_0, \mathbf{A}^{-1})$$

Yaklaşıklamayı genel olarak konuştuğumuza göre şimdi Bayesçi kestirimde sonsal dağılım için nasıl kullanabiliriz. $p(\boldsymbol{w}|\boldsymbol{r}, \mathbf{X})$ sonsal dağılımının $\boldsymbol{w}_{EBS}$ tepe noktasını merkez alırız; eşdeğişinti dizeyi de olabilirliğin logaritmasının eksisinin ikinci türev dizeyinin tersidir:

$$\mathbf{S}_N = -\nabla\nabla \log p(\boldsymbol{w}|\boldsymbol{r}, \mathbf{X}) = \mathbf{S}_0^{-1} + \sum_t y^t(1-y^t)\boldsymbol{x}^t(\boldsymbol{x}^t)^T$$

Ardından sınıf olasılığını hesaplamak için bu Gauss üzerinden integral alırız:

$$P(C_1|\boldsymbol{x}) = y = \int \text{sigmoid}(\boldsymbol{w}^T\boldsymbol{x})q(\boldsymbol{w})d\boldsymbol{w}$$

öyle ki $q(\boldsymbol{w}) \sim \mathcal{N}(\boldsymbol{w}_{EBS}, \mathbf{S}_N^{-1})$. Ama ne yazık ki içeride sigmoid işlevinin olması işi karıştırır ve yerine (yine S biçimli) *probit işlevini* kullanmamız gerekir (Bishop 2006).

14.4 Gauss Süreçleri

Elimizde $y = \boldsymbol{w}^T\boldsymbol{x}$ doğrusal modeli olsun. Bu durumda her $\boldsymbol{w}$ için bir doğru vardır. Bir önsel $p(\boldsymbol{w})$ dağılımı doğrular üzerine bir dağılımı tanımlar, ve her $\boldsymbol{x}$ için $p(\boldsymbol{w})$ dağılımdan gelen $\boldsymbol{w}$ değerlerine bağlı $y(\boldsymbol{x}|\boldsymbol{w})$ değerlerinden oluşan bir y değerleri dağılımı vardır. Gauss süreçlerinden bahsederken bahsettiğimiz budur. $p(\boldsymbol{w})$ Gauss dağılmışsa her y de Gauss dağılımlarının doğrusal birleşimi olduğu için yine Gauss dağılır; bizi ilgilendiren N tane, $\boldsymbol{x}^t, t = 1, \ldots, N$, veri noktasında y değerlerinin birleşik dağılımıdır (MacKay 1998).

0 ortalamalı Gauss önsel varsayalım:

$$p(\boldsymbol{x}) \sim \mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbf{I})$$

$N \times d$ büyüklüğünde $\mathbf{X}$ veri dizeyi ve $d \times 1$ büyüklüğünde w ağırlık yöneyini kullanarak y çıktı yöneyini yazalım:

$$y = \mathbf{X}w \tag{14.24}$$

Bu değerler birleşik olarak N değişkenli bir Gauss dağılım oluşturur, öyle ki:

$$\begin{aligned} E[y] &= \mathbf{X}E[w] = \mathbf{0} \\ \text{Cov}(y) &= E[yy^T] = \mathbf{X}E[ww^T]\mathbf{X}^T = \frac{1}{\alpha}\mathbf{X}\mathbf{X}^T \equiv \mathbf{K} \end{aligned} \tag{14.25}$$

Burada $\mathbf{K}$, öğeleri

$$K_{i,j} \equiv K(x^i, x^j) = \frac{(x^i)^T x^j}{\alpha}$$

olan (Gram) dizeyidir. Gauss süreçleri yazınında buna *eşdeğişinti işlevi* denir ve altta yatan düşünce çekirdek işleviyle aynıdır: $\phi(x)$ taban işlevleri kullanarak asıl uzay yerine çekirdek işlevinin tanımladığı taban işlevinin uzayında iç çarpım alırız:

$$K_{i,j} = \frac{\phi(x^i)^T \phi(x^j)}{\alpha}$$

Gözlenen gerçek r değeri doğruya gürültü eklenerek oluşur, $r = y + \epsilon$; gürültünün de Gauss dağıldığını varsayıyoruz: $\epsilon \sim \mathcal{N}(0, \beta^{-1})$. Tüm N veri noktasının birleşik dağılımını yazalım:

$$r \sim \mathcal{N}_N(\mathbf{0}, \mathbf{C}_N) \text{ öyle ki } \mathbf{C}_N = \beta^{-1}\mathbf{I} + \mathbf{K} \tag{14.26}$$

Öngörü yaparken yeni $(N+1)$. veri noktasını (x', r') veri çifti olarak veriye katıp tüm $N+1$ veri noktası için birleşik dağılımı yazarız:

$$r_{N+1} \sim \mathcal{N}_N(0, \mathbf{C}_{N+1}) \tag{14.27}$$

öyle ki

$$\mathbf{C}_{N+1} = \begin{bmatrix} \mathbf{C}_N & k \\ k^T & c \end{bmatrix}$$

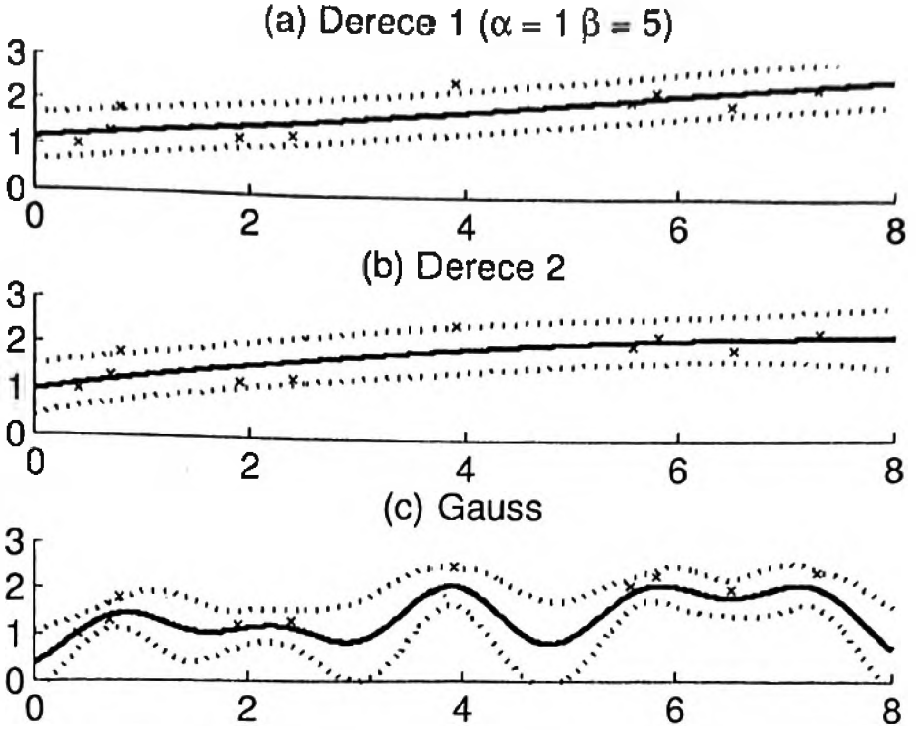
Burada, k, $N \times 1$ boyutludur ve $K(x', x^t), t = 1, \ldots, N$ değerlerinden oluşur. $c = K(x', x') + \beta^{-1}$ sabittir. Ardından öngörü yaparken $p(r'|x', \mathbf{X}, r)$ dağılımını hesaplarız, ki parametreleri

$$\begin{aligned} E[r'|x'] &= k^T \mathbf{C}_N^{-1} r \\ \text{Var}(r'|x') &= c - k^T \mathbf{C}_N^{-1} k \end{aligned}$$

olarak türetilebilir.

Şekil 14.6'da doğrusal, ikinci derece, ve Gauss çekirdekleriyle bir örnek gösterilmiştir. İlk iki çekirdek, karşılık gelen taban işlevlerinin iç çarpımı olarak tanımlıdır; Gauss çekirdeğiyse doğrudan tanımlanmıştır:

$$K_G(x^i, x^j) = \exp\left[-\frac{\|x^i - x^j\|^2}{s^2}\right]$$



Şekil 14.6: Gauss süreç bağlanımı, bir standart sapma hata çubuğuyla gösterilmiştir: (a) doğrusal çekirdek, (b) ikinci derece çekirdek, (c) yarıçap değeri $s^2 = 0.5$ olan Gauss çekirdeği.

Nokta kestirimimiz olan ortalama değer, çekirdek işleviyle tanımlanan etkilerin ağırlıklı toplamıdır:

$$E[r'|x'] = \sum_t a^t K(x^t, x') \tag{14.28}$$

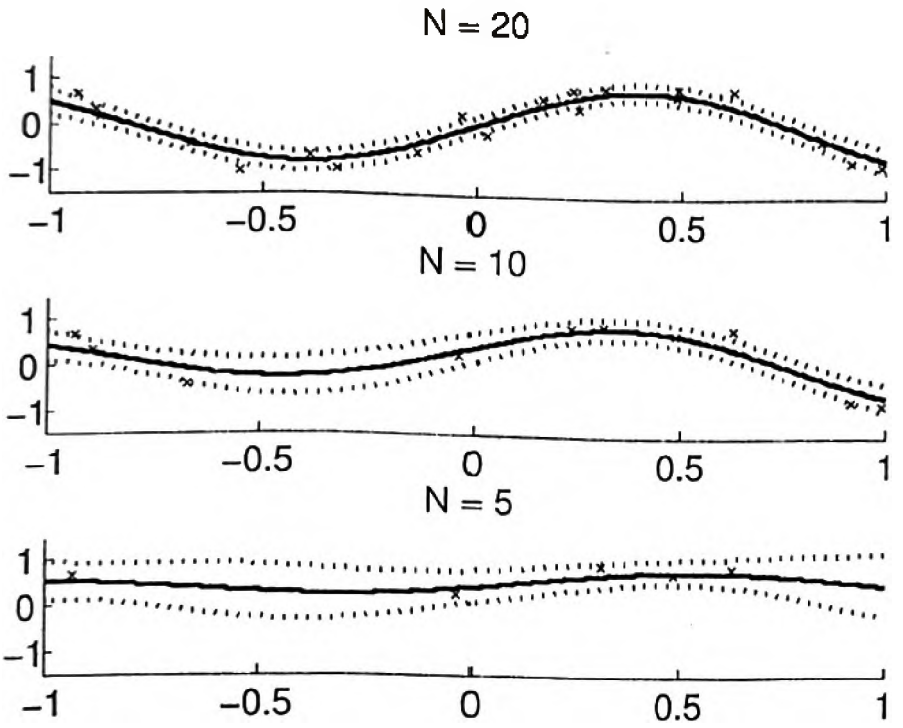
Burada a^t değeri $\mathbf{C}_N^{-1}\boldsymbol{r}$ yöneyinin t. öğesidir. Ya da başka bir açıdan bakarsak çıktı, öğrenme örneklerinin çıktılarının ağırlıklı toplamıdır ve ağırlıklar çekirdek değerleriyle tanımlanmıştır:

$$E[r'|x'] = \sum_t r^t w^t \tag{14.29}$$

ve burada w^t değeri $\boldsymbol{k}^T\mathbf{C}_N^{-1}$ yöneyinin t. öğesidir.

Herhangi bir noktada öngörü yanında o noktadaki öngörünün değişintisini de hesaplayabiliriz; bu değişinti değeri bilinmezliği göstermesi açısından yararlıdır, ve o noktadaki kestirimi etkileyen örneklere bağlıdır. Örneğin Gauss çekirdeğinde her örnek yerel olarak etkili olduğundan çevresinde az öğrenme noktası olan bölgelerde değişinti yüksektir (bkz. şekil 14.7).

Daha önce bölüm 13'te çekirdek makinelerindeki gibi, uygulamaya bağlı olarak farklı çekirdek işlevleri tanımlanıp kullanılabilir. Bir taban işlevi tanımlamaya, hesaplamaya, saklamaya gerek olmadan doğrudan çekirdek işlevinin kullanılabilmesi büyük bir esneklik sağlar. Geleneksel yaklaşımda önce öğrenme kümesinden, örneğin



Şekil 14.7: Farklı büyüklükte öğremöe kümeleriyle Gauss süreç bağlanımı örnekleri. Yarıçap değeri $s^2 = 0.5$ alınmıştır. Az veri olan bölgelerde değişintinin daha fazla olduğunu görüyoruz.

denklem 14.12'yle, parametreleri hesaplar, ardından, örneğin denklem 14.13'ü kullanarak bu parametrelerle, bir daha öğrenme kümesine gerek duymadan, öngörü yaparız. Çoğunlukla parametre sayısı $\mathcal{O}(d)$ olup öğrenme kümesi büyüklüğü olan N değerinden küçüktür ve bu yaklaşım mantıklıdır.

Taban işlevleri kullandığımızdaysa taban işlevlerinin boyutu çok yüksek, hatta sonsuz olabilir ve parametreleri ayrı olarak gösterip hesaplayamayız. Bu durumda eşlenik gösterimi kullanmak ve çekirdek işleviyle öğrenme örnekleri üzerinden hesaplamak çok daha kolaydır. Aynı düşünce, dağılımdan bağımsız düzleştiriciler (bölüm 8) ve çekirdek makinelerinde de (bölüm 13) geçerlidir.

Buradaki ister, $\mathbf{C}_N$ dizeyinin tersi alınabilir ve kesin artı olmasıdır. Bunun için $\mathbf{K}$ dizeyinin yarı kesin artı olması gerekir ki köşegene uygun $\beta^{-1} > 0$ eklenerek kesin artılık kazanılabilsin. En pahalı işlem bu $N \times N$ dizeyin tersinin alınma işidir ama bir kez (öğrenme sırasında) yapılıp öyle saklanabilir. Yine de çok büyük N için yaklaşıklamak gerekebilir.

İki sınıf için sınıflandırmada kullandığımızda çıktı S biçimli işlevden geçtiği için, $y = \text{sigmoid}(w^T x)$, birleşik y dağılımı artık Gauss değildir. Yine aynı yaklaşımla bir Gauss süreci sınıflandırıcısı türetilebilir; tek fark, $p(r_{N+1}|x_{N+1}, \mathbf{X}, r)$ koşullu dağılımı Gauss olmayacağından bir yaklaşıklama, örneğin Laplace yaklaşıklama, gerekmesidir (Bishop 2006; Rasmussen ve Williams 2006).

14.5 Kısa Kısa

Son yıllarda bilgisayarların hesap gücünün artmasıyla sonsal dağılımı örnekleyen ya da yaklaşıklayan Bayesçi yaklaşımlar daha yaygın kullanılır olmuştur. Aklın yolu birdir. Basitliğin yeğlenmesi kendini farklı bağlamlarda Bayesçi kestirim, düzenlileştirme, en kısa tanım uzunluğu, ya da düzleştirme adı altında gösterir ve istatistiksel çıkarım, ve dolayısıyla yapay öğrenmenin temelidir.

Öte yandan önsellerin öznelliği rahatsız edicidir ve Bayesçi yaklaşım bu yüzden eleştirilir; örneğin bkz. (Gelman 2008). Tekdüze bir önsel yararlı mıdır? Dar bir tepe yapmış önselimiz varsa veri örneklemeye gerek var mıdır? Eşlenik önseli doğru olduğu için mi kullanıyoruz, işimizi kolaylaştırdığı için mi?

Destek yöney makinelerindeki gibi Gauss süreçlerinde de, örneğin ağırlıklı toplam alarak, bilinen çekirdeklerden yenileri oluşturulabilir ve bu ağırlıkları ya da çekirdek parametrelerini (örneğin yarıçapları) *2. tür en büyük olabilirlik kestirimiyle* hesaplayabiliriz; buna *ikinci* tür denmesi, parametreler (örneğin, a^t ya da w^t) için değil ikinci dereceden üstparametreler için yapılmasıdır (Bishop 2006; Rasmussen ve Williams 2006).

14.6 Alıştırmalar

1. Şekil 14.3'teki örnekte N, σ^2, ve σ_0^2 değiştiğinde sonsalın nasıl değiştiğini gözleyin.
2. n büyüklüğündeki bir örneklemdeki istenmeyen eposta sayısı x olsun. İstenmeyen eposta oranı için önsel q dağılımının $[0, 1]$ aralığında tekdüze olduğunu varsayarsak sonsal $p(q|x)$ dağılımını hesaplayın.
3. Yukarıdaki alıştırmanın aynısını varsayın, tek fark, $p(q) \sim \mathcal{N}(\mu_0, \sigma_0^2)$ olsun. Ayrıca, n değerinin büyük olduğunu ve merkezi limit kuramıyla binom dağılımını Gauss ile yakınsayabildiğimizi varsayın. Bu durumda $p(q|x)$ dağılımını türetin.
4. En büyük olabilirilik kestiricisi kullanıldığında $\text{Var}(r')$ ne olur? Denklem 14.17'yle karşılaştırın.
5. Şekil 14.6'da s^2 değiştiğinde oturtulan işlev nasıl değişir?
6. Gauss süreçlerinde öğrenme kümesinin bir altkümesini seçen bir süzme algoritması önerin.
7. *Etkin öğrenmede* öğrenici, edilgen bir biçimde öğrenme kümesi beklemek yerine kendisi x girdileri oluşturup gözetmenden karşılık gelen r değerlerini sorgulayabilir. Gauss süreçlerinde etkin öğrenme nasıl yapılabilir? (İpucu: Bilinmezliğin en çok nerede olduğunu düşünün.)
8. İki farklı gösterimden girdilerimiz olduğunu düşünelim. Bu bölümde gördüğümüz yaklaşımları bu durumda nasıl kullanabiliriz?

14.7 Kaynaklar

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning.* New York: Springer.

Figueiredo, M. A. T. 2003. "Adaptive Sparseness for Supervised Learning." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25: 1150–1159.

Gelman, A. 2008. "Objections to Bayesian statistics." *Bayesian Statistics* 3: 445–450.

MacKay, D. J. C. 1998. "Introduction to Gaussian Processes." *Neural Networks and Machine Learning*, ed. C. M. Bishop, 133–166. Berlin: Springer.

MacKay, D. J. C. 2003. *Information Theory, Inference, and Learning Algorithms*. Cambridge, UK: Cambridge University Press.

Rasmussen, C. E., C. K. I. Williams. 2006. *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press.

Tibshirani, R. 1996. "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society B* 58: 267–288.

Bölüm 15

Saklı Markov Modelleri

Öğrenme kümesindeki örneklerin bağımsız olma koşulunu terk edip girdi dizilerinin belirli parametrelerce tanımlanan rastsal bir süreç tarafından oluşturulduğunu varsayan Markov modellerini tanıtacağız. Hem böylesi modellemenin nasıl yapıldığını, hem de örnek dizilerden model parametrelerinin nasıl öğrenileceğini tartışacağız.

15.1 Giriş

Bu bölüme dek, örnek kümemizi oluşturan örneklerin bağımsız ve özdeşçe dağıldığını varsaymış ve örneklem olabilirliğini örneklerin birer birer olabilirliklerinin çarpımı olarak yazabilmiştik. Ne yazık ki bu varsayım arka arkaya gelen örneklerin bağımlı olduğu uygulamalarda geçerli değildir. Örneğin, bir sözcükte arka arkaya gelen harfler bağımlıdır; Türkçede 'b' harfinden sonra 'a' harfinin gelme olasılığı yüksekken, 'f' harfinin gelme olasılığı çok düşüktür. Bir sözcükteki harfler ya da DNA içinde taban çiftleri gibi gözlem *dizilerinden* bahsettiğimiz uygulamalarda girdiyi basit olasılık dağılımları olarak modelleyemiyoruz. Bir başka örnek olarak konuşma tanımada, seslerin temel sesbirimciklerinden oluştuğu ve bunların yalnızca belirli biçimlerde sıralanabildiği varsayılır, ki bunlar da dilin sözüklerine karşılık gelir. Daha yüksek bir düzeyde bakarsak sözcüklerin o dilin sözdizimsel ve anlambilimsel kurallarına bağlı olarak ancak belirli biçimlerde sıralanabildiğini ve izin verilen bu sıralamaların da geçerli cümlelere karşılık geldiği söylenebilir.

Bir dizi, belirli parametrelerce tanımlanan *rastsal bir süreççe* üretilmiş olarak betimlenebilir. Bu bölümde, böylesi modellerden, parametrelerinden ve bu parametrelerin verilen bir örnek dizi kümesinden nasıl eğitilebileceğini tartışacağız.

15.2 Kesikli Markov Süreçleri

$S_1, S_2, \ldots, S_N$ ile gösterdiğimiz N tane farklı kesikli durum olduğunu ve dizgenin herhangi bir anda bunların birinde olduğunu düşünelim. t anındaki durumu $q_t, t = 1, 2, \ldots,$ diye yazalım, öyle ki, $q_t = S_i$ yazdığımızda dizgenin t anında S_i durumunda olduğunu anlayalım. Bu bölümde, sıralama sanki yalnızca zaman içinde olurmuş gibi t "anı" ya da "zamanı," diyeceğiz ama t herhangi bir dizin değişkeni olabilir ve burada tartıştığımız modellemeyi zamanda, konumda, DNA dizisi içindeki sıralamada, ya da başka herhangi bir sıralamada uygulayabiliriz.

Dizge, bir "saatin" her vuruşunda geçmişteki durumlarına bağlı olarak belirli bir olasılıkla başka bir duruma geçer:

$$P(q_{t+1} = S_j | q_t = S_i, q_{t-1} = S_k, \cdots)$$

Birinci dereceden bir *Markov modelinde*, $t+1$ anındaki durum yalnızca t anındaki duruma bağlıdır, ve daha önceki anlardaki durumlardan bağımsızdır:

$$P(q_{t+1} = S_j | q_t = S_i, q_{t-1} = S_k, \cdots) = P(q_{t+1} = S_j | q_t = S_i) \tag{15.1}$$

Şu an verildiğinde gelecek geçmişten bağımsızdır. "Bugün kalan hayatımızın ilk günüdür," deyişinde kast edilen tam olarak budur.

Modelimizi daha da basitleştirmek (düzenlileştirmek) için bu *geçiş olasılıklarının* zamandan bağımsız olduğunu varsayarız:

$$a_{ij} \equiv P(q_{t+1} = S_j | q_t = S_i) \tag{15.2}$$

Geçiş olasılıkları eksi olamaz ve bir durumdan olası tüm öteki durumlara geçme olasılıklarının toplamı 1 olmak zorundadır:

$$a_{ij} \geq 0 \text{ and } \sum_{j=1}^{N} a_{ij} = 1 \tag{15.3}$$

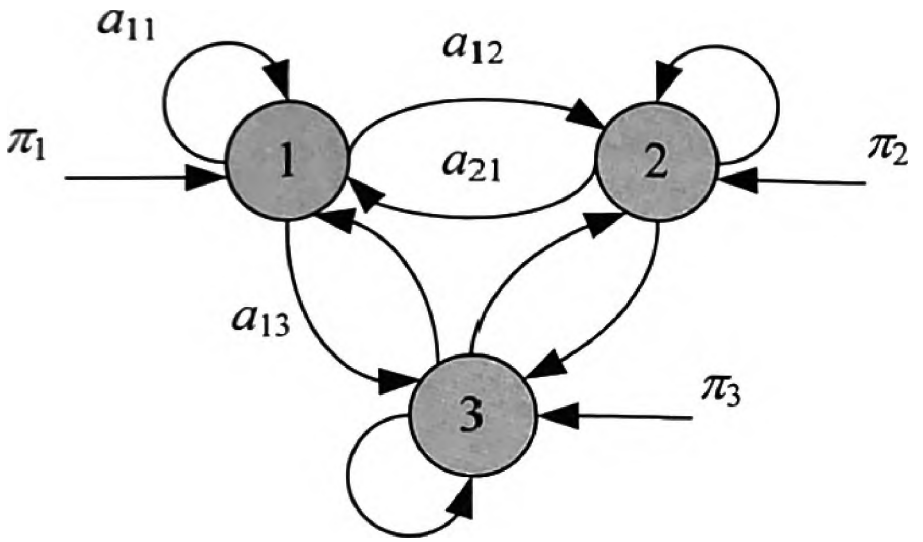
Bu da demek oluyor ki, S_i durumundan S_j durumuna geçme olasılığı, hangi anda ya da dizinin neresinde olursak olalım aynıdır. Geçiş olasılıklarını (satırlarının toplamı 1 olan) $N \times N$ boyutlu $\mathbf{A} = [a_{ij}]$ dizeyi olarak gösteriyoruz.

Böylesi bir model, *rastgele bir özdevinir* (bkz. şekil 15.1) olarak görebiliriz. S_i durumundan S_j durumuna a_{ij} olasılığıyla geçeriz ve bu olasılık tüm t değerleri için aynıdır. Buna tek istisna ilk durumdur. $t = 1$ anındaki durum için π_i ile gösterdiğimiz *başlangıç olasılıkları* tanımlıdır:

$$\pi_i \equiv P(q_1 = S_i) \tag{15.4}$$

ve bunların toplamının 1 olması gerekir:

$$\sum_{i=1}^{N} \pi_i = 1 \tag{15.5}$$



Şekil 15.1: Üç durumlu bir Markov model örneği. π_i dizgenin S_i durumundan başlama olasılığını, a_{ij} ise dizgenin S_i durumundan S_j durumuna geçiş olasılığını gösterir. Böylesi bir dizge rastsal bir özdevinir olarak görülebilir.

$\mathbf{\Pi} = [\pi_i]$, öğlerinin toplamları 1 olan N boyutlu bir yöneydir.

Gözlenebilir bir Markov modelinde, durumlar, adı üstünde, gözlenebilirdir. Herhangi bir t anındaki q_t değerini biliriz. Dizge bir durumdan ötekine geçerken arka arkaya gelen bu durum değerleri bir gözlem dizisi oluşturur. Sürecin çıktısı, her anı gözlenebilen bir durum değerine karşılık gelen bir durumlar dizisidir.

$O = Q = \{q_1 q_2 \cdots q_T\}$ durum (gözlem) dizisinin olasılığı dizideki durumların görülme olasılıklarının çarpımıdır:

$$P(O = Q|\mathbf{A}, \mathbf{\Pi}) = P(q_1) \prod_{t=2}^{T} P(q_t|q_{t-1}) = \pi_{q_1} a_{q_1 q_2} \cdots a_{q_{T-1} q_T} \tag{15.6}$$

π_{q_1}, ilk durumun q_1 olma olasılığı ve $a_{q_1 q_2}$, q_1 durumundan q_2 durumuna geçme olasılığıdır. Tüm dizinin olasılığını bulmak için bu biçimde dizi sonuna kadar durumların önceki durumlara koşullu olasılıklarını çarparız.

Şimdi bir örnek görelim (Rabiner ve Juang 1986). Elimizde N tane kap ve her kapta farklı bir renkte toplar olsun; bir kapta kırmızı toplar, bir başkasında mavi toplar, vs. Bir cin, bir kaptan bir top çekip bize gösteriyor, sonra ötekine geçip, ondan bir top çekiyor, vs. q_t, t anında çekilen topun rengini göstersin. Diyelim ki üç kabımız (durumumuz) var:

$$S_1 : \text{Kırmızı}, S_2 = \text{Mavi}, S_3 : \text{Yeşil}$$

Başlangıç olasılıkları

$$\mathbf{\Pi} = [0.5, 0.2, 0.3]^T$$

ve geçiş olasılıkları verilmiş olsun:

$$\mathbf{A} = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$

a_{ij} burada, j renginde (j kabından) bir top çektikten sonra i renginde bir top çekmek için i kabına geçme olasılığına karşılık gelir.

$\mathbf{\Pi}$ ve $\mathbf{A}$ bilindiğinde, K tane, her biri T uzunluğunda rastsal dizin üretmek kolaydır. Verilmiş bir dizinin olasılığını nasıl hesaplayacağımızı düşünelim: İlk dört topun rengi "kırmızı, kırmızı, yeşil, yeşil," olsun. Bu, $O = \{S_1, S_1, S_3, S_3\}$, gözlem dizisine karşılık gelir. Olasılığı da birer birer gözlemlerin koşullu olasılıklarının çarpımına eşittir:

$$\begin{aligned} P(O|\mathbf{A}, \mathbf{\Pi}) &= P(S_1) \cdot P(S_1|S_1) \cdot P(S_3|S_1) \cdot P(S_3|S_3) \\ &= \pi_1 \cdot a_{11} \cdot a_{13} \cdot a_{33} \\ &= 0.5 \cdot 0.4 \cdot 0.3 \cdot 0.8 = 0.048 \end{aligned} \tag{15.7}$$

Şimdi de, K tane, her biri T uzunluğunda örnek dizi verildiğinde parametrelerin, yani, $\mathbf{\Pi}$ ve $\mathbf{A}$ değerlerinin, nasıl hesaplanabileceğini (öğrenilebileceğini) konuşalım. Başlangıç olasılıkları, K tane diziden kaç tanesinin o durumla başladığıyla kestirilir:

$$\hat{\pi}_i = \frac{\#\{S_i \text{ ile başlayan diziler}\}}{\#\{\text{tüm diziler}\}} = \frac{\sum_k 1(q_1^k = S_i)}{K} \tag{15.8}$$

Burada $1(b)$, b doğru olduğunda 1, değilse 0 değerini alan $1(\cdot)$ işlevidir.

a_{ij} geçiş olasılığı için kestirimimiz S_i durumundan tüm geçişler içinde S_j durumuna geçişlerin oranıdır:

$$\hat{a}_{ij} = \frac{\#\{S_i\text{'den } S_j\text{'ye geçişler}\}}{\#\{S_i\text{'den tüm geçişler}\}} = \frac{\sum_k \sum_{t=1}^{T-1} 1(q_t^k = S_i \text{ VE } q_{t+1}^k = S_j)}{\sum_k \sum_{t=1}^{T-1} 1(q_t^k = S_i)} \tag{15.9}$$

Örneğin $\hat{a}_{12}$, bütün dizi ve anlar içinde, kırmızı top seçilen durumların kaç tanesinde ardından mavi topun seçildiği durumların oranıdır.

15.3 Saklı Markov Modelleri

Saklı Markov modelinde (SMM) durumlar gözlenemez ama her duruma belirli bir olasılıkla bağlı bir gözlem oluşturulur. Gözlemlerimizin kesikli olduğunu ve belli bir simge kümesinden geldiğini varsayıyoruz:

$$b_j(m) \equiv P(O_t = v_m | q_t = S_j) \tag{15.10}$$

$b_j(m)$ ile S_j durumunda $v_m, m = 1, \ldots, M$ simgesinin *gözlem olasılığını* gösteriyoruz. Burada yine sıraya (t değerine) bağlı olmayan bir model varsayıyoruz. Bu

biçimde gözlenmiş değerler O gözlem dizisini oluşturur. Durum dizisi Q gözlenemez, modeli "saklı" yapan da budur ve gözlem dizisinden çıkarılması gerekir. Dikkat edilmesi gereken nokta şudur: Aynı gözlem dizisi birden çok durum dizisi tarafından türetilmiş olabilir, ama bu farklı durum dizilerinin o gözlem dizisini türetme olasılıkları aynı değildir. Bu şuna benzer: Elimizdeki bağımsız ve özdeşçe dağılmış örneklem, birçok farklı normal dağılımdan türetilmiş olabilir; bizim aradığımız bunlar içinde bu örneklemin olabilirliğini en yüksek yapan parametrelere sahip olanıdır.

Saklı Markov modelinde, biri bir durumdan ötekine geçişte, öteki de bir durumda gözlem oluşturmakta olmak üzere iki tane rastsallık kaynağı vardır.

Örneğimize geri dönelim: Eğer kapların içinde farklı renklerde toplar olabilirse, ve topun hangi kaptan çekildiğini değil, yalnızca rengini görebiliyorsak bu, saklı Markov modeliyle betimlenebilir bir uygulama olur Bu durumda, $b_j(m)$ değeri j kabından m renginde bir top çekme olasılığına karşılık gelir. Cinimiz yine bir kaptan başlayıp rastgele bir kaptan ötekine geçiyor ve her kapta rastgele bir top seçiyor, ama artık kaplar bir perde arkasında saklıyor ve kabı görünmeden seçtiği topu bize gösteriyor. Gösterdikten sonra da olasılıkları değiştirmemek için kaba geri koyuyor. Bu saklı durumda olası renk (simge) sayısının kap (durum) sayısından farklı olabileceğini görebiliriz. Örneğin üç kap ve dört renkle bir gözlem dizisi oluşabilir:

$$O = \{\text{kırmızı, kırmızı, yeşil, mavi, sarı}\} \text{ ve } Q = \{S_1, S_2, S_1, S_3, S_1\}$$

Gözlenebilir modelde, gözlemi (topun rengini) bildiğimizde durumu (kabı) da biliyoruz, çünkü kap kadar renk var ve bir renkteki topların hepsi aynı kaptan gelmek zorunda. Bu biçimde baktığımızda gözlenebilir modelin saklı modelin özel bir hali olduğunu söyleyebiliriz: $M = N$ ve $b_j(m)$, $j = m$ olduğunda 1, değilse 0. Ama saklı modelde, (her kapta her renkten en az bir top olduğunu varsayarsak) bir top herhangi bir kaptan seçilmiş olabilir ve bu yüzden verilmiş bir O gözlem dizisi birden çok olası Q durum dizisi tarafından türetilmiş olabilir (bkz. şekil 15.2).

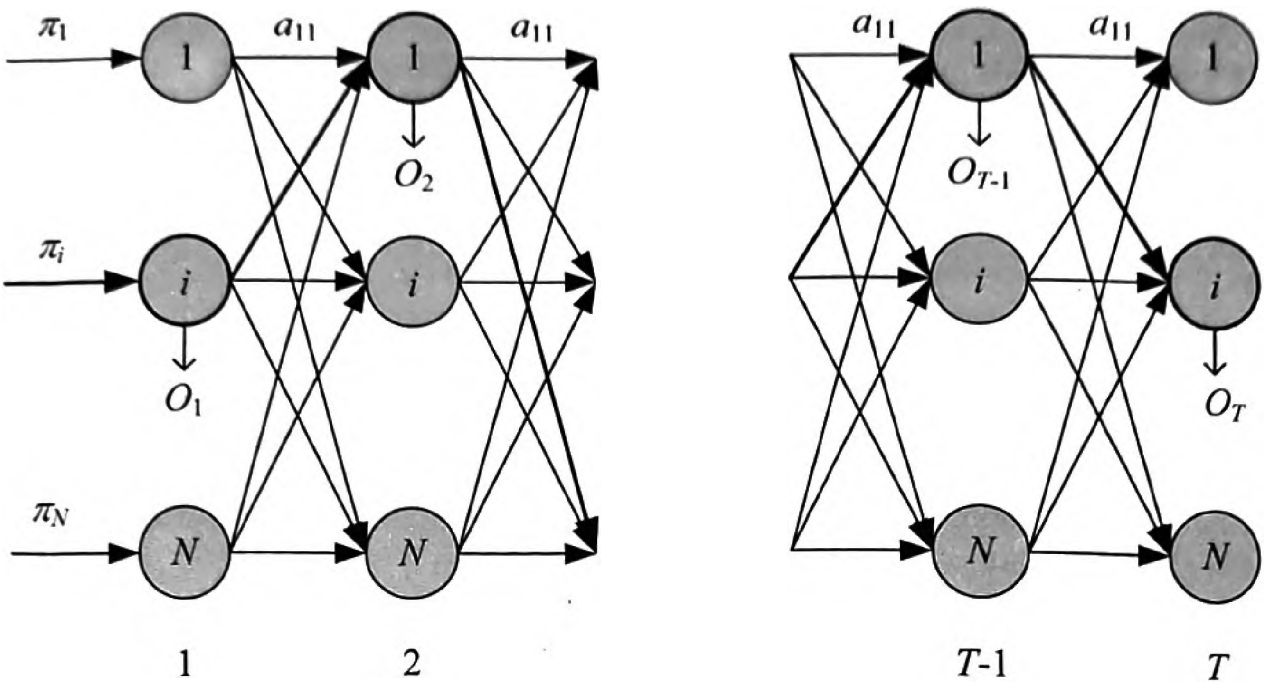
Özetlersek, saklı bir Markov modeli aşağıdaki bileşenlerle tanımlanır:

1. N: Durumu sayısı
$$S = \{S_1, S_2, \ldots, S_N\}$$

2. M: Gözlem alfabesindeki simge sayısı
$$V = \{v_1, v_2, \ldots, v_M\}$$

3. Geçiş olasılıkları:
$$\mathbf{A} = [a_{ij}] \text{ öyle ki } a_{ij} \equiv P(q_{t+1} = S_j | q_t = S_i)$$

4. Gözlem olasılıkları:
$$\mathbf{B} = [b_j(m)] \text{ öyle ki } b_j(m) \equiv P(O_t = v_m | q_t = S_j)$$



Şekil 15.2: Zaman içinde açılmış bir SMM, bir kafes biçiminde çizilebilir. Hiçbir geçiş olasılığının 0 olmadığını varsayarsak baştan sona N^T tane farklı yol vardır ve bunlardan kalın çizilmiş olanı gözlem dizisini oluşturan gerçek durum dizisine karşılık gelir.

5. Başlangıç olasılıkları:

$$\mathbf{\Pi} = [\pi_i] \text{ öyle ki } \pi_i \equiv P(q_1 = S_i)$$

N ve M değerleri öteki parametrelerce dolaylı olarak tanımlandıklardından $\lambda = (\mathbf{A}, \mathbf{B}, \mathbf{\Pi})$, saklı bir Markov modelinin parametre kümesi olarak alınır. λ verildiğinde, istediğimiz sayıda ve uzunlukta durum ve gözlem dizisi üretebiliriz, ama tabii ki burada bizi daha çok ilgilendiren öteki yön, yani, verilmiş örnek gözlem dizilerinden modelin parametrelerini kestirebilmektir.

15.4 Saklı Markov Modellerinde Üç Temel Soru

Örnek gözlem dizileri verildiğinde üç temel soruyla ilgileniriz:

1. λ ile tanımlanmış bir model için, verilmiş bir $O = \{O_1 O_2 \cdots O_T\}$ gözlem dizisinin olasılığını, yani $P(O|\lambda)$ değerini nasıl hesaplayabiliriz?

2. Bir λ modeli ve O gözlem dizisi verildiğinde, bu gözlem dizisinin türetilme olasılığını en fazla yapan durum dizisini nasıl bulabiliriz? İstediğimiz öyle bir O^* bulmak ki $P(Q|O, \lambda)$ en büyük olsun.

3. Gözlem dizilerinden oluşan bir $\mathcal{X} = \{O^k\}_k$ öğrenme kümesi verildiğinde bu gözlem kümesini türetme olasılığı en büyük olan model parametreleri nasıl bulunur? İstediğimiz öyle bir λ^* bulmak ki $P(\mathcal{X}|\lambda)$ en büyük olsun.

Şimdi bunlar için çözümleri sırayla konuşalım; göreceğimiz gibi her sorunun yanıtı bize bir sonraki soruyu yanıtlarken gerekecek, ve en sonuncu soru, bu kitaptaki tüm öteki bölümlerde olduğu gibi, veriden bir model öğrenmeye karşılık gelecek.

15.5 Değerlendirme

Bir $O = \{O_1 O_2 \cdots O_T\}$ gözlem dizisi ve $Q = \{q_1 q_2 \cdots q_T\}$ durum dizisi verildiğinde, Q verilmişken O dizisini gözleme olasılığı kolayca hesaplanabilir:

$$P(O|Q,\lambda) = \prod_{t=1}^{T} P(O_t|q_t,\lambda) = b_{q_1}(O_1) \cdot b_{q_2}(O_2) \cdots b_{q_T}(O_T) \tag{15.11}$$

Bir durum dizisinin olasılığının nasıl hesaplandığını görmüştük:

$$P(Q|\lambda) = P(q_1) \prod_{t=2}^{T} P(q_t|q_{t-1}) = \pi_{q_1} a_{q_1 q_2} \cdots a_{q_{T-1} q_T} \tag{15.12}$$

Birleşik olasılıkları bu ikisini çarparak bulunur:

$$\begin{aligned} P(O,Q|\lambda) &= P(q_1) \prod_{t=2}^{T} P(q_t|q_{t-1}) \prod_{t=1}^{T} P(O_t|q_t) \\ &= \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) \cdots a_{q_{T-1} q_T} b_{q_T}(O_T) \end{aligned} \tag{15.13}$$

Bu durumda gözlem olasılığını, birleşik dağılımı olası tüm Q yolları üzerinden tümleştirerek bulabiliriz:

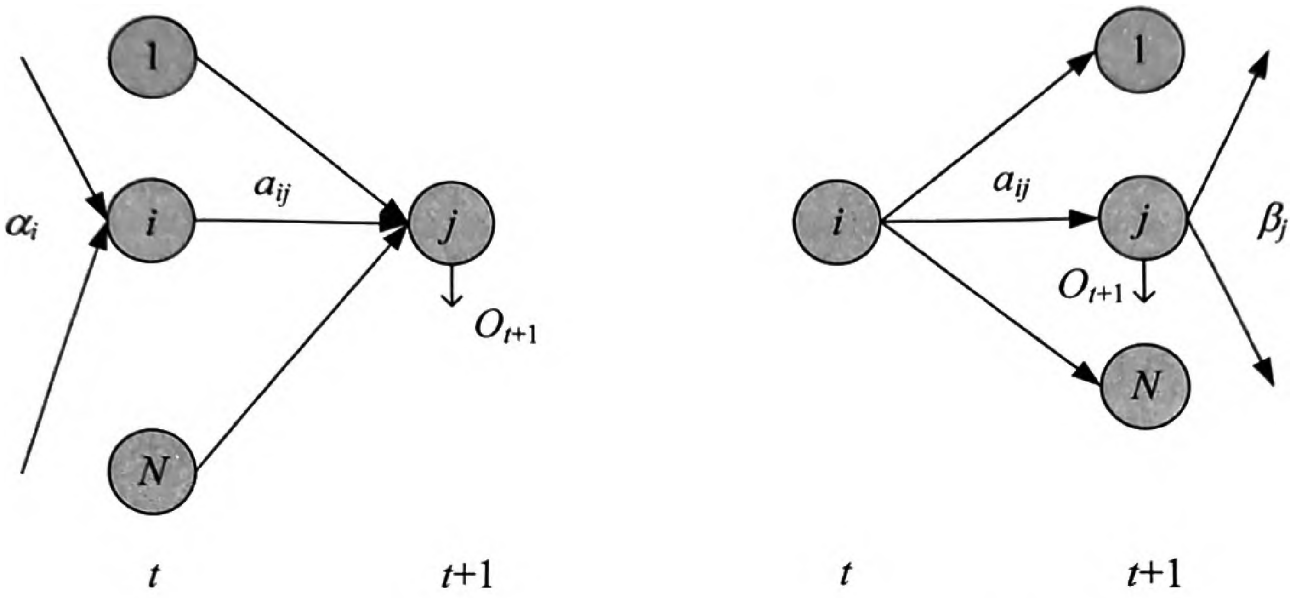
$$P(O|\lambda) = \sum_{\text{olası tüm } Q} P(O,Q|\lambda)$$

Eğer N ve T çok küçük değilse bu uygulanamaz çünkü (hiçbir olasılığın 0 olmaması durumunda) N^T tane olası Q vardır. Birazdan göreceğimiz gibi *İleri-geri yöntemi*, bu hesabı hızlı ve verimli bir biçimde hesaplayabilmemizi sağlıyor (bkz. şekil 15.3). Bu yöntem, diziyi, ilk parçası birden t anına, ikinci parçası da $t+1$ anından T anına kadar olmak üzere iki parçaya ayırma düşüncesine dayanıyor.

İlk olarak, verilmiş bir λ için, gözlem dizisinin ilk $\{O_1 \cdots O_t\}$ parçasını gözleyip t anında S_i durumuna varma olasılığına karşılık gelen $\alpha_t(i)$ *ileri değişkenini* tanımlayalım:

$$\alpha_t(i) \equiv P(O_1 \cdots O_t, q_t = S_i|\lambda) \tag{15.14}$$

Böyle bir tanımın güzel yanı, dizi boyunca ilerlerken sonrakilerin özçağrılı bir biçimde öncekilerden hesaplanabilmesidir:



(a) Ileri (b) Geri

Şekil 15.3: İleri-geri yönteminde, (a) ileri: $\alpha_t(j)$ ve (b) geri: $\beta_t(i)$ değişkenlerinin hesaplanması.

- Başlangıç:

$$\begin{aligned}\alpha_1(i) &\equiv P(O_1, q_1 = S_i|\lambda) \\ &= P(O_1|q_1 = S_i, \lambda)P(q_1 = S_i|\lambda) \\ &= \pi_i b_i(O_1)\end{aligned} \tag{15.15}$$

- Özçağrı (bkz. şekil 15.3a):

$$\begin{aligned}\alpha_{t+1}(j) &\equiv P(O_1 \cdots O_{t+1}, q_{t+1} = S_j|\lambda) \\ &= P(O_1 \cdots O_{t+1}|q_{t+1} = S_j, \lambda)P(q_{t+1} = S_j|\lambda) \\ &= P(O_1 \cdots O_t|q_{t+1} = S_j, \lambda)P(O_{t+1}|q_{t+1} = S_j, \lambda)P(q_{t+1} = S_j|\lambda) \\ &= P(O_1 \cdots O_t, q_{t+1} = S_j|\lambda)P(O_{t+1}|q_{t+1} = S_j, \lambda) \\ &= P(O_{t+1}|q_{t+1} = S_j, \lambda)\sum_i P(O_1 \cdots O_t, q_t = S_i, q_{t+1} = S_j|\lambda) \\ &= P(O_{t+1}|q_{t+1} = S_j, \lambda) \\ &\quad \sum_i P(O_1 \cdots O_t, q_{t+1} = S_j|q_t = S_i, \lambda)P(q_t = S_i|\lambda) \\ &= P(O_{t+1}|q_{t+1} = S_j, \lambda) \\ &\quad \sum_i P(O_1 \cdots O_t|q_t = S_i, \lambda)P(q_{t+1} = S_j|q_t = S_i, \lambda)P(q_t = S_i|\lambda)\end{aligned}$$

$$
\begin{aligned}
&= P(O_{t+1}|q_{t+1} = S_j, \lambda) \\
&\quad \sum_i P(O_1 \cdots O_t, q_t = S_i|\lambda) P(q_{t+1} = S_j|q_t = S_i, \lambda) \\
&= \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \qquad (15.16)
\end{aligned}
$$

$\alpha_t(i)$ değeri ilk t gözlemi açıkladıktan sonra S_i durumunda sonlanan parçayı açıklar. Bu değeri a_{ij} olasılığıyla çarparak bir sonraki S_j durumuna geçiyoruz, ve burada önceki durum için N tane seçenek olduğundan olası tüm S_i durumları üzerinden toplam almak gerekiyor. Ardından, $t+1$ anında S_j durumunda $b_j(O_{t+1})$ olasılığıyla $(t+1)$. gözlemi de gördüğümüzde artık ilk $t+1$ gözlemi açıklamış ve S_j durumuna erişmiş oluyoruz.

İleri değişkenleri dizi sonuna dek hesapladığımızda tüm dizinin görülme olasılığını hesaplayabiliriz:

$$
\begin{aligned}
P(O|\lambda) &= \sum_{i=1}^{N} P(O, q_T = S_i|\lambda) \\
&= \sum_{i=1}^{N} \alpha_T(i) \qquad (15.17)
\end{aligned}
$$

$\alpha_T(i)$ değeri bütün diziyi görüp S_i durumuna ulaşma olasılığı olduğuna göre, olası tüm son durumlar üzerinden toplam alarak tüm dizinin görülme olasılığı hesaplanabilir.

$\alpha_t(i)$ değerlerini hesaplamanın karmaşıklığı $\mathcal{O}(N^2T)$ olacağından, ilk soru olan değerlendirme sorusunun kabul edilebilir karmaşıklıkta bir çözümü olduğunu görüyoruz. Her ne kadar birinci soru için gerekmese de, hazır başlamışken *geri değişkenini* de tanımlayalım. $\beta_t(i)$, ileri değişkeni tamamlayacak biçimde, t anında S_i durumundan başlayarak dizinin sonraki bölümü olan $\{O_{t+1} \cdots O_T\}$ dizi parçasını gözlemenin olasılığı olarak tanımlanır:

$$\beta_t(i) \equiv P(O_{t+1} \cdots O_T|q_t = S_i, \lambda) \qquad (15.18)$$

Bu değerler yine özçağrılı olarak, bu kez ters yönde hesaplanır:

- Başlangıç:

$$\beta_T(i) = 1$$

- Özçağrı (bkz. şekil 15.3b):

$$
\begin{aligned}
\beta_t(i) &\equiv P(O_{t+1} \cdots O_T|q_t = S_i, \lambda) \\
&= \sum_j P(O_{t+1} \cdots O_T, q_{t+1} = S_j|q_t = S_i, \lambda)
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_j P(O_{t+1} \cdots O_T | q_{t+1} = S_j, q_t = S_i, \lambda) P(q_{t+1} = S_j | q_t = S_i, \lambda) \\
&= \sum_j P(O_{t+1} | q_{t+1} = S_j, q_t = S_i, \lambda) \\
&\quad P(O_{t+2} \cdots O_T | q_{t+1} = S_j, q_t = S_i, \lambda) P(q_{t+1} = S_j | q_t = S_i, \lambda) \\
&= \sum_j P(O_{t+1} | q_{t+1} = S_j, \lambda) \\
&\quad P(O_{t+2} \cdots O_T | q_{t+1} = S_j, \lambda) P(q_{t+1} = S_j | q_t = S_i, \lambda) \\
&= \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \qquad (15.19)
\end{aligned}
$$

S_i durumundan sonra, üzerlerinden toplam almak gereken N tane olası sonraki S_j durumu vardır ve bunlara geçiş olasılığı a_{ij} değeriyle verilmiştir. Her birindeyken $(t+1)$. gözlemini görmenin ve ardından kalan dizi parçasının olasılığı olan $\beta_{t+1}(j)$ değerini hesaba katmak gerekir.

Bu algoritmanın gerçeklenmesinde dikkat edilmesi gereken bir nokta var: Hem α_t, hem de β_t değerleri küçük olasılıkları çarparak bulunduğundan diziler kısa değilse kullandığımız değişken veri yapısının hassasiyetinin yetmeme sorunu olabilir. Bu yüzden her t anında, $\alpha_t(i)$ değerlerini

$$c_t = \frac{1}{\sum_j \alpha_t(j)}$$

ile çarparak normalleştirme yapılabilir. Aynı c_t değeriyle çarparak $\beta_t(i)$ değerlerini de normalleştiririz ($\beta_t(i)$ değerlerinin toplamı 1 olmaz). Normalleştirmeden sonra 15.17. denklemi değil normalleştirme çarpanlarını kullanırız (Rabiner 1989):

$$P(O|\lambda) = \frac{1}{\prod_t c_t} \quad \text{ya da} \quad \log P(O|\lambda) = -\sum_t \log c_t \qquad (15.20)$$

15.6 Durum Dizisinin Bulunması

Bu kısımda ikinci soruyu, yani λ modeli verildiğinde, $O = \{O_1 O_2 \cdots O_T\}$ gözlem dizisini türetme olasılığı en yüksek olan $Q = \{q_1 q_2 \cdots q_T\}$ durum dizisinin bulunmasını konuşacağız.

Denklemlerimizi kolaylaştıracak yeni $\gamma_t(i)$ değişkenini, λ ve *tüm* O verildiğinde, arada herhangi bir t anında S_i durumuna olma olasılığı olarak tanımlayalım:

$$
\begin{aligned}
\gamma_t(i) &\equiv P(q_t = S_i | O, \lambda) \qquad (15.21) \\
&= \frac{P(O|q_t = S_i, \lambda) P(q_t = S_i|\lambda)}{P(O|\lambda)}
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{P(O_1 \cdots O_t | q_t = S_i, \lambda) P(O_{t+1} \cdots O_T | q_t = S_i, \lambda) P(q_t = S_i | \lambda)}{\sum_{j=1}^{N} P(O, q_t = S_j | \lambda)} \\
&= \frac{P(O_1 \cdots O_t, q_t = S_i | \lambda) P(O_{t+1} \cdots O_T | q_t = S_i, \lambda)}{\sum_{j=1}^{N} P(O | q_t = S_j, \lambda) P(q_t = S_j | \lambda)} \\
&= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \qquad (15.22)
\end{aligned}
$$

Burada, $\alpha_t(i)$ ve $\beta_t(i)$ değişkenlerinin gözlem dizisini aralarında nasıl paylaştıklarını görüyoruz: $\alpha_t(i)$ ileri değişkeni başlangıçtan t anına kadar gelip S_i durumunda duruyor, $\beta_t(i)$ geri değişkeni de oradan alıp dizinin sonuna kadar olan parçayı açıklıyor.

Paydaki $\alpha_t(i)\beta_t(i)$, t anında S_i durumundan geçecek biçimde tüm gözlem dizisini açıklıyor. Paydada bunun t anında olası tüm durumlar için toplam değeri var. Payı paydaya bölerek normalleştiriyoruz ki (her olasılık değişkeninden beklediğimiz gibi) $\sum_i \gamma_t(i) = 1$ koşulu sağlansın.

Durum dizisinin bulmak için her t adımında olasılığı en yüksek durumu seçebiliriz:

$$q_t^* = \arg\max_i \gamma_t(i) \qquad (15.23)$$

ama bu doğru çözümü vermez, çünkü $a_{ij} = 0$ olmasına rağmen S_i ve S_j, t ve $t+1$ anlarında en yüksek olasılığa sahip durumlar olabilir. En iyi durum *dizisini* (*yolunu*) bulmak için geçiş olasılıklarını gözönüne alan ve devingen programlamaya dayanan *Viterbi algoritması* önerilmiştir.

Bir değişken daha tanımlayalım: $Q = \{q_1 q_2 \cdots q_T\}$ durum dizisi ve $O = \{O_1 \cdots O_T\}$ gözlem dizisi verildiğinde, $\delta_t(i)$, ilk t gözlemi açıklayan ve S_i durumunda biten olası tüm yollar içinde olasılığı en yüksek olanın olasılık değerini göstersin:

$$\delta_t(i) \equiv \max_{q_1 q_2 \cdots q_{t-1}} p(q_1 q_2 \cdots q_{t-1}, q_t = S_i, O_1 \cdots O_t | \lambda) \qquad (15.24)$$

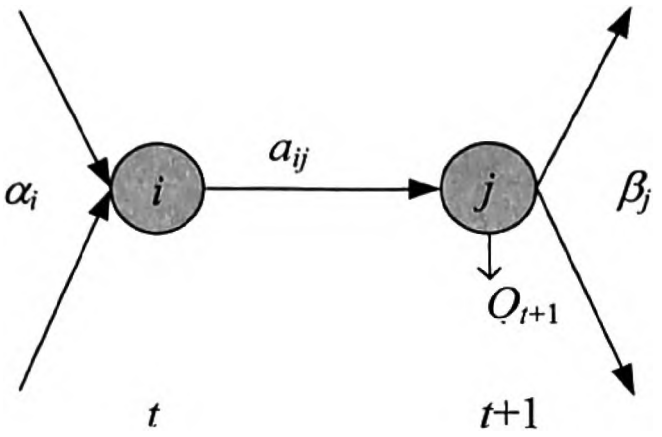
Özçağrılı biçimde $\delta_{t+1}(i)$ değerini hesaplayarak ileri doğru, dizinin sonuna kadar gider ve T anından başlayıp bu kez geriye doğru giderek her adımda en yüksek olasılığı olan durumu seçeriz. Algoritmamızı yazalım:

1. Başlangıç:

$$
\begin{aligned}
\delta_1(i) &= \pi_i b_i(O_1) \\
\psi_1(i) &= 0
\end{aligned}
$$

2. Özçağrı:

$$
\begin{aligned}
\delta_t(j) &= \max_i \delta_{t-1}(i) a_{ij} \cdot b_j(O_t) \\
\psi_t(j) &= \arg\max_i \delta_{t-1}(i) a_{ij}
\end{aligned}
$$



Şekil 15.4: $\xi_t(i,j)$ geçiş olasılıklarının hesaplanması.

3. Son:

$$\begin{aligned} p^* &= \max_i \delta_T(i) \\ q_T^* &= \arg\max_i \delta_T(i) \end{aligned}$$

4. Yol (durum dizisi) geri izleme:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \; t = T-1, T-2, \ldots, 1$$

Şekil 15.2'deki kafes yapısını düşünelim. $\psi_t(j)$ değişkeninde $\delta_t(j)$ değerinin yüksek olmasına neden olan $t-1$ anındaki durumu saklıyoruz. Böylece en sona gelip en yüksek olasılıklı durumu bulduğumuzda, $\psi_t(j)$ değişkenlerini geriye doğru izleyerek her birine nereden geldiğimizi hatırlıyoruz. Viterbi algoritmasının karmaşıklığı, ileri değişkenlerin hesaplanmasının karmaşıklığıyla aynı; zaten dikkat edersek tek fark, toplam almak yerine en büyük olanı bulmak.

15.7 Model Parametrelerinin Öğrenilmesi

Şimdi üçüncü ve en önemli soruya, saklı bir Markov modelinin veriden öğrenilmesine geldik. Yaklaşımımız, sıklıkla daha önceki bölümlerde kullandığımız gibi olabilirliği en yüksek yapan parametre değerlerinin bulunması. $\mathcal{X} = \{O^k\}_{k=1}^K$ örnek gözlem dizisi kümesi verildiğinde öyle bir λ^* bulmak istiyoruz ki $P(\mathcal{X}|\lambda)$ değeri en büyük olsun. Bir değişken daha tanımlıyoruz (bu sonuncu):

$\xi_t(i,j)$ değeriyle tüm gözlem dizisi O ve λ verildiğinde, arada herhangi bir t anında S_i durumundan ve sonraki $t+1$ anında S_j durumundan geçme olasılığını gösterelim:

$$\xi_t(i,j) \equiv P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \tag{15.25}$$

Bunu yine ileri ve geri değişkenler kullanarak hesaplayabiliriz (bkz. şekil 15.4):

$$\begin{aligned}
\xi_t(i,j) &\equiv P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \\
&= \frac{P(O|q_t = S_i, q_{t+1} = S_j, \lambda) P(q_t = S_i, q_{t+1} = S_j|\lambda)}{P(O|\lambda)} \\
&= \frac{P(O|q_t = S_i, q_{t+1} = S_j, \lambda) P(q_{t+1} = S_j | q_t = S_i, \lambda) P(q_t = S_i|\lambda)}{P(O|\lambda)} \\
&= \left(\frac{1}{P(O|\lambda)}\right) P(O_1 \cdots O_t | q_t = S_i, \lambda) P(O_{t+1} | q_{t+1} = S_j, \lambda) \\
&\quad P(O_{t+2} \cdots O_T | q_{t+1} = S_j, \lambda) a_{ij} P(q_t = S_i|\lambda) \\
&= \left(\frac{1}{P(O|\lambda)}\right) P(O_1 \cdots O_t, q_t = S_i|\lambda) P(O_{t+1} | q_{t+1} = S_j, \lambda) \\
&\quad P(O_{t+2} \cdots O_T | q_{t+1} = S_j, \lambda) a_{ij} \\
&= \frac{\alpha_t(i) b_j(O_{t+1}) \beta_{t+1}(j) a_{ij}}{\sum_k \sum_l P(q_t = S_k, q_{t+1} = S_l, O|\lambda)} \\
&= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_k \sum_l \alpha_t(k) a_{kl} b_l(O_{t+1}) \beta_{t+1}(l)}
\end{aligned} \tag{15.26}$$

$\alpha_t(i)$ değişkeni ilk t gözlemi açıkladıktan sonra t anında S_i durumuna gelir. Oradan a_{ij} olasılığıyla S_j durumuna geçer, $(t+1)$. gözlemi türetir ve $t+1$ anından sonra S_j durumundan dizinin kalanını açıklayacak biçimde $\beta_{t+1}(j)$ olasılığıyla devam ederiz. Paydaki bu değeri, t ve $t+1$ anında geçilebilecek olası tüm durum çiftlerin değerlerinin toplamına bölerek normalleştiririz.

İstersek, t anında S_i durumunda olma olasılığını, olası tüm sonraki durumlara geçiş olasılıklarının toplamı olarak da hesaplayabiliriz:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{15.27}$$

Eğer Markov modeli saklı değil de gözlenebilir olsaydı, $\gamma_t(i)$ ve $\xi_t(i,j)$ değerleri 0 ya da 1 olurdu ve bilinirdi. Saklıyken onları sonsal olasılıklar olarak kestiriyoruz ve toplamları bize *eşiksiz sayımları* veriyor. Düşününce buradaki farkın gözetimli sınıflandırmayla gözetimsiz öbekleme arasındaki farkın aynısı olduğunu görebiliriz. Sınıflandırmada sınıf etiketleri veriliyor, öbeklemedeyse bunlar beklenti adımında sonsal olasılıklar olarak (kısım 7.4) kestiriliyor ve parametre kestiriminde 0/1 etiketler yerine büyütme adımında bu sonsal olasılıklar kullanılıyor.

Şimdi konuşacağımız *Baum-Welch algoritması* da bir tür beklenti büyütme algoritmasıdır. Her yinelemenin beklenti adımında, o anki $\lambda = (\mathbf{A}, \mathbf{B}, \Pi)$ değerlerini kullanarak $\xi_t(i,j)$ ve $\gamma_t(i)$ değerlerini, büyütme adımında da $\xi_t(i,j)$ ve $\gamma_t(i)$ değerlerini kullanarak sonraki λ değerini kestiriyoruz. Bu iki adımı yakınsama olana dek yineliyoruz; bu sırada $P(O|\lambda)$ olabilirlik değerinin hiç azalmadığı gösterilmiştir.

Gösterge değişkenlerini tanmlayalım:

$$z_i^t = \begin{cases} 1 & \text{eğer } q_t = S_i \\ 0 & \text{değilse} \end{cases} \tag{15.28}$$

ve

$$z_{ij}^t = \begin{cases} 1 & \text{eğer } q_t = S_i \text{ ve } q_{t+1} = S_j \\ 0 & \text{değilse} \end{cases} \tag{15.29}$$

Gözlenebilir bir Markov modelinde bunlar 0/1 değerini alır ve bilinir; saklı bir Markov modelindeyse bunlar saklı rastsal değişkenlerdir ve beklenti adımında o anki λ değerine bağlı olarak kestirilirler:

$$\begin{aligned} E[z_i^t] &= \gamma_t(i) \\ E[z_{ij}^t] &= \xi_t(i,j) \end{aligned} \tag{15.30}$$

Büyütme adımında ise bu değişkenler için kestirimlerikullanarak yeni parametre değerlerini hesaplarız. S_i durumundan S_j durumuna beklenen geçiş sayısı $\sum_t \xi_t(i,j)$ ve S_i durumundan toplam geçiş sayısı $\sum_t \gamma_t(i)$ ile verilir. Bunların oranı S_i durumundan S_j durumuna geçiş olasılığı için kestirimimizdir:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{15.31}$$

Burada yazdığımızın denklem 15.9'la aynı olduğunu görüyoruz; tek fark, gözlenen gerçek sayımlar yerine kestirilmiş eşiksiz sayımların kullanılmasıdır.

S_j durumunda v_m simgesini görme olasılığı, S_j durumundan geçişlerimiz içinde hangi oranda v_m sigmesini de gözlediğimiz biçiminde kestirilir:

$$\hat{b}_j(m) = \frac{\sum_{t=1}^{T} \gamma_t(j) 1(O_t = v_m)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{15.32}$$

Birden çok gözlem dizisi olduğunda

$$\mathcal{X} = \{O^k\}_{k=1}^K$$

bunların birbirinden bağımsız olduğunu varsayarsak

$$P(\mathcal{X}|\lambda) = \prod_{k=1}^{K} P(O^k|\lambda)$$

parametre değerlerini kestirirken olası diziler üzerinden toplam alırız:

$$\hat{a}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P(O^k|\lambda)} \sum_{t=1}^{T_k-1} \xi_t^k(i,j)}{\sum_{k=1}^{K} \frac{1}{P(O^k|\lambda)} \sum_{t=1}^{T_k-1} \gamma_t^k(i)} \tag{15.33}$$

$$\hat{b}_j(m) = \frac{\sum_{k=1}^{K} \frac{1}{P(O^k|\lambda)} \sum_{t=1}^{T_k} \gamma_t^k(j) 1(O_t^k = v_m)}{\sum_{k=1}^{K} \frac{1}{P(O^k|\lambda)} \sum_{t=1}^{T_k} \gamma_t^k(j)}$$

$$\hat{\pi}_i = \frac{\sum_{k=1}^{K} \frac{1}{P(O^k|\lambda)} \gamma_1^k(i)}{\sum_{k=1}^{K} \frac{1}{P(O^k|\lambda)}}$$

15.8 Sürekli Gözlemler

Şimdiye kadar gözlediğimiz simgelerin kesikli olduğunu ve katlıterimli bir dağılımdan geldiğini varsaydık:

$$P(O_t|q_t = S_j, \lambda) = \prod_{m=1}^{M} b_j(m)^{r_m^t} \tag{15.34}$$

öyle ki

$$r_m^t = \begin{cases} 1 & \text{eğer } O_t = v_m \\ 0 & \text{değilse} \end{cases} \tag{15.35}$$

Eğer dizideki örnekler sürekli değerler alıyorsa onları kesikli gösterime çevirip kullanabiliriz. Bu amaçla sıklıkla bir yöney nicemleme algoritması (bölüm 7.3) kullanılır. Örneğin konuşma tanımada, bir sözcük sesi kısa ses parçacıklarına bölünüp önişlemden sonra bir yöney nicemleme yapılır ve her sözcük, saklı bir Markov modeliyle bir dizi olarak modellenir.

Yöney nicemlemede kullanılan k merkez algoritmasının Gauss karışım modelinin kesikli sürümü olduğunu hatırlayalım:

$$p(O_t|q_t = S_j, \lambda) = \sum_{l=1}^{L} P(\mathcal{G}_l) p(O_t|q_t = S_j, \mathcal{G}_l, \lambda) \tag{15.36}$$

Her bileşen bir Gauss dağılımıdır ve gözlemler süreklidir:

$$p(O_t|q_t = S_j, \mathcal{G}_l, \lambda) \sim \mathcal{N}(\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l) \tag{15.37}$$

Çıkış olasılıklarında bir Gauss karışım dağılımı kullanılıp beklenti büyütme algoritmasıyla bu karışımın parametreleri öğrenilebilir. Burada belki dikkat edilmesi gereken nokta, karmaşıklığı denetim altında tutabilmek için bir düzenlileştirme gerekmesidir (Rabiner 1989).

Basit bir örnek olarak tek boyutlu ($O_t \in \Re$) ve normal dağılmış sürekli gözlemler varsayalım:

$$p(O_t|q_t = S_j, \lambda) \sim \mathcal{N}(\mu_j, \sigma_j^2) \tag{15.38}$$

Böylece S_j durumunda gözlem değerinin ortalaması μ_j ve değişintisi σ_j^2 olan bir Gauss dağılımdan geldiğini varsayarız. Büyütme adımında bu parametreler, beklenti

adımında hesaplanan sonsal olasılıklar kullanılarak kestirilir:

$$\hat{\mu}_j = \frac{\sum_t \gamma_t(j) O_t}{\sum_t \gamma_t(j)} \qquad (15.39)$$
$$\hat{\sigma}_j^2 = \frac{\sum_t \gamma_t(j)(O_t - \hat{\mu}_j)^2}{\sum_t \gamma_t(j)}$$

15.9 Girdiye Bağlı Saklı Markov Modeli

Bazı uygulamalarda, O_t gözlem dizisi yanında elimizde bir x_t girdi dizisi de olabilir ve O_t gözlemi yalnızca o andaki duruma değil o anki girdiye de koşulludur: $P(O_t|q_t = S_j, x_t)$. Gözlemler sürekli ve tek boyutlu olsun; bu durumda denklem 15.38 yerine girdiye bağlı bir model

$$p(O_t|q_t = S_j, x_t, \lambda) \sim \mathcal{N}(g_j(x^t|\theta_j), \sigma_j^2) \qquad (15.40)$$

örneğin, en basit olarak doğrusal bir işlev kullanabiliriz:

$$g_j(x^t|w_j, w_{j0}) = w_j x^t + w_{j0} \qquad (15.41)$$

Gözlemler kesikliyse bir sınıflandırıcının çıktıları olarak modellenebilir. Öğrenme sırasında, büyütme adımında, girdiye bağlı işlevin parametreleri veriden öğrenilir.

Benzer biçimde, durumlar arası geçiş olasılıkları da bir girdiye koşullu olarak yazılabilir. Örneğin, $P(q_{t+1} = S_j|q_t = S_i, x_t)$ olasılığı, t anındaki durum ve girdiye bağlı olarak $t+1$ anındaki durumu seçen bir sınıflandırıcının normalleştirilmiş çıktısı olarak modellenebilir. *Uzmanların Markov karışımı* (Meila ve Jordan 1996) diye adlandırılan bu model, uzmanların karışımının (kısım 12.8) zamana bağlı bir sürümüdür ve geçit ağı, o anki girdi yanında bir önceki andaki geçit çıktı değerlerini de kullanır. *Girdi çıktı SMM* (Bengio ve Frasconi 1996) yine benzer bir yapıdır ve değişik anlarda hep aynı gözlem ya da geçiş olasılıkları kullanmak kısıtı yoktur; hâlâ tek bir modelimiz var ama çıktı, girdiye bağlı olarak belirli parametrelerle tanımlanmış ve bu sayede farklı geçiş ve gözlem olasılıkları verebiliyor. Girdi tek bir değer yerine çok boyutlu olup t anı çevresindeki bir penceredeki girdilerden de oluşabilir; bu yapıldığında girdi ve gözlem dizileri farklı uzunluklarda da olabilir.

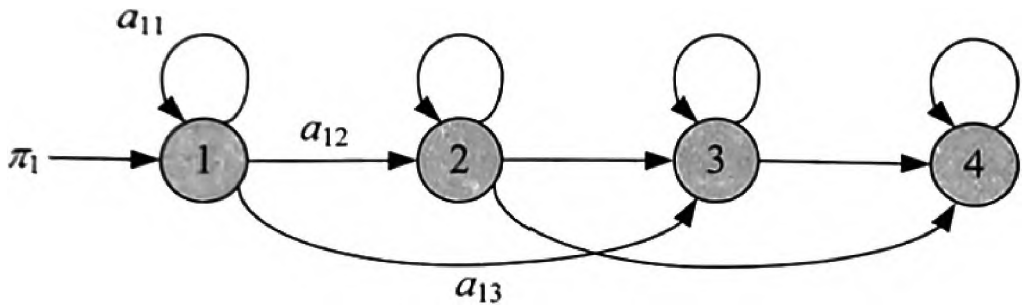
Ayrı bir girdi dizisi yoksa bile, önceki gözlemler üzerinde tanımlanacak bir işlevle girdiye bağlı bir saklı Markov modeli kullanılabilir:

$$x_t = f(O_{t-\tau}, \ldots, O_{t-1})$$

Burada τ parametresi bağlam bilgisini veren geçmiş penceresinin uzunluğudur.

15.10 Saklı Markov Modellerinde Model Seçimi

Öteki tüm öğrenen modellerde olduğu gibi saklı Markov modellerinde de karmaşıklığın eldeki verinin büyüklüğüne ve öbür özelliklerine uyması gerekiyor. Bir olasılık, SMM



Şekil 15.5: Soldan sağa SMM örneği.

yapısının kullanıcı tarafından uyarlanmasıdır. Tümüyle bağlı bir modelde her durumdan öteki tüm durumlara geçiş vardır; bu da A dizeyini $N \times N$ yapar. Bazı uygulamalarda yalnızca bazı geçişlere izin verilir ve izin verilmeyen geçişler için $a_{ij} = 0$ alınır. Olası sonraki durum sayısı $N' < N$ olduğunda, ileri/geri geçişlerin ve Viterbi algoritmasının karmaşıklığı azalır ve $\mathcal{O}(NN'T)$ olur.

Örneğin, konuşma tanımada, *soldan sağa SMM* kullanılır. Durumlar zamana göre sıralanmıştır, öyle ki zaman artarken durum dizin değişkeni ya aynı kalır ya da artar. Böylesi bir kısıt, konuşma gibi özellikleri zaman içinde değişen dizileri modelleyebilmemizi sağlar; böylece, herhangi bir duruma geldiğimizde önceki durumları yaklaşık olarak biliriz. Hiçbir zaman daha düşük dizinli bir duruma dönmeyiz: $j < i$ ise $a_{ij} = 0$. Büyük atlamalara da izin verilmez: $j > i + \tau$ ise $a_{ij} = 0$. Bir örnek, şekil 15.5'te verilmiştir; $\tau = 2$ olmak üzere karşılık gelen durum geçiş dizeyini yazalım:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$

Saklı bir Markov modelinin karmaşıklığı durum sayısı olan N değerine bağlıdır. Durumlar saklı olduğundan bu değer bilinmez, ama öğrenme öncesinde bir değer varsaymak gerekir. Burada önsel bilgi kullanılır ya da çapraz geçerlemeyle geçerleme kümesi üzerindeki olabilirliğe (ya da eğer tanımlıysa, sınıflandırma hatasına) bakarak uyarlanır.

Sınıflandırma için kullandığımızda, her sınıf için o sınıfın örnek dizileriyle eğitilen ayrı bir SMM vardır. Örneğin, konuşma tanımada sözcükler ayrı sınıflardır ve her sözcük için ayrı bir λ_i modeli eğitilir. Yeni bir O ses girdisi verildiğinde, önce tüm sözcük modelleri için $P(O|\lambda_i)$ gözlem olasılıkları, ardından Bayes kuralıyla sonsal olasılıklar hesaplanır:

$$P(\lambda_i|O) = \frac{P(O|\lambda_i)P(\lambda_i)}{\sum_j P(O|\lambda_j)P(\lambda_j)} \tag{15.42}$$

Burada, $P(\lambda_i)$, i. sözcüğün önsel olasılığıdır. Sınıflandırmada hep yapıtğımız gibi sonsal olasılığı en yüksek olan sözcük seçilir. Bu olabilirlik tabanlı yaklaşımdır,

çünkü her SMM modeli yalnızca kendi sınıf örnekleriyle eğitilir. Saklı Markov modellerinin eğitilmesi için ayırtaç tabanlı yaklaşımlar da önerilmiştir. Bir sınıf için birden çok dizi altsınıfı olabiliyorsa, örneğin, aynı sözcüğün farklı söylenişleri varsa, bunlar model içinde farklı yollar olarak tanımlanır.

Konuşma gibi sürekli değer dizileri gözlenen uygulamalarda zor olan sinyali küçük kesikli gözlemlere bölmektir. Genelde, *sesbirimcik* diye adlandırabileceğimiz küçük parçaların olduğu ve daha uzun dizilerin, örneğin, sözcüklerin, bunların arka arkaya gelmesiyle oluştuğu varsayılır. Sesbirimcikler eş anlı olarak bir yöney nicemleyiciyle kesiklileştirildikten sonra saklı bir Markov modeli bunları sıralamayı öğrenir. Ses parçacıkları ne kadar kısa alınırsa sonraki Markov modelinin işi o kadar zorlaşır. Sözcüklerin birbirinden sessizlikle ayrıldığı bağlı konuşma tanıma uygulamasında, saklı Markov modelleri farklı düzeylerde çalışabilir: Biri sesbirimcikleri birleştirerek sözcükleri, bir başkası bir dil modeliyle sözcükleri birleştirerek cümleleri tanıyabilir.

Konuşma tanıma için yapay sinir ağı ve saklı Markov modellerini birleştiren melez yapılar da önerilmiştir (Morgan ve Bourlard 1995). Böylesi bir yapıda, çok katmanlı algılayıcı (bölüm 11) zaman içinde yerel ama karmaşık temel ses parçacıklarını tanırken saklı Markov modeli zaman içindeki sıralamalarını öğrenir. Yapay sinir ağı bir önişleyici olarak çalışır ve ham gözlemleri, bir yöney nicemleyiciye kıyasla daha kolay modellenebilir bir gösterime çevirir.

Saklı bir Markov modeli, çizgesel bir model olarak görselleştirilebilir ve SMM'deki değerlendirme algoritması, bölüm 16'da göreceğimiz inanç yayılım algoritmasının özel bir sürümüdür. Saklı Markov modelini ayrı bir bölümde,tek başına tartışmamızın nedeni bu modelin başta konuşma tanıma olmak üzere birçok gerçek uygulamada uzun yıllardır başarıyla kullanılıyor olmasıdır. Çizgesel modellerden bahsederken saklı Markov modellerinin nasıl geliştirilebileceğini, örneğin farklı gözlem dizilerini nasıl işleyebileceğini, ya da gösterimi daha da basitleştirebilecek saklı değişkenleri modele nasıl ekleyebileceğimizi konuşacağız.

15.11 Kısa Kısa

Saklı Markov modelleri hem kuramsal olarak olgundur, hem de başarılı mühendislik uygulamaları vardır. SMM tabanlı ticari konuşma tanıma yazılımları gerçek yaşamda kullanılmaktadır (Rabiner ve Juang 1993; Jelinek 1997). 11.12. kısımda çok katmanlı algılayıcıların dizileri tanımayı öğrenebileceğini konuşmuştuk; SMM'nin gecikmeli ağa üstünlüğü belirli bir zaman penceresi belirlemeye gerek duymamasında ve daha hızlı öğrenebilmesindedir. SMM çok farklı uygulamalarda, örneğin, biyolojide (Baldi ve Brunak, 1998), doğal dil işlemede (Manning ve Schütze, 1999) başarıyla kullanılmıştır. Çevrimiçi yazı tanımada da kullanılmaktadır; çevrimiçi yazıda harfler basınca duyarlı bir tablet üzerine yazılır ve elimizdeki girdi, bir görüntü değil yazarken örneklenen kalem ucu (x, y) eksen değerlerinden oluşur. Bunlar zaman içinde sıralı olduklarından tanımada SMM kullanılması çok uygundur. Bengio ve ötekiler (1995), çok katmanlı bir algılayıcının harfleri, SMM'nin de bunları birleştirerek sözcükleri tanıdığı çevrimiçi bir yapıyı sunar. SMM'nin farklı uygulama ve sürümleri,

örneğin ayırtaç tabanlı SMM, (Bengio, 1999)'da verilmiştir. SMM'nin neleri yapıp neleri yapamayacağı hakkında daha yakın tarihli bir kaynak için bkz. (Bilmes, 2006)

Böylesi her modelde en önemli soru, işin ne kadarının eş anlı, ne kadarının sırayla yapılacağıdır. Konuşma tanımada sesbirimcikler, sanki sesin o küçük parçası bir anda ağızdan çıkıyormuş gibi eş anlı tanınır. Ardından bir sözcük sesbirimcikleri sırayla birleştirerek tanınır. Başka bir seçenek olarak sesbirimcikler de, eğer onlar için de değişik sıralamalar olabiliyorsa, daha küçük parçacıkların sıralaması olarak tanımlanabilir. Eş anlı yapmak işlemi hızlandırdığı için yeğlenir, ama ancak bir yere kadar; uygulama kısıtlarına göre koşut ve sırayla işlemenin dengesinin bulunması gerekir. Herkesi tek bir düğmeye basarak aramak istersek telefon üstüne milyonlarca düğme koymamız gerekir; onun yerine telefon üstüne on tane düğme konmuş ve istediğimiz kişinin "numarası" bu düğmelerin farklı sıralamaları olarak tanımlı.

Çizgesel modellerden bahsettiğimiz 16. bölümde, saklı Markov modellerinin bir tür çizgesel model olarak yorumlanabileceğini ve SMM için kullanılan çıkarım ve öğrenme işlemlerinin çizgesel modeller için önerilmiş olanlarla benzerliğini göreceğiz (Smyth, Heckerman, ve Jordan 1997). Bu bağlamda incelendiğinde SMM'nin farklı biçimlerde geliştirilebileceğini, örneğin bir gözlemin farklı durumların etkileşimiyle oluştuğu etmensel SMM'yi, ya da durumlar arasında düzeyler tanımlayan ağaç yapılı SMM'yi konuşacağız. Bu genel tanımlama, durumların sürekli olduğu doğrusal devingen dizgelerin de işlenebilmesini sağlıyor, ama böylesi modeller için her zaman kesin çözüm bulunamıyor ve yaklaşıklama ya da örnekleme kullanmak gerekebiliyor (Ghahramani, 2001).

15.12 Alıştırmalar

1. Parametreleri aşağıdaki gibi verilmiş üç durumlu, gözlenebilir bir Markov modelinden her biri 1,000 uzunluğunda, 100 tane dizi üretin:

$$\mathbf{\Pi} = [0.5, 0.2, 0.3]^T$$

$$\mathbf{A} = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$

2. Bir önceki alıştırmada ürettiğiniz veriyle $\mathbf{\Pi}$ ve $\mathbf{A}$ parametrelerini kestirin ve gerçek değerlerle karşılaştırın.

3. İkinci dereceden bir Markov modelini tanımlayın: Parametreleri nelerdir? Verilmiş bir durum dizisinin olasılığı nasıl hesaplanır? Gözlenebilir olduğunu varsayarsak parametreler nasıl öğrenilebilir?

4. Herhangi bir ikinci (ya da daha yüksek) dereceden Markov modeline denk birinci dereceden bir modelin yazılabileceğini gösterin.

5. Bazı araştırmacılar, gözlemin durumlarda değil bir durumdan ötekine geçiş sırasında ortaya çıktığını varsayan bir saklı Markov modeli tanımlar. Sizce bu, bizim bu bölümde tanımladığımız modelden daha güçlü bir model midir?

6. Bir SMM tanımlayın, ondan öğrenme ve geçerleme kümeleri üretin. Ardından aynı öğrenme verisiyle farklı durum sayıları olan farklı saklı Markov modelleri eğitin ve bunların geçerleme verisi üzerindeki olabilirlik değerlerini hesaplayın. Durum sayısı değiştiğinde geçerleme olabilirliğinin değişimine bakın.

7. Denklem 15.38'de gözlemlerimiz çok boyutlu olsaydı büyütme denklemleri nasıl değişirdi?

8. Toplar ve kaplar örneğinde seçtiğimiz topu kabına geri koymazsak ne fark eder?

9. Elimizde iki değişik alfabeden iki dizin eş anlı geliyor olsun. Örneğin, her gün farklı iki döviz için kur bilgilerini gözlüyor olalım. Böyle bir veriyi saklı bir Markov modeliyle nasıl gerçekleyebiliriz?

10. Gerektiğinde yeni durumların eklendiği artımlı bir saklı Markov modeli olabilir mi? Nasıl?

15.13 Kaynaklar

Baldi, P., S. Brunak. 1998. *Bioinformatics: The Machine Learning Approach.* Cambridge, MA: MIT Press.

Bengio, Y. 1999. "Markovian Models for Sequential Data." *Neural Computing Surveys* 2: 129–162.

Bengio, Y., P. Frasconi. 1996. "Input-Output HMMs for Sequence Processing." *IEEE Transactions on Neural Networks* 7: 1231–1249.

Bengio, Y., Y. Le Cun, C. Nohl, C. Burges. 1995. "LeRec: A NN/HMM Hybrid for On-line Handwriting Recognition." *Neural Computation* 7: 1289–1303.

Bilmes, J. A. 2006. "What HMMs Can Do." *IEICE Transactions on Information and Systems* E89-D: 869–891.

Ghahramani, Z. 2001. "An Introduction to Hidden Markov Models and Bayesian Networks." *International Journal of Pattern Recognition and Artificial Intelligence* 15: 9–42.

Jelinek, F. 1997. *Statistical Methods for Speech Recognition.* Cambridge, MA: MIT Press.

Manning, C. D., H. Schütze. 1999. *Foundations of Statistical Natural Language Processing.* Cambridge, MA: MIT Press.

Meila, M., M. I. Jordan. 1996. "Learning Fine Motion by Markov Mixtures of Experts." *Advances in Neural Information Processing Systems 8*, ed. D. S. Touretzky, M. C. Mozer, M. E. Hasselmo, 1003–1009. Cambridge, MA: MIT Press.

Morgan, N., H. Bourlard. 1995. "Continuous Speech Recognition: An Introduction to the Hybrid HMM/Connectionist Approach." *IEEE Signal Processing Magazine* 12: 25–42.

Smyth, P., D. Heckerman, M. I. Jordan. 1997. "Probabilistic Independence Networks for Hidden Markov Probability Models." *Neural Computation* 9: 227–269.

Rabiner, L. R. 1989. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." *Proceedings of the IEEE* 77: 257–286.

Rabiner, L. R., B. H. Juang. 1986. "An Introduction to Hidden Markov Models." *IEEE Acoustics, Speech, and Signal Processing Magazine* 3: 4–16.

Rabiner, L. R., B. H. Juang. 1993. *Fundamentals of Speech Recognition.* New York: Prentice Hall.

Bölüm 16

Çizgesel Modeller

Çizgesel modellerde değişkenler arasındaki etkileşimi görselleştir ve koşullu bağımsızlıkları kullanarak çok sayıda değişken içeren bir çıkarsamayı az sayıda değişkeni içeren yerel hesaplamalara ayrıştırabiliriz. Birkaç örnek üzerinde çıkarsamanın elle nasıl yapıldığını gördükten sonra, d ayrımı kavramını ve farklı çizge türleri üzerinde inanç yayılım algoritmasını tartışacağız.

16.1 Giriş

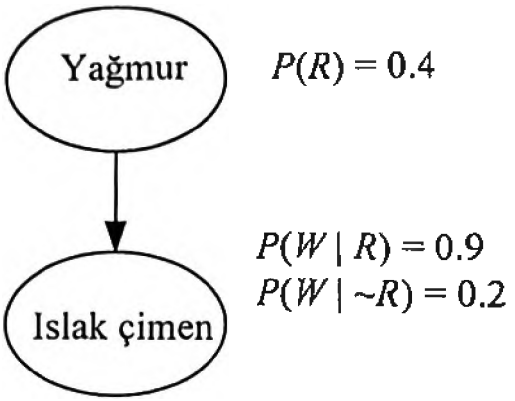
Çizgesel modeller, ya da öteki adlarıyla, *Bayesci ağlar*, *inanç ağları*, *olasılıksal ağlar*, düğümler ve aralarındaki ayrıtlardan oluşur. Her düğüm bir X rastsal değişkenini gösterir ve $P(X)$ olasılığına karşılık gelen bir değer alır. X düğümünden Y düğümüne bağlanan bir ayrıt, X değişkeninin Y değişkeni üzerinde *doğrudan bir etkisi* olduğunu gösterir ve bu etki, $P(Y|X)$ koşullu olasılığıyla gösterilir. Ağ, *yönlü çevrimsiz bir çizgedir*. Düğümler ve ayrıtlar ağın *yapısını*, koşullu olasılıklarsa yapıya bağlı *parametreleri* tanımlar.

Şekil 16.1'deki basit örnek yağmurun çimleri ıslattığını modeller. Günlerin yüzde 40'ında yağmur yağar ve yağdığında çimenlerin ıslanma olasılığı yüzde 90'dır; belki kalan yüzde 10'da çimenleri ıslak niteleyemeyeceğimiz kadar kısa yağıyordur. Bu örnekte rastsal değişkenler ikilidir, doğru ya da yanlıştır. Yağmur yağmadığında çimenlerin ıslanma olasılığı yüzde 20'dir; havanın sıcak olduğu günlerde belki bir fıskiye kullanılıyordur.

Bu üç değerle birleşik $P(R, W)$ dağılımını tümüyle tanımlarız. $P(R) = 0.4$ ise $P(\sim R) = 0.6$ olur ve benzer biçimde, $P(\sim W|R) = 0.1$ ve $P(\sim W|\sim R) = 0.8$ çıkarılabilir. Birleşik dağılımı bu iki dağılım kullanılarak yazabiliriz:

$$P(R, W) = P(R)P(W|R)$$

Çimenlerin ıslak olma olasılığını ebeveyn düğümün değerleri üzerinden tümleştirerek



Şekil 16.1: Yağmurla çimenlerin ıslanması arasındaki ilişkiyi gösteren çizgesel model.

bulabiliriz:

$$\begin{aligned} P(W) &= \sum_R P(R,W) = P(W|R)P(R) + P(W|\sim R)P(\sim R) \\ &= 0.9 \cdot 0.4 + 0.2 \cdot 0.6 = 0.48 \end{aligned}$$

Yağmur yağsaydı çimenlerin ıslak olma olasılığı 0.9, yağmadığındaysa 0.2 kadar düşük olurdu; yağıp yağmadığını bilmediğimizde (tümleşik) olasılık 0.48 olur.

Şekil 16.1, örnek bir *nedensel çizgedir* ve ıslak çimenin nedenini yağmur olarak tanımlar. Bayes kuralı, bağımlılıkların yönünü tersine çevirerek *tanı koymayı* sağlar. Örneğin, çimenlerin ıslak olduğunu bildiğimizde yağmurun yağmış olma olasılığını hesaplayabiliriz:

$$P(R|W) = \frac{P(W|R)P(R)}{P(W)} = 0.75$$

Çimenlerin ıslak olduğunu bilmek yağmur olasılığını 0.4'ten 0.75'e çıkardı; bunun nedeni, $P(W|R)$ olasılığının yüksek, $P(W|\sim R)$ olasılığınınsa düşük olmasıdır.

Düğümler ve ayrıtlarla bu düğümlerin aralarına bağımlılıklar ekleyerek daha büyük çizgeler oluşturabiliriz. X ve Y, *bağımsız olaylar* olduğunda birleşik olasılık tümleşik olasılıkların çarpımıdır:

$$p(X,Y) = P(X)P(Y) \tag{16.1}$$

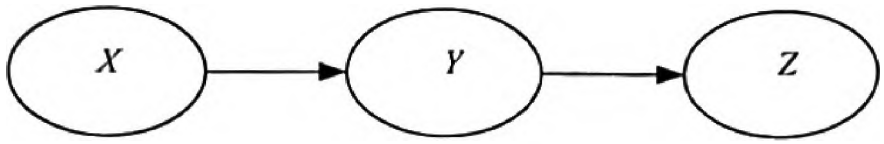
X ve Y, üçüncü bir Z olayı verildiğinde *koşullu bağımsız* olabilir:

$$P(X,Y|Z) = P(X|Z)P(Y|Z) \tag{16.2}$$

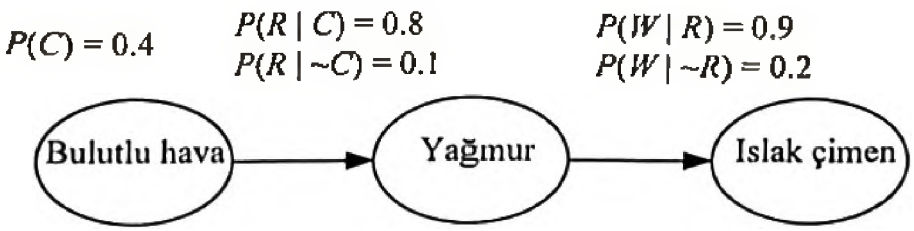
Bu durumda, Z değişkeninin değeri bilindiğinde Y değişkeninin önemi kalmaz:

$$P(X|Y,Z) = P(X|Z) \tag{16.3}$$

Çizgesel bir modelde tüm düğümler birbirine bağlı değildir; çoğu zaman bir düğüm az sayıda düğüme bağlıdır. Bazı altçizgeler koşullu bağımsızlıkları gösterir ve böylece karmaşık bir çizge üzerindeki çıkarsamayı daha küçük değişken altkümelerindeki yerel hesaplara bölüp sonuçlarını birleştirilebiliriz. Üç asal altçizge vardır ve daha büyük çizgeler bunların birleştirilmesiyle oluşturulur.



(a) Model



(b) Örnek

Şekil 16.2: Baş-kuyruk bağı. (a) Üç düğüm sırayla bağlıdır ve ara Y düğümü bilindiğinde, X ve Z bağımsızdır: $P(Z|Y,X) = P(Z|Y)$. (b) Örnek: Bulutlu hava yağmura, yağmur da çimenlerin ıslanmasına neden olur.

16.2 Koşullu Bağımsızlığın Asal Örnekleri

Örnek 1: Baş-kuyruk Bağı

Üç olay, şekil 16.2a'daki gibi sırayla bağlı olabilir. Burada, Y değişkenini verildiğinde (değeri bilindiğinde) X ve Z bağımsızdır: Y, Z değişkenine her şeyi söyler; X değişkenini bilmek, Z değişkenine ek bir bilgi vermez. Bu durumda, $P(Z|Y,X) = P(Z|Y)$ yazılabilir. Y, X değişkeninden Z değişkenine giden yolu *engeller*, ya da bir başka deyişle, ikisini *ayırır*, yani Y çıkarılırsa X ile Z arasında bir yol kalmaz. Bu durumda birleşik olasılığı önsel olasılık ve iki koşullu olasılığı kullanarak yazarız:

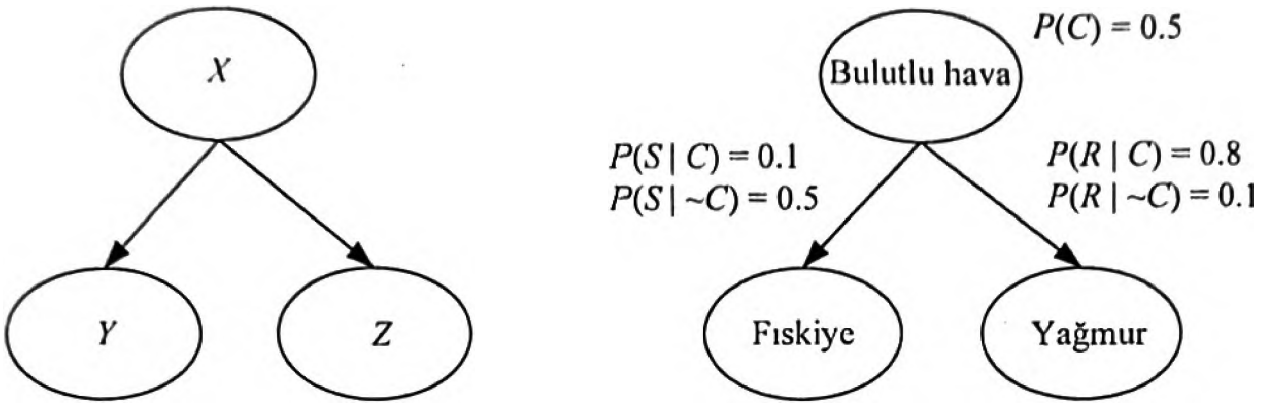
$$P(X,Y,Z) = P(X)P(Y|X)P(Z|Y) \tag{16.4}$$

Birleşik olasılığı bu biçimde yazmak koşullu bağımsızlığı gösterir:

$$P(Z|X,Y) = \frac{P(X,Y,Z)}{P(X,Y)} = \frac{P(X)P(Y|X)P(Z|Y)}{P(X)P(Y|X)} = P(Z|Y) \tag{16.5}$$

Genelde, X, Y değişkeninin, Y de Z değişkeninin nedenidir. Örneğin, şekil 16.2b'de, X bulutlu hava, Y yağmur ve Z ıslak çimenlerdir. Bilgi bu zincir boyunca ilerler. Havada bulut var mı, yok mu bilinmediğimizde yağmur ve ıslak çimen olasılıklarını hesaplayabiliriz:

$$\begin{aligned} P(R) &= P(R|C)P(C) + P(R|\sim C)P(\sim C) = 0.38 \\ P(W) &= P(W|R)P(R) + P(W|\sim R)P(\sim R) = 0.47 \end{aligned}$$



(a) Model (b) Örnek

Şekil 16.3: Kuyruk-kuyruk bağı. X, Y ve Z düğümlerinin ebeveynidir. Ebeveyn verildiğinde çocuklar bağımsız olur: $P(Y|X,Z) = P(Y|X)$. Bu örnekte, bulutlu hava hem yağmura neden olur, hem de fıskiye kullanılma olasılığını azaltır.

Sabah havada bulut gördüysek çimenlerin ıslanma olasılığı nedir? Bunu hesaplamak için "havada bulut var" kanıtını önce ara R düğümüne, ardından da W sorgu düğümüne iletmemiz gerekir:

$$P(W|C) = P(W|R)P(R|C) + P(W|\sim R)P(\sim R|C) = 0.76$$

Havanın bulutlu olma olasılığı çimenlerin ıslanma olasılığını artırıyor. Kanıtı Bayes kuralıyla ters yönde de yayabiliriz. Yolculuktan dönüp çimenleri ıslak bulduğumuzu düşünelim; o gün havanın bulutlu olmuş olma olasılığı nedir? Bayes kuralıyla yönü tersine çeviririz:

$$P(C|W) = \frac{P(W|C)P(C)}{P(W)} = 0.65$$

Çimenlerin ıslak olduğunu bilmek bulutlu hava olasılığını önsel 0.4 değerinden 0.6'ya çıkardı.

Örnek 2: Kuyruk-kuyruk Bağı

Şekil 16.3a'daki gibi X, Y ve Z düğümlerinin ortak ebeveyni olabilir. Birleşik dağılım, iki önsel dağılım ve koşullu dağılımı kullanır:

$$P(X,Y,Z) = P(X)P(Y|X)P(Z|X) \tag{16.6}$$

Y ve Z, X değişkeninden dolayı bağımlıdır; ama X bilinirse bağımsız olurlar:

$$P(Y,Z|X) = \frac{P(X,Y,Z)}{P(X)} = \frac{P(X)P(Y|X)P(Z|X)}{P(X)} = P(Y|X)P(Z|X) \tag{16.7}$$

Değeri verildiğinde X, Y ve Z arasındaki yolu engeller, ya da bir başka deyişle, ikisini ayırır.

Şekil 16.3b'de gördüğümüz örnekte, bulutlu hava hem yağmura artı, hem de fıskiyenin kullanılmasına eksi yönde neden olmaktadır. Yağmur yağdığını bildiğimizde Bayes kuralıyla bağımlılığı tersine çevirip nedeni çıkartabiliriz:

$$\begin{aligned} P(C|R) &= \frac{P(R|C)P(C)}{P(R)} = \frac{P(R|C)P(C)}{\sum_C P(R,C)} \\ &= \frac{P(R|C)P(C)}{P(R|C)P(C) + P(R|{\sim}C)P({\sim}C)} = 0.89 \end{aligned} \tag{16.8}$$

Bu hesapladığımız $P(C)$ önsel olasılığından büyüktür; yağmur yağdığını bilmek havanın bulutlu olma olasılığını artırır.

Şekil 16.3a'da, X bilinmediğinde, Y biliniyorsa, önce X değişkenini, ardından Z değişkeninin olasılığını hesaplayabiliriz. Şekil 16.3b'de fıskiyenin açık olup olmaması yağmurun yağmış olma olasılığını etkiler. Fıskiye açık diyelim:

$$\begin{aligned} P(R|S) &= \sum_C P(R,C|S) = P(R|C)P(C|S) + P(R|{\sim}C)P({\sim}C|S) \\ &= P(R|C)\frac{P(S|C)P(C)}{P(S)} + P(R|{\sim}C)\frac{P(S|{\sim}C)P({\sim}C)}{P({\sim}S)} \\ &= 0.22 \end{aligned} \tag{16.9}$$

Bu değer, tümleşik $P(R) = 0.45$ değerinden küçüktür; fıskiyenin açık olması yağmur olasılığını azaltır, çünkü yağmur ve fıskiye havanın bulut durumunun farklı değerlerine göre oluşur. Fıskiyenin kapalı olduğunu bildiğimizde, aynı yaklaşımla, yağmur olasılığını $P(R|{\sim}S) = 0.55$ olarak buluruz; yağmur olasılığı bu kez artar.

Örnek 3: Baş-baş Bağı

Şekil 16.4a'daki baş-baş düğümünde, Z düğümünün X ve Y ile gösterdiğimiz iki ebeveyni vardır. Birleşik dağılım, iki önsel dağılım ve iki ebeveyne koşullu dağılım kullanılarak yazılır:

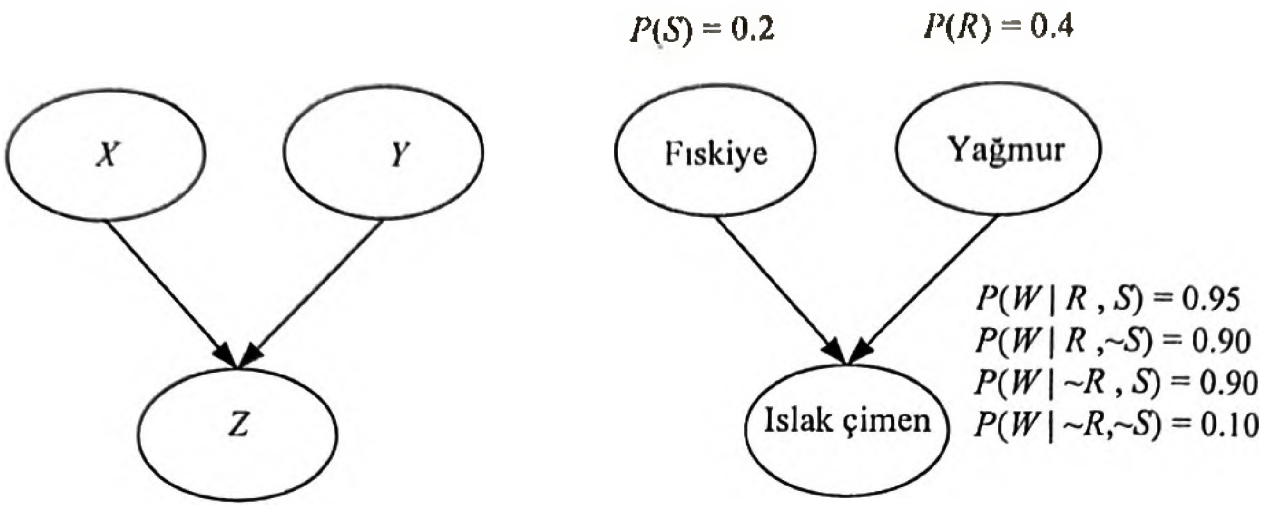
$$P(X,Y,Z) = P(X)P(Y)P(Z|X,Y) \tag{16.10}$$

X ve Y bağımsızdır: $P(X,Y) = P(X) \cdot P(Y)$ (alıştırma 2); Z bilindiğinde bağımlı olurlar. Engelleme ya da ayırmanın tanımı bu örnekte farklıdır: Değeri bilinmediğinde Z, X ve Y arasındaki yolu engeller, ya da iki düğümü ayırır; Z ya da soyundan gelen bir düğümün değeri gözlendiğinde, engelleme, ayırma, ya da bağımsızlık söz konusu olmaz.

Şekil 16.4b'deki örnekte, W düğümünün R ve S adlı iki ebeveyni vardır ve olasılığı bu ikisine, $P(W|R,S)$ koşullu dağılımıyla bağlıdır.

Başka bir şey bilmiyorsak çimenlerin ıslak olma olasılığını birleşik dağılım üzerinden tümleştirmeyle hesaplayabiliriz:

$$P(W) = \sum_{R,S} P(W,R,S)$$



(a) Model (b) Örnek

Şekil 16.4: Baş-baş bağı. Düğümün iki ebeveyni çocuk bilinmediğinde bağımsızdır. Örneğin, bir olayın iki bağımsız nedeni olabilir.

$$\begin{aligned} &= P(W|R,S)P(R,S) + P(W|\sim R,S)P(\sim R,S) \\ &\quad + P(W|R,\sim S)P(R,\sim S) + P(W|\sim R,\sim S)P(\sim R,\sim S) \\ &= P(W|R,S)P(R)P(S) + P(W|\sim R,S)P(\sim R)P(S) \\ &\quad + P(W|R,\sim S)P(R)P(\sim S) + P(W|\sim R,\sim S)P(\sim R)P(\sim S) \\ &= 0.52 \end{aligned}$$

Fiskiyenin açık olduğunu bildiğimizde çimenlerin ıslak olma olasılığını hesaplamak *nedensel* bir çıkarsamadır:

$$\begin{aligned} P(W|S) &= \sum_R P(W,R|S) \\ &= P(W|R,S)P(R|S) + P(W|\sim R,S)P(\sim R|S) \\ &= P(W|R,S)P(R) + P(W|\sim R,S)P(\sim R) \\ &= 0.92 \end{aligned}$$

$P(W|S) > P(W)$ olur, çünkü fıskiyenin açık olması ıslak çimen olasılığını artırır.

Çimenler ıslakken fıskiyenin açık olma olasılığını da hesaplayabiliriz. Bu da *tanısal* bir çıkarsamadır:

$$P(S|W) = \frac{P(W|S)P(S)}{P(W)} = 0.35$$

$P(S|W) > P(S)$ olur, çünkü çimenlerin ıslak olması fıskiyenin açık olmuş olma olasılığını artırır. Şimdi bir de yağmur yağdığını varsayalım:

$$\begin{aligned} P(S|R,W) &= \frac{P(W|R,S)P(S|R)}{P(W|R)} = \frac{P(W|R,S)P(S)}{P(W|R)} \\ &= 0.21 \end{aligned}$$

ki bu değer, $P(S|W)$ değerinden küçüktür. Yağmur yağdığını bilmek çimenlerin ıslanma nedeni olarak fıskiyeyi daha az olası yapar; buna *örtbas etmek* denir. Çimenlerin ıslak olduğunu bildiğimizde yağmur ve fıskiye bağımlı olur. $P(S|{\sim}R, W) > P(S|W)$ olduğu da benzer biçimde gösterilebilir. $P(R|W)$ ve $P(R|W, S)$ olasılıklarında da (alıştırma 3) yine aynı davranışı görebiliriz.

Böyle altçizgeleri birleştirerek daha büyük çizgeler oluşturabiliriz. Örneğin, şekil 16.5'te iki altçizgeyi birleştirdik ve örneğin, hava bulutlu olduğunda çimenlerin ıslak olma olasılığını hesaplayabiliriz:

$$\begin{aligned}
P(W|C) &= \sum_{R,S} P(W, R, S|C) \\
&= P(W, R, S|C) + P(W, {\sim}R, S|C) \\
&\quad + P(W, R, {\sim}S|C) + P(W, {\sim}R, {\sim}S|C) \\
&= P(W|R, S, C)P(R, S|C) \\
&\quad + P(W|{\sim}R, S, C)P({\sim}R, S|C) \\
&\quad + P(W|R, {\sim}S, C)P(R, {\sim}S|C) \\
&\quad + P(W|{\sim}R, {\sim}S, C)P({\sim}R, {\sim}S|C) \\
&= P(W|R, S)P(R|C)P(S|C) \\
&\quad + P(W|{\sim}R, S)P({\sim}R|C)P(S|C) \\
&\quad + P(W|R, {\sim}S)P(R|C)P({\sim}S|C) \\
&\quad + P(W|{\sim}R, {\sim}S)P({\sim}R|C)P({\sim}S|C)
\end{aligned}$$

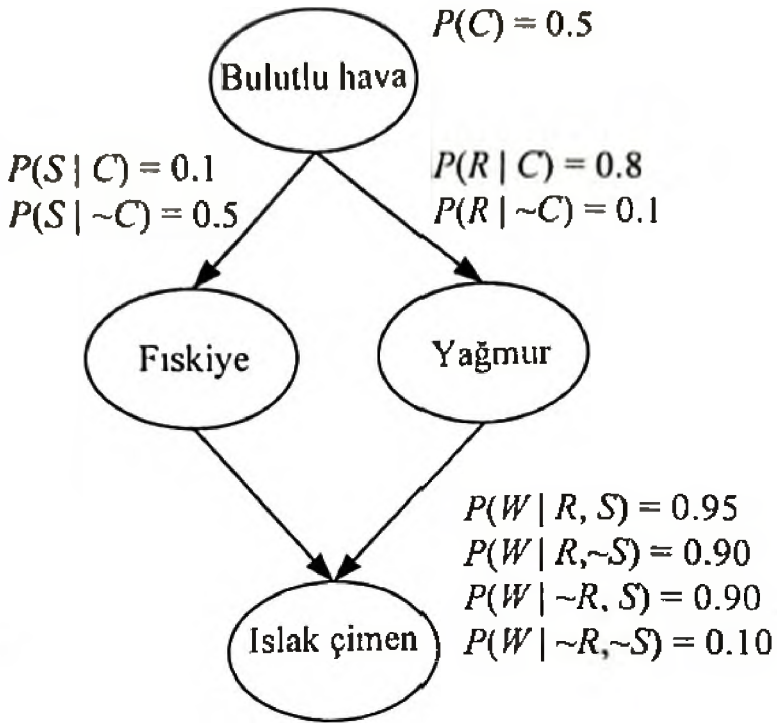
Burada, $P(W|R, S, C) = P(W|R, S)$ yazdık, çünkü R ve S verildiğinde, W ve C bağımsızdır: R ve S birlikte W ve C arasındaki yolu engeller. Benzer biçimde, $P(R, S|C) = P(R|C)P(S|C)$ yazılır; C verildiğinde R ve S bağımsızdır. Çizgesel ağların üstünlüğü işte buradadır: Bağımsızlıkları göstermemizi ve bu bağımsızlıklarla çıkarsamayı, kanıt düğümlerinden sorgu düğümlerine doğru yayılan ve her biri az sayıda değişken içeren hesaplamalara bölmemizi sağlar.

$P(C|W)$ değerini hesaplayarak tanısal bir çıkarsama yapabiliriz:

$$P(C|W) = \frac{P(W|C)P(C)}{P(W)}$$

Çizgesel gösterim görseldir ve anlaması kolaydır. Ağ, koşullu bağımsızlık bildirimlerinin bir gösterimidir ve tüm değişkenleri içeren birleşik dağılımı, her biri az sayıda değişken kullanan *yerel* altçizgelere bölmeye izin verir; bu hem çözümlemeyi, hem de hesaplamayı kolaylaştırır. Şekil 16.5, dört ikili değişkenli bir birleşik dağılımı gösterir ve bunun için normalde 15 değer $(2^4 - 1)$ saklamak gerekir, ama bu çizgede yalnızca 9 tane değer yeterlidir. Eğer her düğümün az sayıda ebeveyni varsa karmaşıklık (düğüm sayısı yönünden) üstelden doğrusala iner. Yukarda gördüğümüz gibi çıkarsama da daha kolaydır, çünkü birleşik dağılım az sayıda değişkenli koşullu dağılımlara ayrışmıştır:

$$P(C, S, R, W) = P(C)P(S|C)P(R|C)P(W|S, R) \tag{16.11}$$



Şekil 16.5: Küçük altçizgeleri birleştirerek oluşturulan büyük çizgelerde çıkarsama, altçizgelerin tanımladığı koşullu bağımsızlıklar kullanılarak daha kolay yapılır.

Genel durumda değişkenlerimizi $X_1, \ldots, X_d$ olarak gösterirsek birleşik dağılım, düğümlerin ebevenylerine koşullu olasılıklarının çarpımıdır:

$$P(X_1, \ldots, X_d) = \prod_{i=1}^{d} P(X_i | \text{ebeveynler}(X_i)) \tag{16.12}$$

Değişkenlerin herhangi bir X_i altkümesi verildiğinde, yani, değerleri kanıt olarak sabitlendiğinde, başka bir değişken altkümesinin dağılımı, birleşik dağılım üzerinden tümleştirmeyle bulunabilir. Ama bu karmaşık bir işlemdir çünkü her biri denklem 16.11'yi kullanarak daha basit hesaplansa da üstel sayıda birleşim üzerinden toplam gerektirir. Burada değişik X_i değerleri için aynı kanıtlarla, aynı ara değerlerin (koşullu olasılıkların çarpımı ve bilinmeyen değişkenler üzerinden toplamı) birçok kez hesaplandığını görebiliriz ve 16.5. bölümde inanç yayılımı algoritmasını tartışırken bu yerel ara hesapları bir kez yapıp sonuçları saklayarak işlemi basitleştirebildiğimizi göreceğiz.

Bu örnekteki değişkenler yalnız iki değer alsalar da kesikli değişkenlerin ikiden çok değer alabildikleri durum için de genelleme yapabiliriz. Her biri m farklı değer alabilen k ebeveyn olduğunda koşullu dağılım m^k değer içerir. Değişkenler sürekli de olabilir ve parametrelerle tanımlanabilir, örneğin, $p(Y|x) \sim \mathcal{N}(\mu(x|\theta), \sigma^2)$ (bkz. bölüm 16.3.3).

Çizgesel ağ kullanmanın en önde gelen üstünlüklerinden biri bazı değişkenleri özellikle girdi, bazılarını da özellikle çıktı değişkeni olarak tanımlamak gerekmeme-

sidir. Herhangi bir küme değişkenin değeri kanıt olarak verildiğinde başka herhangi bir değişken kümesinin değerleri için olasılıklar hesaplanabilir ve bu bağlamda, gözetimli ve gözetimsiz öğrenme arasındaki fark kaybolur. Bu bakış açısıyla çizgesel modeli, rastsal değişkenlerin değerleriyle ilgili sorguları yanıtlayan "olasılıksal bir veri tabanı" (Jordan 2009) olarak düşünebiliriz.

Bir uygulamada, değerlerini hiçbir zaman gözleyemeyeceğimiz *saklı değişkenler* olabilir. Bazen böylesi saklı değişkenler bağımlılık yapısını daha kolay tanımlamamızı sağlar. Örneğin, farklı mallar arasındaki ilişkiyi öğrenmek istediğimiz sepet çözümlemesinde, "mama," "çocuk bezi," ve "süt" arasında ikili ayrıtlar koymak yerine, "evde bebek var," diye bir saklı değişkeni bu üç gözlenen değişkenin nedeni olan ebeveyn düğüm olarak tanımlayabiliriz. Saklı düğümlerin değerlerini gözlenebilenleri kullanarak hesaplayıp doldururuz.

Bu noktada, X düğümünden çıkan bir ayrıtın ille de bir *nedensellik* göstermediğini aklımızda tutmamız gerekir. Ayrıt yalnızca X ile Y arasında *doğrudan bir etki* olduğunu gösterir ve Y değişkeninin alacağı değerlerin olasılığının X değişkeninin değerine bağlı olduğu anlamına gelir; iki değişken aralarında bir neden olmadan da bağlı olabilir. Veriyi üreten süreçle ilgili nedensel ilişkilerle ağı kurmak (Pearl 2000) yeğlense de böylesi nedenler her zaman bilinmeyebilir.

16.3 Örnek Çizgesel Modeller

16.3.1 Saf Bayes Sınıflandırıcı

Sınıflandırmaya karşılık gelen çizgesel modelde (bkz. şekil 16.6a), $\boldsymbol{x}$ girdiyi, C de K farklı sınıftan birini seçen katlıterim dağılımlı değişkeni gösterir. Bayes kuralı, şekil 16.1'deki yağmur ve ıslak çimen örneğindeki gibi tanı koymamızı sağlar:

$$P(C|\boldsymbol{x}) = \frac{P(C)p(\boldsymbol{x}|C)}{P(\boldsymbol{x})}$$

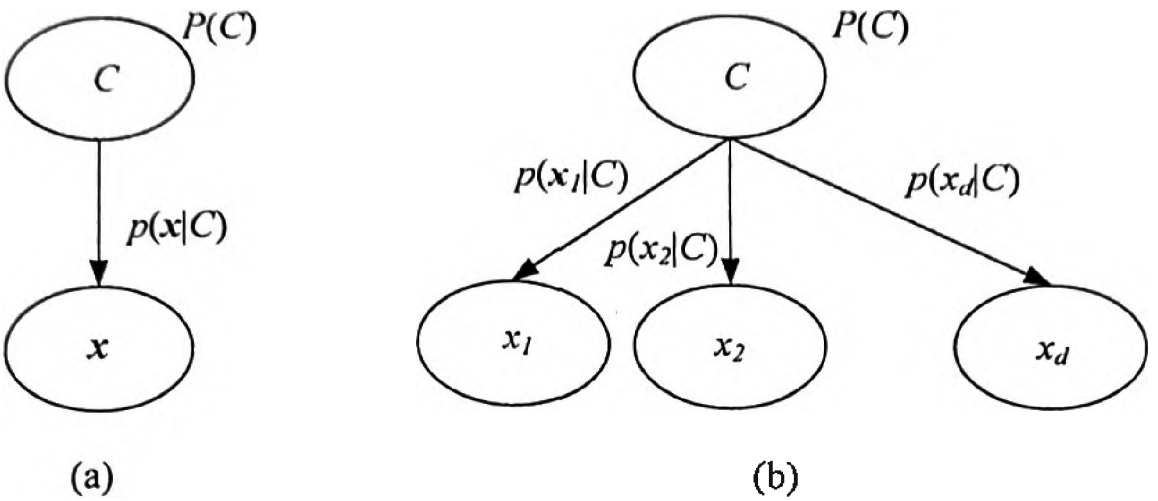
Şekil 16.6b'de çizdiğimiz *saf Bayes sınıflandırıcısı*, girdiler arasındaki bağımlılıkları, yani ilintileri, yok sayarak çok değişkenli bir dağılımı birden çok tek değişkenli dağılıma indirger:

$$p(\boldsymbol{x}|C) = \prod_{j=1}^{d} p(x_j|C)$$

Bu durumda sınıflandırmanın nasıl yapıldığını bölüm 5.5 ve 5.7'de sırasıyla sayısal ve kesikli girdiler için gördük.

Öbekleme de sınıflandırmaya benzer; tek farkı sınıflandırmada değerini bildiğimiz C katlıterim değişkenine karşılık gelen Z öbek gösterge değişkeninin gözlenmemesidir. Beklenti büyütme algoritmasının (bölüm 7.4) beklenti adımı, Bayes kuralıyla ayrıtın yönünü tersine çevirir ve öbek gösterge değerini girdiye bağlı kestirir.

Şekil 16.6a, *üretici bir modeli* gösterir ve sürecin veriyi üretmesini modeller. Sanki önce $P(C)$ önselinden örnekleyerek bir C sınıfı seçer, ardından bu C değerine göre,



Şekil 16.6: (a) Sınıflandırma için çizgesel model. (b) Bağımsız girdiler varsayan saf Bayes sınıflandırıcısı.

$p(\boldsymbol{x}|C)$ dağılımından bir $\boldsymbol{x}$ örnekleriz. Verinin, bir çizgeyle görselleştirilen nedensel bir üretici modelden örneklendiğini düşünmek hem anlamayı, hem de bazı alanlarda çıkarsamayı kolaylaştırır. Örneğin belge sınıflandırmada, bir belgenin oluşmasını, yazarın önce bir konu seçmesi, ardından o konuya göre bir sözcükler kümesi seçmesi biçiminde modelleyebiliriz. Biyoinformatikte, çizgesel yaklaşımların çeşitli uygulamalarından bir tanesi *soy ağacı* modellemektir; bu yönlü çizgenin yaprakları bilinen şu anki türlere, yaprak olmayan düğümlerse evrimsel çeşitleme olaylarıyla birden çok türe bölünmüş geçmiş atalara karşılık gelir ve koşullu olasılıklar, bir türle atası arasındaki evrimsel uzaklığa bağlıdır (Jordan 2004).

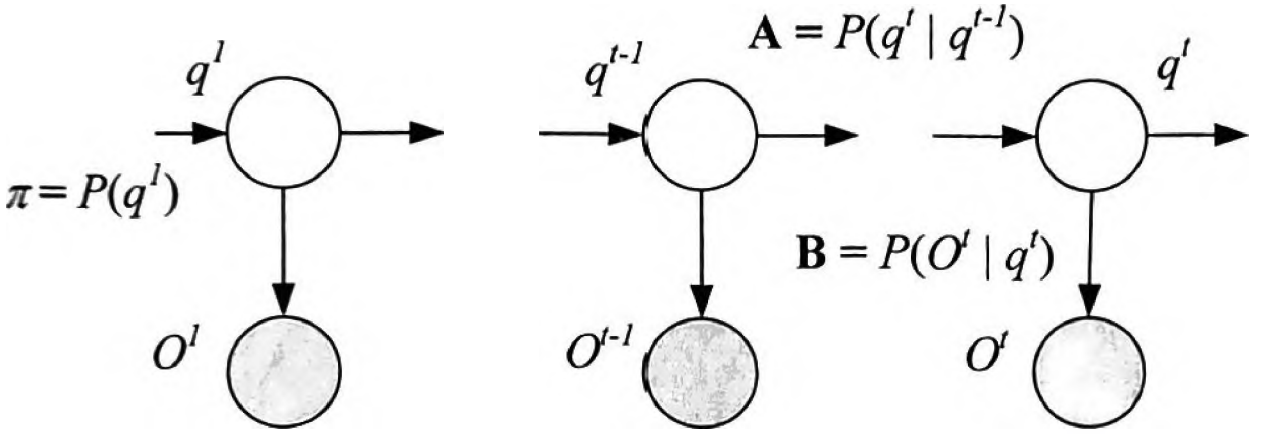
16.3.2 Saklı Markov Modeli

Daha önce 15. bölümde de gördüğümüz *saklı Markov modeli* (SMM), ardışık üç q_{t-2}, q_{t-1}, q_t, durumunun örnek 1'deki gibi zincir biçiminde bağlandığı bir çizge olarak gösterilebilir. t anındaki durumu gösteren q_t, yalnızca $t-1$ anındaki durumu gösteren q_{t-1} değerine bağlıdır ve q_{t-1} verildiğinde q_t, q_{t-2} durumundan bağımsızdır:

$$P(q_t|q_{t-1}, q_{t-2}) = P(q_t|q_{t-1})$$

Bu koşullu olasılık değerleri, **A** ile gösterdiğimiz durum geçiş olasılık dizeyi içinde verilmiştir (bkz. şekil 16.7). Her saklı değişken, **B** gözlem olasılık dizeyi tarafından belirlenen olasıklıklara göre kesikli bir gözlem oluşturur. Saklı Markov modellerinde gördüğümüz ileri geri yordamı, az sonra göreceğimiz inanç yayılımı algoritmasının bir örneğidir.

Farklı HMM türleri farklı çizgesel modellerle gösterilir. Şekil 16.8a'da gösterilen *girdi çıktı SMM*'de (bkz. bölüm 15.9) gözlenen iki ayrı girdi ve çıktı dizisi yanında bir de saklı durum dizisi vardır. Çıktı değişkeni, hem duruma, hem de girdiye bağlıdır; bunu, öğeleri sayı yerine girdinin işlevleri olan bir **B** dizeyi olarak da düşünebiliriz.



Şekil 16.7: Saklı Markov modeli, q^t değişkenlerinin saklı, gölgeli O^t değişkenlerinin gözlenebilir olduğu çizgesel bir model olarak gösterilebilir.

Bir başka bakış açısıyla bu yapıyı, geçiş çıktısı (saklı durum değeri) bir önceki andaki geçiş çıktısına bağlı bir uzman karışım modeli olarak da düşünebiliriz.

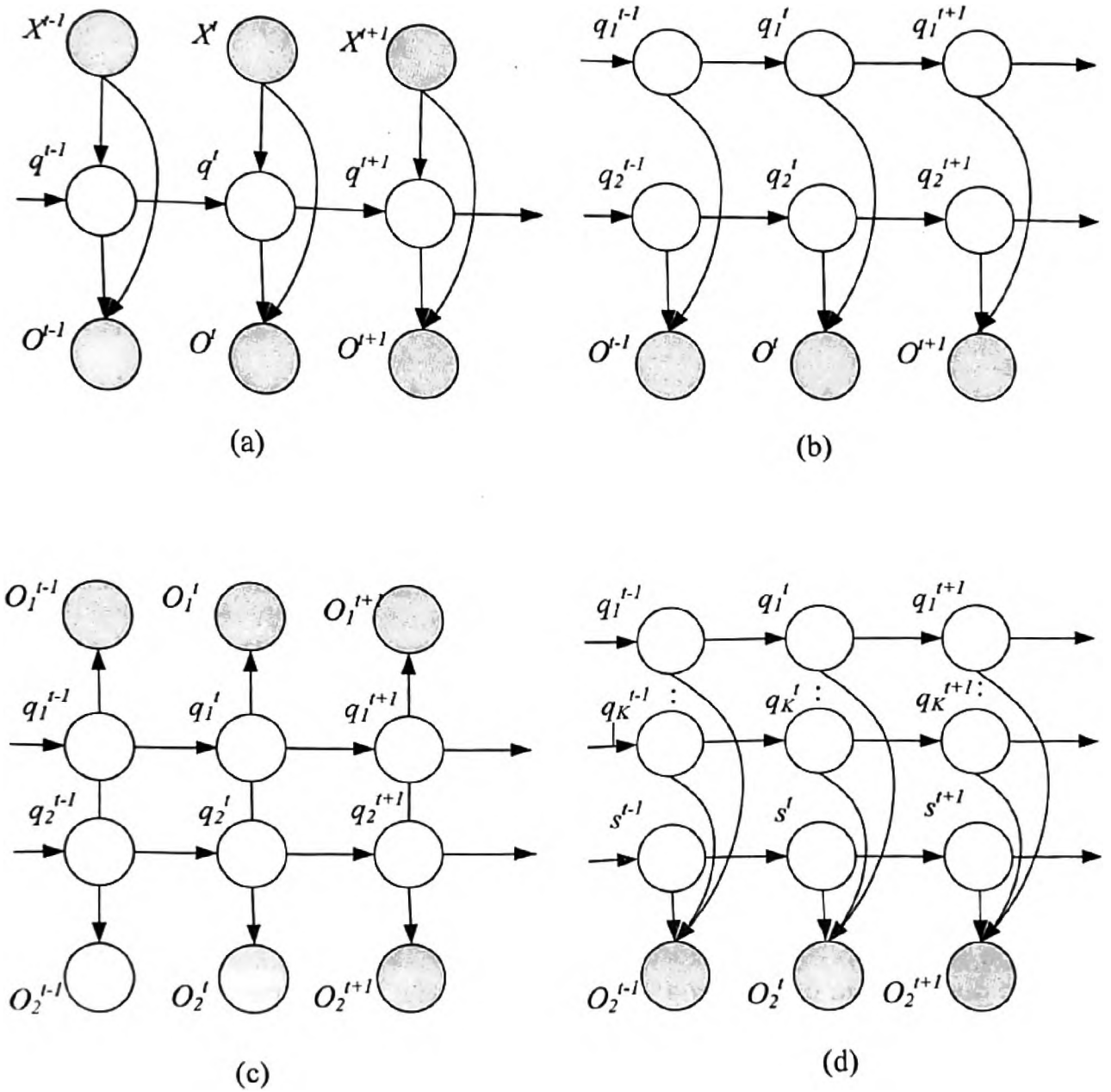
Kolay görselleştirilebilen bir başka SMM türü olan *etkensel SMM*'de,, etkileşerek tek bir gözlem dizisi oluşturan farklı saklı diziler vardır (bkz. şekil 16.8b). Buna bir örnek, ebeveyn-çocuk arasındaki *soy* ilişkisidir (Jordan 2004); *eşeyli bölünmede*, iki dizi anne ve babanın (bağımsız) kromozomlarına karşılık gelir ve her gende çocuk, eş genlerden birini babadan, birini anneden alır.

Şekil 16.8c'de gösterilen *bağlı SMM*'de, iki koşut ama bağımlı saklı dizi, iki koşut gözlem dizisi oluşturur. Örneğin konuşma tanımada, ağızdan çıkan ses bilgisiyle dudakların görsel hareketi, kendi saklı durumları olan iki ayrı gözlem dizisidir ve birbirine bağımlıdır.

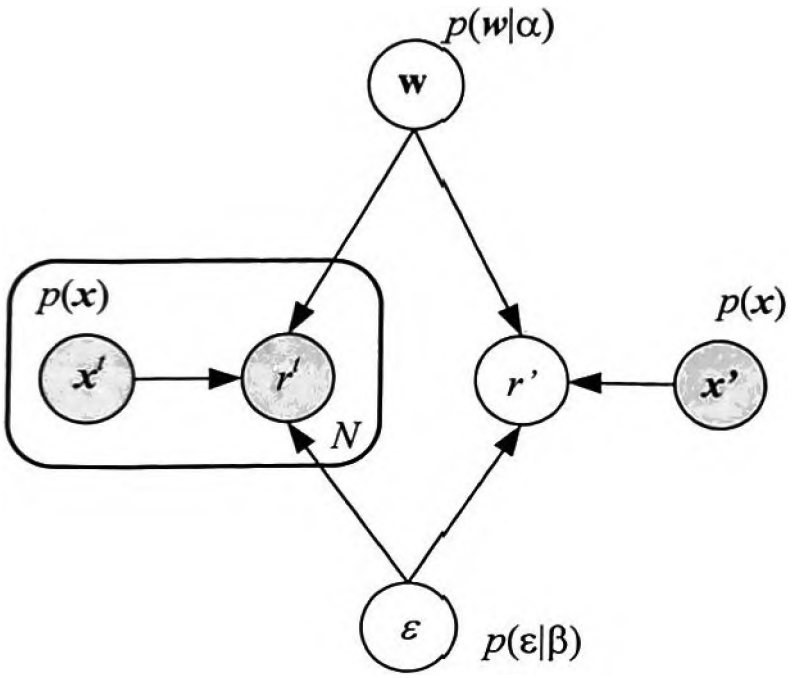
Geçişli SMM'de, K tane koşut, bağımsız, saklı durum dizisi vardır ve S durum değişkeni herhangi bir anda bunlardan birini seçer ve seçilen o anki gözlemi üretir (şekil 16.8d). Yani, zaman içinde durum dizilerinden birinden ötekine atlayarak ilerleriz.

Yaygın kullanılan SMM'de gözlemler sürekli olsa da durum kesiklidir. *Doğrusal devingen dizge*, ya da öteki adıyla *Kalman süzgecinde*, hem durum, hem de gözlem süreklidir. Temel olarak t anındaki durum, $t-1$ anındaki durumun sıfır ortalamalı Gauss gürültü eklemeli doğrusal bir işlevidir ve her durumda gözlem, durumun, yine sıfır ortalamalı Gauss gürültü eklemeli başka bir doğrusal bir işlevidir. Bu iki doğrusal eşleme ve iki gürültü kaynağının eşdeğişintileri parametreleri oluşturur. Yukarıda gördüğümüz tüm SMM sürümleri benzer biçimde sürekli durumlar için genellenebilir.

Çizgesel modeli uygun biçimde değiştirerek yapısını veriyi üreten sürece uydurabiliriz. Modelin veriye bu uydurulma süreci her zamanki gibi yanlılık ve değişinti arasında ödünleşim yapılan bir model seçim yordamıdır. Bir çekince, bu biçimde değiştirilmiş SMM'de artık kesin çıkarsama yapılamaması ve yaklaşıklama ya da örnekleme kullanmak zorunda kalınabilmesidir (Ghahramani 2001; Jordan 2009).



Şekil 16.8: Farklı SMM türleri, (gölgeli düğümlerle gösterilen) gözlenen verinin üretilmesiyle ilgili farklı varsayımlar yapar: (a) girdi çıktı SMM, (b) etkensel SMM, (c) bağlı SMM, (d) geçişli SMM.



Şekil 16.9: Doğrusal bağlanım için çizgesel model.

16.3.3 Doğrusal Bağlanım

Doğrusal bağlanım, şekil 16.9'daki gibi çizgesel bir model olarak gösterilebilir. $\boldsymbol{x}^t$ girdisi $p(\boldsymbol{x})$ önsel dağılımından çekilmiştir. Çıktıyı gösteren r^t bağımlı değişkeni, $\boldsymbol{x}$ girdisine, (α parametreli bir önsel dağılımdan çekilmiş: $p(\boldsymbol{w}) \sim \mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbf{I})$—) $\boldsymbol{w}$ ağırlıklarına ve (β parametreleri bir önsel dağılımdan çekilmiş: $p(\epsilon) \sim \mathcal{N}(0, \beta^{-1})$) ϵ gürültüsüne bağlıdır:

$$p(r^t|\boldsymbol{x}^t, \boldsymbol{w}) \sim \mathcal{N}(\boldsymbol{w}^T\boldsymbol{x}^t, \beta^{-1}) \tag{16.13}$$

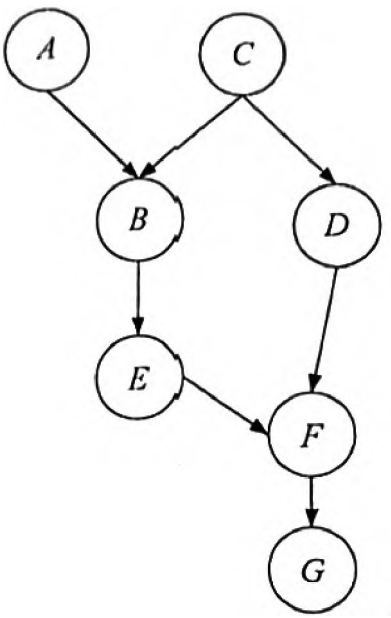
Öğrenme kümesinde böyle N bağımsız çift vardır ve bunları şekilde dikdörtgen biçiminde bir *tabaka* olarak gösteriyoruz. Yeni $\boldsymbol{x}'$ girdisi geldiğinde amaç r' düğümünü doldurmak, yani $E[r'|\boldsymbol{x}', \boldsymbol{w}]$ değerini kestirmektir.

$\boldsymbol{w}$ ağırlıklarını bilmesek de öğrenme için verilen $[\mathbf{X}, \boldsymbol{r}]$ kümesinden kestirebiliriz. Hatırlarsanız denklem 16.9'da C, R ve S değişkeninin nedeniydi ve bir sonucu ötekinden kestirirken

$$P(R|S) = \sum_C P(R, C|S) = P(R|C)P(C|S) + P(R|{\sim}C)P({\sim}C|S)$$

S değişkenini kullanıp C değişkenini doldurmuş ve bununla R değişkenini kestirmiştik. Burada da aynı yaklaşımı kullanabiliriz:

$$\begin{aligned} p(r'|\boldsymbol{x}', \boldsymbol{r}, \mathbf{X}) &= \int p(r'|\boldsymbol{x}', \boldsymbol{w}) p(\boldsymbol{w}|\mathbf{X}, \boldsymbol{r}) d\boldsymbol{w} \\ &= \int p(r'|\boldsymbol{x}', \boldsymbol{w}) \frac{p(\boldsymbol{r}|\mathbf{X}, \boldsymbol{w}) p(\boldsymbol{w})}{p(\boldsymbol{r})} d\boldsymbol{w} \end{aligned}$$



Şekil 16.10: d ayrılma örnekleri. C verildiğinde $BCDF$ yolu engellenmiştir, çünkü C kuyruk-kuyruk düğümüdür. $BEFG$ yolu, F tarafından engellenir çünkü F baş-kuyruk düğümüdür. F (ya da G) verilmediği zaman $BEFD$ engellenmiştir.

$$\propto \int p(r^t|x^t, w) \prod_t p(r^t|x^t, w)p(w)dw \tag{16.14}$$

Yukarıda ikinci eşitlik Bayes kuralından, üçüncü de öğrenme örneklerinin birbirinden bağımsız olmasından gelir.

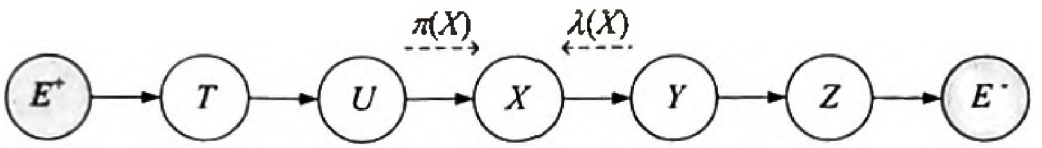
16.4 d Ayrımı

Şimdi engelleme ve ayrılma kavramlarını, *d ayrımı* adıyla genelleyeceğiz. Amacımız, üç düğüm kümesi A, B, ve C, için C verildiğinde A ve B kümelerinin bağımsız olup olmadığını anlayabilmek olacak. Jordan (2009) bunu çizge üstünde dolaşan bir *Bayes topu* olarak düşünür ve C kümesindeki düğümlere değer atandığında A kümesindeki her düğüme bir top koyar, belirli kurallara göre bu topların gezinmesine izin verir ve toplardan herhangi birinin B kümesindeki düğümlerden birine ulaşıp ulaşmadığına bakar. Eğer ulaşıyorsa bağımlı, ulaşmıyorsa bağımsız oldukları kararını verir.

C verildiğinde A ve B kümelerinin d ayrıldığını denetlerken A kümesindeki tüm düğümlerle B kümesindeki tüm düğümler arasındaki olası tüm yollara bakarız. Bir yolun *engellenmiş* olması için ya

(a) yol üzerindeki ayrıtların yönleri baş-kuyruk (örnek 1) ya da kuyruk-kuyruk (örnek 2) bağıyla buluşur ve o düğüm C içindedir, ya da

(b) yol üzerindeki ayrıtların yönleri baş-baş (örnek 3) bağıyla buluşur ve ne o düğüm, ne de onun soyundan gelenler C içinde değildir.



Şekil 16.11: Zincir boyunca çıkarsama.

Eğer olası tüm yollar engellenmişse A ve B kümeleri d ayrılmıştır, yani C verildiğinde bağımsızdır, aksi durumda bağımlı olduklarını anlarız. Şekil 16.10'da bazı örnekler gösterilmiştir.

16.5 İnanç Yayılımı

Elle basit çıkarsamaların nasıl yapıldığını gördükten sonra şimdi de bunu nasıl programlayabileceğimizi, yani, E ile değerleri bilinen *kanıt düğümlerini*, X ile de *sorgu düğümlerini* gösterirsek $P(X|E)$ biçindeki sorguları yanıtlamayı tartışalım. Pearl (1988) izinde en basit zincirlerden başlayıp giderek daha karmaşık yapılara ulaşacağız. Amacımız, çıkarsamayı genel amaçlı çizge algoritmalarıyla gerçekleyebilmek için Bayes kuralı ve tümleştirmeye karşılık gelen çizge işlemlerini bulmak olacak.

16.5.1 Zincir

Zincir, bir dizi baş-kuyruk bağından oluşur ve ebeveynsiz *kök* düğümü dışındaki tüm düğümlerin tek ebeveyni ve en sondaki *yaprak* düğümü dışındaki tüm düğümlerin tek çocuğu vardır. Eğer kanıt, X düğümünün atalarından birindeyse tanısal bir çıkarsama yapıp zincir boyunca kanıtı iletebiliriz; eğer kanıt, X düğümünün soyundan bir düğümdeyse bu kez nedensel bir çıkarsama yapıp Bayes kuralıyla kanıtı ters yönde geriye iletebiliriz. En genel durumda, her iki yönde, hem zincirin üstünde E^+, hem de altında E^- kanıtımız olabilir (bkz. şekil 16.11). Herhangi bir kanıt (hangi yönde olursa olsun) X düğümüne göre kendisinden daha ilerideki düğümleri ayırır ve o düğümler $p(X)$ değerini etkilemez.

Her düğümü, komşularından iletiler alıp yerel bir hesap yaptıktan sonra sonucunu çevresine yayan bir işlemci olarak düşünebiliriz. Her düğüm, yerel bir biçimde iki değer hesaplayıp saklar: $\lambda(X) \equiv P(E^-|X)$, X düğümünün çocuğundan aldığı ve ebeveynine ilettiği E^- bilgisi, ve $\pi(X) \equiv P(X|E^+)$, X düğümünün ebeveyninden alıp çocuğuna geçirdiği E^+ bilgisidir.

$$\begin{aligned} P(X|E) &= \frac{P(E|X)P(X)}{P(E)} = \frac{P(E^+,E^-|X)P(X)}{P(E)} \\ &= \frac{P(E^+|X)P(E^-|X)P(X)}{P(E)} \\ &= \frac{P(X|E^+)P(E^+)P(E^-|X)P(X)}{P(X)P(E)} \end{aligned}$$

$$= \alpha P(X|E^+)P(E^-|X) = \alpha\pi(X)\lambda(X) \tag{16.15}$$

Burada α, X düğümüne bağlı olmayan bir normalleştirme sabitidir. İkinci eşitlik, X verildiğinde E^+ ve E^- düğümlerinin bağımsız olmasından, üçüncüyse Bayes kuralından gelir.

Bir E düğümü, belirli bir $\tilde{e}$ değeri almış, yani somutlaşmışsa, $\lambda(\tilde{e}) \equiv 1$ ve $\lambda(e) \equiv 0, \forall e \neq \tilde{e}$ olur. Somutlaşmamış yaprak düğümü X, tüm x değerleri için $\lambda(x) \equiv 1$ değerini alır. Somutlaşmamış kök düğümü X, önsel olasılık değerlerini π olarak alır: $\pi(x) \equiv P(x), \forall x$.

Bu ilk koşullara bağlı olarak zincir boyunca kanıtın iletilmesi için özçağrılı kurallar yazabiliriz.

π iletileri için:

$$\begin{aligned} \pi(X) &\equiv P(X|E^+) = \sum_U P(X|U, E^+)P(U|E^+) \\ &= \sum_U P(X|U)P(U|E^+) = \sum_U P(X|U)\pi(U) \end{aligned} \tag{16.16}$$

Burada ikinci eşitlik, U düğümünün X ve E^+ arasındaki yolu engellemesinden dolayıdır.

λ iletileri için:

$$\begin{aligned} \lambda(X) &\equiv P(E^-|X) = \sum_Y P(E^-|X, Y)P(Y|X) \\ &= \sum_Y P(E^-|Y)P(Y|X) = \sum_U P(Y|X)\lambda(Y) \end{aligned} \tag{16.17}$$

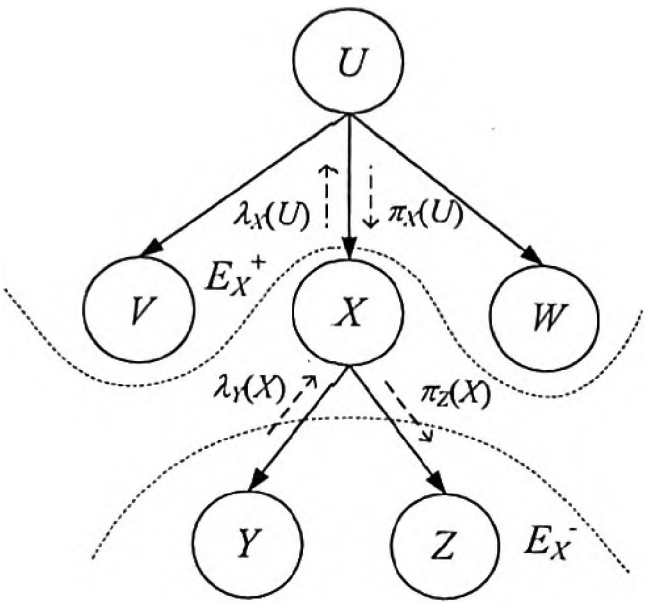
Burada da ikinci eşitlik, Y düğümünün X ve E^- arasındaki yolu engellemesinden dolayıdır.

Kanıt düğümlerine değer atandığında ağ üzerinde bir trafik başlar ve düğümler bir yakınsama olana dek değerlerini günceller. Pearl (1988) bunu, her düğümün bir işlemciyle gerçeklendiği ve eş zamanlı çalıştığı öbür işlemcilerle λ ve π iletileri değiştokuş ettiği koşut bir bilgisayar olarak düşünür.

16.5.2 Ağaç

Her düğümün yalnız bir ebeveyni ve bir çocuğu, yani bir nedeni ve bir sonucu olabildiği için zincir kısıtlı bir yapıdır. *Ağaçta*, tek bir kök düğümü dışında her düğümün tek bir ebeveyni vardır ama bir düğümün birden çok çocuğu olabilir. Zincirde gördüğümüz inanç yayılım algoritması ağaçta da geçerlidir; farkı, bir düğümün farklı çocuklarından ayrı λ iletileri almasıdır. X düğümü, çocuğu olan Y düğümünden $\lambda_Y(X)$ iletisini alır ve ona $\pi_Y(X)$ iletisi geçirir.

Yine olası kanıtı iki parçaya bölelim. E^-, X sorgu düğümünün kök olduğu altağaçtaki kanıt düğümleri göstersin; E^+ de öteki kanıtlar olsun (bkz. şekil 16.12). Bu ikincisinin ille de X düğümünün bir atası olmasının gerekmez, X düğümünün



Şekil 16.12: Bir ağacın düğümlerinin birden çok çocuğu olabilir ama (kök hariç) tek bir ebeveyni vardır.

kardeşlerinden birinin soyunda da olabilir. Önemli olan yine X düğümünün E^+ ve E^- düğümlerini ayırmasıdır, öyle ki $P(E^+, E^-|X) = P(E^+|X)P(E^-|X)$ yazabilelim ve bir normalleştirme sabiti α ile

$$P(X|E) = \alpha\pi(X)\lambda(X)$$

yazılabilsin.

$\lambda(X)$, X düğümünün kök olduğu altağaçtaki kanıtı gösterir, ve X düğümünün iki çocuğunu Y ve Z ile gösterirsek (bkz. şekil 16.12) onların λ iletileriyle hesaplanabilir:

$$\begin{aligned}\lambda(X) &\equiv P(E_X^-|X) = P(E_Y^-, E_Z^-|X) \\ &= P(E_Y^-|X)P(E_Z^-|X) = \lambda_Y(X)\lambda_Z(X)\end{aligned} \tag{16.18}$$

Genel durumda, X düğümünün $Y_j, j = 1, \ldots, m$ ile gösterdiğimiz m tane çocuğu varsa tümünün λ değerlerini çarparız:

$$\lambda(X) = \prod_{j=1}^{m} \lambda_{Y_j}(X) \tag{16.19}$$

X düğümü, çocuklarının λ iletilerini birleştirip λ kanıtını oluşturur ve ebeveynine kendi λ iletisi olarak geçirir:

$$\lambda_X(U) = \sum_X \lambda(X)P(X|U) \tag{16.20}$$

Benzer biçimde ama ters yönde, soyu dışındaki kanıt, $P(U|E^+)$, X düğümüne bir π iletisi olarak gelir:

$$\pi(X) \equiv P(X|E_X^+) = \sum_U P(X|U)P(U|E_X^+) = \sum_U P(X|U)\pi_X(U) \tag{16.21}$$

Hesaplanan bu π değeri sonra X düğümünün çocuklarına yayılır. Y düğümünün X düğümünün aldığı hem U düğümünden gelen, hem de öteki çocuğu Z düğümünden aldığıdır; bunlar birlikte E_Y^+'yı oluşturur (bkz. şekil 16.12):

$$\begin{aligned}\pi_Y(X) &\equiv P(X|E_Y^+) = P(X|E_X^+, E_Z^-) \\ &= \frac{P(E_Z^-|X, E_X^+)P(X|E_X^+)}{P(E_Z^-)} = \frac{P(E_Z^-|X)P(X|E_X^+)}{P(E_Z^-)} \\ &= \alpha\lambda_Z(X)\pi(X)\end{aligned} \tag{16.22}$$

Y'nin tek değil birden çok kardeşi olduğunda hepsinin λ değerleri üzerinden çarpım alırız:

$$\pi_{Y_j}(X) = \alpha \prod_{s \neq j} \lambda_{Y_s}(X)\pi(X) \tag{16.23}$$

16.5.3 Çoklu Ağaç

Ağaçta her düğümün tek ebeveyni yani tek nedeni vardır. *Çoklu ağaçta*, bir düğümün birden çok ebeveyni olabilir ama çizge tekli bağlıdır, yani herhangi iki düğüm arasında olası tek bir yol vardır. X düğümünü çıkardığımızda çizge iki parçaya ayrılır. Bunu, E_X kümesini E_X^+ ve E_X^- olarak ikiye ayırabilmek ve X verildiğinde bağımsız olmaları için isteriz (bkz. şekil 16.13).

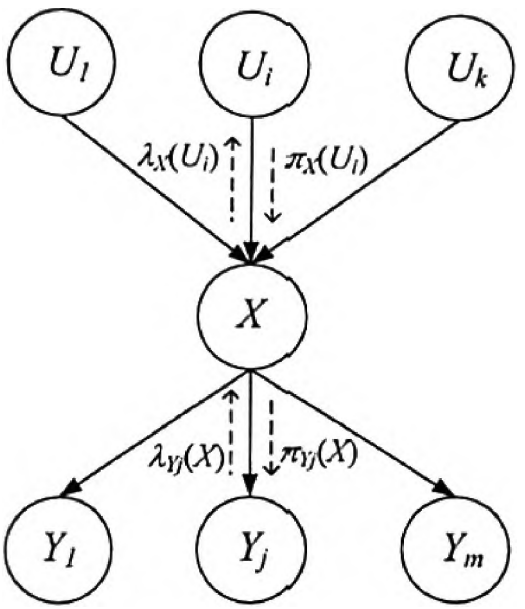
X düğümünü, ebeveynleri olan $U_i, i = 1, \ldots, k$ düğümlerinden $\pi_X(U_i)$ alıp birleştirir

$$\begin{aligned}\pi(X) &\equiv P(X|E_X^+) = P(X, E_{U_1X}^+, E_{U_2X}^+, \ldots, E_{U_kX}^+) \\ &= \sum_{U_1}\sum_{U_2}\cdots\sum_{U_k} P(X|U_1, U_2, \ldots, U_k)P(U_1|E_{U_1X}^+)\cdots P(U_k|E_{U_kX}^+) \\ &= \sum_{U_1}\sum_{U_2}\cdots\sum_{U_k} P(X|U_1, U_2, \ldots, U_k)\prod_{i=1}^{k}\pi_X(U_i)\end{aligned} \tag{16.24}$$

ve $Y_j, j = 1, \ldots, m$ ile gösterdiğimiz tüm çocuklarına iletir:

$$\pi_{Y_j}(X) = \alpha \prod_{s \neq j} \lambda_{Y_s}(X)\pi(X) \tag{16.25}$$

Birden çok ebeveyni varsa, X düğümünün ebeveynlerinden biri olan U_i düğümüne geçirdiği λ iletisi, X düğümünün yalnızca çocuklarından gelen kanıta değil, öteki



Şekil 16.13: Çoklu ağaçta bir düğümün birden çok çocuğu ve birden çok ebeveyni olabilir ama çizge tekli bağlı olmalı, yani U_i ve Y_j arasında yalnızca X düğümünden geçen bir yol olması gerekir.

$U_r, r \neq i$ ebeveynlerinden gelen π iletilerine de bağlıdır; bunlar birlikte $E^-_{U_i X}$'yi oluşturur:

$$
\begin{aligned}
\lambda_X(U_i) &\equiv P(E^-_{U_i X}|X) \\
&= \sum_X \sum_{U_{r\neq i}} P(E^-_X, E^+_{U_{r\neq i}X}, X, U_{r\neq i}|U_i) \\
&= \sum_X \sum_{U_{r\neq i}} P(E^-_X, E^+_{U_{r\neq i}X}|X, U_{r\neq i}, U_i) P(X, U_{r\neq i}|U_i) \\
&= \sum_X \sum_{U_{r\neq i}} P(E^-_X|X) P(E^+_{U_{r\neq i}X}|U_{r\neq i}) P(X|U_{r\neq i}, U_i) P(U_{r\neq i}|U_i) \\
&= \sum_X \sum_{U_{r\neq i}} P(E^-_X|X) \frac{P(U_{r\neq i}|E^+_{U_{r\neq i}X}) P(E^+_{U_{r\neq i}X})}{P(U_{r\neq i})} P(X|U_{r\neq i}, U_i) P(U_{r\neq i}|U_i) \\
&= \beta \sum_X \sum_{U_{r\neq i}} P(E^-_X|X) P(U_{r\neq i}|E^+_{U_{r\neq i}X}) P(X|U_{r\neq i}, U_i) \\
&= \beta \sum_X \sum_{U_{r\neq i}} \lambda(X) \prod_{r\neq i} \pi_X(U_r) P(X|U_1, \ldots, U_k) \\
&= \beta \sum_X \lambda(X) \sum_{U_{r\neq i}} P(X|U_1, \ldots, U_k) \prod_{r\neq i} \pi_X(U_r) \qquad (16.26)
\end{aligned}
$$

Ağaçtaki gibi toplam λ için ebeveyn, çocuklarından gelen λ iletilerini çarpar:

$$\lambda(X) = \prod_{j=1}^{m} \lambda_{Y_j}(X) \tag{16.27}$$

Birden çok ebeveyn olduğunda tüm ebeveynlere koşullu $p(X|U_1, \ldots, U_k)$ olasılığını kullanmak büyük k değerleri için masraflıdır. Karmaşıklığı üstelden doğrusala indirmek için farklı yaklaşımlar önerilmiştir. Örneğin, *gürültülü VEYA* kullanıldığında nedenlerden en az birinin gerçekleşmesi yeter ve birden çok nedenin gerçekleşmesi olasılığı azalmaz. Yalnızca U_i olduğunda X olayının olma olasılığı $1 - q_i$ ise

$$P(X|U_i, \sim U_{p\neq j}) = 1 - q_i \tag{16.28}$$

T altkümesindeki nedenler olduğunda X olasılığı

$$P(X|T) = 1 - \prod_{u_i \in T} q_i \tag{16.29}$$

olur. Islak çimenin yağmur ve fiskiye diye iki nedeni ve bunlar için $q_R = q_S = 0.1$ olsun; yani her biri kendi başına yüzde 90 olasılıkla ıslatsın. Bu durumda, $P(W|R, \sim S) = 0.9$ ve $P(W|R, S) = 0.99$ olur.

Bir başka yaklaşım, koşullu olasılığı parametrelerce tanımlanan bir model olarak tanımlamaktır. Örneğin, doğrusal bir modelde

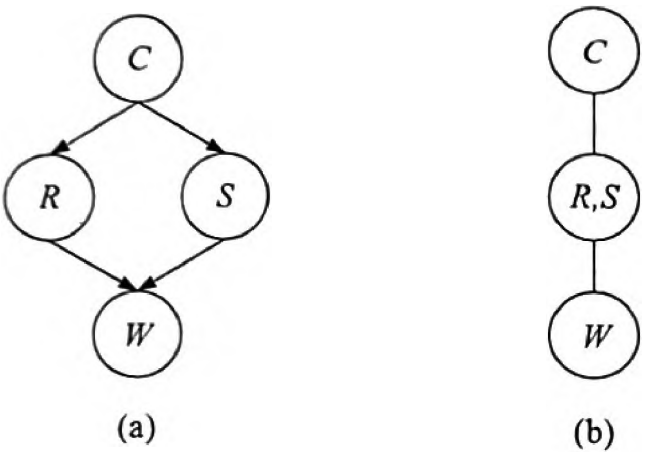
$$P(X|U_1, \ldots, U_k, w_0, w_1, \ldots, w_k) = \text{sigmoid}\left(\sum_{i=1}^{k} w_i U_i + w_0\right) \tag{16.30}$$

sigmoid işlevi çıktının 0 ile 1 arasında olmasını sağlar. Öğrenme sırasında $w_i, i = 0, \ldots, d$ parametreleri, örneğin örneklem olabilirliğini yükseltecek biçimde kestirilir.

16.5.4 Kavşak Ağacı

Eğer bir döngü varsa, yani altta yatan yönsüz çizgede bir çevrim varsa—örneğin, X düğümünün ebeveynlerinin ortak bir atası varsa—bu algoritma çalışmaz. Böylesi bir durumda kanıt birden çok yol üzerinden iletilebilir ve örneğin X düğümündeki olasılığı hesaplarken X, E kümesini nedensel (üstteki) E_X^+ ve tanısal (alttaki) E_X^- kanıt olarak ikiye bölemez; X düğümünü çıkarmak çizgeyi ikiye ayırmaz. X düğümüne koşullamak bu iki tür kanıtı bağımsız yapmaz, X düğümünü içermeyen bir yol üstünden engelsiz etkileşebilirler.

Yine de eğer çizgeyi çoklu ağaça çevirebilirsek aynı algoritmayı kullanabiliriz. Elimizdeki değişkenlerin kümelerinden oluşan *hizip düğümleri* tanımlayıp onları bir ağaç oluşturacak biçimde bağlarsak (bkz. şekil 16.14) aynı inanç yayılım algoritmasını birkaç değişiklikle çalıştırabiliriz. *kavşak ağacı* algoritmasındaki temel düşünce budur (Lauritzen ve Spiegelhalter 1988; Jensen 1996; Jordan 2009).



Şekil 16.14: (a) Çoklu bağlı bir çizge, ve (b) karşılık gelen düğümleri öbeklenmiş kavşak ağacı.

16.6 Yönsüz Çizgeler: Markov Rastsal Alanları

Şimdiye dek etkilerin tek yönlü olduğu yönlü çizgeleri tartıştık ve yönü tersine çevirmek için Bayes kuralını kullandık. Etkiler çift yönlü olduğunda, *Markov rastsal alanları* da denilen yönsüz çizgesel modelleri kullanırız. Örneğin bir resimde, komşu noktalar genelde aynı renktedir ve bu ilinti, çift yönlüdür.

Yönlü ve yönsüz çizgeler koşullu bağımsızlıkları farklı tanımlar ve bu yüzden yönlü çizgelerle gösterilip yönsüzlerle gösterilemeyecek (ya da tersi) dağılımlar vardır (Pearl 1988).

Ayrıtın başıyla kuyruğu arasında fark olmadığı için yönsüz çizgelerle uğraşmak daha kolaydır. Örneğin, C kümesi verildiğinde A ve B kümelerinin bağımsız olup olmadığını görmek daha kolaydır. Eğer C içindeki tüm düğümleri çizgeden attığımızda A içindeki bir düğümle B içindeki bir düğüm arasında bir yol kalıyorsa bağımlılardır, eğer A'daki düğümlerle B'deki düğümleri arasındaki tüm yollar C'nin düğümlerinden geçiyorsa ve C'nin düğümlerinin budanması A'nın düğümleriyle B'nin düğümlerini ayrı bileşenlerde bırakıyorsa bağımsızlardır.

Yönsüz bir çizgede, ebeveyn ve çocuk yerine *hizip* kavramını kullanırız. Hizip, ikişer ikişer aralarında bağ olan düğümler kümesi, *en büyük* hizipse en çok düğüm içeren hiziptir. Bir yön gösteren koşullu olasılık yerine yönsüz çizgelerde *gerilim işlevi* kavramını kullanırız: X_C, C hizibindeki değişkenlerse $\psi_C(X_C)$ karşılık gelen gerilim işlevini gösterir. Çizge üstündeki birleşik dağılımı çizgenin en büyük hiziplerinin gerilimlerinin çarpımı olarak yazabiliriz:

$$p(X) = \frac{1}{Z}\prod_C \psi_C(X_C) \tag{16.31}$$

Burada Z, $\sum_X p(X) = 1$ koşulunu sağlayacak biçimde tanımlanmış normalleştirme sabitidir:

$$Z = \sum_X \prod_C \psi_C(X) \tag{16.32}$$

Yönlü bir çizgenin zaten normalleştirilmiş olduğu gösterilebilir (alıştırma 5).

Yönlü çizgelerin aksine, yönsüz çizgelerdeki gerilim işlevlerinin olasılıklar olarak bir yorumu olması gerekmez ve bu bize onları tanımlamakta daha fazla özgürlük sağlar. Genelde gerilim işlevleri yerel kısıtlar tanımlar, yani bazı değer birliktelik-lerini ötekilere göre daha yeğlememizi sağlar. Örneğin bir imgede komşu noktalar arasında ikili tanımlanan bir gerilim işlevi, iki komşu noktanın farklı renkleri için aynı oldukları duruma göre daha yüksek değer alabilir (Bishop 2006). Bu durumda bazı noktaların değerleri kanıtla somutlaştığında, bilinmeyen başka noktaların (örneğin resimde örtülü bölgelerin) değerleri kestirilebilir.

Ayrıtlardaki yönleri atarak yönlü bir çizgeyi yönsüz çizebiliriz ve bu durumda her düğümün tek ebeveyni varsa ikili gerilim işlevlerini koşullu olasılıklara eşit olarak tanımlayabiliriz. Bir düğümün birden çok ebeveyni varsa, baş-baş bağından dolayı gelen "örtbas etme" özelliği ebeveynleri bağımlı yapar; bu durumda ebeveynlerin tek bir hizipte olması gerekir ve hizip gerilimi tüm ebeveynleri kapsar. Bunu sağlamak için bir düğümün tüm ebeveynlerini aralarında ikişer ikişer bağlarız ve böylece aralarında bağlı olur ve bir hizip oluştururlar. Buna ebeveynleri "evlendirmek," sürece de *ahlaklılaştırma* denir. Yeri gelmişken söyleyelim: Ebeveynlerin evlendirilmesi kavşak ağacı oluşturma aşamalarından biridir ve kavşak ağacı da yönsüzdür.

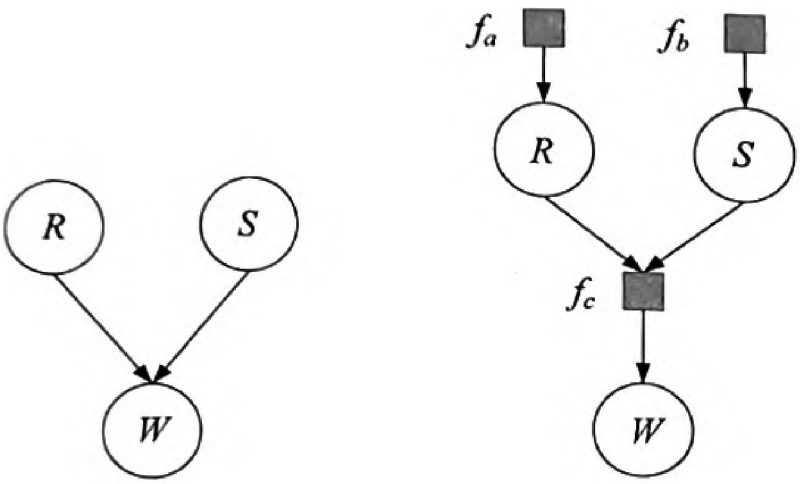
İnanç yayılım algoritması zahmetsiz bir biçimde yönsüz çizgelere uyarlanabilir; gerilim işlevleri her iki yönde de aynı olduğu için nedensel ve tanısal çıkarsama farkı yoktur. Böylece yönsüz zincir ve ağaçlarda çıkarsama yapabiliriz. Çoklu ağaçlarda bir düğümün birden çok ebeveyni olup ebeveynleri evlendirdiğimizde döngüler oluşur ve bu algoritma çalışmaz. Bir yaklaşım olarak kullanılan *etken çizgesinde*, değişken düğümleri yanında ikinci bir tür olarak *etken düğümleri* kullanılır ve birleşik dağılım etkenlerin çarpımı olarak yazılır (Kschischang, Frey, ve Loeliger 2001)

$$p(X) = \frac{1}{Z} \prod_{S} f_S(X_S) \tag{16.33}$$

X_s, S etkeni tarafından kullanılan değişkenlerin kümesini gösterir. Yönlü çizgeler etkenlerin yerel koşullu olasılık dağılımlarına, yönsüz çizgeler de etkenlerin en büyük hiziplerinin gerilim işlevlerine karşılık geldiği iki özel durumdur. Üstünlüğü, şekil 16.15'te gördüğümüz gibi ebeveynlerin evlenmesinden sonra bile ağaç yapısının korunabilmesidir.

İnanç yayılım algoritması, etken çizgeleri üzerinde çalışmak için genellenebilir ve buna *toplam çarpım algoritması* (Bishop 2006; Jordan 2009) denir; bu algoritmada da aynı biçimde, yerel hesapları bir kez yapıp sonuçları iletiler olarak çizge üzerinde yayma düşüncesi kullanılır. Buradaki fark, değişkenler ve etkenler olarak iki ayrı tür düğüm olduğundan iletileri arasında da ayrım yapmak gerekmesidir. Etken çizgesi iki parçalıdır; Bir türdeki düğüm yalnızca kendi türünden bir düğümle bağlı olabilir.

İnanç yayılımı ya da toplam çarpım algoritmasında amaç, E ile gösterilen bazı kanıt düğümlerine değer atanmışken X ile gösterilen başka bir küme düğümün olasılığının, yani, $P(X|E)$ değerinin hesaplanmasıdır. Bazı uygulamalardaysa tüm birleşik olasılık $p(X)$ değerini en büyük yapan X değerini bulmak isteyebiliriz.



Şekil 16.15: (a) Ebeveynler evlendirildiğinde döngü oluşan yönlü bir çizge, ve (b) karşılık gelen, ağaç yapısındaki etken çizgesi. Üç tane etken tanımlıdır: $f_a(R) \equiv P(R)$, $f_b(S) \equiv P(S)$, ve $f_c(R, S, W) \equiv P(W|R, S)$.

Örneğin, gerilim işlevlerinin yerel kısıtları gösterdiği yönsüz durumda bu yaklaşım, yerel kısıtları tüm çizge boyunca yayar ve küresel olarak en tutarlı çözümü bulur. Düğümleri imgedeki noktalara karşılık gelen ve ikili gerilim işlevleri ilintileri yeğleyen bir çizgede bu yaklaşım görüntüdeki gürültüyü temizler (Bishop 2006). Bunu yapan *Büyütme çarpım algoritması* (Bishop 2006; Jordan 2009), toplam çarpım algoritmasıyla aynıdır; tek fark, (tümleştirme için) toplam almak yerine (en olası değeri bulmak için) en büyüğü seçmektir. Bu fark, 15. bölümde gördüğümüz saklı Markov modellerinde ileri geri algoritmasıyla Viterbi algoritması arasındaki farkla benzeşir.

Düğümlerin imge noktaları gibi alt düzey kavramlara karşılık gelmesi gerekmez; bir görüntü tanıma uygulamasında, farklı tür köşelere ya da farklı yöndeki kenarlara karşılık gelen düğümler olabilir ve gerilim işlevleri bunların uyumluluğunu, örneğin aynı yorumun parçası olup olmadıklarını—Necker kübünü hatırlayın—denetler, öyle ki yerel kanıtların çizge boyunca yayılıp tümleşmesiyle küresel olarak tutarlı bir çözüm oluşur.

Çoklu ağaç ya da kavşak ağaçlarında çıkarsama algoritmalarının karmaşıklığı en büyük ebeveyn sayısına, ya da en büyük hizibin büyüklüğüne bağlıdır ve bu değer büyükse kesin çıkarsama yapılamayabilir. Bu durumda ya yaklaşık çözüm veren ama karmaşıklığı düşük bir algoritma, ya da bir örnekleme algoritması kullanılır (Jordan 1999; Bishop 2006; Jordan 2009).

16.7 Çizgesel Modelin Yapısının Öğrenilmesi

Öteki yöntemlerde olduğu gibi çizgesel modelin öğrenilmesi de iki aşamadan oluşur. Birincisi, yapı verildiğinde parametrelerin öğrenilmesidir, ki bu daha kolaydır (Buntine 1996). Çizgesel modellerde koşullu olasılık çizelgelerinin ya da (16.30. denklemdeki gibi) olasılıkları tanımlayan model parametrelerinin olabilirliği büyütecek biçimde, ya da önsel dağılımlar biliniyorsa Bayesçi bir yaklaşımla (bölüm 14) kestirilmesidir.

İkinci, daha zor ve daha ilginç aşamaysa çizge yapısının öğrenilmesidir (Cowell ve ötekiler 1999). Bu, temelde bir model seçimi uygulamasıdır ve çok katmanlı algılayıcıların yapısının öğrenilmesindeki (bölüm 11.9) artımlı yaklaşımlar gibi, bu da çizge uzayında arama olarak görülebilir. Örneğin, ayrıt ya da saklı düğümler ekleyen ya da budayan işlemler denenip değişiklikten (ve çıkan yapı üstünde parametreler hesaplandıktan) sonra iyileşmeye bakılarak değişiklik kabul ya da reddedilebilir. Ezberlemeyi engellemek için uygun bir biçimde düzenlileştirme yapmak gerekir ki bu, basit çizgeleri yeğleyen Bayesçi bir yaklaşıma eşittir (Neapolitan 2004). Yine de, çizge uzayı çok büyük olabileceği için, nedensel ilişkileri belirleyip az sayıda değişkenli altçizgeler oluşturabilecek bir uzmanın olması çok faydalıdır.

16.8 Etki Çizemleri

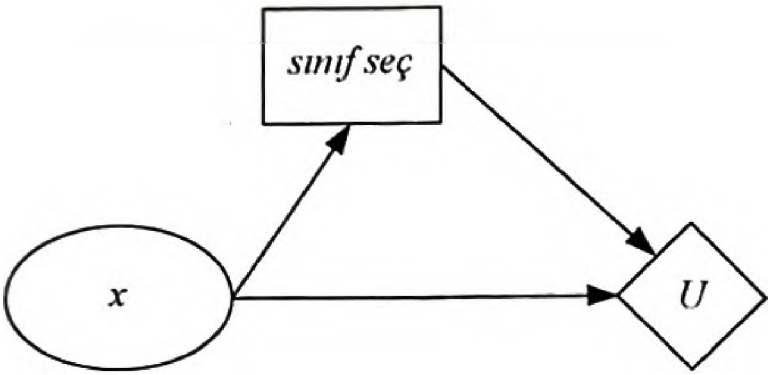
3. bölümde olasılıklardan riskli eylemlere geçtiğimiz gibi, *etki çizemleri*, çizgesel modellerin karar ve faydaları içerecek biçimde genellenmesidir. Bir etki çizemi, çizgesel modeldeki gibi rastsal değişkenleri gösteren *olasılık düğümleri* yanında karar düğümleri ve bir fayda düğümü içerir (bkz. şekil 16.16). *Karar düğümü*, eylemler arasında bir seçim olduğunu gösterir ve *fayda düğümünde* fayda hesaplanır. Kararlar, olasılık düğümlerine bağlıdır; başka olasılık düğümlerini ve faydayı etkilerler.

Etki çizemlerinde çıkarsama, çizgesel modellerdeki inanç yayılımı algoritmasının geliştirilmişidir. Bazı olasılık düğümlerine değer verildiğinde bu kanıt yayılır; her karar için fayda hesaplanır ve en faydalı karar seçilir. Bir girdinin sınıflandırılması için etki çizemi şekil 16.16'da gösterilmiştir. Girdi verildiğinde karar düğümü sınıfı seçer ve her karar için bir fayda (risk) oluşur.

16.9 Kısa Kısa

Çizgesel modellerin iki üstünlüğü vardır. Bir tanesi, değişkenler arasındaki etkilenişimin görselleştirilmesi ve böylece sürecin (örneğin, nedensel üretici bir modelle) daha iyi anlaşılabilmesidir. İkincisiyse, Bayes kuralı ve tümleştirme gibi temel olasılık hesaplarına karşılık gelen çizge işlemleriyle çıkarsamanın kolayca gerçeklenebilen genel amaçlı çizge algoritmalarına eşlenmesidir.

Değişkenler ve aralarındaki bağımlılıkların bir çizgeyle gösterilmesi, ve çok sayıda değişken içeren küresel bir işlevin her biri az sayıda değişkenli yerel işlevlerin çarpımı



Şekil 16.16: Sınıflandırmaya karşılık gelen etki çizemi. x girdisine bağlı olarak bir sınıf seçilir ve bir fayda (risk) oluşur.

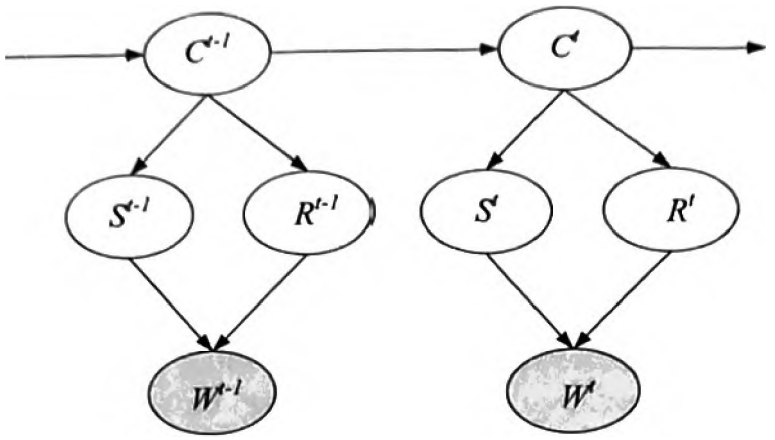
olarak tanımlanması, karar verme, işaret işleme ve gizyazma gibi farklı alanlarda kullanılmaktadır; Kschischang, Frey, ve Loeliger (2001) bunların bir değerlendirmesini verir. Değişimsel yaklaşıklama ve Markov zinciri Monte Carlo algoritmaları hakkında bilgi, (Jordan ve ötekiler 1999, MacKay 2003, Andrieu ve ötekiler 2003, Bishop 2006, Jordan 2009)'da bulunabilir.

Çizgesel modeller, değişkenleri gösteren düğümler dışında gözlenen değişkenleri etkileyen saklı değişkenleri de içeren Bayesçi yaklaşımları göstermek için özellikle uygundur. Üstparametrelere karşılık gelen düğümlerin olduğu ağaç benzeri bir yapı, örneğin, birinci düzey parametrelerinin önsellerini tanımlayan ikinci düzey parametreleri içerebilir.

Saklı Markov modeli de bir tür çizgesel modeldir ve gerçekte herhangi bir çizgesel model zaman içinde açılarak ardışık kopyaları arasında bağımlılık tanımlanabilir. Böylesi *devingen çizgesel modeller*, zamanın bir boyut olduğu uygulamalarda, örneğin konuşma tanımada, kullanılabilir. Gerçekte saklı Markov modeli, t anındaki öbek dizin değerinin yalnızca t anındaki gözleme değil, bir de $t-1$ anındaki öbek dizin değerine bağlı olduğu ardışık öbeklemeden başka bir şey değildir, ve Baum-Welch algoritması, beklenti büyütme algoritmasının zaman içindeki bu bağımlılığı göz önüne alan sürümüdür. 6.4. bölümde az sayıda etkenin gözlemi oluşturduğu etken çözümlemesini tartışmıştık; benzer biçimde, doğrusal devingen dizge, şu anki etkenlerin bir önceki andaki etkenlere bağlı olduğu bir etken çözümleme dizisi olarak görülebilir.

Gerektiğinde böyle bir devingen bağımlılık eklenebilir. Örneğin, şekil 16.5 herhangi bir günde ıslak çimenin nedenini modeller. Eğer dünkü hava durumunun bugünkü hava durumunu etkilediğini düşünüyorsak (ki öyle olmalı—ardışık günlerde hava durumu birbirine benzer), bu bağımlılığı şekil 16.17'deki devingen çizgesel modelle gösterebiliriz.

Çizgesel model, SMM'den daha genel yapılar tanımlamamızı ve örneğin konuşma tanımada daha yüksek başarımlara ulaşmamızı sağlar (Zweig 2003; Bilmes ve Bartels 2005). Çizgesel modeller görüntü tanımada, örneğin görüntü erişimi (Barnard ve ötekiler 2003) ve görüntü çözümlemesinde (Sudderth ve ötekiler 2008) kullanılmaktadır. Çizgesel modellerin biyoinformatikte kullanılması (ve ilgili yazılımlarla) ilgili bilgi,



Şekil 16.17: Ardışık günlerdeki hava durumu bağımlılığını gösteren bir çizge zinciri kullandığımız devingen bir sürüm.

(Donkers ve Tuyls 2008)'de bulunabilir.

16.10 Alıştırmalar

1. İki bağımsız girdili bir sınıflandırma uygulamasında, yani, $p(x_1, x_2|C) = p(x_1|C)p(x_2|C)$, $p(x_1|x_2, C)$ olasılığını nasıl hesaplarız ? $p(x_j|C_i) \sim \mathcal{N}(\mu_{ij}, \sigma^2_{ij})$ için genelleyin.
2. Baş-baş düğümü için denklem 16.10, $P(X, Y) = P(X) \cdot P(Y)$ eşitliğini gösterir. Neden ?
3. Şekil 16.4'te $P(R|W)$, $P(R|W, S)$, ve $P(R|W, \sim S)$ değerlerini hesaplayın.
4. Denklem 16.30'da, X değişkeni iki değerlidir. K kesikli değerden birini alıyor olsaydı nasıl değiştirilmesi gerekirdi ?
5. Birleşik dağılımın denklem 16.12'deki gibi yazıldığı yönlü çizgede $\sum_{\boldsymbol{x}} p(\boldsymbol{x}) = 1$ olduğunu gösterin.
6. Necker kübünü, farklı köşe yorumları arasında birbirini destekleyen ya da ketleyen bağların tanımlandığı bir çizgesel model olarak gösterin.
7. Şekil 16.17'deki devingen hava durumu çizgesi üstünde nasıl çıkarsama yapabiliriz ?
8. İki sınıf için doğrusal sınıflandırmayı şekil 16.9'daki doğrusal bağlanımdaki gibi bir çizgesel model olarak gösterin.
9. Çizge yapısını ararken kullanılabilecek uygun bir iyilik işlevi tanımlayın. Yapıda değişiklikler yapan işlemler neler olabilir ?
10. Bir gazeteci aynı konuda hikâye geliştikçe, ardışık günlerde bir dizi haber yazabilir. Bunu çizgesel bir modelle nasıl gösterebiliriz ?

16.11 Kaynaklar

Andrieu, C., N. de Freitas, A. Doucet, M. I. Jordan. 2003. "An Introduction to MCMC for Machine Learning." *Machine Learning* 50: 5–43.

Barnard, K., P. Duygulu, D. Forsyth, N. de Freitas, D. M. Blei, M. I. Jordan. 2003. "Matching Words and Pictures." *Journal of Machine Learning Research* 3: 1107–1135.

Bilmes, J., C. Bartels. 2005. "Graphical Model Architectures for Speech Recognition." *IEEE Signal Processing Magazine* 22: 89–100.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. New York: Springer.

Buntine, W. 1996. "A Guide to the Literature on Learning Probabilistic Networks from Data." *IEEE Transactions on Knowledge and Data Engineering* 8: 195–210.

Cowell, R. G., A. P. Dawid, S. L. Lauritzen, D. J. Spiegelhalter. 1999. *Probabilistic Networks and Expert Systems*. New York: Springer.

Donkers, J., K. Tuyls. 2008. "Belief Networks in Bioinformatics." *Computational Intelligence in Bioinformatics*, ed. A. Kelemen, A. Abraham, Y. Chen, 75–111. Berlin: Springer.

Ghahramani, Z. 2001. "An Introduction to Hidden Markov Models and Bayesian Networks." *International Journal of Pattern Recognition and Artificial Intelligence* 15: 9–42.

Jensen, F. 1996. *An Introduction to Bayesian Networks*. New York: Springer.

Jordan, M. I., ed. 1999. *Learning in Graphical Models*. Cambridge, MA: MIT Press.

Jordan, M. I. 2004. "Graphical Models." *Statistical Science* 19: 140–155.

Jordan, M. I. 2009. *An Introduction to Probabilistic Graphical Models*. Forthcoming.

Jordan, M. I., Z. Ghahramani, T. S. Jaakkola, L. K. Saul. 1999. "An Introduction to Variational Methods for Graphical Models." *Learning in Graphical Models*, ed. M. I. Jordan, 105–161. Cambridge, MA: MIT Press.

Kschischang, F. R., B. J. Frey, H.-A. Loeliger. 2001. "Factor Graphs and the Sum-Product Algorithm." *IEEE Transactions on Information Theory* 47: 498–519.

Lauritzen, S. L., D. J. Spiegelhalter. 1988. "Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems." *Journal of Royal Statistical Society B* 50: 157–224.

MacKay, D. J. C. 2003. *Information Theory, Inference, and Learning Algorithms*. Cambridge, UK: Cambridge University Press.

Neapolitan, R. E. 2004. *Learning Bayesian Networks*. Upper Saddle River, NJ: Pearson.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA: Morgan Kaufmann.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge, UK: Cambridge University Press.

Sudderth, E. B., A. Torralba, W. T. Freeman, A. S. Willsky. 2008. "Describing Visual Scenes Using Transformed Objects and Parts." *International Journal of Computer Vision* 77: 291–330.

Zweig, G. 2003. "Bayesian Network Structures and Inference Techniques for Automatic Speech Recognition." *Computer Speech and Language* 17: 173–193.

Bölüm 17

Modellerin Birleştirilmesi

Önceki bölümlerde birçok öğrenme algoritmasından bahsettik. Bu algoritmalarla eğitilmiş modeller genelde başarılı olsalar da her uygulamada en başarılı modeli oluşturacak bir algoritma yoktur. Bu bölümde birbirini tamamlayan modeller oluşturmayı ve bu modelleri daha yüksek başarı elde edecek biçimde birleştirmeyi göredeceğiz.

17.1 Gerekçe

Bir uygulamada farklı öğrenme algoritmalarından herhangi birini ve her algoritmada model üstparametre değerlerinden herhangi birini kullanabiliriz. Örneğin bir sınıflandırma uygulamasında, olasılık tabanlı bir sınıflandırıcı ya da çok katmanlı bir algılayıcı kullanabiliriz, ve eğer çok katmanlı algılayıcı kullanırsak saklı birim sayısı şu ya da bu olabilir. Olası tüm uygulamalar üzerinde en başarılı olacağına emin olduğumuz bir algoritma yoktur ve bu yüzden her uygulamada olası algoritmaların hepsini geçerleme kümesi üzerinde deneyip en başarılı olanı seçmek gerekir; İngilizcede "Bedava yemek yoktur," denir, Türkçede "Taş yerinde ağırdır," deriz.

Her öğrenme algoritması belirli varsayımlara dayanan bir model kullanır ve bu varsayımlar eldeki veri için geçerli değilse hataya neden olur. Öğrenme eksik konumlanmış bir problemdir ve sonlu sayıda veriyle her algoritma farklı bir sonuca yakınsar ve farklı durumlarda hata yapar. Bir modelin geçerleme hatasını en aza indirmek için ayarlamalar yapılabilir ama hem bu ayarlamalar maliyetlidir, hem de ne kadar ayarlasak da, en başarılı algoritmalar bile bazı örneklerde hata yapacaktır. Ama başka bir model bu örneklerde doğru sonuç verebilir. Öyleyse, birden çok *temel modeli* birleştirerek başarıyı artırabiliriz. Son yıllarda bellek ve hesaplama maliyetlerinin azalmasıyla bu biçimde, birden çok modeli birleştiren yaklaşımlar daha çok kullanılır olmuştur (Kuncheva 2004).

Burada, temel olarak iki soru vardır:

1. Birbirini tamamlayan temel modelleri nasıl oluşturabiliriz?

2. Başarıyı artırmak için modellerin çıktılarını nasıl birleştirebiliriz?

Bu bölümdeki tartışmamız bu iki sorunun çevresinde olacak. Özellikle vurgulamak istediğimiz nokta, model birleştirmenin başarıyı her zaman artıran sihirli bir değnek olmadığıdır. Bir yerine birden çok modeli birleştirmek bellek ve hesap gereksinimini her zaman artırır, ama temel modeller dikkatli eğitilip kararları zekice birleştirilmezse başarıda bir artma görmeden yalnızca karmaşıklığı ve maliyeti artırmış oluruz.

17.2 Farklı Modellerin Oluşturulması

Hep aynı kararları veren modelleri birleştirmenin bir anlamı yoktur; amaç, aralarında *çeşitlilik* olan, farklı karar verip birbirlerini tamamlayan modeller oluşturabilmektir. Bununla birlikte, sonucun başarısının yüksek olması için karara katılan modellerin (en azından kendi sorumluluk alanlarında) başarılı olmaları gerektiğini de biliyoruz. Dolayısıyla hem modellerin birer birer başarılarını, hem de aralarındaki çeşitliliği artırmak gibi çifte amacımız var. Gelin, bunun nasıl sağlanabileceğini konuşalım.

Farklı Algoritmalar

Farklı temel modelleri eğitmek için farklı öğrenme algoritmaları kullanabiliriz. Farklı algoritmalar veriyle ilgili farklı varsayımlar yaptığından farklı modeller (örneğin, sınıflandırıcılar) oluştururlar. Örneğin, temel modellerden biri çok katmanlı bir ağ, bir başkası bir karar ağacı olabilir. Tek bir algoritmayı seçtiğimizde kendimizi tek bir yönteme bağlıyor ve ötekileri gözardı ediyoruz. Farklı algoritmalara dayanan farklı modelleri birleştirdiğimizde böyle bir karar vermemize gerek kalmıyor ve bütün yumurtalarımızı aynı sepete koymuyoruz.

Farklı Üstparametreler

Aynı algoritmayı farklı üstparametre değerleriyle kullanan modeller de üretebiliriz. Örnekler, çok katmanlı algılayıcıda saklı birim sayısı, en yakın k komşu algoritmasında k değeri, karar ağaçlarında hata eşiği, destek yöney makinelerinde çekirdek işlevi, vs. Gauss dağılımları varsayan sınıflandırıcıda, eşdeğişinti dizeylerinin ortak olup olmaması bir üstparametredir. Eğer bir hata işlevi üzerinde iyileme yaparak öğreniyorsak ve kullandığımız, hata geri yayma gibi son durumun ilk duruma bağlı olduğu yinelemeli bir yordamsa, ilk değerler, örneğin ilk bağlantı ağırlıkları, üstparametredir. Farklı üstparametre değerleri kullanan modelleri birleştirdiğimizde bu etkenin olası değerleri üzerinden birleştirme yaptığımız için bu etkenden dolayı oluşan değişintiyi, dolayısıyla hatayı, azaltmış oluruz.

Farklı Girdi Gösterimleri

Farklı temel modeller, farklı *girdi gösterimleri* kullanabilir. Bu farklı gösterimler, farklı alıcılardan farklı kiplerde gelebilir. Farklı gösterimler, girdi nesne ya da ol-

gusunun farklı özelliklerini belirginleştirerek tanımayı kolaylaştıracaktır. Birçok uygulamada farklı bilgi kaynakları vardır ve başarıyı artırmak için olası tüm veriyi kullanarak daha fazla bilgi kullanmak istenir.

Örneğin konuşma tanımada, sözcükleri tanırken ses bilgisi yanında konuşmacının dudak hareketlerini görüntü olarak alıp kullanabiliriz. Farklı alıcılardan gelen verilerin daha fazla bilgi çıkarmak için birleştirilmesine *alıcı kaynaştırma* denir.

En basit yaklaşım, farklı kaynaklardan gelen verileri yan yana koyup sanki tek bir kaynaktan gelen uzun, tek bir girdiymiş gibi tek bir modele vermektir, ama bu kuramsal olarak uygun olmaz, çünkü tüm verinin çok boyutlu tek bir dağılımdan geldiğini varsayar. Üstelik girdi boyut sayısının artması, modelin karmaşıklığını ve kestirmenin başarılı olması için gereken örneklem büyüklüğünü artırır. Bunun yerine önerdiğimiz yaklaşım, farklı kaynakları gelen girdileri farklı modellere girdi olarak verip onların kararlarını birleştirmektir.

Tek bir gösterim olsa bile, eğer boyut sayısı çoksa, rastgele öznitelik altkümeleriyle farklı girdiler oluşturulup farklı modeller eğitilebilir; buna *rastsal altuzay yöntemi* (Ho, 1998) diyoruz. Böylece, farklı modeller aynı probleme farklı açılardan bakabilecek ve verilen karar daha gürbüz olacaktır; bir başka güzel yanı da, girdi boyutları azalacağı için çok boyutluluğun lanetinden sakınmamız olacaktır.

Farklı Öğrenme Kümeleri

Modelleri farklı yapmak için bir başka olasılık da farklı temel modellerin aynı öğrenme kümesinin farklı altkümeleri üzerinde eğitilmesidir. Öğrenme kümesinden rastgele örnekler seçebiliri, ya da modelleri sırayla eğitip daha önce üzerinde hata yapılan örneklere ağırlık verilecek biçimde yeni kümeler oluşturabiliriz; böylece, modellerin birbirlerini tamamlamasını şansa bırakılmak yerine daha etkin bir biçimde sağlamış oluruz.

Girdi uzayında tanımlanacak bir yerellik işlevine bağlı olarak girdileri modeller arasında paylaştırılabiliriz; böyle bir durumda her model, girdi uzayında sorumlu olduğu örneklerle, uzayın bir bölümünde uzmanlaşacak biçimde eğitilir. Bölüm 12'de tartıştığımız *uzmanların karışımı* bu düşünceye dayanır ve bu bölümde aynı algoritmayı bir kez daha, bu kez model birleştirme bakış açısından tartışacağız. Benzer biçimde bir görevi daha basit modellerce yüklenilebilecek altgörevlere bölmenin bir başka yolu da *hata düzelten çıktı gizyazılarıdır*.

Çeşitlilik ya da Başarı

Farklı temel modellerin kendi başlarına başarılı olmaları gerekiyor ama çok da değil. Temel modellerin karmaşıklığı artırmamak ve az veri olduğunda ezberlememek için basit olmaları gerekiyor. Çeşitli olmalarını ve farklı örneklerde başarılı olmalarını, böylece problemin farklı altproblemlerine özelleşmelerini istiyoruz. Bizim için önemli olan temel modellerin kendi bireysel başarıları değil, birleştirmeden sonraki başarı. Elimizde yüzde 80 başarılı bir sınıflandırıcı varken yanına ikinci bir sınıflandırıcı eklediğimizde önemli olan bu ikincinin genel başarısı değil, kalan yüzde 20'de ne

kadar başarılı olduğudur (hangi örnek için ikisinden hangisini seçmemiz gerektiğini bildiğimiz varsayımıyla).

Bu da gösteriyor ki, modellerin başarı ve çeşitliliğinden bahsederken onların nasıl birleştirileceği de önemli. Eğer oylama gibi her girdi için her model kullanılıyorsa, modellerin her yerde başarılı olmasını ve çeşitliliğin her yerde sağlanmasını bekleriz; girdi uzayını modeller arasında uzmanlık alanlarına göre böldüğümüzdeyse çeşitlilik zaten bu bölünmeyle sağlanmış olur ve her modelin yalnızca kendi alanında başarılı olması yeterlidir.

17.3 Model Birleştirme Yöntemleri

Temel modellerin çıktılarını birleştirip en son çıktıyı oluşturmak için farklı yöntemler vardır:

- *Koşut model birleştirme* yöntemleri kendi içlerinde ikiye ayrılır:
  - *Küresel* yaklaşımda, girdiye bağlı olarak her model kendi çıktısını verir ve sonra bu çıktılar birleştirilir; buna *model kaynaşımı* da denir. *Oylama* ve *yığma*, bunun örnekleridir.
  - *Yerel* yaklaşımdaysa, örneğin, *uzmanların karışımında*, girdiye bağlı olarak modellerden birini ya da birkaçını seçen bir *geçit modeli* vardır. Bu yaklaşıma bazen *devingen model seçimi* de denir.

- *Sıralı model birleştirme* yöntemlerinde bir model, ancak öncekiler yeterince emin değilse kullanılır. Temel modeller (ya da kullandıkları girdi gösterimleri) karmaşıklıklarına göre sıralanmışlardır ve her model, çıktısıyla birlikte çıktısından ne kadar emin olduğuyla ilgili bir *güven değeri* de üretir; öyle ki, karmaşık bir model ancak ondan önceki daha basit modeller kendinden emin değilse kullanılır. Bu yaklaşıma örnek, *ardışık* modellerdir.

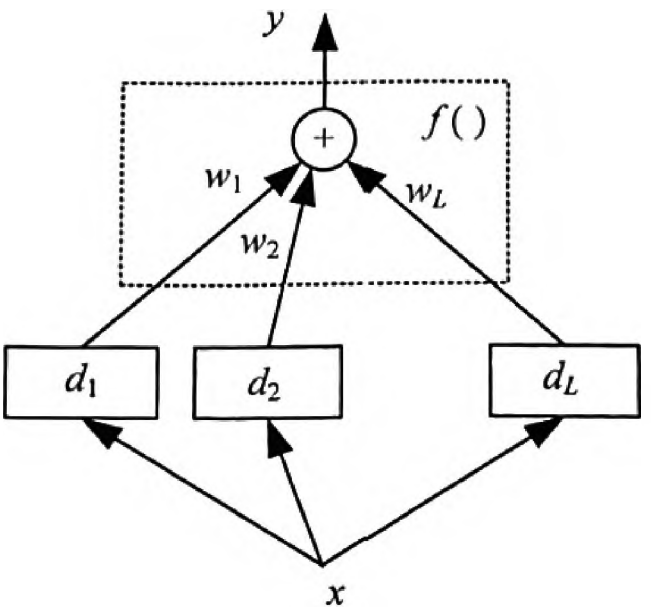
L tane temel modelimiz olsun. $d_j(x), j = 1, \ldots, L$, $\mathcal{M}_j$ modelinin girdi x için verdiği çıktıyı göstersin (farklı gösterimler varsa her $\mathcal{M}_j$ kendi x_j gösteriminini kullanır). Bu çıktıları birleştirip en son çıktıyı hesaplarız:

$$y = f(d_1, d_2, \ldots, d_L|\Phi) \tag{17.1}$$

Burada, $f(\cdot)$ birleştirme işlevini, Φ de onun parametrelerini gösterir.

Her modelin K tane çıktısı, $d_{ji}(x), i = 1, \ldots, K$, $j = 1, \ldots, L$, olduğunda bunlar birleştirilerek K tane son çıktı, $y_i, i = 1, \ldots, K$, hesaplanır. Örneğin sınıflandırmada, birleştirme ardından en yüksek y_i değerli sınıf seçilir:

$$C_i \text{ seçilir eğer } y_i = \max_{k=1}^{K} y_k$$



Şekil 17.1: Temel modellerin d_j ile gösterdiğimiz çıktıları $f(\cdot)$ işleviyle birleştirilir. Ağırlıklı oylamada w_j model ağırlıklarıdır. Basitlik için modellerin tek çıktısı gösterilmiştir; sınıflandırmada her temel modelin K tane çıktısı vardır, bunlar ayrı ayrı birleştirilerek sınıf öngörüleri y_i oluşturulur ve sonra en büyük olanı seçilir. Bu şekilde tüm temel modeller aynı girdiyi kullanmaktadır; farklı modeller aynı girdinin farklı gösterimlerini de kullanabilir.

17.4 Oylama

Farklı sınıflandırıcıları birleştirmenin en kolay yolu olan *oylamada* model çıktılarının doğrusal bir birleşimi hesaplanır (bkz. şekil **17.1**):

$$y_i = \sum_j w_j d_{ji} \text{ öyle ki } w_j \geq 0 \text{ ve } \sum_j w_j = 1 \tag{17.2}$$

Basit oylamada tüm modellerin ağırlığı birbirine eşittir ve bu ortalama almaya eştir. Ağırlıklı oylamada ağırlıkların belirlenmesi gerekir. Birleştirmede (ağırlıklı) toplam almak, model birleştirme için kullanılabilecek sabit kurallardan yalnızca biridir; başka kurallar da (bkz. tablo 17.1) önerilmiştir (Kittler ve ötekiler 1998). Çıktılar sonsal olasılık değillerse aynı ölçeğe gelecek biçimde normalleştirilmeleri gerekir (Jain, Nandakumar, ve Ross 2005).

Bu kuralların kullanımına bir örnek tablo 17.2'de verilmiştir. Toplama, en kolay anlaşılır ve gerçek yaşamda en çok kullanılan birleştirme kuralıdır. Ortanca, aykırı değerlere karşı gürbüzdür. En küçük ve en büyük işlevleri sırasıyla en kötümser ve iyimser olanlardır. Çarpımda her modelin veto gücü vardır; ötekilerin çıktısı ne olursa olsun, bir model 0 çıktısı verdiğinde, son çıktı 0 olur. Dikkat: Birleştirme kurallarının uygulanmasından sonra y_i değerlerinin toplamı 1 olmayabilir.

Ağırlıklı toplamda, d_{ji} değeri, j modelinin C_i sınıfı için verdiği oy, w_j değeri

Tablo 17.1: Sınıflandırıcı birleştirme kuralları.

Kural	Birleştirme işlevi $f(\cdot)$
Toplam	$y_i = \frac{1}{L}\sum_{j=1}^{L} d_{ji}$
Ağırlıklı toplam	$y_i = \sum_j w_j d_{ji}$, öyle ki $w_j \geq 0, \sum_j w_j = 1$
Ortanca	$y_i = \text{median}_j d_{ji}$
En küçük	$y_i = \min_j d_{ji}$
En büyük	$y_i = \max_j d_{ji}$
Çarpım	$y_i = \prod_j d_{ji}$

Tablo 17.2: Üç sınıfı olan üç modelli bir örnekte model birleştirme kurallarının verdiği değerler.

	C_1	C_2	C_3
d_1	0.2	0.5	0.3
d_2	0.0	0.6	0.4
d_3	0.4	0.4	0.2
Toplama	0.2	**0.5**	0.3
Ortanca	0.2	**0.5**	0.4
En küçük	0.0	**0.4**	0.2
En büyük	0.4	**0.6**	0.4
Çarpım	0.0	**0.12**	0.032

de o oyun ağırlığıdır. Basit oylamada herkesin oyu eşit ağırlıktadır: $w_j = 1/L$. Sınıflandırmada *çoğulluk oyuyla* karar verildiğinde en fazla oy alan kazanır; iki sınıf olduğunda bu *çoğunluk oyuna*, yani, oyların yarısından fazlasını alanın seçilmesine eş değerdir (alıştırma 1). Eğer oy verenler, oyları yanında her sınıf için çıktılarından ne kadar emin olduklarını da belirtebilirlerse, normalleştirmeden sonra bu güven değerleri oy ağırlıkları olarak *ağırlıklı oylamada* kullanılabilir. Benzer biçimde, eğer d_{ji} değerleri $P(C_i|x, \mathcal{M}_j)$ sınıf sonsal olasılıklarına eşit olursa doğrudan toplamlarını alıp ($w_j = 1/L$), en büyük y_i değerli sınıfı seçebiliriz.

Bağlanımda, temel modellerin çıktılarını birleştirmek için basit ya da ağırlıklı ortalama, ya da ortanca değer kullanılabilir.

w_j ağırlıklarını bulmanın bir başka yolu da temel modelleri (sınıflandırma ya da bağlanım için) farklı bir geçerleme kümesi üzerinde deneyip başarılı olanlara daha fazla ağırlık vermektir. Bu ağırlıklar, 17.9. kısımda yığılmış genelleyicilerden bahsederken göreceğimiz gibi doğrudan veriden de öğrenilebilir. Oylama yöntemlerine Bayesçi bir açıdan baktığımız *Bayesçi model birleşiminde*, ağırlıkları modellerin önsel olasılıkları olarak yorumlarız. Örneğin sınıflandırmada, $w_j \equiv P(\mathcal{M}_j)$ ve

$d_{ji} = P(C_i|x, \mathcal{M}_j)$ olarak görülebilir ve denklem 17.2, Bayesçi olarak yazılabilir:

$$P(C_i|x) = \sum_{\text{tüm modeller } \mathcal{M}_j} P(C_i|x, \mathcal{M}_j)P(\mathcal{M}_j) \tag{17.3}$$

Basit oylama, tekdüze bir önsel dağılıma karşılık gelir. Basit modelleri yeğleyen bir önsel dağılım basitlere daha fazla ağırlık verir. Olası tüm modeller üzerinden toplam alamayacağımızdan, uygulanabilir olmak için sonsal olasılığının yüksek olduğunu düşündüğümüz bir altküme örnekleriz.

Hansen ve Salamon (1990), başarı olasılıkları 1/2'den büyük (rastgele seçmekten daha başarılı) bağımsız sınıflandırıcılar verildiğinde, model sayısı artarken çoğunluk oyuyla karar vermenin başarıyı artırdığını göstermiştir.

Temel modellerin bağımsız ve özdeş olduğunu, beklenen değerlerinin $E[d_j]$ ve değişintilerinin $\text{Var}(d_j)$ ile gösterildiğini varsayalım. Basit ortalama aldığımızda ($w_j = 1/L$), çıktının beklenen değeri aynı kalırken değişintisi L artarken azalır:

$$\begin{aligned} E[y] &= E\left[\sum_j \frac{1}{L} d_j\right] = \frac{1}{L} L E[d_j] = E[d_j] \\ \text{Var}(y) &= \text{Var}\left(\sum_j \frac{1}{L} d_j\right) = \frac{1}{L^2}\text{Var}\left(\sum_j d_j\right) = \frac{1}{L^2} L \text{Var}(d_j) \\ &= \frac{1}{L}\text{Var}(d_j) \end{aligned} \tag{17.4}$$

Beklenen değerin değişmemesi yanlılığın da değişmediğini gösterir. Ama değişinti, ve dolayısıyla ortalama kare hata, oylamaya katılanların sayısı arttıkça azalır. Eğer oy verenler bağımsız değilse, eşdeğişintiler de işe karışır:

$$\text{Var}(y) = \frac{1}{L^2}\text{Var}\left(\sum_j d_j\right) = \frac{1}{L^2}\left[\sum_j \text{Var}(d_j) + 2\sum_j \sum_{i<j} \text{Cov}(d_j, d_i)\right] \tag{17.5}$$

Bu denklem, modeller artı ilintili olduğunda değişinti ve dolayısıyla hatanın artacağını gösteriyor. Farklı algoritmalar ya da girdi gösterimleri kullanmamız ve çeşitliliği artırmaya çalışmamız zaten bu yüzden, modeller arasındaki artı ilintiyi yok etmek ya da en azından olabildiğince azaltabilmek içindir. 17.10. kısımda aralarında yüksek artı ilinti olan modellerin nasıl budanabileceğini tartışacağız.

Aynı denkleme baktığımızda, oy verenlerin bağımsız yerine eksi ilintili olmaları durumunda değişintinin çok daha fazla azaltılabileceğini görüyoruz. Eğer bu sırada yanlılık çok artmazsa bu hatanın da azalmasını sağlayabilir ama bu iki amaç genelde çelişir. Hepsi, hem başarılı, hem de eksi ilintili modeller oluşturamayız. Uzmanların karışımında örneğin uzmanlar eksi ilintilidir ama yanlılardır da (Jacobs, 1997).

Eğittiğimiz her modeli, gerçek ayırtaç/bağlanım işlevinin üzerine eklenmiş bir gürültü, gürültüyü de 0 ortalamalı ve ilintisiz düşünürsek, modeller üzerinden ortalama almak gürültü üzerinden ortalama almaya karşılık gelir. Bu bağlamda, oylamanın işlev uzayında düzleştişrme etkisi vardır ve gerçek işlevin düz olduğunu varsayan bir düzgünleştirici olarak düşünebilir (Perrone, 1993). Buna bir örneği şekil 4.5d'de görmüştük; yüksek değişintili modeller üzerinden ortalama aldığımızda birer birer modellerden daha az hatalı bir sonuç çıkıyordu. Oylamadaki mantık da bu: Yüksek değişintili ama yanlılığı az bağımsız modeller üzerinden oylama yaptığımızda yanlılık düşük kalırken ortalama sayesinde değişinti azalıyor. Temel modeller yanlı olsa bile, değişintideki daha büyük azalma yanlılığı geçip hatanın azalmasını sağlıyor.

17.5 Hata Düzelten Çıktı Gizyazıları

Hata düzelten çıktı gizyazılarında (Dietterich ve Bakiri 1995) sınıflandırma problemi, temel modeller tarafından gerçeklenecek altproblemlere bölünür. Bir sınıfı tüm öteki sınıflardan ayırmak zor bir iş olabilir; yerine bu işi, her biri ayrı bir model tarafından gerçeklenecek daha basit işlere bölebilirsek, bu basit modelleri birleştirerek sınıflandırıcımızı oluşturabiliriz.

Her temel model, çıktısı $-1/+1$ olan ikili bir sınıflandırıcıdır ve bu modeller yan yana L boyutlu bir yöney oluştururlar. $\mathbf{W}$ ise $K \times L$ *gizyazı dizeyidir* ve K satırının her biri, L temel model tarafından tanımlanmış sınıf gizyazılarına karşılık gelir. Örneğin, ikinci satırı $[-1,+1,+1,-1]$ ise, bir örneğin $\mathcal{C}_2$ sınıfından olduğunu söyleyebilmemiz için, d_1 ve d_4 modellerinin eksi, d_2 ve d_3 modellerinin de artı tarafında olması gerekir. Benzer biçimde, gizyazı dizeyinin sütunları temel modellerin yükümlülüklerini tanımlar. Örneğin, üçüncü sütun $[-1,+1,+1]^T$ ise, d_3 modelinin görevinin, $\mathcal{C}_1$ sınıfından örnekleri $\mathcal{C}_2$ ve $\mathcal{C}_3$ sınıflarının örneklerinden ayırmak olduğu anlaşılır. Bu durumda, $\mathcal{C}_2$ ve $\mathcal{C}_3$ sınıflarından tüm örnekler birlikte $\mathcal{X}_3^+$ artı örnek kümesini, $\mathcal{C}_1$ sınıfından örnekler de $\mathcal{X}_3^-$ eksi örnek kümesini oluşturur; ardından d_3 modelini $x^t \in \mathcal{X}_3^+$ için $+1$ ve $x^t \in \mathcal{X}_3^-$ için -1 çıktı verecek biçimde eğitiriz.

Dolayısıyla, gizyazı dizeyi $\mathbf{W}$, $K > 2$ sınıflı bir problemi, çok sayıda iki sınıf problemi olarak tanımlamamızı sağlar. İki sınıflı modelleri, herhangi bir algoritmayla, (tek çıktısı olan) tek ya da çok katmanlı algılayıcı, karar ağacı, ya da (zaten iki sınıf için önerilmiş) destek yöney makineleriyle gerçekleyebiliriz.

Hep kullandığımız yaklaşımda her sınıf için bir ayırtaç vardır. Bunu, $L = K$ olan köşegen bir gizyazı dizeyi olarak yazabiliriz:

$$\mathbf{W} = \begin{bmatrix} +1 & -1 & -1 & -1 \\ -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 \\ -1 & -1 & -1 & +1 \end{bmatrix}$$

Her iki sınıflı sınıflandırıcı, bir sınıfın örneklerini öteki tüm sınıfların örneklerinden ayırır. Bu dizeyde sınıf gizyazıları birbirine benzediğinden ikili sınıflandırıcıların hatalı olması durumunda yanlış sınıflandırma yapılabilir. Hata düzelten gizyazılarındaki

yaklaşım, $L > K$ alarak dizgeye fazlalık eklemek ve böylece sınıf gizyazıları arasındaki Hamming uzaklığını artırmaktır. Bir olasılık, sınıfları *ikili ayırmaktır*. Her C_i ile C_j sınıf çiftini $(i < j)$ ayırmak için ayrı bir ikili sınıflandırıcı tanımlandığında (bölüm 10.4) $L = K(K-1)/2$ olur ve örneğin $K = 4$ için gizyazı dizeyinde 6 sütun bulunur:

$$\mathbf{W} = \begin{bmatrix} +1 & +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & 0 & +1 & +1 & 0 \\ 0 & -1 & 0 & -1 & 0 & +1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{bmatrix}$$

0 değerleri önemsenmez. Örneğin yukarıdaki örnekte, d_1 modeli, C_1 sınıfıyla C_2 sınıfını ayırır ama öteki sınıfların örneklerini kullanmaz. Benzer biçimde, bir örneğin C_2 sınıfından olduğunu söyleyebilmemiz için $d_1 = -1$ ve $d_4 = d_5 = +1$ olmalıdır, ama d_2, d_3, ve d_6 modellerinin değerlerinin ne olduğu önemli değildir. Buradaki sorun, L değerinin $\mathcal{O}(K^2)$ olması ve K küçük olmadığı zaman bu yaklaşımın çok pahalı olmasıdır.

Önerilen, L değerini bellek ve hesap karmaşıklığı açılarından gerçeklenebilir bir sayı olarak belirledikten sonra, satırları ve sütunları arasındaki Hamming uzaklıkları olabildiğince büyük olan gizyazı dizeyini bulmaktır. K sınıfımız varsa, olası $2^{(K-1)} - 1$ tane iki sınıf problemi (sütun) tanımlanabilir. Çünkü K bit uzunluğunda 2^K tane farklı ikili dizi yazılabilir, tümleyenler atılır (örneğin, "0101" ve "1010" aynı sınıflandırıcıdır) ve hepsi 0 ya da hepsi 1 olan sınıflandırıcı hiçbir işe yaramaz. Örneğin, $K = 4$ için 7 tane sütun vardır:

$$\mathbf{W} = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & +1 & +1 & +1 & +1 \\ -1 & +1 & +1 & -1 & -1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 \end{bmatrix}$$

K büyük olduğunda, verilen bir L değeri için olası $2^{(K-1)} - 1$ sütun içinden L tanesini seçeriz. Temel modellerin birbirinden olabildiğince farklı problemler öğrenmesi (çeşitlilik) için sütunların birbirlerinden farklı olmasını isteriz. Bununla birlikte, daha fazla model hatasını düzeltebilmek için sınıf gizyazılarına karşılık gelen satırların da olabildiğince birbirinden farklı olmasını isteriz.

Hata düzelten çıktı gizyazılarında en son kararı verirken kullandığımız Hamming uzaklığı ölçme, iç çarpım olarak da gerçeklenebilir, ve bu durumda işlem, gizyazı dizey öğeleri olan w_{ij} değerlerinin ağırlıklar olduğu bir tür oylama olarak da görülebilir:

$$y_i = \sum_{j=1}^{L} w_{ij} d_j \tag{17.6}$$

ve ardından en büyük y_i değerine karşılık gelen sınıfı seçeriz. Birebir aynı olmasını istemek yerine ağırlıklı toplam alıp ardından en büyük olanı seçtiğimizde, d_j değerlerinin ikili olması gerekmez ve $[-1, +1]$ aralığında eşiksiz bir değer alabilirler. Eşiksiz bir değer, dolaylı olarak her sınıflandırıcının kararından ne kadar emin olduğunu da

gösterir. Eğer sınıflandırıcı çıktısı 0 ile 1 arasındaysa (örneğin, sonsal olasılıksa), bu değeri kolayca -1'le $+1$ arasına eşleyebiliriz:

$$d_j = 2p_j - 1$$

Denklem 17.6'yla denklem 17.2'deki genel oylama arasındaki fark, ilkinde bir modelin sınıflar için oy ağırlıklarının farklı olabilmesidir (çünkü w_j yerine w_{ij} olarak kullanıyoruz); bir başka fark da, $w_j \geq 0$ koşulu yerine w_{ij} değerlerinin olası iki $(-1, +1)$ ya da üç $(-1, 0, +1)$ değerden birini almasıdır.

Hata düzelten gizyazılarıyla ilgili belki en önemli çekince, gizyazı dizeyinin önceden belirlenmesi ve belirlenirken sütunlarının tanımladığı sınıflandırma problemlerinin kolaylığı konusunda bir güvencemiz olmamasıdır. Dietterich ve Bakiri (1995), sütunlar için öğrenilen karar ağaçlarının çok sınıflı ağaca göre bazen daha karmaşık olabildiğini ve çok katmanlı algılayıcılar kullanıldığında yakınsamanın daha yavaş olabildiğini bildirmiştir.

17.6 Örnekleme ve Oylama

Modelleri birbirlerinden farklı yapmanın bir yolunun öğrenme kümelerininin farklı olması olduğunu daha önce konuşmuştuk. Farklı öğrenme kümeleri oluşturmak için N örnek içeren $\mathcal{X}$ öğrenme kümesinden N tane örneği rastgele seçiyoruz, ama her örneği seçip kaydettikten sonra yeniden geri koyuyoruz. Böylece bazı örnekleri birden çok kez seçerken bazı örnekleri de hiç seçmeyebiliyoruz. Bu biçimde oluşturduğumuz $\mathcal{X}_j, j = 1, \ldots, L$ kümeleri aynı kümeden yeniden örneklemeyle türetildiğinden birbirlerine benziyor ama rastgelelik yüzünden biraz da farklılar. Ardından her d_j temel modeli kendi $\mathcal{X}_j$ kümesiyle eğitiliyor.

Öğrenme kümesindeki küçük farklar oluşturulan modelde büyük farklara neden olabiliyorsa, yani modelin öğrenme kümesine bağlı değişkenliği yüksekse, öğrenme algoritmasına *kararsız* diyoruz. Breiman tarafından önerilen (1996) ve *bagging* (*bootstrap aggregating*) adını verdiği algoritmada, yukarıda bahsettiğimiz gibi yeniden örneklemeyle L tane öğrenme kümesi oluşturuluyor, bunlarla kararsız bir öğrenme algoritması kullanılarak L tane farklı model oluşturuluyor ve ardından kararları oylamayla birleştiriliyor, ya da bağlanımda ortalamaları alınıyor. Aykırı değerlere gürbüzlük için ortalama yerine ortanca değer de kullanılabilir.

Daha önce, modeller arasındaki artı ilinti düşük olduğunda ortalama almanın değişintiyi azalttığını gördük; bir algoritma kararlıysa yeniden örneklenmiş öğrenme kümeleri üzerinde eğitilen modeller arasındaki artı ilinti yüksek olacaktır. Karar ağaçları ve çok katmanlı algılayıcılar kararsızdır; en yakın komşu algoritması kararlıdır ama seyrek en yakın komşu kararsızdır (Alpaydın 1997). İlk öğrenme kümesi büyükse yeniden örneklemeyle daha küçük $N' < N$ kümeler oluşturmak gerekebilir, yoksa kümeler birbirine fazla benzeyecek ve modeller ilintili olacaktır.

17.7 Sırayla Eğiterek İteleme

Önceki kısımda gördüğümüz yöntemde, temel modellerin birbirini ne kadar tamamladığı şansa ve modellerin kararsızlığına bırakılmıştır. Bu kısımdaysa, her bir modelin öncekinin hataları üzerinde eğitildiği bir yöntemi göreceğiz. *İteleme* algoritması (Schapire, 1990), üç zayıf modeli birleştirerek güçlü bir model oluşturur. Hata olasılığı 1/2'den az, yani rastgele seçmekten biraz daha iyi bir modele *zayıf*, hata olasılığını istediğimiz kadar küçültebildiğimiz modele de *güçlü* diyoruz.

Büyük bir öğrenme kümesini rastgele üç parçaya böldükten sonra, önce $\mathcal{X}_1$ kümesini kullanıp d_1 modelini eğitiriz. Ardından $\mathcal{X}_2$ kümesini geçerleme kümesi gibi alıp d_1 modeline verir, içinden d_1'in hata yaptığı örnekleri ve aynı sayıda başarılı olduğu örnekleri toplayıp bunlarla d_2 modelini eğitiriz. Ardından $\mathcal{X}_3$ kümesini alıp d_1 ve d_2 modellerine verir, bunların içinden iki modelin üzerinde farklı karar verdikleriyle d_3 modelinin öğrenme kümesini oluştururuz. Deneme sırasında bir örneği d_1 ve d_2 modellerine birlikte veririz; aynı kararı verirlerse bu karar çıktımız olur, yoksa d_3 modeline verip onun çıktısını kullanırız. Schapire (1990), böyle bir birleştirmenin hata olasılığını azalttığını ve bu sistem özyineli olarak (d_1 yerine üçlü böyle bir sistem) kullanıldığında hatanın istendiği kadar azaltılabildiğini göstermiştir.

Başarılı olsa da yöntemin bu ilk sürümü, büyük öğrenme kümesi gerektirdiğinden yaygın kullanılamaz. Kümenin üçe bölünmesi gerekir, üstelik ikinci ve üçüncü modeller yalnızca yanlışlarla eğitildiklerinden ilk küme çok büyük olmazsa onlar için öğrenme kümeleri çok küçük kalabilir. Drucker ve ötekiler (1994), elle yazılmış sayıları tanımak için çok katmanlı algılayıcıları bu yöntemle birleştirirken 118.000 örnek içeren bir küme kullanır.

Freund ve Schapire (1996) tarafından önerilen *AdaBoost* (*adaptive boosting*) algoritmasında hep aynı küme kullanılır, ve bölünmediği için çok büyük olması gerekmez. Ama bu durumda modellerin veriyi ezberlememesi için basit olması gerekir. Bir başka fark olarak bu yeni sürümde birleştirilen modellerin üç tane olma kısıtı yoktur.

AdaBoost algoritmasının, bizim burada göreceğimiz ilk AdaBoost.M1'den (bkz. şekil 17.2) sonra, birçok sürümü önerilmiştir. Ana düşünce, model eğitimi sırasında örneklerin seçilme olasılıklarının hataya bağlı olarak değiştirilmesidir.

$(\boldsymbol{x}^t, r^t)$ örnek çiftinin j taban modelinin eğitilmesinde kullanılma olasılığını p_j^t ile gösterelim. İlk modeli eğitirken tüm olasılıklar eşittir: $p_1^t = 1/N$. Sonraki modeller $j = 1$'den başlayarak sırayla eklenir: ϵ_j ile d_j modelinin hatasını gösterelim. AdaBoost yöntemi, modellerin zayıf olduğunu ($\epsilon_j < 1/2, \forall j$) varsayar; bu sağlanmazsa dururuz. Burada, bu hatanın ilk öğrenme kümesi üzerinde değil, j. adımda oluşturulmuş öğrenme kümesi üzerinde hesaplandığını aklımızda tutalım. $\beta_j = \epsilon_j/(1-\epsilon_j) < 1$ olarak tanımlanır ve ardından buna göre bir sonraki modelin eğitilmesinde kullanılacak ağırlıklar güncellendir: Eğer j modeli, $\boldsymbol{x}^t$ örneğini doğru sınıflandırırsa, $p_{j+1}^t = \beta_j p_j^t$ olur, yani azalır; değilse, $p_{j+1}^t = p_j^t$ alınır ve değişmez. Ardından, p_{j+1}^t değerleri (olasılıklar oldukları için) toplamları 1 olacak biçimde $\sum_t p_{j+1}^t$ değerine bölünerek normalleştirilir ve bu sayede doğru sınıflandırılmış bir örneğin bir sonraki modelin

Öğrenme:
  Tüm $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$ için $p_1^t = 1/N$ ile başla
  Tüm taban modeller $j = 1, \ldots, L$
    $\mathcal{X}_j$'yi $\mathcal{X}$'den p_j^t olasılıklarıyla örnekleyerek oluştur
    d_j'yi $\mathcal{X}_j$ ile eğit
    Her (x^t, r^t) için $y_j^t \leftarrow d_j(x^t)$ hesapla
    Hatayı hesapla: $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$
    Eğer $\epsilon_j > 1/2$ ise $L \leftarrow j - 1$; dur
    $\beta_j \leftarrow \epsilon_j/(1 - \epsilon_j)$
    Tüm (x^t, r^t) için, çıktı doğruysa olasılıkları azalt:
      Eğer $y_j^t = r^t$ ise $p_{j+1}^t \leftarrow \beta_j p_j^t$ değilse $p_{j+1}^t \leftarrow p_j^t$
    Olasılıkları toplamları 1 olacak biçimde normalleştir:
      $Z_j \leftarrow \sum_t p_{j+1}^t$; $p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$
Deneme:
  x için model çıktılarını hesapla: $d_j(x), j = 1, \ldots, L$
  Sınıf çıktılarını hesapla; tüm $i = 1, \ldots, K$:
    $y_i = \sum_{j=1}^L \left(\log \frac{1}{\beta_j}\right) d_{ji}(x)$

Şekil 17.2: AdaBoost algoritması.

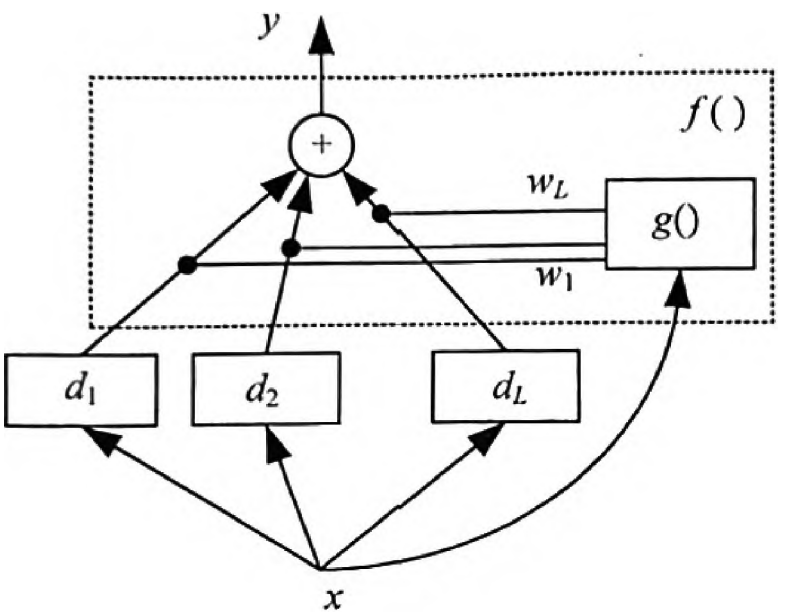
eğitilmesinde kullanılma olasılığı azalırken yanlış sınıflandırılmış bir örneğin olasılığı artar. Bu yeni olasılık değerleri p_{j+1}^t ile asıl kümeden yeni bir öğrenme kümesi örneklenir ve d_{j+1} modelini eğitmek için kullanılır.

Bunun sonucu olarak d_{j+1} modeli, önceki d_j tarafından yanlış sınıflandırılmış örneklere odaklanacaktır. İşte bu yüzden modellerin birer birer başarılı olmalarından çok basit olmalarını istiyoruz, yoksa sonraki öğrenme kümesi yanlış sınıflandırılmış, gürültülü ya da aykırı birkaç örneğin birçok kopyasından oluşacaktır. Örneğin karar ağaçları kullandığımızda bir ya da iki düzeyli *karar ağaççıkları* kullanırız. Bu modelin yanlı olacağını biliyoruz ama birleştirme sırasında azalan değişkenliğin bunu karşılayacağını umuyoruz. Doğrusal bir ayırtacın, örneğin, değişkenliği düşüktür ve bu yüzden AdaBoost ile çoğaltılıp birleştirilmesi bir şey kazandırmaz.

Öğrenme bittikten sonra birleştirme için oylama kullanılır. Öğrenme sırayla olsa da deneme koşuttur. Bir girdi için bütün d_j çıktıları hesaplanır ve ağırlıklı oylamayla birleştirilir. Ağırlıklar, modellerin (öğrenme kümesi üzerindeki) başarılarına orantılıdır: $w_j = \log(1/\beta_j)$. Freund ve Schapire (1996), yirmi iki denek veri kümesinde başarının arttığını, birinde aynı kaldığını, ve dördünde kötüleştiğini belirtir.

Schapire ve ötekiler (1998), AdaBoost algoritmasının başarısının *kenar payını* artırmaktan dolayı olduğunu savunur. Kenar payı artınca öğrenme örnekleri birbirlerinden daha iyi ayrılmakta ve hata azalmaktadır. Bu da AdaBoost algoritmasını destek yöney makinelerine (bölüm 13) benzer yapar.

Farklı taban modellerinin öğrenme kümeleri arasındaki fark, modellerin hata-



Şekil 17.3: Uzmanların karışımı, ağırlıkların bir g geçit modeli tarafından girdiye bağlı olarak belirlendiği bir ağırlıklı oylama gerçekler. Birleştirme işlevi f, bu geçit modelini de içerir.

larına bağlıdır ve herhangi bir uygulamada AdaBoost algoritmasının ne kadar başarılı alacağı veriye ve kullanılan model algoritmasına bağlıdır. Yeterince örnek olmalı, model zayıf olmalı ama çok da zayıf olmamalıdır. Veride gürültülü ve aykırı örnekler çoksa AdaBoost sürekli bunlara yoğunlaşacağından başarısı yüksek olmayacaktır.

AdaBoost, bağlanım için genellenmiştir: Bunun için en kolay yol (Avnimelech ve Intrator 1997), kare hatayı bir eşik değeriyle karşılaştırıp eğer büyükse bunu sınıflandırma hatası gibi görmektir. Bir başka sürümde (Drucker 1997), olasılıklar kare hatanın büyüklüğüne oranla güncellenir. Taban modellerini birleştirirken ortalama ya da ortanca değer kullanılır.

17.8 Uzmanların Karışımına Farklı Bir Bakış

Oylamada w_j ağırlık değerleri sabittir, girdiye bağlı değildir. Bölüm 12.8'de yerel bir model olarak gördüğümüz *uzmanların karışımı* yapısında, çıktıları uzmanların ağırlıklarını belirleyen bir geçit modeli vardır. Bu biçimde baktığımızda bu yapıyı, ağırlıkların girdiye bağlı belirlendiği bir tür ağırlıklı oylama olarak görebiliriz. Böylece, bir uzmanın ağırlığı farklı girdiler için farklı olabilir. Uzmanların karışımındaki yerel yaklaşım, uzmanları girdi uzayının farklı bölgelerine yerleştirir ve her uzmanın ağırlık değeri olan $w_j(x)$ değeri kendi uzmanlık alanında 1 değerine yaklaşır. Sonuç çıktı, oylamada olduğu gibi ağırlıklı ortalamayla bulunur:

$$y = \sum_{j=1}^{L} w_j(x) d_j \tag{17.7}$$

Oylamadan farkı, burada hem taban modellerin, hem de onların ağırlıklarının girdiye bağlı olmasıdır (bkz. şekil 17.3).

Jacobs (1997), öğrenme sonunda uzmanların yanlı ama eksi ilintili olduğunu göstermiştir. Öğrenme sürdükçe uzmanların yanlılıkları azalır ama değişintileri artar, ve girdi uzayının farklı bölgelerinde uzmanlaştıkça, eşdeğişinti değerleri eksiye döner. Bu sayede, 17.5. denklemde yazıldığı gibi, toplam değişinti ve dolayısıyla hata azalır. Bölüm 12.8'de gördüğümüz yapıda hem uzmanlar, hem de geçit modeli doğrusaldı; ama bunlar için doğrusal olmayan bir model, örneğin, çok katmanlı algılayıcılar da kullanılabilir. Böylece uzman yanlılıkları azalır ama değişkenliğin artması ve ezberleme riski ortaya çıkar.

Devingen sınıflandırıcı seçiminde, uzmanların karışımındaki geçit modeline benzer bir biçimde modellerin girdiye bağlı olarak yeterlilikleri kestirilir. Ardından en yeterli model seçilerek yalnızca onun çıktısı alınır. Woods, Kegelmeyer, ve Bowyer (1997), deneme girdisine en yakın k öğrenme örneğini bulur, taban modellerinin bunlar için verdiği çıktının doğruluğuna bakar ve buna göre modeller arasından en başarılıyı seçer. Yalnız en yetkin model için çıktı hesaplanır. Değişkenliği azaltmak için karmaşıklığı artırmak pahasına bir yerine en yetkin birkaç model üzerinde oylama da yapılabilir.

Bu yöntemin başarılı olması için girdi uzayının her bölgesinde yeterli en az bir model gerekir, yani, modeller uzayı aralarında paylaşmalıdır. Uzmanların karışımında bu, uzmanları yerleştiren geçit modeliyle uzmanların eğitilmesinin bağlı yapılması sayesinde sağlanır. Böyle bir devingen model seçimi, bağlanım için de kolayca genellenebilir (alıştırma 5).

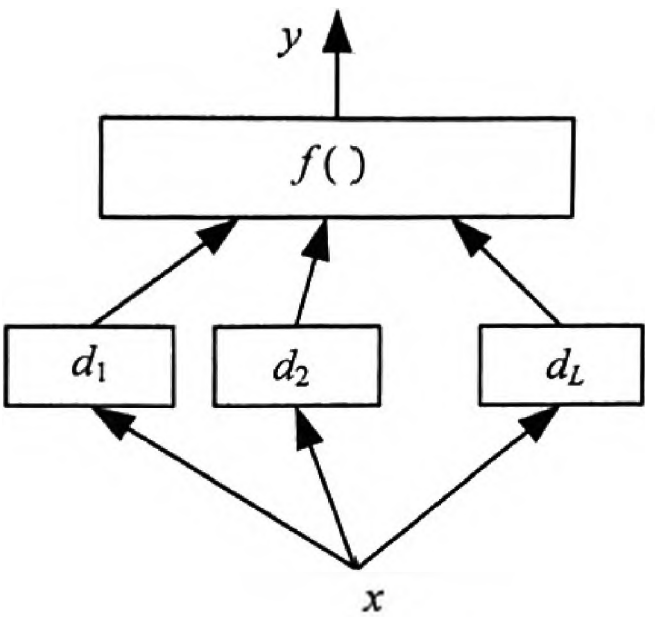
17.9 Yığılmış Genelleme

Yığılmış genellemede (Wolpert 1992), modelleri birleştiren $f(\cdot|\Phi)$, Φ parametreleriyle tanımlanmış başka bir modeldir ve eğitilir (bkz. şekil 17.4):

$$y = f(d_1, d_2, \ldots, d_L|\Phi) \tag{17.8}$$

Taban modeller belli çıktılar verdiğinde gerçek çıktının ne olması gerektiğini birleştirici belirler. Birleştiriciyi taban modellerle aynı veri üzerinde eğitmemek gerekir çünkü bazı taban modeller öğrenme verisini ezberlemiş olabilir; birleştiricinin gerçekte öğrenmesi gereken hangi taban modellerin nerede ve nasıl hata yaptığıdır. Yığmanın amacı temel olarak taban modellerin yanlılıklarını kestirmek ve düzeltmektir. Bu yüzden birleştiricinin taban modellerin eğitiminde kullanılan veriden farklı bir veri üzerinde eğitilmesi gerekir.

$f(\cdot|w_1, \ldots, w_L)$ doğrusal bir model olduğunda ağırlık değerleri, $w_i \geq 0, ve \sum_j w_j = 1$ kısıtlarını sağlayacak biçimde bağlanımla bulunabilir. Ama tabii ki birleştirmenin doğrusal olmasını (ya da öteki kısıtları) zorunlu tutmamıza gerek yok; yığmada birleştirici herhangi bir işlev olabilir ve oylamanın aksine, $f(\cdot)$ doğrusal olmayan bir model olabilir. Örneğin, çok katmanlı bir algılayıcı ve Φ ile gösterdiğimiz bu algılayıcının bağlantı ağırlıkları olabilir. d_j taban modellerinin her birinin K tane



Şekil 17.4: Yığılmış genellemede birleştirici de veriden eğitilen başka bir modeldir ve oylamadan farklı olarak doğrusal olmak zorunda değildir.

çıktısı olduğunda hep birlikte $L \cdot K$ boyutlu bir uzay tanımlarlar ve birleştirici model, bu uzayda çıktıyı belirleyen ayırtaç/bağlanım işlevini öğrenir.

Yığılmış genelleyicilerde taban modellerin birbirlerini tamamlayabilmek için birbirlerinden olabildiğince farklı olmasını isteriz ve bunun için en iyi yol farklı öğrenme algoritmaları kullanmalarıdır. Eğer sınıflandırıcılarımız sürekli çıktı, örneğin sonsal olasılık verebiliyorsa, 0/1 kararları yerine girdi olarak bunları birleştirmek bilgi kaybına neden olmayacağından yeğlenir.

Yığmadaki gibi eğitilmiş bir birleştirici modelle oylamadaki gibi sabit bir kuralı karşılaştırdığımızda ikisinin de üstün yanları olduğunu görüyoruz: Eğitilmiş bir birleştirici daha esnektir ve daha az yanlıdır, ama ek parametreler gerektirdiği için değişkenlik artar ve eğitmek için ek veri ve zaman gerekir. Bir ek fark da yığmadan önce sınıflandırıcı çıktılarının normalleştirilmesinin gerekmemesidir.

17.10 Model Kümesinin Ayarlanması

Bölümün en başında bahsettiğimiz gibi, birden çok modeli birleştirmek hatayı her zaman azaltacak sihirli bir formül değildir; taban modellerin çeşitli ve başarılı olması, yani işe yarar bilgi vermesi gerekir. Eğer bir model başarıya katkı yapmıyorsa atılabilir; birbiriyle ilintisi yüksek iki modelden birine gerek yoktur. Başarısız bir modeli tutmakta ısrar etmek sonuç başarısını kötüleştirebilir; örneğin, çoğunluk oyunda sınıflandırıcıların en az yarısının doğru karar vermesi gerekir. Bunlar gösteriyor ki elimizde taban modeller olduğunda hepsini kullanmak yerine bir altkümesini kullanmak, karmaşıklığı azaltmak yanında başarıyı da yükseltebilir.

Bir model kümesinden altküme seçilmesi, girdide öznitelik seçimine benzer ve

model kümesi seçimi için kullanılan yaklaşımlar aynıdır. İleriye doğru (artımlı ya da büyüyen) bir yaklaşımla, verilmiş bir model kümesi içinden en başarılı, ya da çeşitli olanı, ya da eklendiğinde başarıyı en fazla artıranı seçilen altkümeye birer birer eklenir, geriye doğru (azalımlı ya da budayan) bir yaklaşımla bu kez yokluğu en yararlı olan modeli birer birer atabilir. İstersek hem ileri, hem geri adımlara da izin verebiliriz. Altküme içinde seçilmiş modelleri birleştirirken oylama gibi sabit bir kural ya da öğrenen bir birleştirici kullanabiliriz. Böyle bir altküme seçimi, başarısız ya da öteki modellerle ilintili olanları atacaktır (Caruana ve ötekiler 2004, Ruta ve Gabrys 2005). Farklı modeller farklı gösterimler kullandığında böylesi bir seçim, birbirlerini en iyi tamamlayan gösterimleri bulmamızı sağlayacaktır (Demir ve Alpaydın 2005).

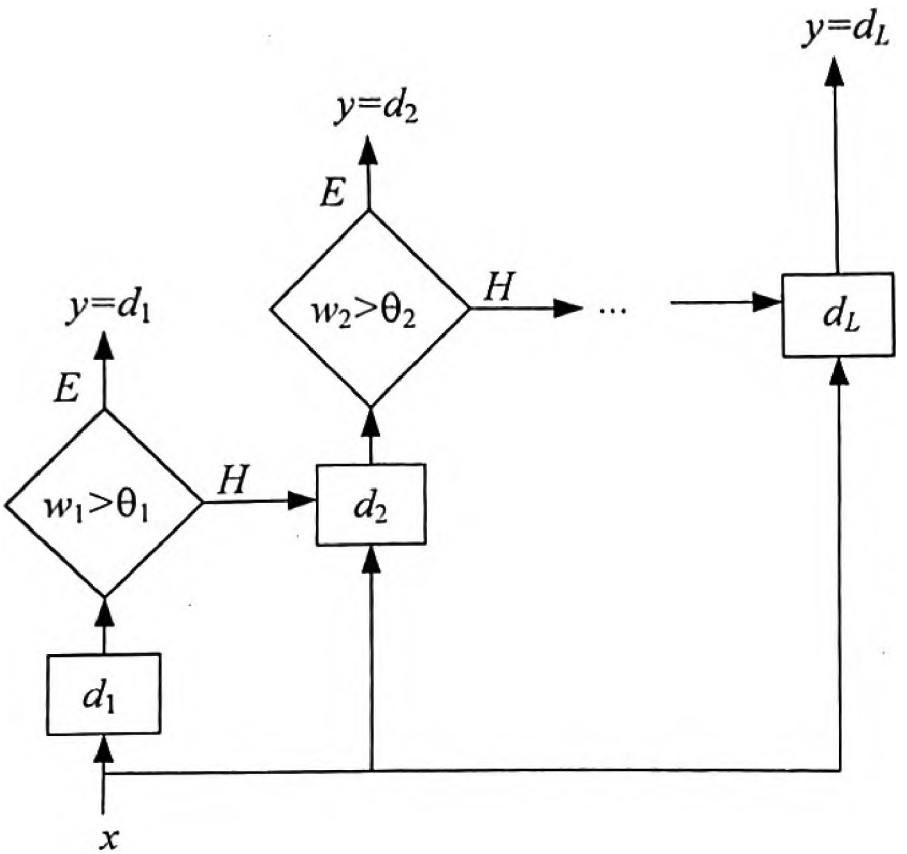
Eğer birleştiriciyi, yığılmış genelleyicilerde düşündüğümüz gibi, taban modellerinin çıktılarını girdi olarak alan bir model olarak düşünürsek, burada amaçlanan girdide boyut azaltmadır. Bölüm 6'da gördüğümüz gibi bir olasılık, bilgi vermeyen girdilerin atılmasıdır, ki bu yukarıda bahsettiğimiz model kümesi seçimine karşılık gelir (tek fark, bir modelin bazı çıktılarını tutup bazılarını atabilmemizdir). Örneğin, model birleştirme için bir karar ağacı kullanırsak, ağaç, hem seçici, hem de birleştirici görevi yapar (Ulaş ve ötekiler, 2009).

İkinci olasılıksa taban modellerin çıktılarının tanımladığı uzaydan gereksiz girdileri ve ilintileri atarak daha düşük boyutlu, yeni bir uzaya geçmektir. Merz (1999), SCANN algoritmasında sınıflandırıcıların 0/1 çıktılarına temel bileşen çözümlemesinin (bölüm 6.3) türevi bir yaklaşım olan uygunluk çözümlemesi uyguladıktan sonra en yakın ortalama sınıflandırıcısı kullanır. Gerçekte bu amaç için doğrusal ya da doğrusal olmayan herhangi bir boyut azaltma yöntemi (6. bölüm) kullanılabilir ve çıktısı herhangi bir modele (sınıflandırma ya da bağlanım için) beslenebilir. Sözün kısası, her birinin K çıktısı olan L modelin tanımladığı $K \cdot L$ boyutlu uzaydan daha düşük boyutlu ve boyutları ilintisiz bir "özmodeller" uzayına geçip birleştiriciyi bu uzayda öğrenebiliriz. Hem boyut azaltma, hem de ardından gelen modelin 0/1 yerine (olabildiğinde) sürekli girdi alması bilgi kaybını engeller. Dikkat edilmesi gereken bir başka nokta da boyut azaltma ve birleşticinin ayrı veriler üzerinde eğitilmesinin gerekmesidir.

Bu yaklaşım, taban modellerin bazılarını tutup bazılarını atmak yerine hepsini kullanır; bilgi kaybetmez ama karmaşıklıkta da bir azalma sağlamaz.

17.11 Ardışık Sınıflandırıcılar

Ardışık sınıflandırıcılarda taban modellerini bellek ya da hesap karmaşıklığına, ya da kullandıkları gösterimin maliyetine göre bir *sıraya* koyar ve gerekmedikçe pahalı olanları kullanmamaya çalışırız (Kaynak ve Alpaydın 2000). d_j modeli, d_{j-1} modeline göre daha pahalıdır ve toplam maliyeti azaltmak için d_j modelini yalnızca ondan önce gelen $d_k, k < j$ modelleri çıktılarından emin değillerse kullanmak isteriz. Bu amaç doğrultusunda, her model için bir w_j *güven* değeri hesaplanır ve bu güven değeri bir eşikten yüksekse, $w_j > \theta_j$ $(1/K < \theta_j \leq \theta_{j+1} < 1)$, d_j modeli çıktısından emindir, deriz. Sınıflandırmada güven değerini doğrudan en yüksek son-



Şekil 17.5: Ardışık sınıflandırıcılar, sıralı bir birleştirme yöntemidir. Modeller maliyetlerine göre artan biçimde sıralanmıştır ve bir modeli ancak ondan öncekiler emin değilse kullanırız.

sal olasılık olarak alabiliriz: $w_j \equiv \max_i d_{ji}$. Zaten 3.3. bölümden hatırlayacağımız gibi karar vermeyi reddettiğimiz şüpheli örnekleri de yine sonsal olasılığı bir eşik değeriyle karşılaştırarak buluyorduk.

d_j modelini önceki modeller emin değilse kullanıyoruz:

$$y_i = d_{ji} \text{ eğer } w_j > \theta_j \text{ ve } \forall k < j, w_k < \theta_k \tag{17.9}$$

$j = 1$ değeriyle başlayarak asıl öğrenme kümesiyle d_j modelini eğitiriz. Sonra, ayrı bir geçerleme kümesinden d_j modelinin emin olmadığı tüm örnekleri ayırırız ve bunlar d_{j+1} modelinin öğrenme kümesini oluşturur. Burada AdaBoost algoritmasından fark, yalnızca yanlış sınıflandırılanları değil emin olunmayan örneklerin de ayrılmasıdır; böylece doğru sınıflandırılan ama sınıra fazla yakın olanları da (kenar payına düşenleri) sonraki aşamada yeniden öğreniyoruz.

Buradaki düşünce, örneklerin çoğunluğu için basit bir modelle erken (ve ucuz) sınıflandırma yapılabileceği ve pahalı modellerin küçük bir yüzde için gerekeceği ve böylece toplam maliyetin çok yükselmeyeceğidir. Oylama gibi koşut bir birleştirmede, her girdi için bütün modeller çıktı verdiğinden toplam maliyet çok yüksek olabilir. İstendiği kadar model karmaşıklığa göre sıralanmış olarak birleştirilebilir. En sonda,

bütün modeller tarafından reddedilen örneklerin sayısı az olacağından bunlar için genel bir kural öğrenmeye çalışmak yerine, en yakın k komşu algoritmasıyla oldukları gibi saklanmaları doğru olacaktır.

Ardışık sınıflandırıcılardaki varsayım, sınıfların artan karmaşıklıkta birkaç "kural" ve az sayıda "istisna" ile gösterilebileceğidir. Kuralları basit modellerle, örneğin, artan karmaşıklıktaki algılayıcılarla öğreniriz ve bu kurallar girdi uzayının her yerinde geçerlidir. İstisnalarsa yereldir ve bu yüzden, en yakın k komşu gibi yerel bir modelle öğrenilir.

Dolayısıyla ardışık sınıflandırıcı yaklaşımı, dağılıma bağlı olan ve olmayan kestirimin arasındadır. Doğrusal bir model dağılıma bağlıdır ve bütün örnekler için tek bir model öğrenir. En yakın k komşu algoritması ise hiçbir dağılım varsaymaz ve genel bir kural öğrenmeden tüm örnekleri olduğu gibi saklar. Ardışık sınıflandırmada kural, ya da kurallar, örneklerin çoğunu ucuz bir biçimde açıklar ve kalanlar istisnalar olarak tutulur. Böylesi bir varsayım çoğu uygulamada geçerlidir: Örneğin İngilizcede fiillerin geçmiş zaman çekiminde çoğunlukla fiilin köküne "–d" ya da "–ed" eklenir; "go"/"went" gibi istisnalar da vardır ama bunlar az sayıdadır. Türkçede sesli uyumu çoğunlukla geçerli bir kuraldır, ama az sayıda istisnası da vardır.

17.12 Kısa Kısa

Model birleştirmedeki düşünce, karmaşık bir işi ayrı ayrı eğitilmiş modeller arasında paylaştırmaktır. Her model bir altişi öğrenir. Tüm modelleri içerecek kadar büyük bir model veriyi ezberleyeceğinden iyi genelleme yapamaz. Örneğin, her biri bir saklı katmanlı üç tane çok katmanlı algılayıcıyı, iki saklı katmanlı bir algılayıcı olarak yeniden yazarsak bu modelin ezberleme olasılığı çok yüksektir. Üç modeli ayrı ayrı eğitmek, büyük modelin ikinci saklı katmanındaki birimler için istenen çıktı tanımlamamıza karşılık gelir. Modele bu biçimde ek kısıtlarla bir yapı eklemenin (ya da bir başka yorumla, ipuçları tanımlamanın) öğrenmeyi kolaylaştırdığını daha önceden görmüştük.

Model birleştirmeyle ilgili bir çekince, birleştirilen modelin yorumlanamaz olmasıdır. Örneğin, karar ağaçları yorumlanabilir ama AdaBoost ile oluşturulan "karar ormanı" kurallar olarak yazılamaz. Hata düzelten çıktı gizyazıları, kullandıkları $-1/0/+1$ ağırlıkları sayesinde biraz olsun yorumlanabilir. Mayoraz ve Moreira (1997), hata düzelten çıktı gizyazıları için gerektiğinde gizyazı dizeyine yeni sütunların eklendiği artımlı yöntemler önerir. Allwein, Schapire, ve Singer (2000), çok sınıflı problemleri iki sınıflı problemlere çevirmek için değişik yöntemleri tartışır. Alpaydın ve Mayoraz (1999), modellerin doğrusal olduğu durumda, hata düzelten çıktı gizyazılarının çok katmanlı bir algılayıcı olarak yazılabildiğini ve böylesi bir yapıda, gizyazı dizeyiyle temel modellerin birlikte öğrenilebileceğini gösterir.

En yaygın kullanılan ve en anlaşılır birleştirme yöntemi oylamadır. Kittler ve ötekiler (1998), sabit kuralları değerlendirir ve farklı gösterimlerin birleştirildiği bir örnek uygulamayı tartışır: Amaç, cepheden yüz görüntüsü, profil görüntü ve sesten kişilerin kimliklerinin saptanmasıdır, ve oylamanın sonuç hatası gösterimlerin teker teker hatalarından küçüktür. Bir başka uygulamada, Alimoğlu ve Alpaydın (1997),

elle yazılmış sayıların tanınması için kalem hareketiyle sonuç görüntüyü birleştirmiştir: Gösterimler kendi başlarına yaklaşık yüzde 5 hata yaparken ikisinin birleştirilmesiyle hata yaklaşık yüzde 3'e düşmüştür. Ayrıca, önemli olanın birbirini tamamlayan gösterimlerin bulunması olduğu ve böylesi farklı gösterimlerin birleştirme için kullanılacak yönteme göre çok daha önemli olduğu sonucuna varılmıştır.

Farklı kiplerin birleştirilmesi *biyometride* kullanılır. Amaç, kişiyi farklı bilgi kaynaklarından gelen özellikleri birleştirerek tanımaktır. Bunlar, parmak izi, imza, yüz görüntüsü, vs. olabilir. Farklı kipler, farklı sınıflandırıcılar tarafından tanındıktan sonra kararları birleştirilir. Bu hem başarıyı artırır, hem de tanıyıcı sistemin aldatılmasını zorlaştırır.

Noble (2004), farklı gösterimlerin ya da kiplerin birleştirilmesinde üç yaklaşım arasındaki farkı ortaya koymuştur:

- *Erken birleştirmede* bütün girdiler yan yana konarak tek bir girdi oluşturulur ve bu tek bir modele verilir. Bunun neden her zaman iyi bir fikir olmayabileceğini daha önce tartışmıştık.

- *Geç birleştirmede*, bu bölümde uzun uzun açıkladığımız ve savunduğumuz gibi, farklı girdiler farklı modellere verilir ve sonra kararları, oylama, yığma, ya da başka bir yöntem kullanılarak birleştirilir.

- *Ara birleştirme* (Noble 2004), çekirdek makinelerinde gördüğümüz *çoklu çekirdek öğrenme* yöntemine karşılık gelir (bkz. bölüm 13.8). Tek bir sınıflandırıcı vardır ama farklı çekirdekler farklı girdileri işler. Erken birleştirmedeki gibi girdi uzayında, ya da geç birleştirmedeki gibi karar uzayında değil, çekirdeklerin taban uzaylarında birleştirme yapılır. Farklı kaynaklardan gelen veride farklı çekirdekler farklı benzerlik ölçütleri hesaplar ve sınıflandırıcı bunları birlikte kullanır.

L tane bağımlı uzman ancak $L' < L$ tane bağımsız uzman değerindedir (Jacobs 1995). Oylamayla Bayesçi yöntemlerin belirli koşullar altında eşdeğerliği gösterilmiştir (Jacobs 1995). 17.3. denklemdeki önsel dağılımlar da üstparametrelerce tanımlanan daha üst düzeydeki başka dağılımlardan örneklenirmiş gibi tanımlanabilir. Böyle tanımlanan modelde çözümsel sonuç olmayabilir ve yaklaşıklama ya da örnekleme gerekebilir. Bayesçi istatistik kuramındaki gelişmelere koşut olarak bu üst Bayesçi yöntemlerin gelecekte daha da sık kullanılabilmesini bekleriz.

Farklı modellerin birleştirilmesi, yapay öğrenmede, 1990'ların başından bu yana rağbet gören ve üzerinde çokça araştırma yapılan bir konudur. Kuncheva, sınıflandırıcı birleştirmenin farklı yönlerini ayrıntılı olarak tartıştığı kitabında (2004) (bizim bu bölümde hiç bahsetmediğimiz) öbekleme algoritmalarının birleştirilmesi konusunu da anlatır.

AdaBoost ile birleştirilmiş karar ağaçları, en hızlı ve başarılı yapay öğrenme yöntemlerinden biri olarak kullanılmaktadır. AdaBoost algoritmasının, her modelin öncekinden kalan hata üzerinde eğitildiği bir sürümü de vardır (Hastie, Tibshirani, ve Friedman 2001). Farklı modellerin birleştirilmesinin her zaman başarıyı artırmadığı

son zamanlarda gittikçe daha çok görülmüş ve birleştirilecek bir model kümesinin sağlaması gereken özellikler daha kuramsal olarak araştırılmaya başlanmıştır. Bu bağlamda çeşitlilik (ne olduğu ve nasıl sağlanabileceği) ilk öne çıkan isterlerden biridir (Kuncheva 2005).

17.13 Alıştırmalar

1. Eğer her model bağımsız ve özdeş dağılmış ve doğru çıktı verme olasılığı 1/2'den yüksekse, L sınıflandırıcı üzerinden coğunluk oyuyla verilen kararın doğru olma olasılığı ne kadardır?
2. Yeniden örnekleme ve oylamada L tane farklı öğrenme kümesini oluştururken L kat çapraz geçerleme kullansak ne fark eder?
3. Hata düzelten çıktı gizyazılarında, gerektiğinde yeni sütünların eklenebildiği artımlı bir algoritma önerin.
4. Uzmanların birleşiminde uzmanlar farklı gösterimleri de kullanabilir. Bu durumda geçit modeli nasıl olabilir?
5. Devingen bir bağlanım modeli seçme algoritması önerin.
6. Birleştiricinin doğrusal olduğu bir yığılmış genelleyiciyle oylama arasındaki fark nedir?
7. Ardışık sınıflandırıcılarda neden $\theta_{j+1} \geq \theta_j$ koşulu vardır?
8. Ardışık sınıflandırıcıları bağlanımda kullanabilmek için bir bağlanım modelinin deneme sırasında verdiği çıktıdan ne kadar emin olduğunu söyleyebilmesi gerekir. Bu nasıl yapılabilir?
9. Farklı öbekleme sonuçlarını nasıl birleştirebiliriz?
10. Bölüm 17.10'da yığılmış genelleyicide birleştirme için karar ağacı kullandığımızda hem seçme hem de birleştirme yaptığını yazdık. Böylesi bir yaklaşımın hoş ve nahoş yanlarını yazın.

17.14 Kaynaklar

Alimoğlu, F., E. Alpaydın. 1997. "Combining Multiple Representations and Classifiers for Pen-Based Handwritten Digit Recognition." *Fourth International Conference on Document Analysis and Recognition*, 637–640. Los Alamitos, CA: IEEE Computer Society.

Allwein, E. L., R. E. Schapire, Y. Singer. 2000. "Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers." *Journal of Machine Learning Research* 1: 113–141.

Alpaydın, E. 1997. "Voting over Multiple Condensed Nearest Neighbors." *Artificial Intelligence Review* 11: 115–132.

Alpaydın, E., E. Mayoraz. 1999. "Learning Error-Correcting Output Codes from Data." *Ninth International Conference on Artificial Neural Networks*, 743–748. London: IEE Press.

Avnimelech, R., N. Intrator. 1997. "Boosting Regression Estimators." *Neural Computation* 11: 499–520.

Breiman, L. 1996. "Bagging Predictors." *Machine Learning* 26: 123–140.

Caruana, R., A. Niculescu-Mizil, G. Crew, A. Ksikes. 2004. "Ensemble Selection from Libraries of Models." *Twenty-First International Conference on Machine Learning*, ed. C. E. Brodley, 137–144. New York: ACM.

Demir, C., E. Alpaydın. 2005. "Cost-Conscious Classifier Ensembles." *Pattern Recognition Letters* 26: 2206–2214.

Dietterich, T. G., G. Bakiri. 1995. "Solving Multiclass Learning Problems via Error-Correcting Output Codes." *Journal of Artificial Intelligence Research* 2: 263–286.

Drucker, H. 1997. "Improving Regressors using Boosting Techniques." *Fourteenth International Conference on Machine Learning*, ed. D. H. Fisher, 107–115. San Mateo, CA: Morgan Kaufmann.

Drucker, H., C. Cortes, L. D. Jackel, Y. Le Cun, V. Vapnik. 1994. "Boosting and Other Ensemble Methods." *Neural Computation* 6: 1289–1301.

Freund, Y., R. E. Schapire. 1996. "Experiments with a New Boosting Algorithm." *Thirteenth International Conference on Machine Learning*, ed. L. Saitta, 148–156. San Mateo, CA: Morgan Kaufmann.

Hansen, L. K., P. Salamon. 1990. "Neural Network Ensembles." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12: 993–1001.

Hastie, T., R. Tibshirani, J. Friedman. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer.

Ho, T. K. 1998. "The Random Subspace Method for Constructing Decision Forests." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20: 832–844.

Jacobs, R. A. 1995. "Methods for Combining Experts' Probability Assessments." *Neural Computation* 7: 867–888.

Jacobs, R. A. 1997. "Bias/Variance Analyses for Mixtures-of-Experts Architectures." *Neural Computation* 9: 369–383.

Jain, A., K. Nandakumar, A. Ross. 2005. "Score Normalization in Multimodal Biometric Systems." *Pattern Recognition* 38: 2270–2285.

Kaynak, C., E. Alpaydın. 2000. "MultiStage Cascading of Multiple Classifiers: One Man's Noise is Another Man's Data." *Seventeenth International Conference on Machine Learning*, ed. P. Langley, 455–462. San Francisco: Morgan Kaufmann.

Kittler, J., M. Hatef, R. P. W. Duin, J. Matas. 1998. "On Combining Classifiers." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20: 226–239.

Kuncheva, L. I. 2004. *Combining Pattern Classifiers: Methods and Algorithms*. Hoboken, NJ: Wiley.

Kuncheva, L. I. 2005. Special issue on Diversity in Multiple Classifier Systems. *Information Fusion* 6: 1–115.

Mayoraz, E., M. Moreira. 1997. "On the Decomposition of Polychotomies into Dichotomies." *Fourteenth International Conference on Machine Learning*, ed. D. H. Fisher, 219–226. San Mateo, CA: Morgan Kaufmann.

Merz, C. J. 1999. "Using Correspondence Analysis to Combine Classifiers." *Machine Learning* 36: 33–58.

Noble, W. S. 2004. "Support Vector Machine Applications in Computational Biology." *Kernel Methods in Computational Biology*, ed. B. Schölkopf, K. Tsuda, J.-P. Vert, 71–92. Cambridge, MA: MIT Press.

Perrone, M. P. 1993. "Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure." Ph.D. thesis, Brown University.

Ruta, D., B. Gabrys. 2005. "Classifier Selection for Majority Voting." *Information Fusion* 6: 63–81.

Schapire, R. E. 1990. "The Strength of Weak Learnability." *Machine Learning* 5: 197–227.

Schapire, R. E., Y. Freund, P. Bartlett, W. S. Lee. 1998. "Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods." *Annals of Statistics* 26: 1651–1686.

Ulaş, A., M. Semerci, O. T. Yıldız, E. Alpaydın. 2009. "Incremental Construction of Classifier and Discriminant Ensembles." *Information Sciences* 179: 1298–1318.

Wolpert, D. H. 1992. "Stacked Generalization." *Neural Networks* 5: 241–259.

Woods, K., W. P. Kegelmeyer Jr., K. Bowyer. 1997. "Combination of Multiple Classifiers Using Local Accuracy Estimates." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19: 405–410.

Bölüm 18

Pekiştirmeli Öğrenme

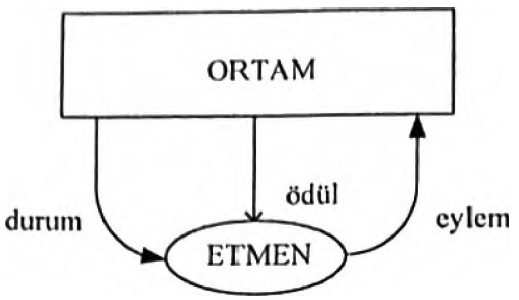
Pekiştirmeli öğrenmede, ortam içinde belli bir politikaya göre eylemler yapan ve eylemlerin çözüme etkisine bağlı olarak ödül (ya da ceza) alan bir etmen eğitiriz. Amaç, denemeler yaparak en yüksek ödülü getiren politikayı öğrenmektir.

18.1 Giriş

Satranç oynamayı öğrenecek bir program yazmak istediğimizi düşünelim. Bu durumda iki nedenden dolayı gözetimli bir öğrenme algoritması kullanamayız: Öncelikle, bizi birçok oyun boyunca gözetip her durumda en iyi hamleyi söyleyecek bir gözetmen çok masraflıdır. Ayrıca, çoğu zaman en iyi hamle diye bir şey yoktur; bir hamlenin iyiliği onu izleyen öteki hamlelere bağlıdır. Tek hamlenin bir anlamı yoktur; bir hamle dizisi, eğer onları sırayla oynadığımızda bize oyunu kazandırıyorsa iyidir. Alabileceğimiz tek geri besleme, en sonda oyunu kazanmamız ya da kaybetmemizdir.

Bir başka örnek, bir labirent içine yerleştirilmiş robottur. Dört ana yönde gidebilir ve çıkışı bulana dek hareket eder. Robot, labirentin içindeyken herhangi bir geri besleme almaz ve çıkışı bulana dek birçok hareket dener; çıkışı bulabildiğinde ancak ödül alır. Bu durumda bir rakip yoktur ama çıkışı daha çabuk bulmak istediğimizde zamana karşı oynuyor oluruz.

Bu iki uygulamada ortak noktalar vardır: *Etmen* dediğimiz bir karar verici, bir *ortam* içindedir (bkz. şekil 18.1). Satrançta, etmen oyunu oynar, rakip ve satranç tahtası ortamdır; ikinci örnekte, robotun ortamı labirenttir. Herhangi bir anda ortamın belirli bir *durumu* vardır—örneğin, satranç tahtasında taşlar belirli bir biçimde yerleşmiştir, ya da robot labirentin belirli bir noktasındadır. Etmenin yapabileceği belirli *eylemler* vardır; bunlar, satrançta izin verilen hamlelere ya da robotun duvarlara çarpmadan yapabileceği hareketlere karşılık gelir. Bir eylem seçilip yapıldığında durum değişir. Çözüm, bir dizi eylem gerektirir. Çok seyrek olarak geri besleme alırız; *ödül* genelde tüm eylem dizisi yapıldıktan sonra gelir. Problemi ödül tanımlar



Şekil 18.1: Etmen ortamla etkileşimdedir. Ortamın belli bir durumunda etmen bir eylem yapar. Bu eylemle ortam değişir ve belki bir ödül gelir.

ve *öğrenen* bir etmen istiyorsak gereklidir. Etmenin öğrenmesi gereken "en iyi," yani, toplamda en fazla ödül getiren eylem dizisini üretmektir. İşte bunlar bir pekiştirmeli öğrenme senaryosunun parçalarıdır.

Pekiştirmeli öğrenme, daha önce gördüğümüz öğrenme algoritmalarından bazı noktalarda farklıdır: Gözetmen yerine "eleştirmenle öğrenme" denir. *Eleştirmen*, gözetmenden farklı olarak gelecekte ne yapılması gerektiğini söylemez, ancak geçmiş yaptığımız eylemimizin iyi olup olmadığını söyler; eleştirmen işin öncesinde bilgi vermez. Geri beslemesi seyrektir ve gelirse, geç gelir. Bu da *sorumluluk atamada* sorun yaratır. Birçok eylem yapıp ödül aldıktan sonra, geçmişteki eylemlerimizi birer birer inceleyip hangilerinin ödülü kazanmakta işe yaradığını bilmek isteriz ki bunları kaydedip ileride yineleyebilelim. Birazdan göreceğimiz gibi pekiştirmeli öğrenme algoritmasının yaptığı, ara durumların (ya da ara durumlardaki eylemlerin) iyiliğini, yani amaca ne kadar yakın olduklarını, gösteren iç bir değer öğrenmektir. Böyle bir iç ödül düzeneği öğrenildiğinde eylem bunu en yüksek yapacak biçimde eylemlerini seçebilir.

Çözüm, bir eylem *dizisidir*, ve bu bakış açısından 15. bölümde gördüğümüz Markov modellerini anımsatır. Gerçekten de etmeni modellerken Markov karar süreçlerini kullanacağız, ama Markov modellerinde modellediğimiz sinyali, örneğin konuşmayı, yaratan dış bir süreç vardı; buradaysa eylem dizisini yaratan etmenin kendisidir. Daha önce, gözlenebilir ve saklı Markov modelleri arasındaki farkı ele almıştık. Benzer biçimde kısmen gözlenebilir Markov karar süreçlerinde etmen ortamı eksiksiz bilmez ve alıcılarından görebildiği kadarıyla çıkarım yapması gerekir. Örneğin oda içinde hareket eden bir robot, odanın neresinde olduğunu ya da çıkışın nerede olduğunu bilmeyebilir ama üstündeki kamera çevresiyle ilgili kısıtlı bir görüntü sağlar.

18.2 Tek Durum: K Kollu Kumar Makinesi

Basit bir örnekle başlayalım. *K kollu kumar makinesinde*, kollardan rastgele birini seçip çekeriz; şanslıysak bu eylem sonucunda bir miktar para kazanırız. Amaç, en fazla para kazandıran kolu bulmaktır. Bunu bir sınıflandırma olarak görebiliriz: K koldan birini seçiyoruz. Gözetimli öğrenme yapıyor olsaydık gözetmen hangi

kolu seçmemiz gerektiğini söylerdi. Ama pekiştirmeli öğrenmede tek yapabileceğimiz farklı kolları deneyip en iyiyi bulmaya çalışmaktır. Bu oldukça basitleştirilmiş bir örnektir, çünkü yalnız tek bir durum, tek bir makine var, ve yalnızca eyleme karar vermemiz gerekiyor. Bunu basit bir örnek yapan başka bir özellikse ödülün eylem (kol çekme) sonunda hemen gelmesi ve gecikmemesi, böylece eylemin iyiliğini gecikmesiz görebilmemizdir.

$Q(a)$ değeriyle a eyleminin ne kadar iyi olduğunu gösterelim. Başta, tüm a için $Q(a) = 0$ olur. Sonra, a eylemini denediğimizde $r_a \geq 0$ olan bir ödül alırız. Ödüller gerekirciyse (rastgele değilse), a eylemini her seçtiğimizde aynı r_a kadar ödül alırız ve bu durumda $Q(a) = r_a$ olarak saklayabiliriz. Sonra bu bilgimizi kullanmak istersek, $Q(a) > 0$ sağlayan bir a bulduğumuzda sürekli onu seçip her çekişte r_a kazanabiliriz. Ama daha fazla ödüllü bir kol da olabilir ve o yüzden arama da yapmamız gerekir.

Farklı eylemleri seçip her a için $Q(a)$ değerlerini saklarız. Bilgimizi kullanmak istediğimizde değeri en yüksek eylemi seçeriz:

$$\text{Seç } a^* \text{ eğer } Q(a^*) = \max_a Q(a) \tag{18.1}$$

Ödüller gerekirci değil de rastgeleyse, aynı eylem için farklı denemelerde farklı ödüller alırız. Bu durumda ödül değerini, $p(r|a)$ olasılık dağılımdan çekilen bir rastsal değişken olarak modelleriz. $Q_t(a)$, t anında a eylemi için yaptığımız kestirim, t anından önce a eylemi için alınan ödüllerin ortalamasıdır. Çevrimiçi bir güncelleme yapılabilir:

$$Q_{t+1}(a) \leftarrow Q_t(a) + \eta[r_{t+1}(a) - Q_t(a)] \tag{18.2}$$

Burada $r_{t+1}(a)$, $(t+1)$ anında a eylemi için alınan ödülü gösterir. Denklem 18.2, daha önceki bölümlerde gördüğümüz *delta kuralıdır*: η (yakınsama için zaman içinde azaltılan) öğrenme çarpanı, r_{t+1} istenen çıktı, ve $Q_t(a)$ şu anki öngörüdür. $Q_{t+1}(a)$, a eylemi için $t+1$ anındaki *beklenen* değerdir ve t arttıkça $p(r|a)$ dağılımının ortalamasına yakınsar.

Genelde bir pekiştirmeli öğrenme uygulamasında, birden çok durum vardır. Bu, farklı ödül olasılıklı, $p(r|s_i, a_j)$, farklı kumar makinelerine karşılık gelir. Amacımız, s_i durumunda (makinesinde) a_j eylemini (kolunu) seçmenin beklenen ödülü olan $Q(s_i, a_j)$ değerlerini öğrenmektir. Ayrıca, eylemler yalnızca ödülleri değil sonraki durumları da belirler ve bir durumdan ötekine geçeriz. Üçüncü olarak ödüller geciktirilebilir ve gelecek ödülleri şimdiden kestirebilmemiz gerekir.

18.3 Pekiştirmeli Öğrenmenin Öğeleri

Öğrenen karar verici olan *etmen* dışındaki her şey *ortamı* oluşturur. Etmen, ortamın *durumunu* alıcılarıyla algılar ve bir *eyleme* karar verir. Etmen, eylemi yaptığımda farklı bir duruma geçer ve bazen ortam bir *ödül* döndürür. Zaman, $t = 0, 1, 2, \ldots$ biçiminde kesiklidir. $\mathcal{S}$, olası tüm durumların kümesini, $s_t \in \mathcal{S}$ ise etmenin t anındaki durumunu gösterir. $a_t \in \mathcal{A}(s_t)$, etmenin t anındaki eylemidir—$\mathcal{A}(s_t)$, s_t durumunda olası tüm eylemlerin kümesidir. Etmen, s_t durumunda a_t eylemini yaptığında saat

bir birim ilerler, $r_{t+1} \in \Re$ ödülü alınır ve etmen, bir sonraki s_{t+1} durumuna geçer. Bu işlem, *Markov karar süreci* (MKS) kullanarak modellenir. Ödül ve sonraki durum değerleri $p(r_{t+1}|s_t, a_t)$ ve $P(s_{t+1}|s_t, a_t)$ olasılık dağılımlarından örneklenir. Sonraki durum ve ödül yalnızca şu anki durum ve eyleme bağlı olduğu için bu bir *Markov* sürecidir. Bazı uygulamalarda ödül ve sonraki durum gerekircidir ve belli bir durum ve eylem için olası tek bir ödül ve izleyen durum vardır.

Uygulamaya bağlı olarak bazen belli bir durumu *ilk durum* olarak tanımlamış olabiliriz. Bazı uygulamalarda, aramanın sonuçlandığı tek bir *amaç durumu* vardır; bu son durumda tüm eylemler ödülsüz olarak kendine döner. İlk durumdan son duruma dek yapılan eylemlerin dizisine bir *serüven* denir.

π ile gösterdiğimiz *politika* etmenin davranışını tanımlar ve durumlardan eylemlere bir eşlemedir: $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Politika, s_t durumundaki eylemi belirleyen işlevdir: $a_t = \pi(s_t)$. π politikasının *değeri*, $V^\pi(s_t)$, s_t durumundan başlayarak o politikaya göre seçtiğimiz eylemlerin sonucu elimize geçen ödüllerin toplamıdır.

Sonlu serüven modelinde, etmen sonraki T adımda beklenen ödüller toplanır:

$$V^\pi(s_t) = E[r_{t+1} + r_{t+2} + \cdots + r_{t+T}] = E\left[\sum_{i=1}^{T} r_{t+i}\right] \tag{18.3}$$

Bazı işlerse hiç bitmez ve serüven için baştan bir sınır belirlenemez. *Sonsuz serüven* modelinde uzunluk sınırı yoktur ama gelecek ödüllerin etkisi indirimlidir:

$$V^\pi(s_t) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots] = E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right] \tag{18.4}$$

Burada, $0 \leq \gamma < 1$, *indirim oranıdır* ve toplamın sonlu olmasını sağlar. $\gamma = 0$ olduğunda yalnızca anlık ödül sayılır. γ, 1 değerine yaklaştıkça gelecekteki ödüller önem kazanmaya başlar ve etmenin uzak görüşlü olduğunu söyleriz. γ, 1 değerinden küçüktür çünkü işi sonlandırmak için her zaman bir zaman sınırı vardır; robot, pille çalışıyor olabilir. Ödülleri olabildiğince çabuk almak isteriz, çünkü daha ne kadar yaşayacağımızı bilmeyiz.

Her π politikası bir $V^\pi(s_t)$ tanımlar ve aralarından toplam beklenen ödülü en yüksek, *en iyi politikayı*, π^*, bulmak isteriz:

$$V^*(s_t) = \max_\pi V^\pi(s_t), \forall s_t \tag{18.5}$$

Denetim gibi bazı uygulamalarda, $V(s_t)$ ile gösterdiğimiz durum iyilik değerleri yerine $Q(s_t, a_t)$ ile gösterdiğimiz durum-eylem çifti iyilik değerleriyle çalışmak daha kolaydır. $V(s_t)$, etmenin s_t durumunda olmasının ne kadar iyi olduğunu gösterirken $Q(s_t, a_t)$, s_t durumunda a_t eylemini yapmasının ne kadar iyi olduğunu gösterir. $Q^*(s_t, a_t)$ ile s_t durumunda a_t eylemini yapıp ardından en iyi politikayla hareket ettiğimizde elimize geçen beklenen toplam ödülü gösterelim. Bir durumun iyiliği o durumdaki en iyi eylemin iyiliğine eşittir:

$$V^*(s_t) = \max_{a_t} Q^*(s_t, a_t)$$

$$
\begin{aligned}
&= \max_{a_t} E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right] \\
&= \max_{a_t} E\left[r_{t+1} + \gamma \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i+1}\right] \\
&= \max_{a_t} E\left[r_{t+1} + \gamma V^*(s_{t+1})\right] \\
V^*(s_t) &= \max_{a_t} \left(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) V^*(s_{t+1})\right) \qquad (18.6)
\end{aligned}
$$

Her olası sonraki s_{t+1} durumuna $P(s_{t+1}|s_t, a_t)$ olasılığıyla geçer ve o andan sona en iyi politikayı izleyip $V^*(s_{t+1})$ beklenen toplam ödülü kazanırız. Olası tüm sonraki durumlar üzerinden toplar ve geleceği bir adım sonrası olduğu için iskonto ederiz. Şu anki ödülü eklediğimizde a_t için toplam beklenen ödülü buluruz. Ardından olası eylemler içinden en iyisini seçeriz. Denklem 18.6, *Bellman denklemi* (Bellman 1957) olarak bilinir. Benzer biçimde, Bellman denklemini durum-eylem çifti değerlerini kullanarak da yazabiliriz:

$$Q^*(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \qquad (18.7)$$

Elimizde bir kez $Q^*(s_t, a_t)$ değerleri olursa, π politikamızı, olası $Q^*(s_t, a_t)$ içinde en iyi a_t^* eylemini seçmek olarak tanımlayabiliriz:

$$\pi^*(s_t) : \text{Seç } a_t^* \text{ öyle ki } Q^*(s_t, a_t^*) = \max_{a_t} Q^*(s_t, a_t) \qquad (18.8)$$

Bu da demektir ki eğer $Q^*(s_t, a_t)$ değerlerini bilirsek, her *yerel* adımda açgözlü bir arama yaparak *toplam* ödülü en fazla yapacak eylem dizisini bulabiliriz.

18.4 Model Tabanlı Öğrenme

Ortamı tanımlayan $p(r_{t+1}|s_t, a_t)$ ve $P(s_{t+1}|s_t, a_t)$ olasılık dağılımlarını bildiğimiz model tabanlı öğrenmeyle başlayalım. Böyle bir durumda arama yapmaya gerek kalmadan, en iyi değerleri ve politikayı devingen programlama kullanarak hesaplayabiliriz. En iyi değerler için tek çözüm vardır ve denklem 18.6'da verilen denklemlerin birlikte çözümüdür. Elimizde en iyi değerler olduğunda en iyi politika, sonraki durumun iyiliğini en iyi yapan eylemi seçmektir:

$$\pi^*(s_t) = \arg\max_{a_t} \left(E[r_{t+1}|s_t, a_t] + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s_t, a_t) V^*(s_t + 1)\right) \qquad (18.9)$$

```
Rastgele V(s) değerleriyle başla
Yinele
    Tüm s ∈ S
        Tüm a ∈ A
            Q(s,a) ← E[r|s,a] + γ Σ_{s'∈S} P(s'|s,a)V(s')
        V(s) ← max_a Q(s,a)
V(s) değerleri yakınsayana dek
```

Şekil 18.2: Model tabanlı öğrenme için değer yineleme algoritması.

18.4.1 Değer Yineleme

En iyi politika için en iyi değer işlevini kullanabiliriz, ve yinelemeli *değer yineleme* algoritmasının doğru V^* değerlerine yakınsadığı gösterilmiştir. Örnek programı şekil 18.2'de verilmiştir.

Eğer iki yineleme değeri arasındaki fark bir δ eşik değerinden küçükse yakınsama olduğunu söyleriz:

$$\max_{s \in \mathcal{S}} |V^{(l+1)}(s) - V^{(l)}(s)| < \delta$$

ki burada l, yineleme sayacıdır. Yalnızca doğru eylemin görece olarak en büyük değerli olması yettiği için politika, değerlerden daha önce yakınsayabilir. Her yineleme $\mathcal{O}(|\mathcal{S}|^2|\mathcal{A}|)$ karmaşıklığındadır ama genelde olası sonraki durum sayısı daha küçük $k < |\mathcal{S}|$ olduğu için karmaşıklık daha azdır ve $\mathcal{O}(k|\mathcal{S}||\mathcal{A}|)$ olur.

18.4.2 Politika Yineleme

Politika yinelemede, değerler yerine politikayı saklar ve güncelleriz. Örnek program 18.3'te verilmiştir. Rastgele bir politikayla başlar, değişiklik gerekmeyinceye dek güncelleriz. Değerler doğrusal denklemleri çözerek bulunabilir ve bu, politikayı iyileştirir; artık iyileştirme olmayınca eldeki politikanın en iyi olduğu gösterilmiştir. Algoritmanın her adımı $\mathcal{O}(|\mathcal{A}||\mathcal{S}|^2 + |\mathcal{S}|^3)$ karmaşıklığındadır; her ne kadar bu karmaşıklık yinelemeninkinden daha fazlaysa da politika yineleme daha az adımda yakınsar.

18.5 Zamansal Fark Öğrenme

Model, ödül ve sonraki durum olasılık dağılımlarıyla tanımlandığı için, 18.4. bölümde gördüğümüz gibi, bunları bildiğimizde devingen programlamayla en iyi politikayı hesaplayabiliriz. Ama bu yöntemler hem pahalıdır, hem de ortamla ilgili bilgimiz her zaman bunu yapabilecek kadar çok değildir. Pekiştirmeli öğrenmenin daha ilginç ve gerçekçi uygulamaları elimizde model olmadığında ortaya çıkar. Modeli öğrenmek için arama, yani denemeler, yapmamız gerekir. Önce böyle bir aramanın nasıl yapılacağını, ardından model varsaymayan öğrenme algoritmalarını gerekirci

Rastgele π' politikasıyla başla
Yinele
$\pi \leftarrow \pi'$
π politikasıyla doğrusal denklemleri
çözüp değerleri bul
$V^\pi(s) = E[r|s, \pi(s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^\pi(s')$
Tüm durumlar için politikayı iyileştir
$\pi'(s) \leftarrow \arg\max_a (E[r|s, a] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^\pi(s'))$
$\pi = \pi'$ oluncaya dek

Şekil 18.3: Model tabanlı öğrenme için politika yineleme algoritması.

ve rastgele ortamlar için göreceğiz. Her ne kadar modeli bildiğimizi varsaymazsak da sabit olduğunu ve biz öğrenirken değişmeyeceğini varsayacağız.

Az sonra göreceğimiz gibi, arama yaparak ortamı keşfedip sonraki durum ve ödülü görecek ve bu bilgiyi şu anki durumun değerini güncellemek için kullanacağız. Bu algoritmalara *zamansal fark* algoritmaları denmesinin nedeni de sonraki adımdaki kestirdiğimiz değerle şu anki durumun değeri arasındaki farkı kullanmalarıdır.

18.5.1 Arama Yordamları

Arama yaparken bir olasılık, ϵ *açgözlü* davranmaktır; burada, ϵ olasılıkla olası eylemlerden eşit olasılıkla birini seçer, yani arama yapar, $1 - \epsilon$ olasılıkla da en iyi eylemi seçer, yani öğrendiğimiz bilgiyi kullanırız. Hem arama yapıp olası daha iyi eylemleri keşfetmek isteriz, hem de o ana dek öğrendiğimiz iyi eylemlerden faydalanıp ödül toplamak isteriz ve bunun için yüksek bir ϵ değeriyle başlar, yavaşça azaltırız. Politikamızın her zaman *eşiksiz* olmasını, yani herhangi bir durumda herhangi bir eylemin olasılığının sıfırdan büyük olmasını isteriz.

Eşiksiz en büyük işleviyle değerleri olasılıklara çevirebiliriz:

$$P(a|s) = \frac{\exp Q(s, a)}{\sum_{b \in \mathcal{A}} \exp Q(s, b)} \tag{18.10}$$

ve ardından bu olasılıklara göre örnekleme yaparak eylem seçeriz. Aramadan bilgi kullanmaya yavaşça geçiş için bir T "sıcaklık" değişkeni eklenip olasılıklar buna bağlı hesaplanabilir:

$$P(a|s) = \frac{\exp[Q(s, a)/T]}{\sum_{b \in \mathcal{A}} \exp[Q(s, b)/T]} \tag{18.11}$$

T büyük olduğunda tüm olasılıklar aynı olur ve arama yaparız. T azaldıkça daha iyi eylemler yeğlenir; *tavlama* denilen böylesi yordamda büyük bir T ile başlayıp yavaşça azaltır ve zaman içinde aramadan kullanmaya geçeriz.

18.5.2 Gerekirci Ödül ve Eylemler

Modelden bağımsız öğrenmede önce daha basit olan gerekirci ortamı göreceğiz. Her durumda her eylem için olası tek bir ödül değeri ve tek bir sonraki durum vardır, ve denklem 18.7 basitleşir:

$$Q(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{18.12}$$

Öğrenmede bu denklemi, $Q(s_t, a_t)$ değeri için bir atama olarak kullanıyoruz. s_t durumundayken yukarıda gördüğümüz arama yordamlarının biriyle bir eylem a_t seçtiğimizde elimize r_{t+1} ödülü geçer ve sonraki s_{t+1} durumuna atlarız. Ardından bu ödülü ve sonraki beklenen değeri (indirim yapıp) kullanarak *önceki* değeri belirleriz:

$$\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) \tag{18.13}$$

Burada değişkenin üstündeki şapka ile değerin kestirildiğini gösterir. $\hat{Q}(s_{t+1}, a_{t+1})$, daha sonraki (sonuca daha yakın) bir değer olduğu için doğru olma olasılığı daha yüksektir. Bunun üzerinde γ kadar bir indirim yapar, (eğer varsa) o anlık ödülü ekler ve önceki $\hat{Q}(s_t, a_t)$ için kestirim olarak alırız. Sonraki adımdaki eylemin iyiliğini *geriye taşıyıp* şu anki eylem için bir iyilik değeri kestiririz.

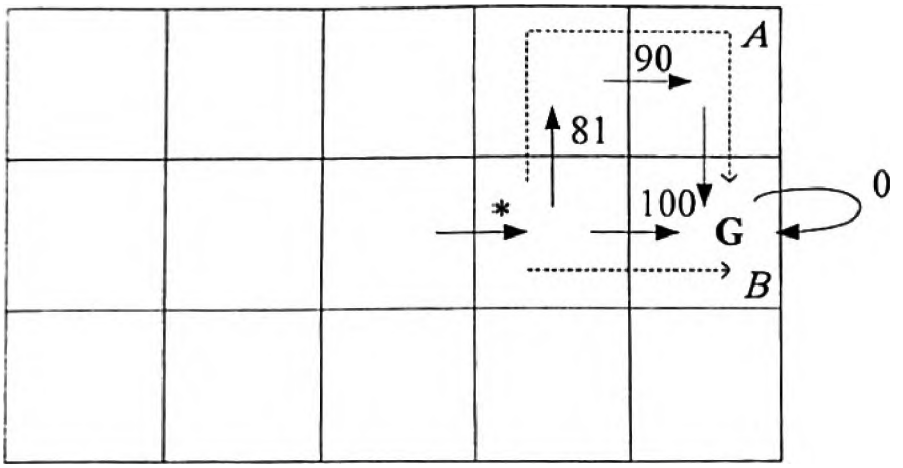
Şimdilik tüm $\hat{Q}(s, a)$ değerlerinin bellekte bir dizeyde saklandığını varsayalım; az sonra $|\mathcal{S}|$ ve $|\mathcal{A}|$ büyük olduğunda bu değerleri nasıl gösterebileceğimizi ele alacağız.

İlk başta tüm $\hat{Q}(s_t, a_t)$ değerleri sıfırdan başlar ve deneme serüvenleri yaşadıkça bu değerler güncellenir. Bir dizi eylem yaptığımızı ve her eylemde, denklem 18.13'ü kullanarak sonraki durum-eylem çiftinin Q değerini kullanarak o anki durum-eylem çiftinin Q değerini güncellediğimizi düşünelim. Ara durumlarda tüm ödüller ve dolayısıyla tüm değerler 0 olacak, ve değerler değişmeyecektir. Amaç duruma geldiğimizde serüven biter, r ödülü elimize geçer ve bir önceki eylem-durum çiftinin Q değeri γr olur. Ardından başka bir serüvende, bu durumdan bir önceki duruma gelirsek, o geçişin değeri de bir adım daha önce olduğu için bir kez daha γ ile çarpılır. Bu biçimde, birçok serüven sonra, bu değerler daha önceki durum-eylem çiftlerine taşınır. Amaca giden daha kısa yollar buldukça Q değerleri artıp en iyi değerlerine yaklaşır, ama hiçbir zaman azalmaz (bkz. şekil 18.4).

Değerleri öğrenirken ödül ve sonraki durum işlevlerini bilmiyoruz. Bu işlevler ortamın bir parçası, ve arama yaparken onlardan değerler alıyoruz yalnızca. Doğrudan bu işlevleri de modellemiyoruz, ki aslında o da yapılabilir; onlar veriliyken durum ya da durum-eylem çifti değerleri kestirip doğrudan en iyi politikayı öğreniyoruz.

18.5.3 Rastgele Ödül ve Eylemler

Eğer ödüller ve eylemlerin sonucu gerekirci değilse ödül değerleri ve sonraki durumlar da bir işlevden değil, sırasıyla, $p(r_{t+1}|s_t, a_t)$ ve $P(s_{t+1}|s_t, a_t)$ olasılık dağılımlarından geliyordur. Bu dağılımlar, ortamda denetimimiz altında olmayan etkenlerin sonucu oluşan bilinmezliği modellememizi sağlar. Örneğin, satrançta bir rakip, tavlada üstüne



Şekil 18.4: Q değerleri artabilir ama azalmaz. Gerekirci bir dünyada olalım. G amaç hücresine ulaşmanın 100, öteki tüm hücreler için ödül 0, ve $\gamma = 0.9$ olsun. Yıldız ile gösterilen geçişin Q değerini düşünelim, ve A ile B yollarına odaklanalım. A yolundan B'den önce geçersek $\gamma \max(0, 81) = 72.9$ olur ve ardından başka bir serüvende B'den geçtiğimizde daha kısa bir yol olduğu için Q değeri $\gamma \max(100, 81) = 90$ olur. Eğer B yolundan, A'dan önce geçilirse Q değeri $\gamma \max(100, 0) = 90$ olur ve ardından A'dan geçilse bile değer değişmez, çünkü $\gamma \max(100, 81) = 90$ aynı kalır.

üstlük bir de zar vardır. Ya da hareket ederken (rüzgar, eğim, sürtünme, aksamındaki aşınma, vs. yüzünden) yönünden sapan ve istenenden daha kısa ya da uzun giden kusurlu bir robotumuz olabilir.

Rastgele durumda hem ödül, hem de sonraki durum için olası değerler üzerinden ortalama almak gerekir:

$$Q(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{18.14}$$

Öğrenirken doğrudan atama yapamayız çünkü aynı durum-eylem çifti için farklı serüvenlerde farklı ödüller ve sonraki durumlar gerçekleşebilir. Bunun yerine zaman içinde ortalama alırız. Buna Q *öğrenme* algoritması denir:

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \eta(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{18.15}$$

$r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$ değerini bir (s_t, a_t) çifti için bir örneklem olarak düşündüğümüzde istediğimiz, $\hat{Q}(s_t, a_t)$ değerinin bunun ortalamasına yakınsamasıdır. Hep yaptığımız gibi yakınsama için η değeri yavaşça azaltılır. Bu algoritmanın en iyi Q^* değerlerine yakınsadığı gösterilmiştir (Watkins ve Dayan 1992). Q öğrenme algoritması için örnek program şekil 18.5'te verilmiştir.

Denklem 18.15'i, şu anki Q değeriyle bir sonraki andan geriye taşınmış değer arasındaki farkı azaltıyor olarak da yorumlayabiliriz. Böylesi algoritmalara *zamansal fark* algoritmaları denir (Sutton 1988).

$Q(s,a)$ değerlerini rastgele başlat
Tüm serüvenler için
    s ilk durum olsun
    Yinele
        Q'ya bağlı bir politika ve (örneğin, ϵ açgözlü) aramayla, a eylemini seç
        a eylemini yap, r ve s' değerlerini gözlemle
        $Q(s,a)$değerini güncelle:
            $Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
        $s \leftarrow s'$
    s amaç durum olana dek

Şekil 18.5: Q öğrenme, politikasız bir zamansal fark algoritmasıdır.

Burada gördüğümüz *politikasız* bir yöntemdir çünkü bir sonraki durum için en iyi değeri kullanır, orada politikayla bir seçme yapmayız. *Politikalı* yöntemde, sonraki durumu seçerken de politikayı kullanırız. Q öğrenmenin politikalı sürümü olan *Sarsa* algoritmasının örnek programı şekil 18.6'da verilmiştir. Olası tüm sonraki a' eylemlerine bakıp en iyisini seçmek yerine politikalı Sarsa, Q değerlerine bağlı bir politikayla arama yapıp sonraki a' eylemini seçer ve gördüğü Q değerini zamansal farkı hesaplamakta kullanır. Politikalı yöntemler, politikayı eylemleri seçerken kullanırken bir yandan da değerlerini öğrenir. Politikasız yöntemlerde bu ikisi ayrılmıştır ve eylemleri belirleyen *davranış* politikası, değerlenen ve güncellenen *kestirim* politikasından farklı olabilir.

Eğer (1) tüm durum-eylem çiftlerinden sonsuz kez geçilir, ve (2) politika sonsuzda açgözlü bir politikaya yakınsarsa (örneğin, $\epsilon = 1/t$ olan ϵ açgöz aramayla) Sarsa, 1 olasılıkla en iyi politika ve en iyi durum-eylem değerlerine yakınsar.

Aynı zamansal fark düşüncesi, $Q(s,a)$ yerine $V(s)$ değerlerini öğrenmek için de kullanılabilir. *TD öğrenme* (Sutton 1988) algoritmasında durum değerleri geriye doğru taşınır:

$$V(s_t) \leftarrow V(s_t) + \eta[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{18.16}$$

$r_{t+1} + \gamma V(s_{t+1})$ değerini sonraki ve daha doğru öngörü, $V(s_t)$ değerini de şu anki kestirim olarak düşünürsek bu da delta kuralıdır. Farkları zamansal farktır ve bu farkı azaltacak biçimde güncelleme yapılır. η öğrenme çarpanı zamanla azaltıldığında TD algoritmasının en iyi $V^*(s)$ değerlerine yakınsadığı gösterilmiştir.

18.5.4 Uygunluk İzleri

Şimdiye dek gördüğümüz algoritmalar hep tek adımlıdır; zamansal fark yalnızca bir önceki (durum ya da durum-eylem çifti için) değeri güncellemekte kullanılır. *Uygunluk izi*, geçmişte geçtiğimiz yerleri akılda tutup onları da güncellememizi ve zaman içinde sorumluluk atayabilmemizi sağlar. Bunun Q değerlerini öğrenirken Sarsa ile nasıl yapıldığını göreceğiz; V değerleri için de kolayca uyarlanabilir.

$Q(s,a)$ değerlerini rastgele başlat
Tüm serüvenler için
    s ilk durum olsun
    Yinele
        Q'ya bağlı bir politika ve (örneğin, ϵ açgözlü) aramayla, a eylemini seç
        a eylemini yap, r ve s' değerlerini gözlemle
        Q'ya bağlı bir politika ve (örneğin, ϵ açgözlü) aramayla, a' eylemini seç
        $Q(s,a)$ değerini güncelle:
            $$Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma Q(s',a') - Q(s,a))$$
        $s \leftarrow s'$, $a \leftarrow a'$
    s amaç durum olana dek

Şekil 18.6: Sarsa algoritması, Q algoritmasının politikalı sürümüdür.

Uygunluk izini saklamak için her durum-eylem çifti için ek bir $e(s,a)$ belleği ayırıp 0 ile başlatacağız. Bir serüvende (s,a) durum-eylem çiftinden geçildiğinde, yani s durumunda a eylemi seçildiğinde, uygunluk değerine 1 atanır ve tüm öteki durum-eylem çiftlerinin uygunlukları da $\gamma\lambda$ ile çarpılır: $0 \leq \lambda \leq 1$, iz sönüm parametresidir:

$$e_t(s,a) = \begin{cases} 1 & \text{eğer } s = s_t \text{ ve } a = a_t, \\ \gamma\lambda e_{t-1}(s,a) & \text{değilse} \end{cases} \tag{18.17}$$

Bir durum-eylem çiftinden hiç geçilmezse uygunluğu 0 olarak kalır; geçilirse zaman geçip sonrasında öteki durum-eylem çiftlerinden geçildikçe uygunluğu γ ve λ değerlerine bağlı olarak söner (bkz. şekil 18.7).

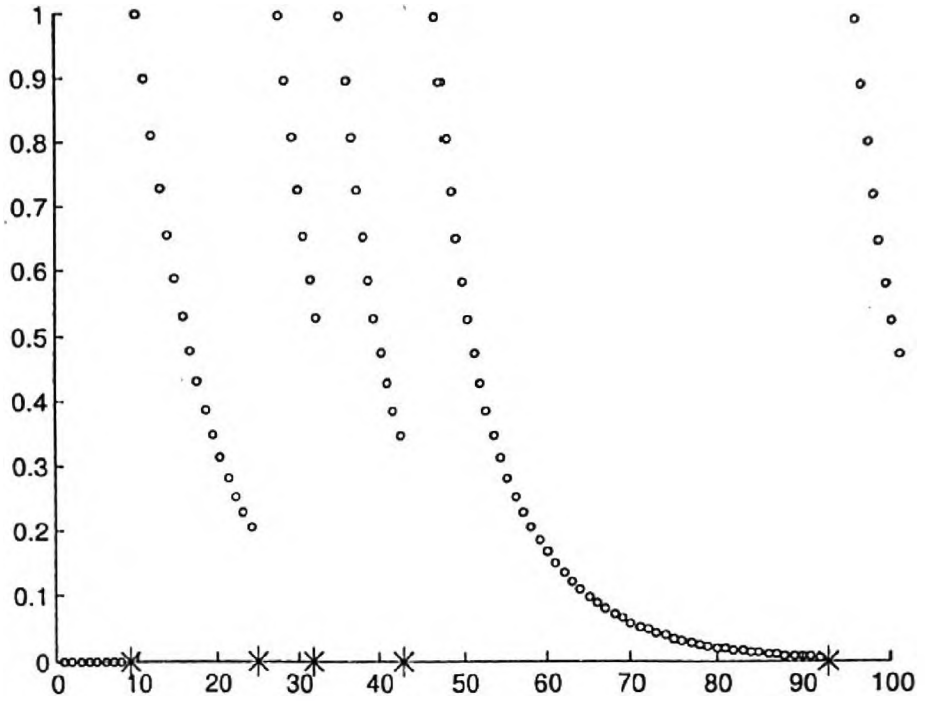
Sarsa'da t anındaki zamansal farkı yazalım:

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{18.18}$$

Sarsa(λ), Sarsa'nın uygunluk izi kullanan sürümüdür ve herhangi bir geçişten sonra *tüm* durum-eylem çift değerleri güncellenir:

$$Q(s,a) \leftarrow Q(s,a) + \eta\delta_t e_t(s,a), \ \forall s,a \tag{18.19}$$

Durum-eylem çiftleri güncellenirken güncelleme ne kadar geçmişte geçildiklerine bağlıdır. λ, zaman içindeki sorumluluğu belirler: $\lambda = 0$ olduğunda yalnızca tek adım güncellenir ve 18.5.3. bölümde gördüğümüz algoritmalar bu yüzden, $Q(0)$, Sarsa(0), ya da TD(0) olarak bilinir. λ değeri bire yaklaştıkça önceki adımlar daha fazla göz önüne alınır. $\lambda = 1$ olduğunda önceki tüm adımlar güncellenir ve sorumlulukları adım başına yalnızca γ kadar azalır. Çevrimiçi güncellemede her adımdan sonra tüm uygun değerler hemen güncellenir; çevrimdışı güncellemede güncellemeler biriktirilip serüven sonunda tek bir güncelleme yapılır. Çevrimiçi güncelleme daha yavaştır ama daha hızlı yakınsar. *Sarsa*(λ) için örnek program şekil 18.8'de verilmiştir. $Q(\lambda)$ ve TD(λ) algoritmaları benzer mantıkla türetilebilir (Sutton ve Barto 1998).



Şekil 18.7: Uygunluk izi örneği. Yıldızlar geçiş zamanlarını gösterir.

$Q(s,a)$ değerlerini rastgele başlat, $e(s,a) \leftarrow 0, \forall s,a$
Tüm serüvenler için
  s ilk durum olsun
  Q'ya bağlı bir politika ve (örneğin, ϵ açgözlü) aramayla, a eylemini seç
  Yinele
    a eylemini yap, r ve s' değerlerini gözlemle
    Q'ya bağlı bir politika ve (örneğin, ϵ açgözlü) aramayla, a' eylemini seç
    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) \leftarrow 1$
    Tüm s,a
      $Q(s,a) \leftarrow Q(s,a) + \eta \delta e(s,a)$
      $e(s,a) \leftarrow \gamma \lambda e(s,a)$
    $s \leftarrow s'$, $a \leftarrow a'$
  s amaç durum olana dek

Şekil 18.8: Sarsa(λ) algoritması.

18.6 Genelleme

Şu ana dek $Q(s,a)$ (ya da $V(s)$) değerlerini bellekte bir dizey olarak sakladığımızı varsaydık. Bu yaklaşımın bazı sakıncaları vardır: (1) durum ve/ya da eylem sayısı çok olduğunda dizey çok büyük olabilir; (2) durum ve/ya da eylem, kesikli değil sürekli olursa, örneğin, kuzey/güney/doğu/batı yerine direksiyon belli bir açıyla döndürülmek istenirse dizey kullanabilmek için kesikleştirme yapılması gerekir ve bu ek hataya neden olur; ve (3) arama uzayı büyük olduğunda olası tüm değerleri kabul edilebilir kesinlikle doldurabilmek için çok fazla serüven yaşamak gerekir.

Q değerlerini olduğu gibi saklamak yerine bunu bir bağlanım uygulaması olarak görebiliriz. $Q(s,a|\theta)$ bağlanım modeli, s ve a değerlerini girdi olarak alır ve θ parametrelerine bağlı olarak Q değerlerini öğrenir. Örneğin modelimiz, tek çıktılı, s ve a değerlerinin girdi ve θ parametrelerinin bağlantı ağırlıkları olduğu bir yapay sinir ağı olabilir.

İyi bir bağlanım, öğrenme örneklerini saklamak yerine bbasit bir modelle iyi bir yaklaşıklama sağlar. Girdileri sürekli olabilir ve genelleme yapar: Eğer benzer (s,a) çiftlerinin Q değerleri de benzerse, geçmiş örneklerden genelleme yaparak daha önce hiç görülmemiş durum-eylem çiftleri için iyi $Q(s,a)$ değerleri kestirebiliriz.

Bağlanım modelini eğitmek için bir öğrenme kümesi gerekir. Sarsa(0) algoritmasında, $Q(s_t,a_t)$ değerinin $r_{t+1}+\gamma Q(s_{t+1},a_{t+1})$ değerine yakın olmasını istediğimizi görmüştük. Dolayısıyla bu biçimde serüven sırasında, girdinin (s_t,a_t) ve istenen çıktının $r_{t+1}+\gamma Q(s_{t+1},a_{t+1})$ olduğu öğrenme örnekleri oluşturabiliriz. Kare hatayı yazalım:

$$E^t(\theta) = [r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]^2 \tag{18.20}$$

$Q(0)$ ya da TD(0) için de benzer biçimde öğrenme kümeleri oluşturabiliriz—$V(s)$ değerlerini kestirmek istediğimizde çıktı $r_{t+1} - \gamma V(s_{t+1})$ olur. Elimizde böyle bir küme olduğunda model oluşturmak için herhangi bir gözetimli bağlanım algoritması kullanabiliriz.

Yapay sinir ağlarında olduğu gibi eğer eğim iniş kullanıyorsak parametre yöneyini adım adım güncelleriz:

$$\Delta\theta = \eta[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]\nabla_{\theta_t} Q(s_t,a_t) \tag{18.21}$$

Bu, tek adımlı bir güncellemedir. Sarsa(λ) kullandığımızda uygunluk izi de kullanılır:

$$\Delta\theta = \eta\delta_t e_t \tag{18.22}$$

Burada, zamansal fark

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)$$

kullanılır ve uygunluk izi yöneyi her adımda güncellenir:

$$e_t = \gamma\lambda e_{t-1} + \nabla_{\theta_t} Q(s_t,a_t) \tag{18.23}$$

ilangıçta $\mathbf{e}_0$ değerlerinin hepsi 0 alınır. Değerlerin bir dizey olarak bellekte sakdığı bir algoritmada, uygunluklar da her durum-eylem çifti için başka bir dizeyde lanır. Bağlanımla kestirdiğimizdeyse uygunluk artık kestiricinin parametreleri ı tanımlıdır. Bu, hata geri yayma algoritmasında kullanılan momentum kavramına ilüm 11.8.1) çok benzer, şu farkla ki, momentumda önceki ağırlık değişiklikleri sakırken burada önceki eğim yöneyi saklanmaktadır. Kullanılan $Q(s_t, a_t)$ modeline :e, ki bu bir yapay sinir ağı da olabilir, hesaplanan eğim yöneyi, denklem 18.23'te :ine konur.

Her ne kadar Q değerlerini kestirmek için herhangi bir model kullanılabilse de bu el uygulamanın kendine özgü koşulları vardır. Öncelikle genellemeye izin vermeli, ınun için de benzer durum ve eylemlerin benzer Q değerleri olmaldır. Bu, s ve a için rdi gösterimini öne çıkarır; girdiler, çıktıdaki benzerlikleri ortaya çıkaracak biçimde isterilmelidir. İkinci olarak, serüven sırasında öğrenme örnekleri birer birer ortaya ktığı için öğrenme algoritmasının bu örneklerle birer birer, eski öğrendiklerini unutıadan güncelleme yapabilmesi gerekir. Örneğin, çok katmanlı bir algılayıcı kulınılıyorsa öğrenme çarpanının küçük olması gerekir. Ya da, böylesi örnekler topınıp hep birlikte öğrenilebilir, ama bu öğrenmeyi yavaşlatır, çünkü bu örnekler oplanırken (serüven sırasında) öğrenme yapılamaz.

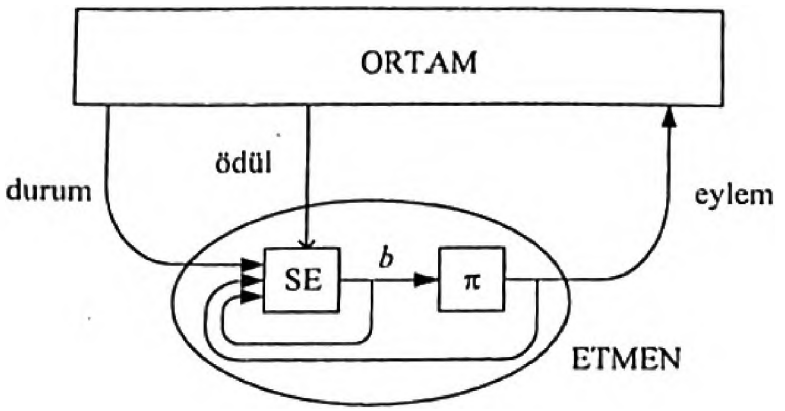
Bu nedenlerden dolayı Q değerlerini öğrenirken yerel modeller kullanmak iyi ir düşüncedir. Örneğin dairesel taban işlevi ağlarında bilgi yereldir, yeni bir örnek ğrenilirken girdi uzayında yalnız o bölgeyle ilgili parametreler güncellenir ve başka ölgelerle ilgili öğrenilmişler bozulmaz. Aynı isterler, $V(s_t|\theta)$ değerlerini kestirirken le geçerlidir.

18.7 Kısmen Gözlenebilen Durumlar

18.7.1 Tanımlar

Bazı uygulamalarda etmen durumunu (ortamı) tümüyle bilmez, ama alıcılarıyla bir *gözlem* oluşturabilir ve bunu kullanarak durumu kestirmesi gerekir. Bir odada gezinen bir robot düşünelim. Robot, odanın neresinde olduğunu ya da odada başka ne olduğunu bilmese de kamerasıyla gözlem yapabilir. Bu, robota durumunu tam olarak söylemese de bazı durumları ötekilere göre daha olası yapar; örneğin robot sağında bir engel olduğunu bilebilir.

Burada yine bir Markov karar süreci kullabiliriz ama şu farkla ki, a_t eyleminden sonraki s_{t+1} durumunu bilmeyiz, yerine, s_t ve a_t değerlerine rastsal, $p(o_{t+1}|s_t, a_t)$ dağılımıyla bağlı bir o_{t+1} gözlemi algılarız. Böyle bir modele, *kısmen gözlenebilir Markov karar süreci* denir. $o_{t+1} = s_{t+1}$ olduğunda durum gözlenebilir ve elimizde bir Markov karar süreci olur. Bu, gözlenebilir ve saklı Markov modelleri arasındaki fark gibidir ve çözümü de benzerdir; gözlemden durumu (daha doğrusu durumların olasılık dağılımını) kestirip bu dağılıma göre karar veririz. Eğer etmen, 0.4 olasılıkla s_1 durumunda, 0.6 olasılıkla s_2 durumunda olduğuna inanıyorsa sola gitmenin iyiliği, 0.4 çarpı s_1 durumunda sola gitmenin iyiliği artı 0.6 çarpı s_2 durumunda sola git-



Şekil 18.9: Kısmen gözlenebilen bir ortamda etmenin, b iç inanç durumu için bir durum kestiricisi vardır ve bu inanç durumlarına göre π politikasını oluşturur.

menin iyiliğine eşittir.

Markov özelliği gözlemler için doğru değildir. Bir sonraki gözlem yalnızca şu anki gözlem ve eyleme bağlı değildir. Algımız kısıtlı olduğunda, iki durum çok farklı olmalarına rağmen benzer görünebilir ve eğer bu iki durum farklı eylemler gerektiriyorsa bu, toplam beklenen ödül olarak hesaplanan başarımda azalmaya neden olabilir. Etmenin bir biçimde geçmiş eylem dizisinden şu an için tek bir durum kestirimi oluşturabilmesi gerekir. Geçmiş gözlemler zaman içinde bekletilip bir pencere biçiminde politikaya girdi olarak verilebilir, ya da özyineli bir ağ (bölüm 11.12.2) ile durum bilgisi, geçmiş gözlemler unutulmadan saklanabilir.

Herhangi bir anda etmen en olası durumu seçip buna göre eylem seçebilir, ya da seçtiği eylem, daha fazla bilgi toplayıp bilinmezliği azaltmak olabilir. Örneğin, bir nirengi noktası arayabilir, ya da durup adres sorabilir. Bu da *bilginin değeri* konusunu bir kez daha gündeme getirir ve kısmen gözlenebilir Markov karar süreçleri, *devingen* etki çizemleri (bölüm 16.8) ile modellenebilir. Etmen eylemler arasında, verdikleri bilgi, kazandırdıkları ödül, ve ortamın durumunu ne kadar değiştirdiklerine göre seçim yapar.

Süreci Markov tutabilmek için etmen, o ana dek deneyimini özetleyen b_t ile gösterdiğimiz bir iç *inanç durumu* saklar (bkz. şekil 18.9). Etmenin, önceki inanç durumu b_t, son eylem a_t ve o anki gözlem o_{t+1} değerlerini kullanarak bir sonraki b_{t+1} inanç durumunu oluşturduğu bir *durum kestiricisi* vardır. π politikası, sonraki a_{t+1} eylemini, tümüyle gözlenebilen ortamdaki gibi gerçek durum yerine, kestirilen bu inanç durumuna göre seçer. İnanç durumu, durumlar üzerinde tanımlanmış bir olasılık dağılımıdır ve (hiçbir eylem yapmadan önceki) ilk inanç değerine ve etmenin gözlem-eylem geçmişine bağlıdır. Böylesi bir uygulamada, Q öğrenme algoritması gerçek durum-eylem çiftleri yerine inanç durumu-eylem çiftleri için değerler kestirir:

$$Q(b_t, a_t) = E[r_{t+1}] + \gamma \sum_{b_{t+1}} P(b_{t+1}|b_t, a_t) V(b_{t+1}) \tag{18.24}$$

18.7.2 Örnek: Saklı Kaplan Problemi

Gelim şimdi Kaelbling, Littman, ve Cassandra (1998) ilk kullanılmış ve Thrun, Burgard, ve Fox (2005) tarafından hafifçe değiştirilmiş *Kaplan* problemini bir örnek olarak tartışalım. Karşımızda, biri solda, biri sağda iki odaya açılan iki kapı olsun. Kapıların bilmediğimiz birinin ardındaki odada bir kaplan, öteki kapının ardındaysa bir hazine olsun. Kaplanın olduğu odanın kapısını açarsak büyük eksi bir ödül, hazinenin odasını açarsak da artı bir ödül kazanacağımızı bilirin. Kaplanın soldaki odada olma olasılığı p olsun; bu durumda kaplanın sağdaki odada olma olasılığı da $1 - p$ olur:

$$p \equiv P(z_L = 1)$$

İki eylemimiz, a_L ve a_R, sırasıyla sol ve sağdaki kapıyı açmaya karşılık gelir. Ödüller için değerler varsayalım:

$r(A, Z)$	Kaplan solda	Kaplan sağda
Solu aç	−100	+80
Sağı aç	+90	−100

İki eylem için beklenen ödül değerlerini hesaplayalım. Gelecekte başka bir ödül yok çünkü kapılardan birini açtığımızda serüven bitiyor:

$$\begin{aligned} R(a_L) &= r(a_L, z_L)P(z_L) + r(a_L, z_R)P(z_R) = -100p + 80(1-p) \\ R(a_R) &= r(a_R, z_L)P(z_L) + r(a_R, z_R)P(z_R) = 90p - 100(1-p) \end{aligned}$$

Bu ödül değerlerine göre, eğer p bire yakınsa, kaplanın solda olma olasılığı yüksek olduğundan sağ kapıyı açmak, benzer biçimde p değeri sıfıra yakınsa da sol kapıyı açmak doğru eylem olur.

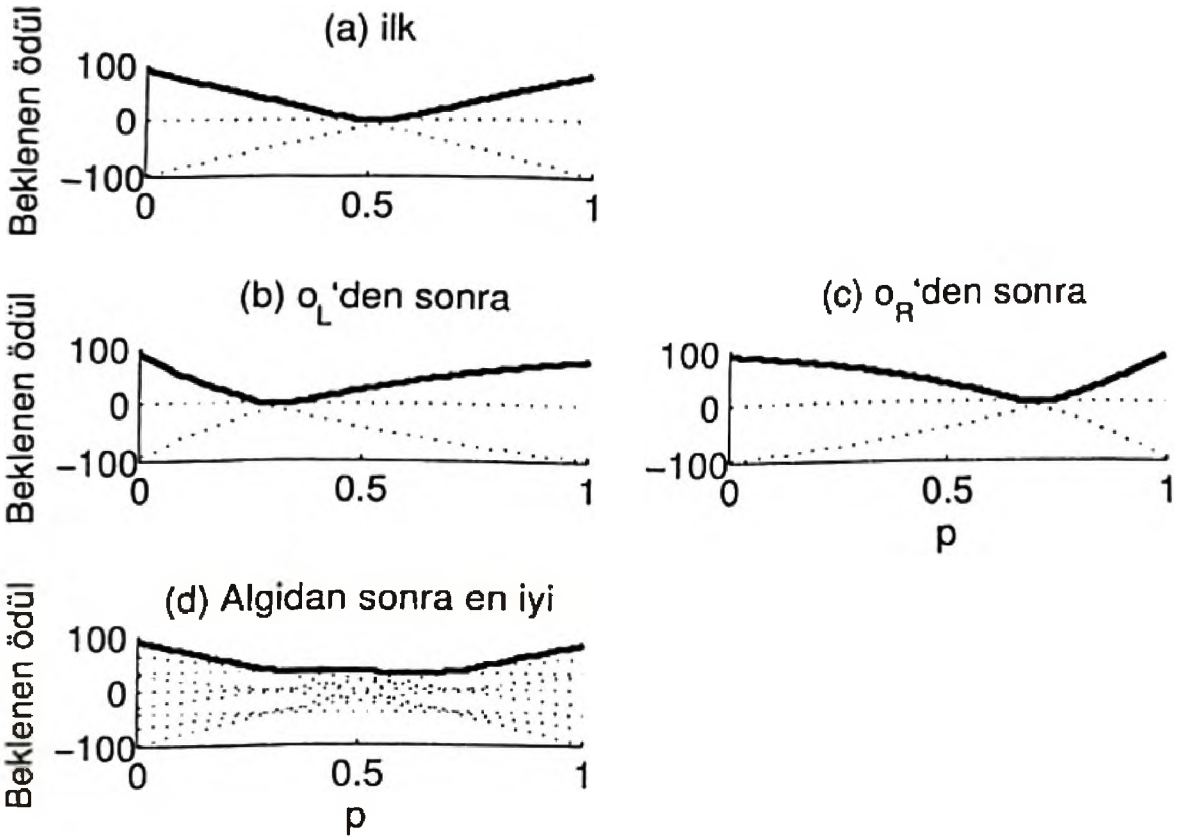
Bu iki eylemin değeri p yaklaşık 0.5 olduğunda kesişir ve burada beklenen ödül değeri yaklaşık -10 olur. Bu da, p yaklaşık 0.5 olduğunda (bilinmezlik olduğunda) bilgi toplamanın önemli olduğunu gösterir. Eğer algıcılar ekleyerek bilinmezliği azaltıp p değerini 0.5 değerinden 0 ya da 1 değerine yaklaştırabilirsek artı ödüllü bir eylem seçebiliriz. Böyle bir a_S algılama eyleminin küçük bir eksi ödülü, $R(a_S) = -1$, vardır, ki bu, algılamanın masrafı, ya da gerçek bir eylemle (kapılardan birini açarak) serüveni tamamlayarak kazanılacak ödülü geciktirmenin indirimi olarak yorumlanabilir.

Böyle bir ekten sonra, beklenen ödüller ve en iyi eylemin değeri

$$V = \max(a_L, a_R, a_S)$$

şekil 18.10a'da gösterilmiştir.

Kaplanın yerini algılayabilmek için her kapıda bir mikrofon olduğunu varsayalım, ama mikrofonlarımız kusurlu olsun (ki kısmen gözlemlenebilirlik senaryomuz geçerli



Şekil 18.10: Kaplan probleminde beklenen ödüller ve algılamanın etkisi.

olmaya devam etsin). Kaplanın varlığını ancak 0.7 olasılıkla algılayabildiğimizi varsayalım:

$$P(o_L|z_L) = 0.7 \qquad P(o_L|z_R) = 0.3$$
$$P(o_R|z_L) = 0.3 \qquad P(o_R|z_R) = 0.7$$

Örneğin, eğer algımız o_L olursa kaplanın konumu hakkındaki inancımız değişir:

$$p' = P(z_L|o_L) = \frac{P(o_L|z_L)P(z_L)}{p(o_L)} = \frac{0.7p}{0.7p + 0.3(1-p)}$$

Bunun etkisi, $R(a_L|o_L)$ değerini çizdiğimiz şekil 18.10b'de görülür. o_L algısı, sağ kapıyı açmayı daha büyük bir aralık için daha iyi eylem yapar. Alıcılarımız iyileştikçe (doğru algılama olasılığı 0.7'den 1'e yaklaştıkça), bu aralık daha büyük olur (alıştırma 9). Benzer biçimde şekil 18.10c'de görüldüğü gibi, algımız o_R olursa sol kapının açılma olasılığı artar. Algılama, (bir kez daha) algılama gerekisinimini de azaltır.

Eylemler için beklenen ödülleri hesaplayalım:

$$\begin{aligned} R(a_L|o_L) &= r(a_L, z_L)P(z_L|o_L) + r(a_L, z_R)P(z_R|o_L) \\ &= -100p' + 80(1-p') \end{aligned}$$

$$
\begin{aligned}
& = -100 \cdot \frac{0.7 \cdot p}{p(o_L)} + 80 \cdot \frac{0.3 \cdot (1-p)}{p(o_L)} \\
R(a_R|o_L) &= r(a_R, z_L)P(z_L|o_L) + r(a_R, z_R)P(z_R|o_L) \\
&= 90p' - 100(1-p') \\
&= 90 \cdot \frac{0.7 \cdot p}{p(o_L)} - 100 \cdot \frac{0.3 \cdot (1-p)}{p(o_L)} \\
R(a_S|o_L) &= -1
\end{aligned}
$$

En iyi eylem bunların en yükseğidir. Benzer biçimde, algımız o_R olduğunda beklenen ödüller de hesaplanabilir:

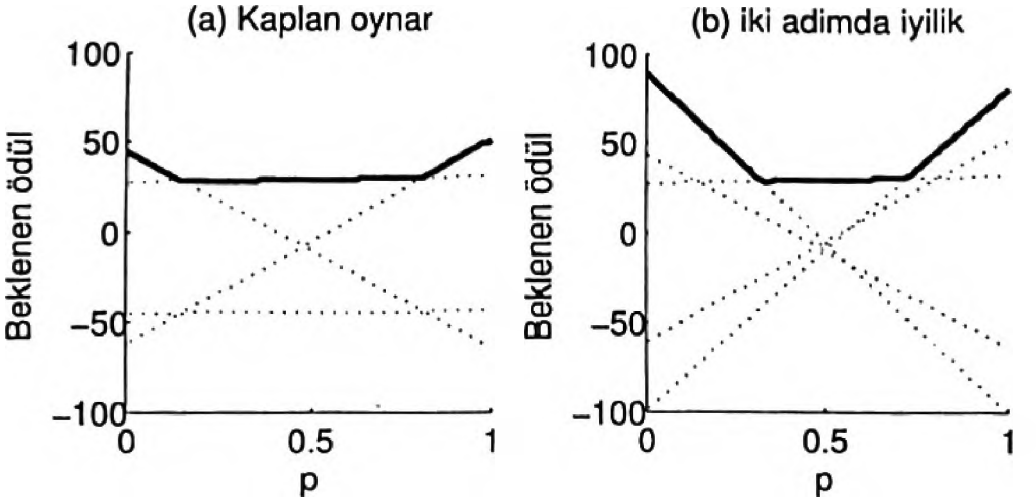
$$
\begin{aligned}
R(a_L|o_R) &= r(a_L, z_L)P(z_L|o_R) + r(a_L, z_R)P(z_R|o_R) \\
&= -100 \cdot \frac{0.3 \cdot p}{p(o_R)} + 80 \cdot \frac{0.7 \cdot (1-p)}{p(o_R)} \\
R(a_R|o_R) &= r(a_R, z_L)P(z_L|o_R) + r(a_R, z_R)P(z_R|o_R) \\
&= 90 \cdot \frac{0.3 \cdot p}{p(o_R)} - 100 \cdot \frac{0.7 \cdot (1-p)}{p(o_R)} \\
R(a_S|o_R) &= -1
\end{aligned}
$$

Beklenen ödülü hesaplayabilmek için ödüllerin algı değerlerinin olasılıklarıyla ağırlıklı ortalamasını almak gerekir:

$$
\begin{aligned}
V' &= \sum_j \left[\max_i R(a_i|o_j)\right] P(O_j) \\
&= \max(R(a_L|o_L), R(a_R|o_L), R(a_S|o_L))P(o_L) + \\
&\quad \max(R(a_L|o_R), R(a_R|o_R), R(a_S|o_R))P(o_R) \\
&= \max(-70p + 24(1-p), 63p - 30(1-p), -0.7p - 0.3(1-p)) + \\
&\quad \max(-30p + 56(1-p), 27p - 70(1-p), -0.3p - 0.7(1-p)) \\
&= \max \begin{pmatrix} -100p & +80(1-p) \\ -43p & -46(1-p) \\ 33p & +26(1-p) \\ 90p & -100(1-p) \end{pmatrix}
\end{aligned}
\tag{18.25}
$$

$P(o_L)$ ile çarpınca sadeleşir ve p cinsinden doğrusal işlev elde ederiz. Burada beş tane doğru oluşur ve bunların her noktada en büyüğünü seçip birleştirdiğimizde parçalı sürekli bir doğru oluşur (bkz. şekil 18.10d). $-40p - 5(1-p)$ doğrusu ve a_S ile ilgili olanlar tüm p değerleri için öbürlerinden aşağıda olduklarından budanmıştır. Şekil 18.10d'nin, şekil 18.10a'dan daha iyi olması *bilginin değerine* örnektir.

Burada, a_S eylemini seçtiğimizde en iyi eylemin değerini hesapladık. Örneğin ilk doğru, a_S eyleminden sonra a_L eylemini seçmenin değeridir. Yani, iki adımlık bir serüvende en iyi kararı verirken bunu a_S eyleminin ödülü olan -1 ekleyerek geriye taşıyıp algı eyleminin değerini hesaplarız. Bir başka biçimde yorumlarsak algı



Şekil 18.11: (a) Saklı durum değiştiğinde, ve (b) iki adımlık serüvenler gözönüne alındığında beklenen ödüller değişir.

eyleminin ödülü 0 olur ama sonraki ödülde $\gamma < 1$ çarpanıyla bir indirim yapılır. Bundan başka anlık iki eylemimiz a_L ve a_R vardır ve bu üç tane (ikisi o anlık, biri de bir adım gecikmiş ve indirim yapılmış) eylem arasından seçim yaparız.

Şimdi örneği daha da ilginç yapmak için (Thrun, Burgard, ve Fox 2005), iki oda arasında bir kapı olduğunu ve biz farkında olmadan kaplanın bir odadan ötekine geçebildiğini varsayalım. Kaplanımız yerinde duramasın ve 0.2 olasılıkla aynı odada kalırken 0.8 olasılıkla öbür odaya geçsin. Bu p olasılığını da değiştirir:

$$p' = 0.2p + 0.8(1 - p)$$

ve denklem 18.25'te a_S eyleminden sonraki en iyi eylemi seçerken bu güncellenmiş p' değerinin kullanılması gerekir:

$$V' = \max \begin{pmatrix} -100p' & +80(1-p') \\ 33p' & +26(1-p') \\ 90p' & -100(1-p') \end{pmatrix}$$

Şekil 18.11b, şekil 18.10d'nin güncellenmiş p' ile hesaplanmışıdır. Şimdi, iki adım uzunluğunda serüvenler için politika oluştururken, anlık iki a_L ve a_R eylemlerinden birini seçebiliriz; ya da bekleyip algılarken p değişir, bir eylem seçeriz ve indirim yapılmış ödül elimize geçer (şekil 18.11b):

$$V_2 = \max \begin{pmatrix} -100p & +80(1-p) \\ 90p & -100(1-p) \\ \max V' - 1 & \end{pmatrix}$$

Şekil 18.11b, şekil 18.10a'ya yeğdir; yanlış eylemlerin cezası yüksek olduğunda kararı erteleyip bilgi toplamak ve ilerisi için plan yapmak daha iyidir. İstersek benzer

biçimde iki adımdan daha uzun serüvenler için de, her adımda p olasılığını güncellediğimiz ve ödülden 1 çıkardığımız $V_t, t > 2$ değerleri de hesaplayabiliriz.

Değerlerin parçalı doğrusal işlevlerle gösterildiği bu algoritma yalnızca durumların, eylemlerin, gözlemlerin ve serüven uzunluğunun sonlu olduğu durumda çalışır. Bunlardan biri sürekliyse, ya da olası değer sayısı küçük değilse, algoritma karmaşıklığı çok yüksek olur ve kabul edilebilir karmaşıklıkta ama yaklaşık bir değer hesaplayan bir algoritmaya başvurmamız gerekir. Böylesi algoritmalar, (Hauskrecht 2000) ve (Thrun, Burgard, ve Fox 2005)'te verilmiştir.

18.8 Kısa Kısa

Pekiştirmeli öğrenme konusunda bir ders kitabı olan (Sutton ve Barto 1998)'de öğrenme algoritmaları ve farklı uygulamalar verilmiştir. (Kaelbling, Littman, ve Moore 1996), öğretici bir makaledir. Pekiştirmeli öğrenmenin robotik ve başka yakın zamanlı, etkileyici uygulamaları (Thrun, Burgard, and Fox 2005)'te anlatılmaktadır.

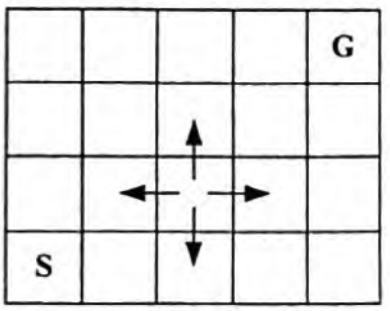
Devingen programlama yöntemleri (Bertsekas 1987, Bertsekas ve Tsitsiklis 1996) içinde ayrıntılı anlatılır. TD(λ) ve Q öğrenme algoritmaları, devingen programlamanın rastgele yaklaşıklamaları olarak görülebilir (Jaakkola, Jordan, ve Singh 1994). Pekiştirmeli öğrenmenin klasik devingen programlamaya göre iki üstünlüğü vardır: Öncelikle, öğrendikçe uzayın önemli bölümlerine odaklanıp gerekmeyen bölümleri gözardı edebilirler; ve ikinci olarak bağlanımla yaklaşık bir model kestirdiklerinden genelleme yapabilir ve bu sayede hızlı öğrenebilirler.

İlgili bir alan *öğrenen özdevinirlerdir* (Narendra ve Thathachar 1974), ki bunlar, deneme yanılmayla öğrenen sonlu durum makineleridir ve K kollu makine gibi uygulamaları çözmeyi öğrenebilirler. Bu bölümde tartıştıklarımız bir denetim uygulaması olarak da görülebilir: Bir donanım (ortam) içinde bir denetleyici (etmen) hatayı en aza indirecek (ödülü en fazla yapacak) eylemi seçer.

Zamansal farkın ilk uygulaması Samuel'in 1959 tarihli dama programıdır (Sutton ve Barto 1998). Her ardışık iki durum için, tahta değerlendirme işlevi hamleden önceki ve sonraki durumlar için tahta iyiliklerini hesaplar ve aradaki farkı azaltacak bir güncelleme yapılır. Oyunlar ile ilgili birçok çalışma yapılmıştır çünkü oyunlar hem kolayca tanımlanabilir, hem de çoğunu oynaması zordur. Satranç gibi bir oyunun programı kolayca yazılabilir: Hamleler muğlak değildir ve amaç kesin tanımlanmıştır. Bu basitliğine rağmen iyi oynaması oldukça zordur.

Pekiştirmeli öğrenmenin en etkileyici uygulamalarından biri, kendisiyle oynayarak öğrenen TD-Gammon tavla programıdır *TD-Gammon* (Tesauro 1995). Bu program, yine Tesauro tarafından yazılmış ve uzman oyunlarıyla gözetimli eğitilmiş önceki yapay sinir ağı tabanlı programa üstündür. Tavla, olası yaklaşık 10^{20} durumu olan zor bir uygulamadır ve zarlardan dolayı rastsallık da içerir. TD(λ) algoritmasıyla bu program, kendisiyle 1.500.000 kez oynadıktan sonra usta seviyesine ulaşmıştır.

Bir başka ilginç uygulama, *sipariş zaman çizelgesi* oluşturmadır ve amaç, zaman ve kaynak kısıtlarına uyacak biçimde işlerin sıralanmasıdır (Zhang ve Dietterich



Şekil 18.12: Etmen, S hücresinden başlayarak dört ana yönde ilerleyebilir. Amaç durumu G hücresidir.

1996). Bazı işler, başka bazı işler başlamadan önce bitirilmelidir ve aynı kaynağı kullanan iki iş aynı anda yapılamaz. Zhang ve Dietterich, kısıtları sağlayan ve kısa çizelgeler oluşturmayı hızla öğrenen programlar için pekiştirmeli öğrenme kullanır. Her durum bir çizelgeye, eylemler çizelge üstündeki değişikliklere karşılık gelir, ve böylece program, yalnız bir iyi çizelgeyi değil ilgili bir çizelgeleme uygulama sınıfı için çizelge oluşturmayı öğrenir.

Yakın zamanlarda, bir işin daha basit alt işlere bölündüğü ağaç yapılı yöntemler önerilmiştir. Bu yaklaşımın üstünlüğü alt işler için geliştirilen politikaların başka işler için paylaşılabilmesidir; bu da yeni işlerin öğrenilmesini hızlandırır (Dietterich 2000). Alt işler daha basittir ve ayrı ayrı hızla öğrenilebilir. Çekincesiyse alt işlerin birleştirilmesiyle oluşturulan çözümün tüm işe bakıldığında olası en iyi çözüm olmayabilmesidir.

Pekiştirmeli öğrenme, gözetimli öğrenmeden daha yavaş olmasına rağmen kullanım alanının daha geniş olduğu ve daha iyi öğrenen dizgeler oluşturabileceği açıktır (Ballard 1997). Gözetim gerektirmediği için bir öğretmenin önyargılarından etkilenmez. Örneğin, Tesauro'nun TD-Gammon programının yaptığı bazı hamlelerin uzmanların o durum için en iyi diye düşündüğü hamlelere üstün olduğu görülmüştür. Pekiştirmeli öğrenme alanı hızla gelişmektedir ve yakın zamanda çok daha etkileyici uygulamalar görmeyi bekliyoruz.

18.9 Alıştırmalar

1. Şekil 18.12'deki ortamda, amaç ödülü 100 ve $\gamma = 0.9$ iken $Q^*(s, a)$ ve $V^*(S)$ değerlerini elle hesaplayın ve en iyi politikanın eylemlerini gösterin.
2. Alıştırma 1 için en iyi politikayı Q öğrenme algoritmasıyla hesaplayın.
3. Alıştırma 1'de alt sağ hücreye bir amaç hücresi daha eklenirse en iyi politika nasıl değişir? Alt sağ hücreye ödülü -100 olan (çok kötü) bir durum eklenirse ne değişir?
4. $\gamma < 1$ yerine $\gamma = 1$ olduğunu ama her ara (amaç olmayan) durum için küçük bir $-c$ ödülü olsa ne fark eder?
5. Alıştırma 1'de amaç hücreye varınca kazandığımız ödülün ortalaması 100, değişintisi 40 olan normal bir dağılımdan geldiğini varsayalım. Eylemlerin de rastsal olduğunu

ve bir eylem yaptığımızda 0.5 olasılıkla istediğimiz yönde, 0.25 olasılıkla da buna dik iki yönün birinde hareket ettiğimizi varsayalım. Bu uygulama için $Q(s, a)$ değerlerini öğrenin.

6. $V(s)$ durum değerlerini kestirmek için TD(λ) algoritmasını türetin.
7. Denklem 18.22'yi kullanarak, Q değerlerini kestiren çok katmanlı bir algılayıcı için ağırlık güncelleme kurallarını yazın.
8. Kısmen gözlenebilir bir uygulamaya örnek verin ve bu uygulama için durumları, eylemleri, gözlemleri ve ödülü tanımlayın.
9. Saklı kaplan örneğinde algılayıcımız iyileştikçe bir kez daha algılamaya olan gereksinimimizin azaldığını gösterin.
10. Kaplan örneğini

$r(A, Z)$	Kaplan solda	Kaplan sağda
Solu aç	−100	+10
Sağı aç	20	−100

ödül çizelgesine göre yeniden hesaplayın:

18.10 Kaynaklar

Ballard, D. H. 1997. *An Introduction to Natural Computation.* Cambridge, MA: MIT Press.

Bellman, R. E. 1957. *Dynamic Programming.* Princeton: Princeton University Press.

Bertsekas, D. P. 1987. *Dynamic Programming: Deterministic and Stochastic Models.* New York: Prentice Hall.

Bertsekas, D. P., J. N. Tsitsiklis. 1996. *Neuro-Dynamic Programming.* Belmont, MA: Athena Scientific.

Dietterich, T. G. 2000. "Hierarchical Reinforcement Learning with the MAXQ Value Decomposition." *Journal of Artificial Intelligence Research* 13: 227–303.

Hauskrecht, M. 2000. "Value-Function Approximations for Partially Observable Markov Decision Processes." *Journal of Artificial Intelligence Research* 13: 33–94.

Jaakkola, T., M. I. Jordan, S. P. Singh. 1994. "On the Convergence of Stochastic Iterative Dynamic Programming Algorithms." *Neural Computation* 6: 1185–1201.

Kaelbling, L. P., M. L. Littman, A. R. Cassandra. 1998. "Planning and Acting in Partially Observable Stochastic Domains." *Artificial Intelligence* 101: 99–134.

Kaelbling, L. P., M. L. Littman, A. W. Moore. 1996. "Reinforcement Learning: A Survey." *Journal of Artificial Intelligence Research* 4: 237–285.

Narendra, K. S., M. A. L. Thathachar. 1974. "Learning Automata—A Survey." *IEEE Transactions on Systems, Man, and Cybernetics* 4: 323–334.

Sutton, R. S. 1988. "Learning to Predict by the Method of Temporal Differences." *Machine Learning* 3: 9–44.

Sutton, R. S., A. G. Barto. 1998. *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press.

Tesauro, G. 1995. "Temporal Difference Learning and TD-Gammon." *Communications of the ACM* 38(3): 58–68.

Thrun, S., W. Burgard, D. Fox. 2005. *Probabilistic Robotics.* Cambridge, MA: MIT Press.

Watkins, C. J. C. H., P. Dayan. 1992. "Q-learning." *Machine Learning* 8: 279–292.

Zhang, W., T. G. Dietterich. 1996. "High-Performance Job-Shop Scheduling with a Time-Delay TD(λ) Network." *Advances in Neural Information Processing Systems 8*, ed. D. S. Touretzky, M. C. Mozer, M. E. Hasselmo, 1024–1030. Cambridge, MA: The MIT Press.

Bölüm 19

Deney Tasarımı ve Sonuçların Çözümlenmesi

Öğrenme algoritmalarının başarılarının ölçülmesi ve karşılaştırılması için yapılacak deneylerin nasıl tasarlanması gerektiğini ve sonuçların çözümlenmesi için kullanılan istatistiksel yöntemleri tartışacağız.

19.1 Giriş

Önceki bölümlerde birçok öğrenme algoritmasını tartıştık. Bu bölümde iki soruya yanıt arayacağız:

1. Belirli bir uygulamada bir öğrenme algoritmasının beklenen hatasının ne kadar olacağını ölçebilir miyiz? Örneğin, bir sınıflandırıcıyı bir veri üzerinde eğittikten sonra, aynı uygulamadan çekilmiş yeni örnekler üzerinde denendiğinde hatasının, örneğin, yüzde ikiden küçük olacağını güvenle söyleyebilir miyiz?

2. Bir uygulama için iki algoritma verildiğinde, birinin hatasının ötekine göre daha küçük olduğunu nasıl söyleyebiliriz? Farklı algoritmalar gibi aynı algoritmanın farklı üstparametrelerini de karşılaştırmak isteyebiliriz. Örneğin, biri 4, öteki 8 saklı birimli çok katmanlı algılayıcıdan hangisinin beklenen hatasının daha az olduğunu, ya da en yakın k komşu algoritmasındaki en iyi k değerinin ne olduğunu nasıl bulabiliriz?

Buraya kadarki bölümlerde birçok kez böyle bir ölçme ya da karşılaştırma yaparken öğrenme kümesi üzerinde hesaplanan hatanın kullanılamayacağından bahsettik. Öğrenme kümesi üzerindeki hata, deneme kümesi üzerindeki hatadan her zaman daha küçüktür ve daha karmaşık modelin öğrenme hatası her zaman daha küçük olur.

Başarıyı ölçerken öğrenme kümesinden farklı bir geçerleme kümesine gerek duyarız. Tek bir geçerleme kümesi de yeterli olmayabilir, çünkü hem öğrenme, hem de deneme kümesi küçük olup istisnai, aykırı, gürültülü örnekler içerebilirler ve bu yüzden tek bir sonuç gösterge olmaz. Üstelik öğrenme algoritması çoğu zaman genellemeyi etkileyen başka rastsal etkenlere de bağlıdır. Örneğin, çok katmanlı algılayıcılardaki hata geri yayma algoritması yerel en küçük noktaların en yakınına yakınsadığından eğim inişe nereden başlandığı önemlidir ve farklı yerlerden başlayınca aynı yapı ve öğrenme kümesini bile kullansak farklı son sınıflandırıcılara ve aynı geçerleme kümesinde farklı hata değerlerine ulaşabiliriz. Bu yüzden, bir yerine birden çok deneme yapmak ve bu rastsallığın üzerinden ortalama almak isteriz. Yalnızca bir kez öğrenme ve deneme yaptığımızda böylesi rastsal etkenlerin etkisini bilemeyiz; bu, ancak öğrenmenin çok pahalı olduğu bir durumda kabul edilebilir.

Bir *öğrenme algoritmasını* bir veri üzerinde kullandığımızda elimizde bir *model* olur. Öğrenme bir kez yapılırsa bir model ve bir geçerleme hata değeri olur. Rastsallık (öğrenme ve geçerleme kümesinde, ilk ağırlıklarda, vs.) üzerinden ortalama almak için aynı algoritmayı birçok kez kullanıp birden çok model oluştururuz. Bunların hepsini denediğimizde elimizde geçerleme hatalarından oluşan bir örneklem olur. (Tabii ki tüm öğrenme ve deneme kümelerinin aynı uygulamadan olması gerekir). Öğrenme algoritmasının iyiliğini ölçmede bu geçerleme değerlerinin *dağılımına* bakarız. Bu dağılıma bakarak algoritmanın *beklenen hatasını* ölçebilir, ya da başka bir ya da birçok algoritmanın beklenen hatasıyla karşılaştırabiliriz.

Bunun nasıl yapılacağına geçmeden önce, bazı noktaları vurgulamak gerekiyor:

1. Varacağımız tüm sonuçlar elimizdeki veriye bağlıdır. Olası tüm uygulama alanlarında ya da bir uygulamanın geneli için karşılaştırma yapmıyoruz. Bir algoritmanın uygulamadan bağımsız hatası, ya da genelde iki algoritmanın genel karşılaştırılması üzerine hüküm vermiyoruz. Vardığımız her sonuç, yalnızca bir uygulama için ve o uygulamadan seçilmiş bir veri kümesi için doğrudur. Zaten daha önce, "Bedava yemek yoktur," kuramından (Wolpert, 1995) bahsederken, "en iyi" öğrenme algoritmasının olmadığını, her algoritmanın çok başarılı olduğu veriler olduğu gibi çok başarısız olduğu verilerin de olduğunu belirtmiştik. Bir öğrenme algoritmasının iyi olması derken kastettiğimiz model varsayımlarının verinin özelliklerine uymasıdır.

2. Verinin öğrenme ve geçerleme kümeleri olarak bölünmesi, yalnızca ölçme ve değerlendirme içindir. Bütün denemeler yapılıp en iyi model ve üstparametreler belirlendikten sonra sonuç modeli eğitirken elimizdeki (daha önce öğrenme ve geçerleme için kullandığımız) tüm veriyi kullanabiliriz.

3. Geçerleme kümesini en iyi model ya da üstparametreyi seçmek için kullanmak, onu da öğrenme kümesinin bir parçası yapar. Bütün denemelerden sonra en iyi model ve üstparametre değeriyle oluşturduğumuz sonuç modelin başarısını ölçmek için bize farklı, daha önce öğrenme ya da geçerlemede kullanmadığımız ve yeterince büyük bir *deneme kümesi* gerekir.

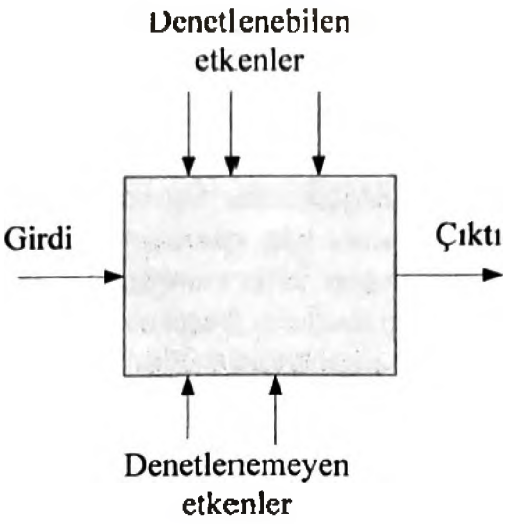
Bu da demek oluyor ki bir veri kümesi verildiğinde önce bir kısmını deneme kümesi olarak ayırmalı, kalanı öğrenme ve geçerleme için kullanmalıyız. Genelde verinin üçte biri deneme kümesi olarak ayrılıp kalan üçte ikiden, bu bölümde biraz sonra göreceğimiz gibi *yeniden örneklemeyle* belli sayıda öğrenme ve geçerleme kümesi çifti oluşturulur. Öğrenme kümesini kullanarak verilmiş bir algoritma ve model yapısı için parametre değerlerini, geçerleme kümesi ile de model yapısını belirleyen üstparametreleri buluruz. Deneme kümesi ise en sonda elde edilen sonuç modelin başarısını ölçmek için, bir kez kullanılır. Örneğin, çok katmanlı bir algılayıcıda öğrenme kümesi ağırlıkları, geçerleme kümesi saklı birim sayısını belirler. En iyi algılayıcı belirlendikten sonra hatası deneme kümesi üzerinde ölçülür. En yakın k komşu algoritmasında öğrenme kümesini içinde arama yapmak için saklarız; en iyi uzaklık ölçütü ve k değeri geçerleme kümesi üzerindeki başarıya bakılarak ayarlanır ve sonuç deneme kümesi üzerinde sonuç model denenir.

4. Öğrenme algoritmalarını çoğu zaman hatalarına göre karşılaştırsak da, gerçek yaşamdaki mühendislik uygulamalarında kararımızı belirleyen tek kıstas hata değildir ve başka kısıtlar da bir algoritmayı diğerine yeğlemekte önemli olur (Turney 2000):

   - 0/1'den farklı bir yitim işlevi tanımlandığında sınıflandırma hatası yerine beklenen risk (bölüm 3.3),
   - öğrenmenin zaman ve bellek karmaşıklığı,
   - denemenin zaman ve bellek karmaşıklığı,
   - yorumlanabilirlik, yani öğrenilmiş modelden bilgi çıkarılması ve modelin uzmanlar tarafından anlaşılabilen kurallara dökülebilmesi,
   - kolay programlanabilme.

   Bunların görece ağırlıkları uygulamaya bağlıdır. Öğrenme tek bir kez, ürünün imalatı sırasında yapılıyorsa öğrenmenin zaman ve bellek karmaşıklığı önemli değildir ama kullanım sırasında uyarlanabilme gerekliyse önemli olur. Son yıllarda, hataya ek olarak başka böylesi kısıtları da kullanabilen *bedel duyarlı öğrenme* algoritmaları önerilmiştir.

Bir süreçten örneklemeyle rastsal veri çekip sonra bununla bir modeli eğitip başka bir geçerleme kümesi üzerinde başarısını ölçüp sonuç çıkardığımızda yaptığımız istatistiksel bir deneydir. İstatistik yazınında böylesi deneylerin nasıl yapılması ve sonuçların anlamlı olması için deney sonunda elde edilen verilen nasıl çözümlenmesi gerektiğini belirleyen yöntemler vardır (Montgomery, 2005). Bu yöntemleri ve yapay öğrenmeye nasıl uyarlandığını bu bölümde göreceğiz.



Şekil 19.1: Süreç, girdi verildiğinde bir çıktı üretir ve denetlenebilen ve denetlenemeyen etkenlere bağlıdır.

19.2 Etkenler, Tepki, ve Deney Yordamları

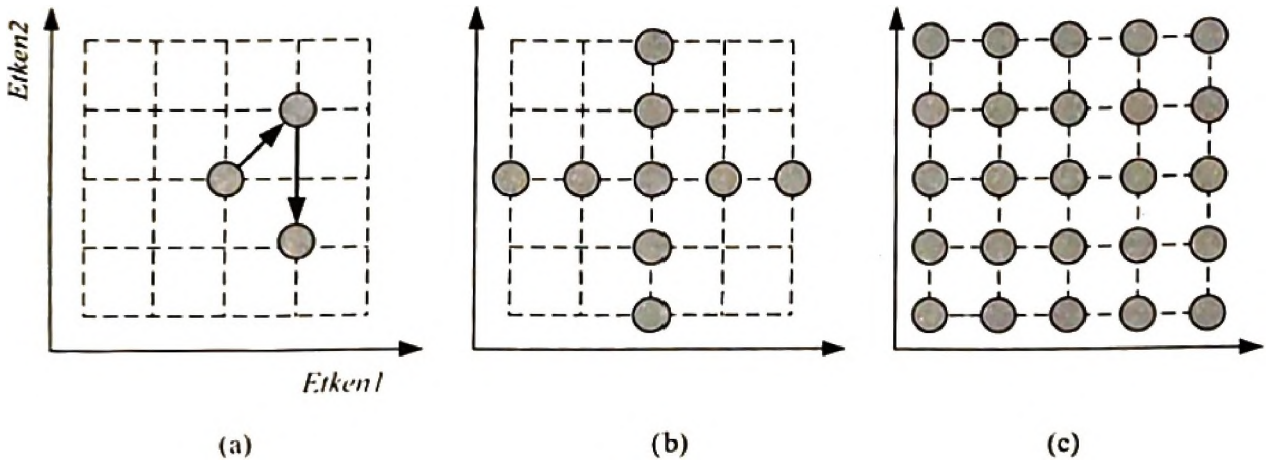
Öteki bilim ve mühendislik dalları gibi yapay öğrenmede de, incelediğimiz süreçle ilgili bilgi edinmek için deney yaparız. Bizde süreç, öğrenme verisiyle eğitilmiş modelin girdiye bağlı olarak çıktı vermesidir. Bir *deney*, çıktıyı etkileyen *etkenlerle* oynadığımız bir dizi denemedir. Yapay öğrenmede etkenler, öğrenme algoritması, öğrenme kümesi, girdi öznitelikleri, vs. olabilir ve bunlara bağlı olarak modelin *tepkisini* gözleriz. Amacımız, tepkiyi, örneğin, deneme kümesindeki sınıflandırma başarısını, en istenen duruma çıkaran etken bileşimini belirlemek, tepkiye etki eden en önemli etkenleri bulmak ya da önemsiz olanları ayıklamak olabilir.

Yapay öğrenme deneyleri tasarlayıp gerçeklemek ve bu deneyler sonucunda ortaya çıkan veriyi çözümleyerek, rastsallıktan en az etkilenen ve *istatistiksel olarak anlamlı* diyebileceğimiz sonuçlara ulaşmak istiyoruz. Amacımız genelleme başarısı olabildiğince yüksek, gerçekleme zaman ve bellek karmaşıklığı olabildiğince düşük ve dış değişkenlik (rastsallık) etkilerine karşı olabildiğince gürbüz bir model oluşturmaktır.

Eğitilmiş bir model şekil 19.1'deki gibi gösterilebilir. Üretilen çıktı iki tür etkene bağlıdır: *Denetlenebilen etkenler*, adı üstünde, değerini istediğimiz gibi belirleyebildiğimiz etkenlerdir; örneğin, öğrenme algoritması, üstparametre değerleri, kullanılan veri kümesi, girdi gösterimi, vs.

Denetlenemeyen etkenler sürece bizim istemediğimiz bir değişkenlik ekler. Örneğin, verideki gürültü, büyük bir küme içinden yeniden örneklemeyle oluşturduğumuz altkümelerdeki oynamalar, öğrenme sırasında eklenebilecek rastsallıklar, örneğin, eğim inişteki ilk ağırlık değerleri, vs, bunlara örnektir. İstenmeyen bu değişkenliği kestirebilmek ve verdiğimiz kararların bundan etkilenmemesini isteriz; "istatistiksel olarak anlamlı sonuç" derken kastettiğimiz budur.

Çıktıyı kullanarak bizim için başarımı belirleyen *tepki* değişkenini hesaplarız;



Şekil 19.2: Her biri beş değer alabilen iki etkende farklı deney yordamlarının çalışması: (a) En iyi öngörü, (b) Her kez bir etken, ve (c) Etkensel tasarım.

örneğin, deneme kümesi üzerindeki sınıflandırma başarısı, ya da bir yitim işlevi varsa beklenen risk, ya da, ileride göreceğimiz gibi, bilgi erişiminde kesinlik ve anma oranları, önemli ölçütler olabilir.

En iyi başarımı veren etken değerlerini ya da bu etkenlerin tepkiye etkilerini belirlemek isteyebiliriz. Örneğin, en yakın k komşu algoritmasından önce temel bileşen çözümlemesiyle boyutu d değerine indiriyorsak, hangi (d, k) bileşiminin en yüksek başarımı sağladığını, ya da destek yöney makinelerinde, C düzenlileştirme parametresi ve örneğin Gauss çekirdeğinin s^2 yarıçap parametresinin nasıl seçilmeleri gerektiğini bulabiliriz.

Şekil 19.2'de gösterildiği gibi farklı *deney yordamları* vardır. *En iyi öngörü* diyebileceğimiz en basit yordamda, iyi olduğunu bildiğimiz bir konumdan (etken değerlerinin bileşimi) başlar ve oradaki tepkiye baktıktan sonra, her adımda bir ya da birkaç etkenin değeriyle oynayarak daha iyi bir konuma geçmeye çalışırız. Yeterince iyi olduğunu düşündüğümüz yerde, ya da tepkide bir iyileşme sağlayamadığımızda dururuz. Eğer uygulamayla ilgili sezgileriniz güçlüyse bu yordam iyi çalışabilir ama denemelerimizde bir düzen olmadığı için iyi bir noktaya yakınsama güvencemiz yoktur.

Bir başka yordamda, *her kez bir etkeni* değiştiririz. Her etken için iyi olduğuna inandığımız bir orta değerden başlar, ve sonrasındaki her adımda, etkenlerden biri dışındakileri sabitleyip serbest olan için olası değerler içinde tepki açısından en iyisini seçeriz. Bu yordamla ilgili en önemli çekince, etkenler arasında etkileşim olmadığını varsaymasıdır. Temel bileşen çözümlemesi ardından en yakın k komşu yapıyorsak örneğin, seçilen her boyut sayısı d farklı bir uzay tanımladığından farklı bir k gerektirebilir.

En doğru deney yordamı, etkenlerin birer birer değil birlikte değiştirildiği *etkensel tasarımdır*. Her biri L olası değer alan F etken, $L \cdot F$ kesişimli bir ızgara tanımlar ve burada birer birer arama yapmak $\mathcal{O}(L \cdot F)$ zaman alırken etkensel tasarımı $\mathcal{O}(L^F)$ zaman alır.

19.3 Tepki Yüzeyi Tasarımı

Deney sayısını azaltmak için bir olasılık, kısmi etkensel tasarımıyla olası bileşimlerin bir altkümesini denemek, bir başka olasılıksa önceki deney sonuçlarını kullanarak iyi tepki değeri vereceğini düşündüğümüz bileşenleri kestirmektir. Etkenleri birer birer denerken, tepkinin o etkene ikinci dereceden bağlı olduğunu varsayabilirsek tepe noktasını çözümsel hesaplayabiliriz: İlk deneylerden sonra gördüğümüz tepki değerlerine ikinci dereceden bir işlev oturtur, onun tepe noktasının nerede olduğunu çözümsel hesaplar, o değerle bir deney yapar, gözlediğimiz tepki sonucunu da katıp yeniden bir ikinci dereceden işlev oturtur ve yeni bir tepe noktası kestirebiliriz. Bir iyileşme olmayınca dururuz.

Bu algoritmanın birden çok etkileşen etken için genellenmişine *tepki yüzeyi tasarımı* denir ve amaç, tepkiyi etken değerlerinin basit bir işlevi olarak kestirebilmektir:

$$r = g(f_1, f_2, \ldots, f_F|\phi)$$

Burada, r değişkeni tepkiyi, $f_i, i = 1, \ldots, F$ etken değerlerini gösterir. Yaptığımız deneylerde gözlemlediğimiz tepki değerlerine bağlanım kullanarak etken değerlerine bağlı bir işlev oturturuz; ϕ, bu işlevin parametreleridir. Denetleyemediğimiz etkenlerin etkisinin sıfır ortalamalı normal gürültü olduğunu varsayarız. $g(\cdot)$, çoğunlukla ikinci dereceden bir bağlanım modelidir ve az sayıda deney ardından $g(\cdot)$ için bir kestirim yapabiliriz. İlk deneyleri, iyi olduğunu düşündüğümüz bir merkez nokta çevresinde bir *deney dizeyi* belirler. Bir kez elimizde $g(\cdot)$ olduğunda, çözümsel olarak g değerini en yüksek yapacak f_i etken değerlerini hesaplayabiliriz. Bu etken değer bileşimi için bir sonraki deneyi yapar, oradaki r değerini gözlemler, bu veriyi öğrenme kümesine ekleyip yeni g işlevini oturtur, ve bu işlemi yakınsama olana, yani r artmayana dek yineleriz. Bu yaklaşımın iyi çalışıp çalışmayacağı, tepki işlev modelinin ne kadar uygun seçildiğine bağlıdır.

19.4 Rastsallaştırma, Çoğaltma, ve Bölükleme

Şimdi deneny tasarımın üç temel ilkesini konuşalım:

- *Rastsallaştırma*, sonuçların bağımsız olabilmesi için deneylerin yapılma sırasının rastgele belirlenmesidir. Bu özellikle gerçek dünyada fiziksel nesnelerin kullanıldığı deneylerde önemlidir; makinelerin ısınması zaman alabilir ya da her an aynı randımanla çalışmayabilirler ve bu zaman bilgisinin sonuçları etkilememesi için deneylerin rastgele sıralanması gerekir. Yazılım deneylerinde genelde (yazılım yaşlanması etkisini gözardı edersek) programların çalıştırılma sırasının bir önemi yoktur.

- *Çoğaltma*, denetlenebilen etkenlerin aynı değerleri için deneyin birden çok kez yinelenerek denetlenemeyen etkenlerin etkisinin anlaşılabilmesi demektir. Yapay öğrenmede, aynı algoritmayı aynı veri kümesinden farklı örneklenmiş

kümeler üzerinde birçok kez çalıştırırız. 19.6. bölümde göreceğimiz bu yaklaşıma *çapraz geçerleme* denir. Aynı deneyin farklı yinelemelerinde tepki değerinin ne kadar değiştiği, denetlenemeyen etkenlerin neden olduğu deney hatasını hesaplamamızı sağlar; böylece, *istatistiksel olarak anlamlı* nitelenebilmesi için tepkideki değişikliğin ne kadardan daha büyük olması gerektiğini söyleyebiliriz.

- *Bölükleme*, *zararlı etkenlerin* tepkide neden olduğu değişkenliğinden kurtulmak için gerekir. Örneğin, bir fabrikada üretilen mallardaki hatalar kullanılan hammaddenin kaynağına, makineye, personele, vs. bağlı olabilir. Yapay öğrenmede algoritmayla ortaya çıkan modelin başarısı örneklemeyle oluşan veri kümesine de bağlıdır ve eğer farklı algoritmaları karşılaştırıyorsak modellerin öğrenme ve geçerleme için kullandığı veri kümelerinin aynı olması gerekir, yoksa başarı (tepki) değerleri arasındaki fark yalnızca algoritmaya değil veriye de bağlı olur; işte buna bölükleme diyoruz. Eğer iki dağılımı karşılaştırıyorsak buna *eşleme* diyoruz ve *eşli sınamada* kullanıyoruz.

19.5 Yapay Öğrenme Deneyleri için İlkeler

Deneylere başlamadan önce neyi araştırdığımızı, nasıl veri toplayacağımızı ve sonuçları nasıl çözümleyip sonuç çıkaracağımızı bilmemiz gerekir. Yapay öğrenmede deneyler öteki alanlarda yapılanlara benzer (Montgomery 2005). Bu bölümde tartıştığımız ilkeler, uygulamanın sınıflandırma ya da bağlanım olduğuna, ya da gözetimsiz mi, pekiştirmeli mi olduğuna bağlı değildir; fark yalnızca deney sonunda ölçülen tepki değerinin aldığı değerin dağılımındadır.

A. Çalışmanın Amacı

Öncelikle çalışmanın amacını açıkça tanımlamak gerekir. Yapay öğrenmede amaç, bir algoritmanın bir uygulamadaki başarısını ölçmek ve örneğin, kabul edilebilir bir düzeyde olup olmadığını sınamak olabilir.

İki öğrenme algoritması ve bir veri kümesi verildiğinde, hangisinin hatasının daha az olduğunu bulmak da isteyebiliriz. Bu ikisi gerçekten farklı algoritmalar olabilir, ya da biri ötekinin geliştirilmiş (örneğin, önüne daha farklı bir öznitelik çıkarıcı konmuş) bir sürümüdür ve bu değişikliğin yararını sınamak istiyor olabiliriz.

Daha genel durumda, elimizde ikiden çok aday algoritma olabilir ve bunlardan en iyisini seçmek, ya da iyiden kötüye sıralamak isteyebiliriz.

Daha da genel durumda istediğimiz, bu karşılaştırmayı tek bir veri üzerinde değil birden çok veri kümesi ya da uygulama üzerinde yapmak olabilir.

B. Tepki Değişkeninin Seçilmesi

Başarımı nasıl ölçeceğimize karar vermemiz gerekir. Çoğu zaman sınıflandırma hatası ya da bağlanımda ortalama kare hata kullanılır. Eğer bir yitim işlevi tanımlıysa bu bir risk ölçütü olabilir. Bilgi erişimi uygulamalarında, 19.7. bölümde göreceğimiz

kesinlik ve anma değerleri kullanılır. Bedel duyarlı uygulamalarda, çıktı yanında sistem özellikleri, örneğin karmaşıklığı, hesaba katılabilir.

C. Etkenlerin ve Düzeylerinin Seçimi

Etkenlerin ne olduğu çalışmanın amacına bağlıdır. Algoritmayı sabitleyip en iyi üst-parametre değerlerini ararsak onlar etken olur. Algoritmaları karşılaştırırken algoritma bir etkendir. Farklı veri kümeleri varsa veri de başka bir etken olur.

Bir etkenin alabileceği düzeyler, iyi bir değeri kaçırmayacak kadar sık ve gereksiz hesaba neden olmayacak kadar seyrek olmalıdır. Etken düzey değerlerini mutlak değil veriye göre ölçeklenmiş seçebilmek her zaman yeğlenir. Örneğin, en yakın k komşu algoritmasında k değeri, 1, 3, 5 olabilecekken, Parzen pencerelerinde h değerini, mutlak olarak 1.0, 2.0, almak yerine veriden bir ölçek belirleyip (örneğin, her örnekle ona en yakın öbür örnek arasındaki uzaklığın bütün örnekler üzerindeki ortalaması) bunun katlarını (yarısı, kendisi, iki katı) denemek daha yeğdir.

Etken düzeylerini belirlerken geçmiş deneyim her ne kadar genelde işe yarasa da, tüm etkenleri ve etken değerlerini denemek ve hiçbir olasılığı atlamamak önemlidir.

D. Deney Yordamının Seçimi

Etkenler arasında etkileşim olmadığına emin değilsek en iyisi etkensel tasarımdır. Çoğaltma sayısı veri büyüklüğüne bağlıdır; büyük olduğunda az, küçük olduğunda çok olmalıdır; bunu az sonra yeniden örneklemeden bahsederken göreceğiz. Çok az çoğaltma yapıldığında dağılım kestirebilecek kadar örnek olmadığından bir dağılımın (örneğin, Gauss) varsayılması ve sınamada kullanılması zor olur.

Daha önce belirttiğimiz gibi genelde, bir veri kümesi verildiğinde, önce bir parçayı deneme için ayırır, kalandan yeniden örneklemeyle birden çok öğrenme ve geçerleme kümesi oluştururuz. Bu bölmenin nasıl yapıldığı önemlidir. Küçük veri kümeleri üzerinde değişkenlik yüksek olacağından farklar anlamlı olmayabilir ve kesin bir sonuç ortaya çıkmayabilir.

Üretilmiş yapay verilerden kaçınmak ve gerçek uygulamalarda, gerçek koşullarda örneklenmiş verileri kullanmak gerekir. Bir ya da iki boyutlu öğretici veriler, sezgi kazanmak konusunda fayda sağlayabilseler de, algoritmaların yüksek boyutlu uzaylardaki davranışları çok farklı olabilir.

E. Deneyin Yürütülmesi

Çok düzeyli birçok etkeni olan etkensel tasarımlı bir deneyi yürütmeye başlamadan önce bazı etken birleşimleri için birkaç deneme yapmak her şeyin düşünüldüğü gibi olduğunu onaylamak açısından yararlıdır. Büyük bir deneyde, ara sonuçların (ya da rastsal sayı üretecinin ilk değerinin) saklanması deneyin bir kısmının gerektiğinde yinelenebilmesini sağlar. Bir deneyin tüm sonuçları gerekirse yeniden oluşturulabilmelidir. Birçok etkeni olan ve dolayısıyla çok zaman alacak büyük bir deneyin gerçeklenmesinde yazılım yaşlanmasının olası zararlı etkisinin farkında olmak gerekir.

Deneyi tasarlayan kişinin deneyi önyargısız yürütmesi önemlidir. Araştırmacı, kendi algoritmasını başkasıyla karşılaştırırken ikisini de aynı özenle değerlendirmeli, ikisine de eşit fırsat vermelidir. Büyük ölçekli bir çalışmada, deneyleri yapan ve sonuçları değerlendiren kişilerin, programları ve algoritmaları geliştiren kişilerden farklı olması düşünülebilir.

Kişinin olabildiğince kendi yazılım kütüphanesini oluşturmaktan kaçınıp güvenilir kaynaklardan edinilmiş yazılımları kullanması yeğlenir; böylesi yazılımlar, daha önce başkaları tarafından kullanılmış ve sınanmıştır.

Herhangi bir yazılım geliştirme çalışmasında olduğu gibi iyi belgelemenin, özellikle birden çok kişi ortak çalışıyorsa, azımsanamayacak faydaları vardır. Kaliteli yazılım mühendisliği için geliştirilmiş tüm yöntemler, yapay öğrenme çalışmalarında da kullanılmalıdır.

F. Verinin İstatistiksel Çözümlemesi

Sonucun öznel ya da rastsallıktan etkilenmiş olmaması için deneyle elde edilen verinin istatistiksel yöntemler kullanılarak çözümlenmesi gerekir. Bunun için sorularımızı, veriyle ilgili savlarımızı tanımlayan denence sınamaları olarak yazar ve deney sonunda elde edilen verilerin bu denenceleri destekleyip desteklemediğine bakarız. Örneğin, "A algoritması, B algoritmasından daha başarılı mıdır?" sorusu, "A algoritmasıyla eğitilen modellerin beklenen hatası, B algoritmasıyla eğitilen modellerin beklenen hatasından anlamlı olarak daha küçük müdür?" diye tanımlanır.

Sonuçları görselleştirmek değerlendirmeyi kolaylaştırır. Örneğin, hata dağılımları çubukçizitlerle gösterilebilir, başarım eğrileri çizilebilir, ya da başka görselleştirme araçları kullanılabilir.

G. Sonuçlar ve Öneriler

Tüm veri toplanıp çözümlendikten sonra nesnel sonuçlara varılır. Sıkça varılan bir sonuç, daha fazla deney yapmanın gerekmesidir. Çoğu istatistiksel ve yapay öğrenme ya da veri madenciliği çalışması yinelemelidir. Bu yüzden tüm deneyleri hemen yapmamak ve ilk deneylerde eldeki kaynakların (örneğin, zaman) en fazla yüzde 25'inin kullanılması salık verilir (Montgomery 2005). İlk deneyler, ilk gözlemler içindir. Yine bu yüzden çalışmaya büyük umutlarla, patrona ya da tez danışmanına gerçeklenmesi zor sözler vererek başlamamak gerekir.

İstatiktiksel sınamalar bir savın doğru ya da yanlış olduğunu söylemez, yalnızca veri tarafından ne kadar desteklendiğini söyler. Kesin bir sonuca varamama, ya da vardığımız sonucun yanlış olma olasılığı her zaman vardır, özellikle de elimizdeki veri küçük ya da gürültülüyse.

Sonuçlar beklediğiniz gibi olmazsa bunun nedeninin iyice araştırılması gerekir. Örneğin, gözbebeğimiz A algoritmasının bazı durumlarda neden iyi çalışmadığını anladığımızda, A'nın daha iyi bir sürümünü oluşturmak için aklımıza iyi bir fikir gelebilir. Her iyileştirme, önceki sürümdeki bir eksiklik sayesindedir; bulduğumuz her eksiklik, bize iyileştirme yapabileceğimiz yeri gösteren bir ipucudur.

Önceki deneylerimizin sonuçlarını tam olarak inceleyip öğrenebileceğimiz her şeyi öğrenmeden bir sonraki deneyi tasarlamaya geçmememiz gerekir. Fikirler ucuzdur, deneyip işe yaradıkları gösterilmeden değersizdir; deney yapmak pahalıdır.

19.6 Çapraz Geçerleme ve Yeniden Örnekleme

Çoğaltmada, verilen $\mathcal{X}$ kümesinden (bir kısmını deneme için ayırdıktan sonra) birden çok öğrenme ve geçerleme kümesi çifti oluşturmamız gerekir. Eğer $\mathcal{X}$ yeterince büyükse, önce rastgele K parçaya, ardından her parçayı rastgele ikiye bölüp K tane öğrenme/geçerleme küme çifti oluşturabiliriz. Ne yazık ki çoğu zaman eldeki veri bunu yapabileceğimiz kadar büyük değildir ve aynı kümeyi birden çok kez, farklı işler için kullanmamız gerekir; buna *çapraz geçerleme* denir. Buradaki çekince, aynı veriyi birden çok kez kullanmaktan dolayı onları kullanarak hesaplanan istatistiklerin bağımlı olmasıdır.

$\mathcal{X}$ kümesinden K tane öğrenme/geçerleme küme çifti $\{\mathcal{T}_i, \mathcal{V}_i\}_{i=1}^{K}$ oluştururken, hesaplanan istatistiklerin (örneğin, hata değerlerinin) daha güvenilir olması için öğrenme ve geçerleme kümelerinin olabildiğince büyük olmasını, bağımlılığı azaltmak içinse olabildiğince az örtüşmelerini isteriz. Bu farklı altkümelerde sınıf önsel olasılıklarının korunması gerekir; buna *katmanlaştırma* diyoruz: Eğer bir sınıfın örnekleri tüm verinin beşte biri kadarsa, her altkümede de yaklaşık olarak beşte biri kadar olmalıdır.

19.6.1 K Kat Çapraz Geçerleme

K *kat çapraz geçerlemede*, $\mathcal{X}$ veri kümesi rastgele olarak K tane eşit büyüklükte $\mathcal{X}_i, i = 1, \ldots, K$ parçalarına bölünür. Her *katta*, K parçadan biri geçerleme için ayrılırken kalan $K - 1$ parça birleştirilerek öğrenme kümesi oluşturulur; böylece elimize K tane çift geçer:

$$\begin{array}{ll} \mathcal{V}_1 = \mathcal{X}_1 & \mathcal{T}_1 = \mathcal{X}_2 \cup \mathcal{X}_3 \cup \cdots \cup \mathcal{X}_K \\ \mathcal{V}_2 = \mathcal{X}_2 & \mathcal{T}_2 = \mathcal{X}_1 \cup \mathcal{X}_3 \cup \cdots \cup \mathcal{X}_K \\ \vdots & \\ \mathcal{V}_K = \mathcal{X}_K & \mathcal{T}_K = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \cdots \cup \mathcal{X}_{K-1} \end{array}$$

Burada iki tane sorun vardır: Birincisi, öğrenme kümesini büyük tutmaya çalışırken geçerleme kümesinin küçük kalabilmesidir. İkinci sorun da öğrenme kümelerinin oldukça örtüşmeleridir; herhangi iki öğrenme kümesinde $K - 2$ parça ortaktır.

K sıklıkla 10 ya da 30 alınır. K arttıkça öğrenme kümelerinin yüzdesi artar ve daha gürbüz kestiriciler oluşur, ama geçerleme kümeleri küçülür. Ayrıca, K artarken gereken toplam öğrenme ve geçerleme karmaşıklığı da artar. Veri kümesi büyüklüğü olan N artarken K küçülebilir; N küçükse öğrenmede yeterli veri olabilmesi için K değerinin daha büyük olması gerekir. En uç durumda $K = N$ alınarak her katta geçerleme için yalnızca bir örnek dışarıda bırakılırken $N - 1$ örnekle

öğrenme yapılır. Bu, tıbbi tanı gibi etiketli verinin bulunmasının çok zor olduğu uygulamalarda kullanılır. Birini dışarıda bıraktığımızda katmanlaştırma yapamayız.

Son yıllarda hesaplamanın gittikçe ucuzlamasıyla, K kat çapraz geçerleme, birden çok kez, örneğin, 10×10 kat, yapılabilir olmuştur. Böylece ortalamaların ortalamasını kullanıp daha güvenilir hata kestirimleri yapılabilir (Bouckaert 2003).

19.6.2 5×2 Çapraz Geçerleme

Dietterich (1998), aynı büyüklükte öğrenme ve geçerleme kullanan 5×2 *çapraz geçerlemeyi* önermiştir. $\mathcal{X}$ veri kümesini rastgele olarak ikiye, $\mathcal{X}_1^{(1)}$ ve $\mathcal{X}_1^{(2)}$ olarak bölünür ve bunlar bizim ilk öğrenme ve geçerleme kümelerimiz olur: $\mathcal{T}_1 = \mathcal{X}_1^{(1)}$ ve $\mathcal{V}_1 = \mathcal{X}_1^{(2)}$. Sonra bu iki parçanın rollerini değiş tokuş edip ikinci çifti oluştururuz: $\mathcal{T}_2 = \mathcal{X}_1^{(2)}$ ve $\mathcal{V}_2 = \mathcal{X}_1^{(1)}$. Bu ilk kattır; $\mathcal{X}_i^{(j)}$, i. kattaki j. parçayı gösterir.

İkinci katı oluşturmak için $\mathcal{X}$ kümesini bir kez daha rastgele iki parçaya, $\mathcal{X}_2^{(1)}$ ve $\mathcal{X}_2^{(2)}$, böleriz. Bu bölme, $\mathcal{X}$ kümesinden geri yerine koymayarak örneklemeyle gerçeklenir: $\mathcal{X}_1^{(1)} \cup \mathcal{X}_1^{(2)} = \mathcal{X}_2^{(1)} \cup \mathcal{X}_2^{(2)} = \mathcal{X}$. Ardından yine rolleri değiş tokuş ederek ikinci çifti oluştururuz. Bunu üç kat için daha yapıp, her kattan yine iki çift çıkardığımda elimize toplam beş kattan on tane öğrenme ve geçerleme çifti geçer:

$$
\begin{array}{ll}
\mathcal{T}_1 = \mathcal{X}_1^{(1)} & \mathcal{V}_1 = \mathcal{X}_1^{(2)} \\
\mathcal{T}_2 = \mathcal{X}_1^{(2)} & \mathcal{V}_2 = \mathcal{X}_1^{(1)} \\
\mathcal{T}_3 = \mathcal{X}_2^{(1)} & \mathcal{V}_3 = \mathcal{X}_2^{(2)} \\
\mathcal{T}_4 = \mathcal{X}_2^{(2)} & \mathcal{V}_4 = \mathcal{X}_2^{(1)} \\
\vdots & \\
\mathcal{T}_9 = \mathcal{X}_5^{(1)} & \mathcal{V}_9 = \mathcal{X}_5^{(2)} \\
\mathcal{T}_{10} = \mathcal{X}_5^{(2)} & \mathcal{V}_{10} = \mathcal{X}_5^{(1)}
\end{array}
$$

Tabii ki istersek bunu beş kattan daha çok kez yaparak daha çok öğrenme ve geçerleme çifti de oluşturabiliriz ama Dietterich (1998), beşten fazla yapıldığında kümelerin fazla örtüşemeye başlamasından dolayı hesaplanan istatistiklerin (hataların) bağımlılaştığını ve ek bilgi getirmediğini söyler. Gerçi beş kat için bile bu doğrudur, ama beşten de az kat yaparsak üzerinde sınama yapabilecek kadar verimiz olmaz.

19.6.3 Yerine Geri Koyarak Örnekleme

Tek bir veri kümesinden birden çok veri kümesi oluşturmak için kullanılabilecek bir başka yöntem, *yerine geri koyarak örneklemedir*. Adı üstünde, veriden rastgele örnek seçer, kaydeder, ama geri de koyarız; böylece aynı veriyi birden çok kez seçebildiğimiz gibi bazı verileri de hiç seçmeyebiliriz. 17.6. bölümde model birleştirmeyi konuşurken farklı modeller için bu yöntemle aynı veri kümesinden farklı kümeler oluşturmayı

Tablo 19.1: İki sınıf için hata dizeyi.

	Öngörülen Sınıf		
Gerçek Sınıf	Artı	Eksi	Toplam
Artı	*da*: doğru artı	*ye*: yanlış eksi	*a*
Eksi	*ya*: yanlış artı	*de*: doğru eksi	*e*
Toplam	a'	e'	N

örmüştük. Bu biçimde oluşturulmuş veriler, çapraz geçerlemeyle oluşturulmuş ve-
lere göre daha fazla örtüşürler ve bu yüzden üzerlerinden hesaplanan istatistikler
aha fazla bağımlı olurlar; yine de bu yaklaşım, çok küçük veri kümelerinde sıklıkla
ullanılır.

N örnekten bu biçimde N kez öğrenme örneği seçtiğimizi düşünelim; içinden
nekleme yaptığımız küme de geçerleme kümemiz olsun. Herhangi bir örneği seçme
asılığımız $1/N$, seçmeme olasılığımız da $1 - 1/N$ olur. N kez seçim yaptıktan sonra
r örneğin hiç seçilmemiş olma olasılığı

$$\left(1 - \frac{1}{N}\right)^N \approx e^{-1} = 0.368$$

ır. Bu da, öğrenme kümesinin ilk kümenin yaklaşık olarak yüzde 63.2'sini içerdiğini,
verinin yüzde 36.8'inin öğrenmede hiç kullanılmadığını ve dolayısıyla sonuç hata
stiriminin kötümser olacağını gösterir. Bunu engellemenin yolu çoğaltma yapmak,
ni bu işlemi birçok kez yapıp ortalamalara bakmaktır.

19.7 Sınıflandırma Başarımının Ölçülmesi

ıflandırmada, özellikle iki sınıf için çeşitli başarım ölçüleri önerilmiştir. Tablo
1'de görüldüğü gibi dört olası sonuç vardır: Artı bir örnek için sınıflandırıcı da
kararı verirse bu *doğru bir artı*, sınıflandırıcı eksi kararı verirse bu *yanlış bir*
olur. Eksi bir örnek için sınıflandırıcı artı derse bu *yanlış bir artı*, eksi derse bu
ru bir eksi olur.

Bazı iki sınıflı uygulamalarda, iki sınıf için yapılan hatalar, yani yanlış artılarla
ış eksiler arasında yitim açısından fark vardır. Bu farkı öne çıkaran başarım
tleri tablo 19.2'de verilmiştir. İnsanların sesle giriş yaptığı bir kimlik doğrulama
ılaması düşünelim. Bu durumda yanlış artı, sahtekar birinin yanlışlıkla kabulu,
ış eksiyse geçerli bir kullanıcının reddine karşılık gelir. Tabii ki bu ikisi aynı
lükte değildir; ilki çok daha kötüdür ve bu yüzden ikinciye göre çok çok daha
ek olması istenir. Bu uygulamada, doğru artı (*da*) oranı, ya da *isabet oranı*,
rli kullanıcıların ne kadarının kabul edildiği, yanlış artı (*ya*), ya da *yanlış alarm*
ıysa kabul edilen sahtekarların oranıdır.

Tablo 19.2: İki sınıflı uygulamalarda kullanılan başarım ölçüleri.

Ad	Kural
hata	$(ya + ye)/N$
başarı	$(da + de)/N = 1-$hata
da oranı	da/a
ya oranı	ya/e
kesinlik	da/a'
anma	$da/a = da$ oranı
duyarlılık	$da/a = da$ oranı
özgüllük	$de/e = 1-ya$ oranı

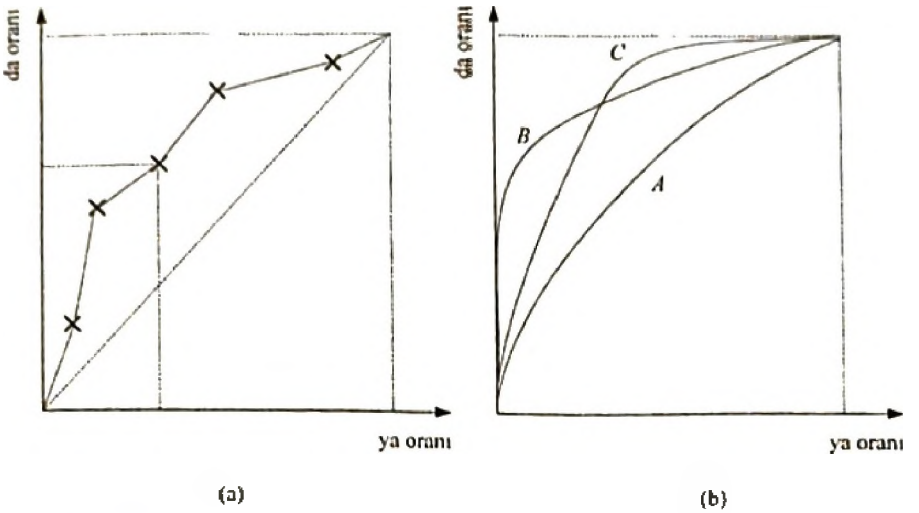
Sınıflandırıcının çıktı olarak $\hat{P}(C_1|x)$ ile gösterdiğimiz artı sınıfın sonsal olasılığını kestirdiğini varsayalım. Bu durumda eksi sınıf olasılığı $\hat{P}(C_2|x) = 1 - \hat{P}(C_1|x)$ olur. Eğer $\hat{P}(C_1|x) > \theta$ ise artı sınıfı seçeriz. Eşik θ, 1 değerine yaklaştıkça artı sınıfı seçme olasılığımız azalır; bu durumda hem doğru artı, hem de yanlış artıların sayısı az olur. θ değerini aşağı çektikçe doğru artıların sayısı artar, ama bu çoğu zaman beraberinde yanlış artıların sayısının artmasını da getirir.

Farklı θ değerleri için (da, ya) oran çiftlerini hesaplayıp *alıcı işletim özellikleri* eğrisini, şekil 19.3a'daki gibi çizdirebiliriz. Farklı θ değerleri iki hata türü için farklı yitim dizeylerine karşılık gelir (alıştırma 1) ve alıcı işletim özellikleri eğrisi, yitim dizeyi değiştiğinde sınıflandırıcının davranışının nasıl değiştiğini gösterir.

Mükemmel sınıflandırıcının da oranı **1**, ya oranı 0 olduğundan, eğrisi sol üst köşeye yaklaşan sınıflandırıcı daha iyidir. Köşegen üzerinde, yanlışlar ve doğrular birbirine eşittir ve bu olabilecek en kötü (rastsal) sınıflandırıcıdır (köşegenin altındaki sınıflandırıcının çıktısını ters çevirerek köşegenin üstüne çıkarabiliriz). Elimizde iki sınıflandırıcı varsa ve bunlardan birinin eğrisi her zaman ötekinin üstündeyse üstte olan altta kalandan daha iyidir; eğriler kesişiyorsa, şekil 19.3b'deki gibi, sınıflandırıcılar farklı eşik değerlerinde (farklı yitim dizeyleri için) iyidir.

Eğri sayesinde görsel değerlendirebiliriz; *eğri altında kalan alanı* hesaplayarak sınıflandırıcının farklı yitim dizeylerine göre ortalama başarımını tek bir sayıyla gösterebiliriz. Mükemmel sınıflandırıcının eğri altında kalan alanı 1 olur ve farklı sınıflandırıcılar bu alan değerlerine göre sıralanabilirler.

Bilgi erişimi uygulamalarında, bir sorgu verildiğinde (örneğin, anahtar sözcüklerle), veritabanına kaydedilmiş kayıtlar içinden ilgili olduğu düşünülenler getirilir. Getirip getirmeme (öngörülen sınıfın artı ya da eksi olduğu) kararı, sorguya bağlı olarak iki sınıflı bir sınıflandırıcı tarafından verilir. Veri tabanı içinde sorguyla ilgili olan kayıtlar (gerçek sınıfı artı olanlar) vardır ve sınıflandırıcı bunların bir kısmını getirir (doğru artılar) ama hepsini getirmeyebilir (yanlış eksiler); bunlardan başka sorguyla gerçekte ilgisiz olan kayıtları da getirilebilir (yanlış artılar) (bkz. şekil 19.4a). *Kesinlik*, getirilmiş ve ilgili kayıtların sayısının toplam getirilmiş kayıtların

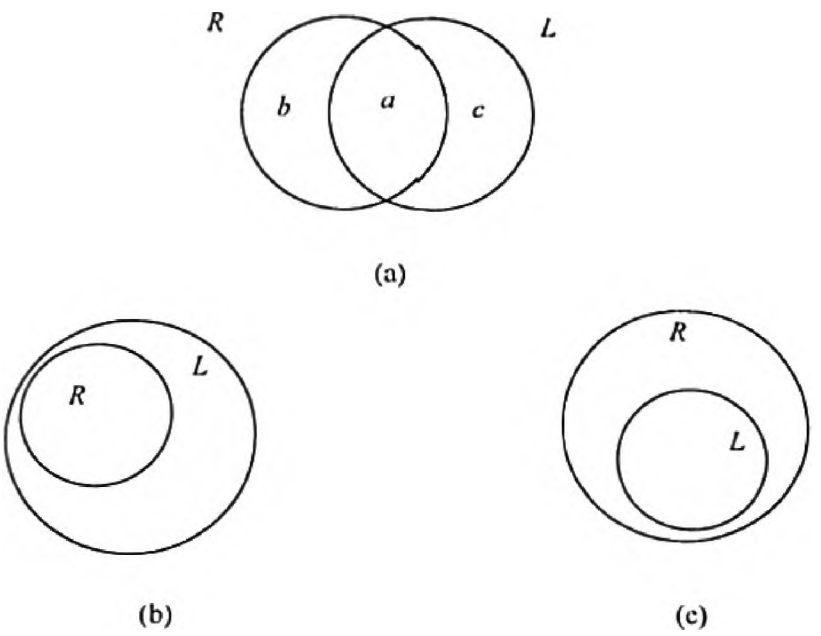


Şekil 19.3: (a) Örnek alıcı işletim özellikleri eğrisi. Her sınıflandırıcının karar verirken kullandığı bir θ eşik değeri vardır ve bu eşiği değiştirerek bu eğri oluşturulur. Doğru artı ve yanlış artıların görece önemlerine göre bu eğri üstünde bir nokta seçeriz. Eğrinin altında kalan alan, eğriyi tek bir değer olarak özetlememizi sağlar. (b) Bir sınıflandırıcının eğrisi sol üst köşeye ne kadar yaklaşırsa o kadar iyidir; bu durumda eğri altında kalan alan 1 değerine yaklaşır. B ve C modelleri, A'dan daha iyidir ve B ile C, farklı eşik değerlerinde (yitim dizeyleri) en iyidir.

sayısına oranıdır; kesinlik 1 olduğunda getirilmiş tüm kayıtlar ilgilidir ama bazı ilgili kayıtlar getirilmemiş olabilir. *Anma*, getirilmiş ve ilgili kayıtların sayısının toplam ilgili kayıtların sayısına oranıdır; anma 1 olduğunda, tüm ilgili kayıtlar getirilmiştir ama ilgisiz olmasına rağmen getirilmiş kayıtlar da olabilir (şekil 19.4c). Getirip getirmeme kararını veren sınıflandırıcı bir eşiğe göre karar verdiği için farklı eşik değerlerinde hesaplanan kesinlik ve anma değerleri bir kesinlik-anma eğrisi olarak çizilebilir.

Duyarlılık ve *özgüllük*, sıkça kullanılan başka bir başarım ölçüt çiftidir. Duyarlılık, doğru artı oranı ve anma ile aynıdır; özgüllükse eksi örnekleri ne kadar iyi belirleyebildiğimizi ölçer; doğru eksilerin sayısının toplam eksilere oranıdır ve 1 eksi yanlış alarm oranına eşittir. Öteki ölçüt çiftleri gibi duyarlılık ve özgüllük değerleri de farklı eşikler için hesaplanıp bir eğri olarak çizilebilir.

$K > 2$ sınıf olduğunda eğer 0/1 yitim işlevini kullanıyorsak, *sınıf hata dizeyinin* (i, j) öğesi, C_i sınıfından olmasına rağmen C_j sınıfına atanan örneklerin sayısını gösterir. Hatasız bir sınıflandırıcıda köşegen dışındaki tüm öğeler 0 değerini alır. Bu $K \times K$ dizey, farklı hata tiplerini ve hangi sınıf çiftlerinin sıklıkla karıştırıldığını gösterir. Bazı durumlarda, K farklı sınıf için, her biri bir sınıfı öteki tüm sınıfların birleşiminden ayıran K tane iki sınıf sınıflandırıcı tanımlayıp başarımlarını o biçimde de ölçebiliriz.



Şekil 19.4: (a) Kesinlik ve anmanın kümeler olarak tanımlanması: R: getirilmiş kayıtlar, L: ilgili kayıtlar. Kesinlik $= a/(a+b)$, Anma: $a/(a+c)$. (b) Kesinlik 1 iken getirilmiş kayıtlar ilgilidir ama bazı ilgili kayıtlar getirilmemiş olabilir. (c) Anma 1 iken tüm ilgili kayıtlar getirilmiştir ama bazı ilgisiz kayıtlar da getirilmiş olabilir.

19.8 Aralık Kestirimi

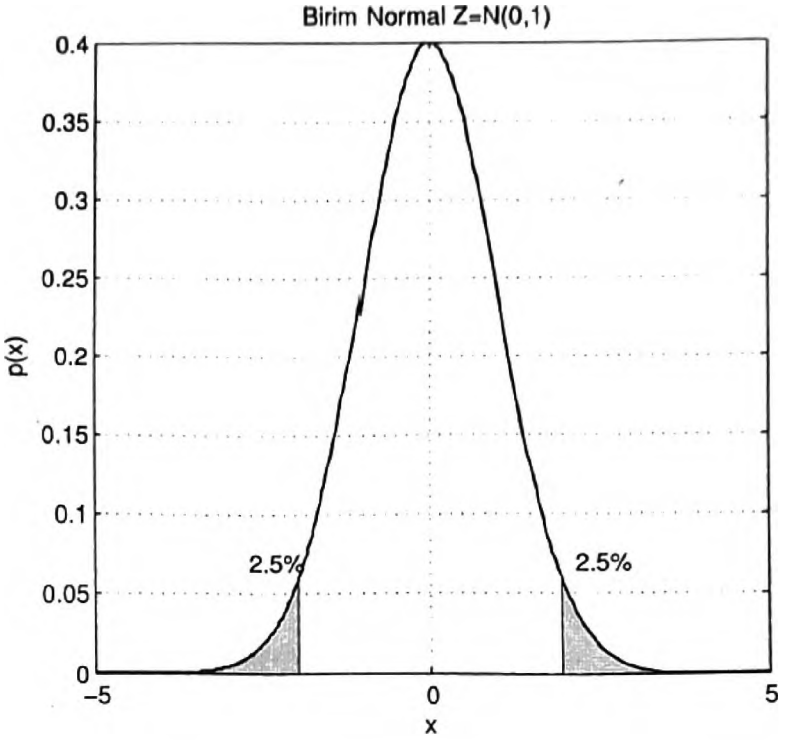
Sınama yöntemlerini tartışmaya geçmeden önce hızla *aralık kestirimi* konusunu gözden geçirelim. Bir nokta kestirici, örneğin en büyük olabilirlik kestiricisi, θ parametresi için bir değer kestirir. Aralık kestirimindeyse θ parametresinin belirli bir güvenle içinde olduğu bir aralık kestirilir. Aralık kestirimi, nokta kestiricinin olasılık dağılımını kullanır.

Örneğin, bir normal dağılımının ortalaması olan μ değerini, verilmiş bir $\mathcal{X} = \{x^t\}_{t=1}^N$ örneklemini kullanarak kestirmek istediğimizi düşünelim. Örneklem ortalaması olan $m = \sum_t x^t/N$, μ için nokta kestiricidir ve normal dağılmış rastsal değişkenlerin toplamı olduğundan normal dağılmıştır: $m \sim \mathcal{N}(\mu, \sigma^2/N)$. Ortalamasını çıkarıp standart sapmasına bölerek *birim normal dağılmış* bir istatistiğe çevrilebilir:

$$\frac{(m-\mu)}{\sigma/\sqrt{N}} \sim \mathcal{Z} \tag{19.1}$$

Birim normal dağılmış $\mathcal{Z}$ rastsal değişkeninin yüzde 95'i $(-1.96, 1.96)$ aralığındadır: $P\{-1.96 < \mathcal{Z} < 1.96\} = 0.95$. Bu durumda (bkz. şekil 19.5)

$$P\left\{-1.96 < \sqrt{N}\frac{(m-\mu)}{\sigma} < 1.96\right\} = 0.95$$



Şekil 19.5: Birim normal dağılımın yüzde 95'i -1.96 ile 1.96 arasındadır.

ya da

$$P\left\{m - 1.96\frac{\sigma}{\sqrt{N}} < \mu < m + 1.96\frac{\sigma}{\sqrt{N}}\right\} = 0.95$$

yazılabilir. "Yüzde 95 güvenle," μ örneklem ortalamasına $1.96\sigma/\sqrt{N}$ uzaklıktadır. Bu, *iki yanlı güven aralığıdır*. Yüzde 99 güvenle, $\mu \in (m-2.58\sigma/\sqrt{N}, m+2.58\sigma/\sqrt{N})$; daha fazla güven istersek aralık büyür. Örneklem büyüklüğü olan N değeri artınca, m kestiricisinin değişkenliği azaldığından aralık da küçülür.

Aralığı istenen herhangi bir $1-\alpha$ güven değeri için genelleyebiliriz. z_α, öyle bir değer olsun ki, birim normal dağılmış bir rastsal değişkenin z_α'dan büyük bir değer alma olasılığı α olsun:

$$P\{\mathcal{Z} > z_\alpha\} = \alpha,\ 0 < \alpha < 1$$

$\mathcal{Z}$ ortalama çevresinde bakışımlı dağıldığı için $z_{1-\alpha/2} = -z_{\alpha/2}$ olur ve $P\{X < -z_{\alpha/2}\} = P\{X > z_{\alpha/2}\} = \alpha/2$ olarak yazabiliriz. Böylece, istenen bir güven aralığı $1-\alpha$ için

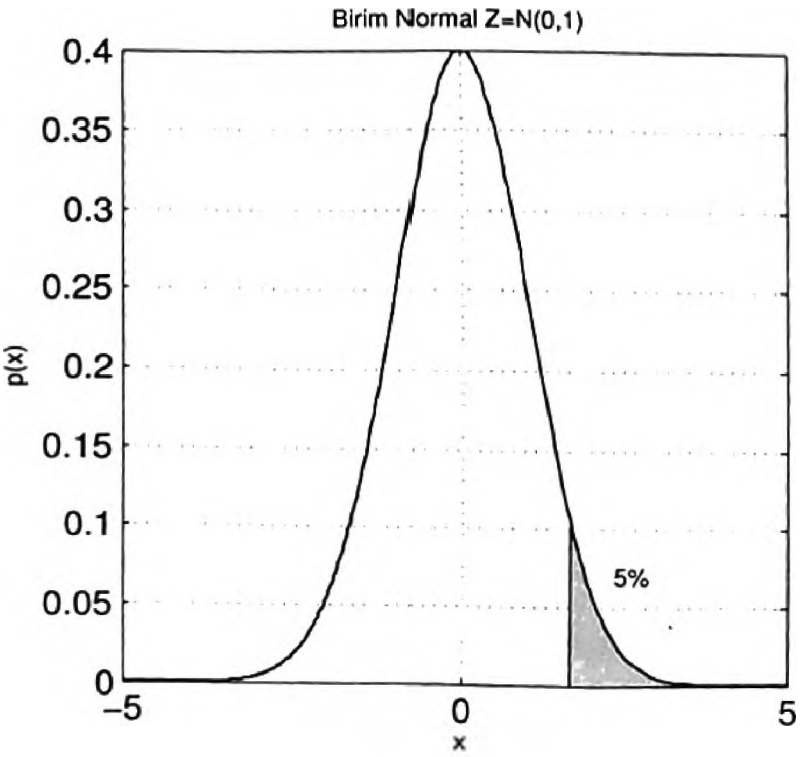
$$P\left\{-z_{\alpha/2} < \mathcal{Z} < z_{\alpha/2}\right\} = 1 - \alpha$$

ve

$$P\left\{-z_{\alpha/2} < \sqrt{N}\frac{(m-\mu)}{\sigma} < z_{\alpha/2}\right\} = 1 - \alpha$$

ya da

$$P\left\{m - z_{\alpha/2}\frac{\sigma}{\sqrt{N}} < \mu < m + z_{\alpha/2}\frac{\sigma}{\sqrt{N}}\right\} = 1 - \alpha \qquad (19.2)$$



Şekil 19.6: Birim normal dağılımın yüzde 95'i 1.64'e kadardır.

yazabiliriz ve bu biçimde μ için iki yanlı, yüzde $100(1-\alpha)$ güven aralığı herhangi bir α için tanımlanabilir.

Benzer biçimde, $P\{Z < 1.64\} = 0.95$ olduğunu bildiğimizden (bkz. şekil 19.6)

$$P\left\{\sqrt{N}\frac{(m-\mu)}{\sigma} < 1.64\right\} = 0.95$$

ya da

$$P\left\{m - 1.64\frac{\sigma}{\sqrt{N}} < \mu\right\} = 0.95$$

yazabiliriz ve $(m - 1.64\sigma/\sqrt{N}, \infty)$, μ için yüzde 95 *tek yanlı güven aralığıdır* ve alt sınırı belirler. Genellersek, μ için yüzde $100(1-\alpha)$ tek yanlı güven aralığı

$$P\left\{m - z_\alpha\frac{\sigma}{\sqrt{N}} < \mu\right\} = 1 - \alpha \tag{19.3}$$

biçiminde yazılır. Üst sınır belirleyen öbür tek yanlı aralık da benzer biçimde tanımlanır.

Bu aralıkları hesaplarken σ değerini kullandık, yani değişkenliği bildiğimizi varsaydık. Bilmediğimizde yerine örneklem değişkenliğini

$$S^2 = \sum_t (x^t - m)^2/(N-1)$$

kullanmamız gerekir. $x^t \sim \mathcal{N}(\mu, \sigma^2)$ ise $(N-1)S^2/\sigma^2$, $N-1$ serbestlik dereceli ki kare bir dağılımdan gelir ve m ile S^2 bağımsızdır. $\sqrt{N}(m-\mu)/S$ ise $N-1$ serbestlik dereceli t dağılmış bir rastsal değişkendir (bölüm A.3.7):

$$\frac{\sqrt{N}(m-\mu)}{S} \sim t_{N-1} \tag{19.4}$$

Bu durumda aralığı belirlerken birim normal $\mathcal{Z}$ yerine t *dağılımını* kullanırız ve herhangi bir $\alpha \in (0, 1/2)$ için iki yanlı aralığı yazabiliriz:

$$P\left\{t_{1-\alpha/2,N-1} < \sqrt{N}\frac{(m-\mu)}{S} < t_{\alpha/2,N-1}\right\} = 1-\alpha$$

t dağılımı da ortalamanın çevresinde bakışımlıdır: $t_{1-\alpha/2,N-1} = -t_{\alpha/2,N-1}$. Böylece aralığımızı yazabiliriz:

$$P\left\{m - t_{\alpha/2,N-1}\frac{S}{\sqrt{N}} < \mu < m + t_{\alpha/2,N-1}\frac{S}{\sqrt{N}}\right\} = 1-\alpha$$

Benzer biçimde tek yanlı güven aralıkları da tanımlanabilir. t dağılımı, birim normal dağılımına göre daha yaygındır (kuyrukları daha uzundur) ve bu yüzden t dağılımını kullanarak hesapladığımız aralık daha geniştir; bu da akla yakındır çünkü değişkenlik de örneklemden kestirildiği için bilinmezlik artar.

19.9 Denence Sınaması

Bazı uygulamalarda, parametreleri kestirmek yerine parametreler hakkındaki savları, ya da denenceleri, sınamak isteriz. Örneğin ortalamayı kestirmek yerine, ortalamanın 0.02'den küçük olup olmadığını sınamak isteriz. Eğer elimizdeki örneklem bu savla tutarlıysa denenceyi "reddetmeyiz," eğer desteklemiyorsa "reddederiz." Ama böyle bir karar vermek savın doğru ya da yanlış olduğunu söylemez; yalnızca savın eldeki veriyle belirli bir güven değerine göre tutarlı olup olmadığını belirtir.

Denence sınamasında yaklaşım şöyledir: Denence doğru olduğunda belli bir dağılımı olacak bir istatistik tanımlarız. Eğer eldeki veriden bu istatistik için hesaplanan değer çok küçük olasılıkla olabilecekse denenceyi reddederiz, yoksa reddedemeyiz.

Elimizde ortalamasını bilmediğimiz ama değişkenliğini bildiğimiz normal $\mathcal{N}(\mu, \sigma^2)$ bir dağılımdan çekilmiş bir örneklem olsun ve μ hakkında, örneğin, belirli bir sabit μ_0 değerine eşit olduğu denencesini sınamak isteyelim. Bu, H_0 diye yazılır ve *sıfır denencesi* denir:

$$H_0 : \mu = \mu_0$$

Tersi de karşıt denencedir:

$$H_1 : \mu \neq \mu_0$$

m, μ parametresinin nokta kestirimidir ve eğer m, μ_0 değerinden çok uzaksa denenceyi reddetmek akla yatkındır; ama ne kadar uzak? İşte aralık kestirimi burada

Tablo 19.3: I. ve II. tür hata ve sınamanın gücü.

	Karar	
Gerçek	Reddedememe	Reddetme
Doğru	Uygun	I. tür hata
Yanlış	II. tür hata	Uygun (güç)

kullanılır. Eğer μ_0, μ için tanımladığımız yüzde $100(1-\alpha)$ güven aralığının dışındaysa denenceyi, α *anlamlılık düzeyinde* reddederiz. Bir başka deyişle,

$$\frac{\sqrt{N}(m-\mu_0)}{\sigma} \in (-z_{\alpha/2}, z_{\alpha/2}) \tag{19.5}$$

ise denenceyi reddedemeyiz. Aralığın iki yanında da sınır olduğundan buna *iki yanlı sınama* denir.

Denence doğru olduğunda reddedersek buna *I. tür hata* denir ve sınamadan önce belirlediğimiz α değeri, ne kadar I. tür hataya izin verebileceğimizi belirler; sıklıkla $\alpha = 0.1, 0.05$, ya da 0.01 değerlerinden biri alınır (bkz. tablo 19.3). *II. tür hata*, gerçek ortalama değer olan μ, μ_0 değerinden farklıyken red kararı vermemek, yani yanlış olan bir denenceyi reddedememektir. Sıfır denencesinin reddedilmeme olasılığı gerçek μ değerine bağlıdır:

$$\beta(\mu) = P_\mu \left\{ -z_{\alpha/2} \leq \frac{m-\mu_0}{\sigma/\sqrt{N}} \leq z_{\alpha/2} \right\} \tag{19.6}$$

$1-\beta(\mu)$ işlevine *güç işlevi* denir ve yanlış olan bir denenceyi reddetmeme olasılığıdır. Gerçek μ, μ_0 değerine yaklaştıkça II. tür hata olasılığı artar ve istersek, belirli bir $\delta = |\mu - \mu_0|$ farkı istenen güçle yakalayabilmek için gereken örneklem büyüklüğünü hesaplayabiliriz.

Tek yanlı sınamada, sıfır denencesi (ve karşıt denence) tek yanlıdır:

$$H_0 : \mu \leq \mu_0 \text{ karşı } H_1 : \mu > \mu_0$$

α anlamlılık düzeyinde tek yanlı sınamanın reddedilmemesi için m değerinin tek yanı sınırlı yüzde $100(1-\alpha)$ güven aralığına düşmemesi gerekir. Eğer

$$\frac{\sqrt{N}}{\sigma}(m-\mu_0) \in (-\infty, z_\alpha) \tag{19.7}$$

sağlanırsa denenceyi reddetmez, bu aralığın dışında kalırsak reddederiz. Sıfır denencesi eşitliği de içerir ve tek yanlı denencenin reddedilmesi bize sıralama bilgisi verir; bu da bize olası iki, tek yanlı denencenin hangisini sınamamız gerektiğini söyler. Bizim için önemli olacak savın H_1 karşıt denencesinde olması gerekir ki sınamanın reddi savımızı destekler olsun.

Eğer değişkenlik bilinmiyorsa, aralık kestiriminde yaptığımız gibi yerine örneklem değişkenliğini koyarız. Bu durumda

$$\frac{\sqrt{N}(m-\mu_0)}{S} \sim t_{N-1} \tag{19.8}$$

olacağından, $H_0 : \mu = \mu_0$ ya da $H_1 : \mu \neq \mu_0$ sınamasını α anlamlılık düzeyinde reddedememiz için

$$\frac{\sqrt{N}(m-\mu_0)}{S} \in (-t_{\alpha/2,N-1}, t_{\alpha/2,N-1}) \tag{19.9}$$

olması gerekir. Buna *iki yanlı* t *sınaması* denir. Tek yanlı t sınaması da benzer biçimde türetilebilir.

19.10 Bir Sınıflandırma Algoritmasının Başarısının Belirlenmesi

Denence sınamasını gördüğümüze göre sınıflandırıcıların hatalarının belirlenmesi ve karşılaştırılmasında kullanılmasını tartışabiliriz. Bu bölümde göreceğimiz sınamaların hepsi sınıflandırma hatasını kullanıyor, ama aynı yaklaşımı başka hata türleri, örneğin bağlanımdaki kare hata, gözetimsiz öğrenmede olabilirlik, pekiştirmeli öğrenmede beklenen ödül üzerinde de kullanabiliriz; tek ister, onlar için de uygun örnekleme dağılımları belirlemektir. Bilinen hiçbir dağılımın uygun olmadığı durumlarda kullanılabilecek dağılımdan bağımsız sınamalarıysa bu bölümün en son kısmında göreceğiz.

Önce bir sınıflandırma algoritmasının beklenen hatasının belirlenmesinden, sonra da iki algoritmanın hatalarının karşılaştırılmasından bahsedeceğiz.

19.10.1 Binom Sınaması

Elimizde yalnız bir tane öğrenme kümesi $\mathcal{T}$ ve bir geçerleme kümesi $\mathcal{V}$ olduğu durumla başlayalım. Sınıflandırıcımızı $\mathcal{T}$ ile eğitip $\mathcal{V}$ üzerinde sınarız. Sınıflandırıcının hata yapma olasılığı p olsun. p değerini bilmiyoruz tabii; kestirmek istediğimiz ya da değeri konusunda sınama yapmak istediğimiz o. Geçerleme kümesi $\mathcal{V}$ içindeki t. örneğin doğruluğunu gösteren değişkeni x^t olarak gösterelim ve sınıflandırıcı hata yaparsa 1, doğru karar verirse 0 değerini alan bir Bernoulli rastsal değişkeni olsun. Bu durumda, N tane geçerleme örneği üstünde toplam hata sayısını gösteren X binom dağılır:

$$X = \sum_{t=1}^{N} x^t$$

p hata olasılık değerinin verilmiş bir p_0 değerinden küçük olup olmadığını sınamak istiyoruz:

$$H_0 : p \leq p_0 \text{ karşı } H_1 : p > p_0$$

Hata olasılığı p ise sınıflandırıcının N örnekten j tanesinde hata yapma olasılığı

$$P\{X = j\} = \binom{N}{j} p^j (1-p)^{N-j}$$

olur. $p \leq p_0$ iken $X = e$ ya da daha fazla hata yapma olasılığı çok küçük bir sayı çıkarsa denenceyi reddetmek akla yakındır. *Binom sınama*, denenceyi eğer

$$P\{X \geq e\} = \sum_{x=e}^{N} \binom{N}{x} p_0^x (1-p_0)^{N-x} \tag{19.10}$$

anlamlılık düzeyi olan α'dan (örneğin, 0.05) küçük olursa reddeder.

19.10.2 Yaklaşık Normal Sınama

Hata olasılığı p için nokta kestiricimiz, $\hat{p} = X/N$ olarak tanımlanır. Bu durumda, $\hat{p}$, p_0 değerinden çok farklıysa sıfır denencesini reddetmek uygun olur. Farkın ne kadar olması gerektiği $\hat{p}$ rastsal değişkeninin örneklem dağılımına ve anlamlılık düzeyi α değerine bağlıdır.

Toplam hata X, aynı dağılımdan bağımsız 0/1 rastsal değişkenlerin toplamıdır, binom dağılır, ve merkezi limit kuramından dolayı, N küçük olmadığında, ortalaması p_0 ve değişkenliği $p_0(1-p_0)$ olan yaklaşık normal bir dağılımdan geldiği varsayılabilir. Bunu birim normal dağılıma çevirebiliriz:

$$\frac{X/N - p_0}{\sqrt{p_0(1-p_0)}} \sim Z \tag{19.11}$$

$\sim$, "yaklaşık olarak dağılır," demektir. Ardından denklem 19.7'yi kullanan *yaklaşık normal sınama*, sıfır denencesini, $X = e$ için bu istatistik z_α değerini geçerse reddeder. Örneğin, $z_{0.05} = 1.64$. Normal dağılımla yaklaşıklama, N çok küçük olmadıkça ve p sıfır ya da bire çok yakın olmadıkça geçerlidir; genelde $Np \geq 5$ ve $N(1-p) \geq 5$ koşulları aranır.

19.10.3 t Sınaması

Yukarıda tartıştığımız iki sınama da tek öğrenme ve geçerleme kümesi kullanır ve dolayısıyla bu kümelerdeki rastsal oynamaları dikkate almaz. Eğer sınıflandırma algoritmasını K kez, K öğrenme/geçerleme küme çiftinde kullanırsak elimizde K geçerleme kümesi üzerinde K hata yüzdesi, $p_i, i = 1, \ldots, K$ olur. $\mathcal{T}_i$ ile eğitilmiş model, $\mathcal{V}_i$ kümesinin t. örneği üzerinde hata yaparsa x_i^t, 1, doğru sınıfı seçerse 0 değerini alsın. Bu durumda, i. kattaki hata kestirimi

$$p_i = \frac{\sum_{t=1}^{N} x_i^t}{N}$$

olur. Katlardaki hataların ortalama ve değişkenliğini hesaplarız:

$$m = \frac{\sum_{i=1}^{K} p_i}{K}, \quad S^2 = \frac{\sum_{i=1}^{K} (p_i - m)^2}{K - 1}$$

ve denklem 19.8'e göre

$$\frac{\sqrt{K}(m - p_0)}{S} \sim t_{K-1} \tag{19.12}$$

olduğundan, t sınaması, sınıflandırıcının hatasının p_0 ya da daha az olduğu denecesini, bu istatistik değeri $t_{\alpha,K-1}$'den daha büyük olursa, α anlamlılık düzeyinde reddeder. Sıklıkla, K, 10 ($t_{0.05,9} = 1.83$) ya da 30 ($t_{0.05,29} = 1.70$) alınır.

19.11 İki Algoritmanın Karşılaştırılması

Elimizde iki tane sınıflandırma algoritması varsa, birini seçebilmek için bunlarla oluşturulan sınıflandırıcıların hatalarının aynı olup olmadığını sınamak isteriz.

19.11.1 McNemar Sınaması

Tek bir öğrenme ve geçerleme kümesi olduğunda iki algoritmayı da öğrenme kümesiyle eğitip geçerleme kümesi üzerinde dener ve hatalarına bakarız. *Olumsallık çizelgesi*, aşağıdaki gibi, belirli koşulların kaç kez sağlandığını sayar:

e_{00}: İkisince de yanlış sınıflandırılanlar	e_{01}: Yalnız 1'in yanlış sınıflandırdıkları
e_{10}: Yalnız 2'nin yanlış sınıflandırdıkları	e_{11}: İkisince de doğru sınıflandırılanlar

İki sınıflandırma algoritmasının eşit hatalı sınıflandırıcılar oluşturduğu sıfır denencesi varsayıldığında $e_{01} = e_{10}$ olmasını ve bu değerin, $(e_{01} + e_{10})/2$ değerine eşit olmasını bekleriz. Bu durumda

$$\frac{(|e_{01} - e_{10}| - 1)^2}{e_{01} + e_{10}} \sim \chi_1^2 \tag{19.13}$$

bir serbestlik dereceli ki kare bir dağılımdan gelir ve *McNemar sınaması*, bu istatistik $\chi_{\alpha,1}^2$'den büyük olursa, iki sınıflandırıcının hatalarının eşit olduğu denencesini α anlamlılık düzeyinde reddeder: $\chi_{0.05,1}^2 = 3.84$.

19.11.2 K Kat Çapraz Geçerlenmiş Eşli t Sınaması

Bu sınama, K kat çapraz geçerlemeyle oluşturulmuş K tane öğrenme/geçerleme küme çiftini kullanır. Her iki algoritmayla $\mathcal{T}_i, i = 1, \ldots, K$ öğrenme kümeleri üzerinde eğittiğimiz sınıflandırıcıları, $\mathcal{V}_i$ geçerleme kümeleri üzerinde dener, ve p_i^1 ve p_i^2 ile gösterdiğimiz hata oranlarını sayarız.

İki algoritma aynı oranda hatalı sınıflandırıcılar oluşturuyorsa, hataların aynı ortalamalı, ya da farklarının ortalamasının 0 olmasını bekleriz. Kat i'de hatalar arasındaki fark $p_i = p_i^1 - p_i^2$ olur. Bu *eşli sınamadır*; her katta iki algoritma da aynı öğrenme ve geçerleme kümesini kullanır. Toplam K kat üzerinde öğrenme ve geçerleme yapınca elimizde K tane p_i fark değeri olur ve p_i^1 ve p_i^2 değerlerinin (yaklaşık) normal dağıldığı varsayımında, farkları olan p_i de (yaklaşık) normal dağılır. Sıfır denencesi bu dağılımın ortalamasının 0 olduğudur:

$$H_0 : \mu = 0 \text{ karşı } H_1 : \mu \neq 0$$

Farkların ortalaması ve değişkenliğini hesaplarız:

$$m = \frac{\sum_{i=1}^{K} p_i}{K}, \quad S^2 = \frac{\sum_{i=1}^{K} (p_i - m)^2}{K - 1}$$

$\mu = 0$ olduğu varsayımında

$$\frac{\sqrt{K}(m - 0)}{S} = \frac{\sqrt{K} \cdot m}{S} \sim t_{K-1} \tag{19.14}$$

$K - 1$ serbestlik dereceli t dağılımından gelir ve K *kat çapraz geçerlenmiş eşli* t *sınaması*, bu istatistik, $(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1})$ aralığının dışındaysa, iki algoritmanın eşit hata oranlı sınıflandırıcılar oluşturduğu denencesini α anlamlılık düzeyinde reddeder. Sık kullanılan değerler: $t_{0.025,9} = 2.26$ ve $t_{0.025,29} = 2.05$.

Eğer birinci algoritmanın ikinciden daha az hatalı sınıflandırıcılar oluşturduğunu sınamak istersek tek yanlı bir denence sınaması kullanırız:

$$H_0 : \mu \geq 0 \text{ karşı } H_1 : \mu < 0$$

Sınama reddederse, birincinin daha az hatalı olduğu savımız desteklenmiş olur.

19.11.3 5×2 Çapraz Geçerlenmiş Eşli t Sınaması

Dietterich (1998) tarafından önerilen 5×2 çapraz geçerlenmiş t sınamasında iki kat çapraz geçerlemeyi beş kez çoğaltırız. Her çoğaltmada, veri kümesi iki tane eşit büyüklükte parçaya bölünür. $p_i^{(j)}$, i. çoğaltmanın $(i = 1, \ldots, 5)$ j. katında $(j = 1, 2)$ iki sınıflandırıcının geçerleme hatalarının farkıdır. Çoğaltma i'deki ortalama $\overline{p}_i = (p_i^{(1)} + p_i^{(2)})/2$, ve kestirilen değişkenlik $s_i^2 = (p_i^{(1)} - \overline{p}_i)^2 + (p_i^{(2)} - \overline{p}_i)^2$ sınamada kullanılan istatistiklerdir.

İki sınıflandırma algoritmasının eşit hatalı sınıflandırıcılar oluşturduğu sıfır denencesini varsayarsak $p_i^{(j)}$ aynı dağılımdan iki tane rastsal değişkenin farkıdır ve bunların gerçekte bağımsız olmadığını unutursak, $p_i^{(j)}$, 0 ortalamalı ve bilinmeyen σ^2 değişkenli yaklaşık normal bir rastsal değişkendir. Bu durumda, $p_i^{(j)}/\sigma$ yaklaşık olarak birim normaldir. Yine $p_i^{(1)}$ ve $p_i^{(2)}$'nin bağımsız normaller olduğunu kabul edersek (öğrenme ve geçerleme kümeleri bağımsız oluşturulmadığından gerçekte bu

doğru değildir), s_i^2/σ^2, bir serbestlik dereceli ki kare dağılır. Tüm s_i^2'lerin bağımsız olduğunu varsayarsak (hepsi aynı veriden hesaplandığından gerçekte değillerdir), toplamları beş serbestlik dereceli ki kare bir dağılımdan gelir:

$$M = \frac{\sum_{i=1}^{5} s_i^2}{\sigma^2} \sim \chi_5^2$$

olduğu için

$$t = \frac{p_1^{(1)}/\sigma}{\sqrt{M/5}} = \frac{p_1^{(1)}}{\sqrt{\sum_{i=1}^{5} s_i^2/5}} \sim t_5 \tag{19.15}$$

beş serbestlik dereceli t dağılmış bir rastsal değişkendir. 5×2 *çapraz geçerlenmiş eşli* t *sınamasının* iki algoritmanın eşit hatalı sınıflandırıcılar oluşturduğu denencesini α anlamlılık düzeyinde reddetmesi için bu istatistiğin $(-t_{\alpha/2,5}, t_{\alpha/2,5})$ aralığının dışına düşmesi gerekir: $t_{0.025,5} = 2.57$.

19.11.4 5×2 Çapraz Geçerlenmiş Eşli F Sınaması

Denklem 19.15'in payında $p_1^{(1)}$ kullanılması nedensizdir; olası on farklı $p_i^{(j)}, j = 1, 2, i = 1, \ldots, 5$, değerinden herhangi biri kullanılabilir ve bu durumda on farklı istatistik değeri ortaya çıkar:

$$t_i^{(j)} = \frac{p_i^{(j)}}{\sqrt{\sum_{i=1}^{5} s_i^2/5}} \tag{19.16}$$

Alpaydın (1999), 5×2 çapraz geçerlenmiş t sınamasını geliştirip bu on farklı değeri birleştiren farklı bir istatistik önerir. Eğer $p_i^{(j)}/\sigma \sim Z$ ise $\left(p_i^{(j)}\right)^2/\sigma^2 \sim \chi_1^2$ olur ve toplamları on serbestlik dereceli ki kare bir dağılımdan gelir:

$$N = \frac{\sum_{i=1}^{5} \sum_{j=1}^{2} \left(p_i^{(j)}\right)^2}{\sigma^2} \sim \chi_{10}^2$$

Denklem 19.15'in payına bunu koyduğumuzda iki ki kare dağılımın oranı olan bir istatistik elimize geçer. Kendi serbestlik derecelerine bölünmüş böyle iki bağımsız ki kare rastsal değişkenin oranı, on ve beş serbestlik dereceli bir F dağılımından gelir (bölüm A.3.8):

$$f = \frac{N/10}{M/5} = \frac{\sum_{i=1}^{5} \sum_{j=1}^{2} \left(p_i^{(j)}\right)^2}{2\sum_{i=1}^{5} s_i^2} \sim F_{10,5} \tag{19.17}$$

5×2 *çapraz geçerlenmiş eşli* F *sınamasının* bu iki öbek sınıflandırıcının eşit hatalı olduğu denencesini α anlamlılık düzeyinde reddetmesi için bu istatistiğin $F_{\alpha,10,5}$ değerinden büyük olması gerekir. $F_{0.05,10,5} = 4.74$.

19.12 İkiden Çok Algoritmanın Karşılaştırılması: Değişkenlik Çözümlemesi

Çoğu uygulamada elimizde ikiden daha çok aday algoritma olabilir. Bunların hatalarını karşılaştırırken, L algoritmayı K öğrenme kümesi üzerinde eğitip her biriyle K tane sınıflandırıcı oluşturur ve ardından bunların K geçerleme kümesi üzerindeki hatalarını kaydederiz. Böylece, her biri K değer içeren L tane öbek oluşur. Yapmamız gereken bu L öbek içindeki değerlerin ortalamaları arasında anlamlı fark olup olmadığını sınamaktır. Öğrenme algoritmasını etken, farklı algoritmaları da bu etkenin düzeyleri olarak düşünürsek bu, L düzeyi olan tek etkenli, K kez çoğaltılmış bir deneydir.

Değişkenlik çözümlemesinde, her biri K örnek içeren L bağımsız örneklem vardır. Örneklerin normal dağıldığı ve dağılımların μ_j ortalama ve σ^2 ortak değişkenlik değerlerinin bilinmediği varsayılır:

$$X_{ij} \sim \mathcal{N}(\mu_j, \sigma^2), j = 1, \ldots, L, \; i = 1, \ldots, K,$$

Sıfır denencesi H_0, tüm öbek merkezlerinin aynı olduğudur:

$$H_0 : \mu_1 = \mu_2 = \cdots = \mu_L \text{ karşı } H_1 : \mu_r \neq \mu_s, \text{en az bir çift } (r, s) \text{ için}$$

İkiden çok algoritmayla eğitilen sınıflandırıcıların ortalama hataları bu biçimde karşılaştıralabilir. Elimizde L algoritma ve her biri için K geçerleme hata değeri var. X_{ij}, j. algoritmanın i. kattaki hatasını göstersin. Her X_{ij} binom ve yaklaşık normal dağılır. Eğer H_0 reddedilmezse L algoritmanın oluşturduğu sınıflandırıcıların hataları arasında anlamlı bir fark yok demektir. Bu bağlamda, değişkenlik çözümlemesi, 19.11. bölümde gördüğümüz ikili sınamaların genellenmesidir. L öbek, farklı algoritmalara karşılık gelebileceği gibi, bir üstparametrenin farklı değerlerine, örneğin, çok katmanlı algılayıcıdaki saklı birim sayısına da karşılık gelebilir.

Değişkenlik çözümlemesinde bilinmeyen σ^2 ortak değişkenliği için iki farklı kestirici tanımlarız. Bunlardan biri yalnızca H_0 doğru olduğunda, öbürüyse her zaman, H_0 doğru olsa da olmasa da geçerlidir. Eğer bu iki kestiriciden gelen değerler birbirinden (şans eseri oluşamayacak kadar) farklıysa değişkenlik çözümlemesi, L örneklemin aynı dağılımdan geldiği savını reddeder.

σ^2 için birinci kestiricimiz yalnızca sıfır denencesi, $\mu_j = \mu, j = 1, \ldots, L$, doğru olduğunda geçerlidir. Eğer $X_{ij} \sim \mathcal{N}(\mu, \sigma^2)$ ise, öbek ortalaması

$$m_j = \sum_{i=1}^{K} \frac{X_{ij}}{K}$$

ortalaması μ ve değişkenliği σ^2/K olacak biçimde yine normal dağılır. Sıfır denencesi doğru olduğunda, $m_j, j = 1, \ldots, L$ değerlerini, $\mathcal{N}(\mu, \sigma^2/K)$ dağılımından çekilmiş L bağımsız örnek olarak görebiliriz. Bu durumda, *onların* ortalama ve değişkenliği

$$m = \frac{\sum_{j=1}^{L} m_j}{L}, \quad S^2 = \frac{\sum_j (m_j - m)^2}{L - 1}$$

olarak hesaplanır, ve σ^2 için bir kestirici $K \cdot S^2$,

$$\hat{\sigma}_b^2 = K \sum_{j=1}^{L} \frac{(m_j - m)^2}{L-1} \tag{19.18}$$

olur. Tüm m_j değerleri normal ve $(L-1)S^2/(\sigma^2/K)$, $(L-1)$ serbestlik dereceli ki kare dağılır. Bunu da

$$\sum_j \frac{(m_j - m)^2}{\sigma^2/K} \sim \chi^2_{L-1} \tag{19.19}$$

biçiminde yazabiliriz. Öbekler arası karelerin toplamını tanımlayalım:

$$SS_b \equiv K \sum_j (m_j - m)^2$$

Bu durumda, H_0 doğru olduğunda

$$\frac{SS_b}{\sigma^2} \sim \chi^2_{L-1} \tag{19.20}$$

olarak yazabiliriz.

σ^2 için ikinci kestirici, öbek değişkenliklerinin (S_j^2) ortalamasıdır. Her bir değişkenlik

$$S_j^2 = \frac{\sum_{i=1}^{K} (X_{ij} - m_j)^2}{K-1}$$

ve ortalamaları

$$\hat{\sigma}_w^2 = \sum_{j=1}^{L} \frac{S_j^2}{L} = \sum_j \sum_i \frac{(X_{ij} - m_j)^2}{L(K-1)} \tag{19.21}$$

olarak yazılır. Öbek içi karelerin toplamını tanımlayalım:

$$SS_w \equiv \sum_j \sum_i (X_{ij} - m_j)^2$$

Normal bir dağılım için

$$(K-1)\frac{S_j^2}{\sigma^2} \sim \chi^2_{K-1}$$

olduğunu ve ki kare rastsal değişkenlerin toplamının yine bir ki kare olduğunu bildiğimiz için

$$(K-1)\sum_{j=1}^{L} \frac{S_j^2}{\sigma^2} \sim \chi^2_{L(K-1)}$$

ya da

$$\frac{SS_w}{\sigma^2} \sim \chi^2_{L(K-1)} \tag{19.22}$$

sağlanır.

Elimizdeki bu iki değişkenlik kestiriminin eşit olup olmadığını anlamak için oranlarının 1 çevresinde olup olmadığına bakarız. Kendi serbestlik derecelerine bölünmüş iki bağımsız ki kare rastsal değişkenin oranı F dağılır ve H_0 doğru olduğunda

$$F_0 = \left(\frac{SS_b/\sigma^2}{L-1}\right) \Big/ \left(\frac{SS_w/\sigma^2}{L(K-1)}\right) = \frac{SS_b/(L-1)}{SS_w/(L(K-1))} = \frac{\hat{\sigma}_b^2}{\hat{\sigma}_w^2} \sim F_{L-1,L(K-1)} \tag{19.23}$$

olur. İstenen anlamlılık düzeyi α için L algoritmanın eşit hata oranlı sınıflandırıcılar oluşturduğu denencesi bu istatistik $F_{\alpha,L-1,L(K-1)}$ değerini geçerse reddedilir.

Bu iki kestiricinin değerleri birbirinden farklıysa reddediyoruz. H_0 doğru değilse, m_j değerlerinin m çevresindeki değişkenliği H_0 doğruykenkine göre daha büyük olacak, ilk kestirici, $\hat{\sigma}_b^2$, gerçek σ^2 değerine göre daha büyük bir değer çıkaracak ve oranları birden büyük olacaktır. $\alpha = 0.05$, $L = 5$ ve $K = 10$ için eşik değeri $F_{0.05,4,45} = 2.6$. X_{ij}, m çevresinde σ^2 değişkenliğiyle dağıldığında, m_j değerleri m çevresinde σ^2/K değişkenliğiyle dağılır. Veriden hesaplanan değişkenliğin bundan büyük olması, m_j değerlerinin m değerinden farkının normal gürültüyle açıklanamayacak kadar olduğunu gösterir ve bu durumda H_0 reddedilmelidir.

Değişkenlik çözümlemesi adı, verideki toplam değişkenliği parçalara ayırıyor olmamızdan gelir:

$$SS_T \equiv \sum_j \sum_i (X_{ij} - m)^2 \tag{19.24}$$

SS_T, serbestlik derecesi olan $K \cdot L - 1$'e bölündüğünde, X_{ij} örneklem değerlerinin değişkenliği çıkar. ($K \cdot L$ tane veri noktasının ortalama değeri m sabitlendiğinde serbestlik derecesi 1 azalır.). Verideki toplam kare hata, öbekler arası kare toplam ve öbek içi kare toplam olarak ikiye ayrılabilir (alıştırma 5):

$$SS_T = SS_b + SS_w \tag{19.25}$$

Değişkenlik çözümlemesi sonuçları, tablo 19.4'teki gibi bir çizelge olarak dökülür. Burada gördüğümüz, tek etkenli (örneğin, öğrenme algoritması) ve *tek yönlü* bir deneyin değişkenlik çözümlemesidir. Birden çok etkenli deneyler de düşünebiliriz; örneğin, sınıflandırma algoritmasından önce farklı boyut azaltma algoritmaları kullanılabilirse, bu, *iki yollu ve etkileşimli* bir deneydir.

Denence reddedildiğinde L öbek arasında bir fark olduğunu anlarız ama farkın kaynağını bilmeyiz. Bu amaç için öbeklerin altkümelerini, örneğin ikişerli karşılaştıran *artçı sınama* yapılır.

Fisher tarafından önerilmiş *en küçük kare fark sınamasında* öbekler ikili olarak karşılaştırılır. Her öbek için, $m_i \sim \mathcal{N}(\mu_i, \sigma_w^2 = MS_w/K)$ ve farkları için $m_i - m_j \sim \mathcal{N}(\mu_i - \mu_j, 2\sigma_w^2)$ olduğundan, $H_0 : \mu_i = \mu_j$ varsayıldığında

$$t = \frac{m_i - m_j}{\sqrt{2}\sigma_w} \sim t_{L(K-1)}$$

dağılır ve $|t| > t_{\alpha/2,L(K-1)}$ ise H_0, karşıt $H_1 : \mu_i \neq \mu_j$ lehine reddedilir. Benzer biçimde, ikili sıralama için tek yanlı sınamalar da tanımlanabilir.

Tablo 19.4: Tek etkenli bir model için değişkenlik çözümlemesi çizelgesi.

Değişkenlik kaynağı	Kareler toplamı	Serbestlik derecesi	Ortalama kare	F_0
Öbekler arası	$SS_b \equiv K\sum_j (m_j - m)^2$	$L-1$	$MS_b = \frac{SS_b}{L-1}$	$\frac{MS_b}{MS_w}$
Öbekler içi	$SS_w \equiv \sum_j \sum_i (X_{ij} - m_j)^2$	$L(K-1)$	$MS_w = \frac{SS_w}{L(K-1)}$	
Toplam	$SS_T \equiv \sum_j \sum_i (X_{ij} - m)^2$	$L \cdot K - 1$		

Birden çok sınama yaparak tek sonuç çıkardığımızda, buna *çoklu karşılaştırma* denir, ve bu durumda, her biri α anlamlılık düzeyinde T sınama yapıyorsak, en az bir tanesinin yanlışlıkla reddedilmesinin olasılığının en fazla $T\alpha$ olduğunu aklımızda tutmamız gerekir. Örneğin, her biri yüzde 95 güvenle hesaplanan aralıkların altısının da doğru olma olasılığı en az yüzde 70'dir. Bu yüzden T sınama sonunda varacağımız sonucun α anlamlılık düzeyinde olması için her sınamanın α/T anlamlılık düzeyinde yapılması gerekir. Buna *Bonferroni düzeltmesi* denir.

Bazen, değişkenlik çözümlemesinin reddetmesine rağmen artçı ikili sınamalardan hiçbiri anlamlı bir fark bulmayabilir. Bu durumda, ortalamaların arasında fark olduğu ama farkın nedenini belirleyebilecek kadar verimiz olmadığı sonucuna varırız.

Bir yapay öğrenme deneyinde maliyetin çoğu L algoritmanın K kez eğitme ve denemede kullanılmasından gelir. Bir kere bunlar yapılıp sonuçlar $K \times L$ bir dizeye kaydedildikten sonra değişkenlik çözümlemesi ya da öteki istatistiksel sınamalar görece çok kolay ve hızlıdır.

19.13 Birden Çok Veri Kümesinde Karşılaştırma

Birden çok algoritmayı birden çok veri kümesi üstünde karşılaştırmak istediğimizi düşünelim. Her algoritma, yaptığı varsayımların veriye ne kadar uyduğuna bağlı olarak farklı veri kümelerinde farklı davranacaktır. Aynı algoritmanın farklı veri kümelerinde oluşturduğu farklı sınıflandırıcıların belirli bir dağılımdan geldiğini, ya da hatalarının ortalama bir başarı değerinin çevresinde normal dağıldığını söyleyemeyiz. Bu yüzden birden çok veri kümesi üzerinde algoritmaları karşılaştırırken önceki sayfalarda gördüğümüz gibi dağılıma bağlı bir sınama yerine *dağılımdan bağımsız sınama* kullanmamız gerekir. Böylesi sınamaların güzel yanı, örneğin, öğrenme zamanı, serbest parametre sayısı, vs. gibi başka tür istatistikler için de kullanılabilmesidir.

Belirli bir dağılım varsayan sınamalar, örneğin, t sınaması, genelde normallikten

küçük sapmalara karşı dayanıklıdır, özellikle de veri kümesi büyükse. Dağılımdan bağımsız sınama daha geneldir ama daha az güvenilirdir ve eğer ikisi de uygulanabiliyorsaa dağılım varsayan sınama yeğlenmelidir. Aynı amaç için kullanılan dağılımdan bağımsız sınamanın denk güce erişmesi için daha büyük örneklem gerekir. Dağılımdan bağımsız sınama verinin belirli bir dağılımdan geldiğini varsaymaz, yalnızca değerlerin karşılaştırılabildiğini ve sıralanabildiğini varsayar, ve biraz sonra göreceğimiz gibi sınama, bu sıra verileri üzerinde yapılır.

Yukarıda bahsettiğimiz gibi, bir algoritmayı birden çok veri kümesi üzerinde eğitim için kullandığımızda oluşan sınıflandırıcıların hatalarının ortalaması anlamlı bir istatistik değildir, ve bu ortalama değerlerini, A ve B algoritmalarını karşılaştırmak için kullanamayız. Karşılaştırmada kullanabileceğimiz tek bilgi, bir veri kümesinde ikisinden hangisinin daha başarılı olduğudur. Bu biçimde, kaç veri kümesinde A'nın B'den daha başarılı olduğunu sayar ve bu değerin, eğer A ile B gerçekten eşit başarılı olsalardı görülme olasılığı yüksek bir sayı olup olmadığına bakarız. İkiden çok algoritma olduğunda algoritmaları başarılarına göre sıralar ve ortalama *sıra* değerleri üzerinden sınama yaparız. Dağılımdan bağımsız sınama, az sonra göreceğimiz gibi mutlak değerleri değil bu sıra değerlerini kullanır.

Bu sınamanın detaylarına geçmeden önce, algoritmaların çok farklı veri kümeleri üzerindeki başarısına bakmanın çok mantıklı olmadığını vurgulamak gerekir. "En iyi algoritma," diye bir şey yoktur ve böylesi savlar her zaman havada kalacaktır. Ama, algoritmaları aynı uygulama için örneklenmiş birden çok veri kümesi üzerinde karşılaştırabiliriz. Örneğin, yüz tanıma için farklı zamanlarda, farklı özelliklerde (çözünürlük, ışıklandırma, özne sayısı, vs.) veri kümeleri olabilir. Bunları tek bir veri kümesi olarak birleştiremeyiz, ama bunlar için ayrı ayrı sınıflandırıcılar eğitip başarılarına göre sıralar, ve dağılımdan bağımsız sınamayla genel bir sonuç çıkarabiliriz.

19.13.1 İki Algoritmanın Karşılaştırılması

İki algoritmayı karşılaştırdığımızı düşünelim. İkisini de $i = 1, \ldots, N$ farklı veri kümesi üzerinde eğitiyor ve geçerliyoruz. Karşılaştırmanın eşli olması gerektiğini, aynı öğrenme ve geçerleme kümelerinin kullanılacağını, ya da daha genel olarak söylersek algoritmalar dışındaki tüm koşulların aynı olması gerektiğini biliyoruz. Aldığımız sonuçları, e_i^1 ve e_i^2 diye gösterelim, ve eğer her veri kümesinde K kat çapraz geçerleme kullanıyorsak, bunlar K sonucun ortalama ya da ortancası olabilir. Eğer iki algoritmanın genel olarak başarıları eşitse (sıfır denencesi), bir veri kümesinde iki algoritmanın da kazanma, ya da daha az hata yapan olma olasılığının $1/2$ olması, ve toplam N veri kümesi üzerinde karşılaştırıyorsak, iki algoritmanın da kazandığı veri kümelerinin sayısının $N/2$ olmasını bekleriz. *İşaret sınaması* bu düşünceye dayanır.

$$X_i = \begin{cases} 1 & \text{eğer } e_i^1 < e_i^2 \\ 0 & \text{değilse} \end{cases} \quad \text{ve } X = \sum_{i=1}^{N} X_i$$

Tek yanlı sınamayı görelim:

$$H_0 : \mu_1 \geq \mu_2 \text{ karşı } H_1 : \mu_1 < \mu_2$$

Sıfır denencesi doğru olduğunda X, N deneme sayılı ve $p = 1/2$ olasılıklı binom bir dağılımdan gelir. Eğer birinci algoritma, $X = e$ veri kümesinde daha az hata yaptıysa, algoritmalar gerçekten eşit hatalı olduğunda bunun olma olasılığını hesaplar

$$P\{X \leq e\} = \sum_{x=0}^{e} \binom{N}{x} \left(\frac{1}{2}\right)^x \left(\frac{1}{2}\right)^{N-x}$$

ve eğer bu olasılık, anlamlılık düzeyi olan α değerinden küçükse sıfır denencesini reddederiz. Bazı veri kümelerinde hatalar eşitse bunların sayısını iki algoritmaya eşit olarak paylaştırırız; t tane eşitlik olduğunda, e değerine $t/2$ ekleriz (t tek sayıysa birini gözardı eder, N değerini de bir azaltırız).

Öteki tek yanlı sınamada

$$H_0 : \mu_1 \leq \mu_2 \text{ karşı } H_1 : \mu_1 > \mu_2$$

$P\{X \geq e\} < \alpha$ ise reddederiz.

İki yanlı sınamada

$$H_0 : \mu_1 = \mu_2 \text{ karşı } H_1 : \mu_1 \neq \mu_2$$

e, çok küçük, ya da çok büyük olduğunda reddederiz. $e < N/2$ ise $2P\{X \leq e\} < \alpha$ olduğunda, $e > N/2$ ise $2P\{X \geq e\} < \alpha$ olduğunda reddederiz—her iki durumda da karşılık gelen kuyruğu bulup iki yanlı sınama olduğu için olasılığı ikiyle çarparız.

Daha önce bahsettiğimiz gibi dağılımdan bağımsız sınamayı herhangi bir ölçüt, örneğin, öğrenme zamanlarını karşılaştırmak için kullanabiliriz. Bunun yapılabilmesi bize sıra bilgisini kullanan dağılımdan bağımsız sınamanın değerlerin kendisini kullanan dağılıma bağımlı sınamaya göre farkını da gösterir. İki algoritmayı on farklı veri kümesinde karşılaştırdığımızı ve bunlardan birinin öğrenme zamanının günler, kalan dokuzunun ise dakikalar sürdüğünü varsayalım. Dağılım bağımlı bir sınama ortalama öğrenme zamanını hesaplar ve büyük tek veri kümesi üzerindeki öğrenme zamanı ağır basıp ötekilerin etkisini yok eder. Dağılım bağımsız sınamaysa her veri kümesinde kimin daha hızlı olduğunu ayrı ayrı sayacağından bir tür normalleştirme yapar ve sonuç karara tüm veri kümelerindeki sonuçların eşit etki etmesini sağlar.

İşaret sınamasını tek örneklemle, örneğin, veri kümeleri üzerindeki ortalama hatanın yüzde ikiden küçük olup olmadığını sınamak için de kullanabiliriz. Bu durumda, her veri kümesinde μ_1 değerini başka bir algoritmanın hatasıyla değil bir μ_0 sabitiyle karşılaştırırız. İkinci algoritmanın sonuçları olarak hep μ_0 değerini koyup yukarıdaki sınamayı aynı biçimde kullanabilir, ve böylece kaç kere 0.02'den daha az hata yaptığımızı ve bununla sıfır denencesi doğru olsaydı ne sıklıkta karşılaşılacağımızı hesaplarız. Büyük bir N değeri için binom dağılımı yerine yaklaşık olarak normal

dağılımını da kullanabiliriz (alıştırma 6), ama çoğu deneyde veri kümesi sayısı yirmiden az olacaktır. İşaret sınaması, ortanca değerini kullanır ve dağılım bakışımlıysa bu değer ortalamaya eşittir.

İşaret sınaması farkın yalnızca işaretini kullanır, boyunu kullanmaz, ama bir algoritmanın kazandığında hep farkla, kaybettiğindeyse kılpayı kaybettiği bir deney olabilir. *Wilcoxon işaretli sıra sınaması*, aşağıda göreceğimiz gibi, farkın hem işaretini, hem de boyunu kullanır:

Farkların işaretleri yanında boylarını da, $m_i = |e_i^1 - e_i^2|$, hesapladığımızı, ve sonra bunları, en küçük, $\min_i m_i$, için 1, sonraki için 2, değerini alacak biçimde sıraladığımızı düşünelim. Eşitlik durumunda ortalama alınır; örneğin farkların boyları $2, 1, 2, 4$ olduğunda, sıraları $2.5, 1, 2.5, 4$ olur. Ardından artı işaretli olanların boylarını toplayıp w_+ değerini ve eksi işaretli olanların boylarını toplayıp w_- değerini buluruz.

Sıfır $\mu_1 \leq \mu_2$ denencesini, karşıt $\mu_1 > \mu_2$ lehine reddetmemiz için w_+ değerinin w_- değerine göre çok küçük olması gerekir. Benzer biçimde, iki yanlı $\mu_1 = \mu_2$ denencesi karşıt $\mu_1 \neq \mu_2$ lehine, ancak w_+ ya da w_-, bir başka deyişle $w = \min(w_+, w_-)$ çok küçük olursa reddedilir. Wilcoxon işaretli sıra sınaması için eşik değerleri anlamlılık düzeylerine göre çizelgelendirilmiştir ve yine $N > 20$ için yaklaşık olarak normal dağılım kullanılabilir.

19.13.2 İkiden Çok Algoritmanın Karşılaştırılması

Kruskal-Wallis sınaması, değişkenlik çözümlemesiyle aynı işi dağılımdan bağımsız olarak yapar ve sıra sınamasının ikiden çok örneklem için genellenmişidir. $M = L \cdot N$ gözlem, örneğin L algoritmanın N veri kümesi üzerinde oluşturduğu $X_{ij}, i = 1, \ldots, L, j = 1, \ldots, N$ geçerleme hataları verildiğinde, *bütün* değerleri küçükten büyüğe sıralar ve bunlara 1 ile M arasında değer alacak biçimde birer R_{ij} sıra numarası veririz; eşitlik durumunda yine ortalama sıra numaraları kullanılır. Sıfır denencesi

$$H_0 : \mu_1 = \mu_2 = \cdots = \mu_L$$

doğru olduğunda, i. algoritmanın aldığı sıra değerlerinin ortalamasının 1 ile M arasında yaklaşık ortada, $(M + 1)/2$ değerine yakın olmasını bekleriz. Algoritma i'nin sıralarının ortalamasını $\overline{R}_{i\bullet}$ ile gösterirsek denenceyi, ortalama sıra değerleri ortadan fark ederse reddederiz. İstatistik

$$H = \frac{12}{(M+1)L} \sum_{i=1}^{L} \left(\overline{R}_{i\bullet} - \frac{M+1}{2} \right)$$

yaklaşık olarak $L-1$ serbestlik dereceli ki kare bir dağılımdan gelir ve hesaplanan istatistik, $\chi_{\alpha,L-1}$ değerini geçerse algoritmaların eşit sıralı sınıflandırıcılar oluşturduğu denencesini reddederiz.

Normal dağılım varsayan değişkenlik çözümlemesi gibi, burada da, herkesin eşit olduğu savı reddedildiğinde, sıralar üstünde ikişer artçı sınamalar yapılabilir. Bunun

için bir olasılık, *studentleştirilmiş açıklık istatistiğini*

$$q = \frac{\overline{R}_{max} - \overline{R}_{min}}{\sigma_w}$$

kullanan *Tukey sınamasıdır*; burada, $\overline{R}_{max}$ ve $\overline{R}_{min}$ sırasıyla L tane sıra ortalamalarının en küçük ve en büyüğüne karşılık gelir ve σ_w^2, sıraların kendi ortalamaları çevresindeki değişkenliğinin ortalamasıdır. i ve j algoritmalarının aynı ortalama sıraları olduğu denencesini, farklı oldukları karşıt denencesi lehine reddetmemiz için

$$|\overline{R}_{i\bullet} - \overline{R}_{j\bullet}| > q_\alpha(L, L(K-1))\sigma_w$$

olması gerekir; $q_\alpha(L, L(K-1))$ değerleri çizelgelendirilmiştir. Benzer biçimde tek yanlı sınamalar da tanımlanabilir.

Demsar (2006) sonuçarı görselleştirmek için bir yöntem önermiştir. Bu yöntemde, 1 ile L arasında $\overline{R}_{i\bullet}$ ortalama değerlerini işaretler ve ardından ortalamalar çevresinde $q_\alpha(L, L(K-1))\sigma_w$ *önemli fark* uzunluğunda çizgiler çizeriz; örtüşen algoritmaların aralarında fark olmadığı anlaşılır.

19.14 Kısa Kısa

Bu bölümde deney tasarımıyla ilgili genel bilgi, (Montgomery, 2005)'in yapay öğrenme için uyarlanmışıdır. Aralık kestirimi, denence sınaması, ve değişkenlik çözümlemesi hakkında daha fazla bilgi, giriş düzeyindeki herhangi bir istatistik kitabında, örneğin (Ross, 1987)'de, bulunabilir.

Dietterich (1998), yapay öğrenmede kullanılan sınamaları tartışır ve farklı uygulamalarda karşılaştırır. Alıcı işletim özellikleri eğrisi altında kalan alan (Fawcett, 2006)'da anlatılır. Demsar (2006), ikiden çok veri kümesi üzerinde sınıflandırma algoritmalarının karşılaştırılması için kullanılan sınamaları karşılaştırır.

İki ya da daha çok algoritmayı karşılaştırdığımızda eğer eşit hatalı olduklarını reddedemezsek, bellek, zaman, ya da başka bir bedel ölçütüne göre en basit olanı seçeriz. Yani eğer veri, hata oranı açısından birini seçmezse biz önsel olarak önemli bulduğumuz bedel kıstasını öne çıkarırız. Örneğin, doğrusal ve doğrusal olmayan iki modeli karşılaştırdıysak ve ikili bir sınama eşit hatalı olduklarını reddedemediyse daha basit olan doğrusal modeli yeğleriz. Hataya bağlı sınama eşit olduklarını reddetse bile hata, bir algoritmayı öbürüne yeğlerken kullanacağımız kıstaslardan yalnızca biridir ve gerçek dünyadaki mühendislik uygulamalarında, öğrenme ya da denemenin bellek ya da hesap karmaşıklığı, yorumlanabilirlik, kolay programlama, vs. gibi kıstaslar daha öne çıkabilir.

İkili artçı sınama sonuçlarını birleştirerek algoritmaları sıralamak için önerilmiş MultiTest yöntemi (Yıldız ve Alpaydın, 2006) bu düşünceye dayanır. Önce $L(L-1)/2$ tane tek yanlı ikili sınama yaparız ama bunların bir bölümü sınamayı reddetmeyecek ve sıra bilgisi vermeyecektir; bu eksik sıraları önsel bedel (örneğin, bellek karmaşıklığı) bilgisini kullanarak doldururuz. Ardından yapılan bir ilingesel sıralama, hata ve bedel bilgilerini birleştiren bir sıra oluşturur.

Zıtlık karşılaştırması için yöntemler de önerilmiştir. 1 ve 2 yapay sinir ağı tabanlı, 3 ve 4 bulanık mantık kullanan sınıflandırıcılar olsun; böylece, 1 ve 2'nin ortalamasının, 3 ve 4'ün ortalamasına eşit olup olmadığını sınayabiliriz.

Burada yanlış sınıflandırmaları karşılaştırıyoruz ve bu, tüm yanlışların aynı yitim değeri olduğu anlamına gelir; eğer bu doğru değilse uygun bir yitim işleviyle beklenen riskleri karşılaştırmamız gerekir. Bu konuda fazla çalışma yapılmamıştır. Benzer biçimde ortalama kare hatalarını kullanan sınamalar bağlanım algoritmalarının karşılaştırılmasını sağlar.

İki sınıflandırma algoritmasını karşılaştırırken yalnızca yaklaşık olarak aynı sayıda hata yapıp yapmadıklarına bakıyoruz. Aynı sayıda hata yapsalar bile bu aynı hataları yaptıkları demek değildir. Bu düşünceyi 17. bölümde farklı sınıflandırıcıların kararlarını birleştirmekte kullanmıştık.

19.15 Alıştırmalar

1. İki sınıflı bir uygulamada yitimler, $\lambda_{11} = \lambda_{22} = 0$, $\lambda_{21} = 1$ ve $\lambda_{12} = \alpha$, olarak verilmiş olsun. Karar eşik değerini α değerine bağlı olarak hesaplayın.
2. Hata olasılığı p olan bir sınıflandırıcıyı Bernoulli dağılımından örneklemeyle benzetimleyebiliriz. Bu biçimde oluşturduğunuz örneklemler üzerinde $p_0 \in (0, 1)$ için binom, yaklaşık normal, ve t sınamalarını gerçekleyin. Bunu 1000 kez yaparak farklı p değerleri için red olasılıklarını hesaplayın; $p_0 = p$ olduğunda red olasılığının ne olmasını beklersiniz?
3. $x^t \sim \mathcal{N}(\mu, \sigma^2)$ olduğunu ve σ^2 değerinin bilindiğini varsayın. $H_0 : \mu \geq \mu_0$ karşı $H_1 : \mu < \mu_0$ sınaması nasıl yapılır?
4. K kat çapraz geçerlenmiş eşli t sınaması, hataların eşit olup olmadığını sınar, ama eğer red kararı çıkarsa kimin daha az hatalı olduğunu bilmeyiz. Birinci algoritmanın hatasının ikinciden büyük olmadığını nasıl sınayabiliz? İpucu: $H_0 : \mu \leq 0$ karşı $H_1 : \mu > 0$.
5. Toplam karelerin öbekler arası karelerle öbek içi karelerin toplamı olarak yazılabildiğini, $SS_T = SS_b + SS_w$, gösterin.
6. İşaret sınamasında binom dağılımı yerine yaklaşık olarak normal dağılımını kullanın.
7. Elimizde üç tane sınıflandırma algoritması olsun. Bunları en iyiden en kötüye nasıl sıralayabiliriz?
8. A algoritmasının iki sürümü ve B algoritmasının üç sürümü olsun. Bütün bu sürümleri gözönüne alarak bu iki algoritmayı nasıl karşılaştırabiliriz?
9. İki bağlanım algoritmasının hatalarını karşılaştırmak için uygun bir sınama önerin.
10. İki pekiştirmeli öğrenme algoritmasının beklenen ödüllerini karşılaştırmak için uygun bir sınama önerin.

19.16 Kaynaklar

Alpaydın, E. 1999. "Combined 5 × 2 cv F Test for Comparing Supervised Classification Learning Algorithms." *Neural Computation* 11: 1885–1892.

Bouckaert, R. R. 2003. "Choosing between Two Learning Algorithms based on Calibrated Tests." *Twentieth International Conference on Machine Learning*, ed. T. Fawcett, N. Mishra, 51–58. Menlo Park, CA: AAAI Press.

Demsar, J. 2006. "Statistical Comparison of Classifiers over Multiple Data Sets." *Journal of Machine Learning Research* 7: 1–30.

Dietterich, T. G. 1998. "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms." *Neural Computation* 10: 1895–1923.

Fawcett, T. 2006. "An Introduction to ROC Analysis." *Pattern Recognition Letters* 27: 861–874.

Montgomery, D. C. 2005. *Design and Analysis of Experiments.* 6th ed., New York: Wiley.

Ross, S. M. 1987. *Introduction to Probability and Statistics for Engineers and Scientists.* New York: Wiley.

Turney, P. 2000. "Types of Cost in Inductive Concept Learning." Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on Machine Learning, Stanford University, Stanford, CA, July 2.

Wolpert, D. H. 1995. "The Relationship between PAC, the Statistical Physics Framework, the Bayesian Framework, and the VC Framework." *The Mathematics of Generalization*, ed. D. H. Wolpert, 117–214. Reading, MA: Addison-Wesley.

Yıldız, O. T., E. Alpaydın. 2006. "Ordering and Finding the Best of $K > 2$ Supervised Learning Algorithms." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28: 392–402.

Ek A

Olasılık

Kısaca olasılık kuramının temellerini, rastsal değişken kavramını ve bazı örnek olasılık dağılımlarını gözden geçireceğiz.

A.1 Olasılık Kuramı

Rastsal bir deneyin sonucu önceden öngörülemez (Ross 1987, Casella ve Berger 1990). Olası tüm sonuçların kümesine *örnek uzayı* denir ve S ile gösterilir. Eğer sonuçlar sonlu ya da sayılabilir sayıdaysa bu *kesikli* bir örnek kümesidir, değilse *süreklidir*. S kümesinin herhangi bir E altkümesi bir *olaydır*. Olaylar küme oldukları için tümleyen, kesişim ya da birleşimlerinden bahsedebiliriz.

Olasılığın bir yorumu *sıklıktır*. Aynı deney aynı koşullar altında birçok kez yinelendiğinde E olayının olma oranı sabit bir değere yakınsar. Olayın olasılığı dediğimiz ve $P(E)$ ile gösterdiğimiz bu sıklık değeridir.

Olasılık bazen bir *inanç derecesi* olarak da yorumlanır. Örneğin, Türkiye'nin 2014 Dünya Kupasını kazanma olasılığından bahsettiğimizde bu bir sıklık değeri olamaz, çünkü bu yarışma yalnızca bir kez yapılacaktır (ve bu kitap yazılırken) henüz yapılmamıştır. Bu durumda kastettiğimiz olayın gerçekleşeceğine olan öznel inanç derecemizdir ve öznel olduğu için farklı kişiler aynı olay için farklı değerler öne sürebilirler.

A.1.1 Olasılığın Belitleri

Belitler, rastsal bir deneyin sonuçlarına atadığımız olasılıkların göreceli sıklıklar olarak yorumlanabilmesini ve bu değerlerin göreceli sıklıklar arasındaki ilişkilerle ilgili sezgisel anlayışımızla tutarlı olmasını sağlar:

1. $0 \leq P(E) \leq 1$. E_1 imkansız bir olaysa, $P(E_1) = 0$, E_2 olması kesin bir olaysa, $P(E_2) = 1$ olur.

2. S, tüm olası sonuçları içeren örnek uzayıdır: $P(S) = 1$.

3. $E_i, i = 1, \ldots, n$ ayrık olaylarsa, herhangi ikisi aynı anda olamaz demektir: $E_i \cap E_j = \emptyset, j \neq i$; burada $\emptyset$, hiçbir olayı içermeyen *boş olayı* gösterir. Böyle bir durumda, olayların herhangi birinin olma olasılığı olayların olma olasılıklarının toplamına eşittir:

$$P\left(\bigcup_{i=1}^{n} E_i\right) = \sum_{i=1}^{n} P(E_i) \tag{A.1}$$

Örneğin, E^c, E olayının *tümleyeniyken* yani, S kümesinin içinde olup E kümesinde olmayan tüm sonuçları içerirken, $E \cap E^C = \emptyset$ olduğu için

$$P(E \cup E^c) = P(E) + P(E^c) = 1$$
$$P(E^c) = 1 - P(E)$$

yazılabilir.

E ve F kümelerinin kesişimleri boş küme değilse, kesişimlerinin olasılığını da dikkate almak gerekir:

$$P(E \cup F) = P(E) + P(F) - P(E \cap F) \tag{A.2}$$

A.1.2 Koşullu Olasılık

$P(E|F)$, F olayının olduğu bilindiğinde E olayının gerçekleşme olasılığıdır:

$$P(E|F) = \frac{P(E \cap F)}{P(F)} \tag{A.3}$$

F olayının olduğunun bilinmesi örnek uzayını olası tüm sonuçlardan F içindekilere indirir ve bunların içinde E kümesine de girenler $E \cap F$ içindekilerdir. Denklem A.3 yalnızca $P(F) > 0$ olduğunda tanımlıdır. Koşullu olasılığın her iki yönü için de bu tanım geçerlidir:

$$P(E \cap F) = P(E|F)P(F) = P(F|E)P(E)$$

ve buradan *Bayes kuralını* yazabiliriz:

$$P(F|E) = \frac{P(E|F)P(F)}{P(E)} \tag{A.4}$$

F_i kümeleri ayrık ve kapsayan ($\bigcup_{i=1}^{n} F_i = S$) olduğunda

$$\begin{aligned} E &= \bigcup_{i=1}^{n} E \cap F_i \\ P(E) &= \sum_{i=1}^{n} P(E \cap F_i) = \sum_{i=1}^{n} P(E|F_i)P(F_i) \end{aligned} \tag{A.5}$$

yazılır. Bayes kuralı sayesinde

$$P(F_i|E) = \frac{P(E \cap F_i)}{P(E)} = \frac{P(E|F_i)P(F_i)}{\sum_j P(E|F_j)P(F_j)} \tag{A.6}$$

yazılabilir. E ve F *bağımsız* olduklarında F olayının gerçekleşmiş olmasının E olayının gerçekleşme olasılığı üzerinde bir etkisi yoktur ve $P(E|F) = P(E)$ ya da

$$P(E \cap F) = P(E)P(F) \tag{A.7}$$

yazılabilir.

A.2 Rastsal Değişkenler

Rastsal değişken, rastsal bir deneyin örnek uzayındaki her sonucuna bir sayı atayan bir işlevdir.

A.2.1 Olasılık Dağılım, Kütle, ve Yoğunluk İşlevleri

X rastsal değişkeninin herhangi bir a değerinden küçük değer alma olasılığı $F(\cdot)$ *olasılık dağılım işlevi* ile gösterilir:

$$F(a) = P\{X \leq a\} \tag{A.8}$$

Buradan

$$P\{a < X \leq b\} = F(b) - F(a) \tag{A.9}$$

yazılabilir. Kesikli X için

$$F(a) = \sum_{\forall x \leq a} P(x) \tag{A.10}$$

öyle ki $P(a) = P\{X = a\}$, olarak tanımlanmış *olasılık kütle işlevidir*. Sürekli X için $p(\cdot)$ *olasılık yoğunluk işlevi* benzer biçimde tanımlanmıştır:

$$F(a) = \int_{-\infty}^{a} p(x)dx \tag{A.11}$$

A.2.2 Birleşik Dağılım ve Yoğunluk İşlevleri

Bazı deneylerde, birden çok rastsal değişken arasındaki ilişki bizim için önemli olabilir ve bu durumda X ve Y değişkenlerinin *birleşik* olasılık dağılım ya da yoğunluk dağılımını kullanırız:

$$F(x, y) = P\{X \leq x, Y \leq y\} \tag{A.12}$$

Ayrı ayrı dağılım ya da yoğunluklar, serbest değişken üzerinden *tümleştirmeyle* bulunabilir:

$$F_X(x) = P\{X \leq x\} = P\{X \leq x, Y \leq \infty\} = F(x, \infty) \tag{A.13}$$

Kesikli durumda

$$P(X = x) = \sum_j P(x, y_j) \tag{A.14}$$

ve sürekli durumda

$$p_X(x) = \int_{-\infty}^{\infty} p(x, y)dy \tag{A.15}$$

olarak yazılır. X ve Y *bağımsız* olduğunda iş basitleşir:

$$p(x, y) = p_X(x)p_Y(y) \tag{A.16}$$

Bu denklemler, ikiden daha çok rastsal değişken için genellenebilir.

A.2.3 Koşullu Dağılımlar

X ve Y rastsal değişkenleri için koşullu olasılığı hesaplarken birleşik dağılım kullanılabilir:

$$P_{X|Y}(x|y) = P\{X = x|Y = y\} = \frac{P\{X = x, Y = y\}}{P\{Y = y\}} = \frac{P(x, y)}{P_Y(y)} \tag{A.17}$$

A.2.4 Bayes Kuralı

Birleşik dağılmış iki rastsal değişkenden birinin değeri bilindiğinde ikincinin değeri *Bayes kuralı* kullanılarak hesaplanabilir:

$$P(y|x) = \frac{P(x|y)P_Y(y)}{P_X(x)} = \frac{P(x|y)P_Y(y)}{\sum_y P(x|y)P_Y(y)} \tag{A.18}$$

Aynı kuralı sözcüklerle de yazabiliriz:

$$\text{sonsal} = \frac{\text{olabilirlik} \times \text{önsel}}{\text{kanıt}} \tag{A.19}$$

Payda, tüm y değerleri için pay toplanarak bulunur. $p(y|x)$ dağılımının "biçimi" paya bağlıdır; paydaysa $p(y|x)$ toplamının 1 olmasını sağlayan normalleştiricidir. Bayes kuralı, x tarafından verilen olabilirlik bilgisini de kullanıp önsel olasılıktan sonsal olasılık oluşturur.

Bayes kuralı bağımlılıkları tersine çevirerek bilinen $p(x|y)$ olasılığından $p(y|x)$ olasılığını hesaplamayı sağlar. y değişkeninin x değişkeninin "nedeni" olduğunu varsayalım; örneğin, y yaz tatiline gitmek, x de yanık bir ten olsun. Bu durumda, $p(x|y)$, yaz tatiline gitmiş bir kişinin yanık tenli olma olasılığıdır ve bu *nedensel* yöndür. Bayes kuralı, $p(y|x)$ değerini, yani yanık tenli olduğunu bildiğimiz birinin tatile gitmiş olma olasılığını hesaplamamızı sağlayarak *tanı koymaya* izin verir. Bu durumda, $p(y)$, herhangi bir kişinin tatile gitmiş olma olasılığı, $p(x)$ ise herhangi bir kişinin yanık tenli olma olasılığına karşılık gelir.

A.2.5 Beklenen Değer

X rastsal değişkeninin *beklenen değeri* ya da *ortalaması* $E[X]$ ile gösterilir ve X değişkeninin çok sayıda deneyde aldığı değerlerin ortalamasıdır:

$$E[X] = \begin{cases} \sum_i x_i P(x_i) & X \text{ kesikliyse} \\ \int x p(x) dx & X \text{ sürekliyse} \end{cases} \tag{A.20}$$

Her değerin o değerin görülme olasılığıyla ağırlıklandırıldığı bir ortalamadır ve $a, b \in \Re$ için

$$\begin{aligned} E[aX + b] &= aE[X] + b \\ E[X + Y] &= E[X] + E[Y] \end{aligned} \tag{A.21}$$

koşullarını sağlar. Herhangi bir gerçek değerli $g(\cdot)$ için beklenen değer hesaplanabilir:

$$E[g(X)] = \begin{cases} \sum_i g(x_i) P(x_i) & X \text{ kesikliyse} \\ \int g(x) p(x) dx & X \text{ sürekliyse} \end{cases} \tag{A.22}$$

$g(x) = x^n$, X değikeninin n. *momenti* olarak adlandırılır:

$$E[X^n] = \begin{cases} \sum_i x_i^n P(x_i) & X \text{ kesikliyse} \\ \int x^n p(x) dx & X \text{ sürekliyse} \end{cases} \tag{A.23}$$

ve sıkça kullanılır. Örneğin, *ortalama* ilk momenttir ve μ ile gösterilir.

A.2.6 Değişinti

Değişinti, X değişkeninin beklenen değeri çevresinde ne kadar değiştiğini ölçer. $\mu \equiv E[X]$ iken değişintiyi

$$\text{Var}(X) = E[(X - \mu)^2] = E[X^2] - \mu^2 \tag{A.24}$$

olarak tanımlarız. Değişinti, ikinci moment eksi birinci momentin karesidir. σ^2 ile de gösterilen değişinti, $a, b \in \Re$ için

$$\text{Var}(aX + b) = a^2 \text{Var}(X) \tag{A.25}$$

koşulunu sağlar. $\sqrt{\text{Var}(X)}$ değerine *standart sapma* denir ve σ ile gösterilir. Standart sapma, X ile aynı ölçekte olduğu için yorumlanması daha kolaydır.

Eşdeğişinti, iki rastsal değişken arasındaki ilişkiyi ölçer. Eğer X olayının olması, Y olayının olma olasılığını yükseltiyorsa eşdeğişinti artı, X'in olması, Y'nin olma olasılığını azaltıyorsa eşdeğişinti eksi olur; aralarında böyle bir bağımlılık yoksa eşdeğişinti 0 olur ($\mu_X \equiv E[X]$ ve $\mu_Y \equiv E[Y]$):

$$\text{Cov}(X, Y) = E[(X - \mu_X)(Y - \mu_Y)] = E[XY] - \mu_X \mu_Y \tag{A.26}$$

$$\begin{aligned}
\text{Cov}(X,Y) &= \text{Cov}(Y,X) \\
\text{Cov}(X,X) &= \text{Var}(X) \\
\text{Cov}(X+Z,Y) &= \text{Cov}(X,Y)+\text{Cov}(Z,Y) \\
\text{Cov}\left(\sum_i X_i, Y\right) &= \sum_i \text{Cov}(X_i,Y) \qquad \text{(A.27)}
\end{aligned}$$

$$\text{Var}(X+Y) = \text{Var}(X)+\text{Var}(Y)+2\text{Cov}(X,Y) \qquad \text{(A.28)}$$

$$\text{Var}\left(\sum_i X_i\right) = \sum_i \text{Var}(X_i)+\sum_i\sum_{j\neq i}\text{Cov}(X_i,X_j) \qquad \text{(A.29)}$$

sağlanan başka eşitliklerdir.

X ve Y bağımsızsa, $E[XY] = E[X]E[Y] = \mu_X\mu_Y$ ve $\text{Cov}(X,Y) = 0$ olur. X_i rastsal değişkenleri bağımsızsa

$$\text{Var}\left(\sum_i X_i\right) = \sum_i \text{Var}(X_i) \qquad \text{(A.30)}$$

olur. *İlinti*, normalleştirilmiştir ve her zaman -1 ile 1 arasında bir değer alır:

$$\text{Corr}(X,Y) = \frac{\text{Cov}(X,Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}} \qquad \text{(A.31)}$$

A.2.7 Büyük Sayıların Zayıf Yasası

$\mathcal{X} = \{X^t\}_{t=1}^N$, ortalamaları μ ve değişintileri σ^2 olan bağımsız ve özdeş dağılmış rastsal değişkenler olsun. Herhangi bir $\epsilon > 0$ için

$$P\left\{\left|\frac{\sum_t X^t}{N} - \mu\right| > \epsilon\right\} \to 0 \text{ öyle ki } N \to \infty \qquad \text{(A.32)}$$

sağlanır. Yani, N deney sonucunun ortalması, N artarken gerçek ortalamaya yakınsar.

A.3 Bazı Özel Rastsal Değişkenler

Bazı rastsal değişkenler o kadar sık karşımıza çıkar ki onlara özel ad verilmiştir.

A.3.1 Bernoulli Dağılımı

Sonucu, "başarı" ya da "başarısızlık" olan bir deney yapıldığında X rastsal değişkeni başarı için 1, başarısızlık için 0 değerini alır. p, sonucun başarı olma olasılığıysa

$$P\{X=1\} = p \text{ ve } P\{X=0\} = 1-p \qquad \text{(A.33)}$$

kısaca

$$P\{X = i\} = p^i(1-p)^{1-i}, i = 0, 1 \tag{A.34}$$

olarak yazılır. Bernoulli dağılmış X için beklenen değer ve değişinti p parametresince tanımlanır:

$$E[X] = p, \ \mathrm{Var}(X) = p(1-p) \tag{A.35}$$

A.3.2 Binom Dağılımı

Aynı Bernoulli deneyi N kez bağımsız yinelendiğinde toplam başarı sayısını gösteren X rastsal değişkeni binom dağılır. N yineleme içinde i başarı görme olasılığı

$$P\{X = i\} = \left(\begin{array}{c} N \\ i \end{array} \right) p^i(1-p)^{N-i}, i = 0 \ldots N \tag{A.36}$$

olarak hesaplanır. X binom değişkeni için beklenen değer ve değişinti N ve p parametrelerince tanımlanır:

$$E[X] = Np, \ \mathrm{Var}(X) = Np(1-p) \tag{A.37}$$

A.3.3 Katlıterimli Dağılım

Bernoulli dağılımının daha genelini, olası iki durum yerine K tane ayrık ve kapsayan durum olduğunu düşünelim. Her durumun olasılığı p_i olsun: $\sum_{i=1}^{K} p_i = 1$. Bu biçimde bir deney N kez yinelendiğinde i. durumun görülme sayısını N_i ile gösterelim: $\sum_{i=1}^{k} N_i = N$. Bu durumda $N_1, N_2, \ldots, N_K$ katlıterimli bir dağılımdan gelir:

$$P(N_1, N_2, \ldots, N_K) = N! \prod_{i=1}^{K} \frac{p_i^{N_i}}{N_i!} \tag{A.38}$$

Bir özel durum, $N = 1$ olduğunda, yani deney tek bir kez yapıldığında oluşur. Bu durumda N_i rastsal değişkenleri 0/1 gösterge değişkenleri olur, öyle ki yalnız bir tanesi 1, öbürleri 0 olur ve denklem A.38 basitleşir:

$$P(N_1, N_2, \ldots, N_K) = \prod_{i=1}^{K} p_i^{N_i} \tag{A.39}$$

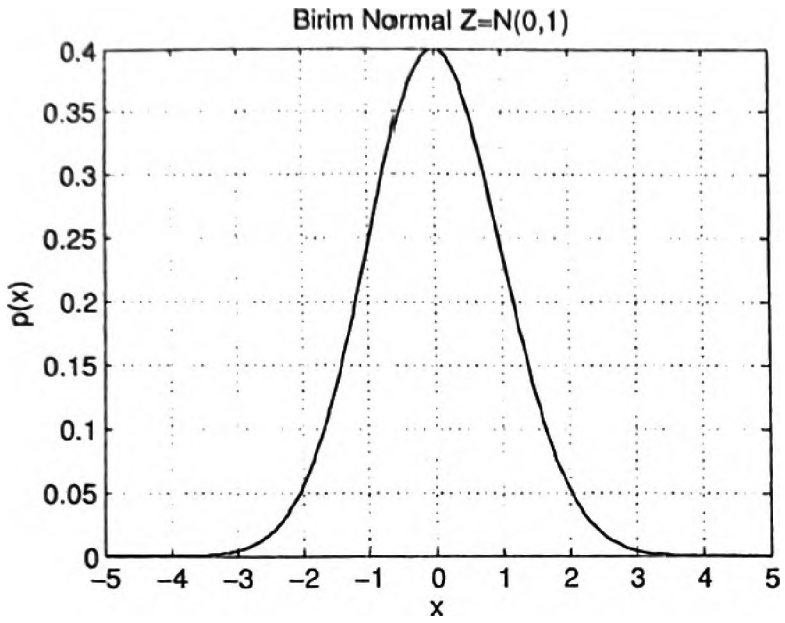
A.3.4 Tekdüze Dağılım

X, $[a, b]$ aralığında tekdüze dağıldığında her nokta için olasılık değeri aynıdır:

$$p(x) = \begin{cases} \frac{1}{b-a} & \text{eğer } a \leq x \leq b \text{ ise} \\ 0 & \text{değilse} \end{cases} \tag{A.40}$$

X tekdüzeyse ortalaması ve değişintisi iki uç değere bağlıdır:

$$E[X] = \frac{a+b}{2}, \ \mathrm{Var}(X) = \frac{(b-a)^2}{12} \tag{A.41}$$



Şekil A.1: Birim normal dağılımı, $\mathcal{Z}$.

A.3.5 Normal (Gauss) Dağılım

Beklenen değeri μ ve değişintisi σ^2 olan normal, ya da Gauss dağılan X rastsal değişkeni $\mathcal{N}(\mu, \sigma^2)$ olarak gösterilir:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right], -\infty < x < \infty \tag{A.42}$$

Doğadaki birçok rastsal gözlem yaklaşık olarak böyle, çan eğrisi biçiminde dağılır ve birçok gözlem belirli bir ideal değer çevresinde küçük oynamalar olarak düşünülebilir; adı herhalde bu nedenden dolayı *normal* dağılımdır. Bu biçimde yorumlarsak μ ideal değeri, σ da örneklerin bu ideal değer çevresinde ne kadar oynayabildiğini gösterir.

Dağılımın yüzde 68.27'si $(\mu-\sigma, \mu+\sigma)$, yüzde 95.45'i $(\mu-2\sigma, \mu+2\sigma)$, ve yüzde 99.73'ü $(\mu-3\sigma, \mu+3\sigma)$ aralığındadır. Yani, $P\{|x-\mu| < 3\sigma\} \approx 0.99$ olur. Gerçeklerken (örneğin bir yazılımda), $x < \mu - 3\sigma$ ya da $x > \mu + 3\sigma$ ise $p(x) \approx 0$ alınabilir. *Birim normal dağılımı* $\mathcal{Z} \equiv \mathcal{N}(0,1)$ olarak tanımlıdır (bkz. şekil A.1):

$$p_Z(x) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{x^2}{2}\right] \tag{A.43}$$

$X \sim \mathcal{N}(\mu, \sigma^2)$ ve $Y = aX + b$ ise $Y \sim \mathcal{N}(a\mu + b, a^2\sigma^2)$ olur. Bağımsız normal değişkenlerin toplamı yine normaldir: $\mu = \sum_i \mu_i$ ve $\sigma^2 = \sum_i \sigma_i^2$. $X \sim \mathcal{N}(\mu, \sigma^2)$ ise

$$\frac{X-\mu}{\sigma} \sim \mathcal{Z} \tag{A.44}$$

Buna z *normalleştirme* denir.

$X_1, X_2, \ldots, X_N$, hepsi ortalaması μ ve değişintisi σ^2 olan dağımsız ve özdeş rastsal değişkenler olsun. Büyük N için *merkezi limit kuramı*, toplamlarının yaklaşık normal dağıldığını söyler:

$$X_1 + X_2 + \ldots + X_N \sim \mathcal{N}(N\mu, N\sigma^2) \tag{A.45}$$

Örneğin, X binom(N, p) ise N Bernoulli deneyin toplamı olduğundan $(X - Np)/\sqrt{Np(1-p)}$ yaklaşık normal dağılmıştır.

Merkezi limit kuramı, bilgisayarda normal dağılmış rastsal değişkenler üretmek için de kullanılabilir. Programlama dillerinin $[0, 1]$ aralığında tekdüze (sahte)rastsal değişken üretmek için altprogramları vardır. U_i böylesi değişkenlerken $\sum_{i=1}^{12} U_i - 6$ yaklaşık olarak Z dağılır.

$X^t \sim \mathcal{N}(\mu, \sigma^2)$ olduğunu varsayalım. Örneklem ortalaması

$$m = \frac{\sum_{t=1}^{N} X^t}{N} \tag{A.46}$$

normal dağılmış bir rastsal değişkendir ve beklenen değeri μ değişintisi σ^2/N olur.

A.3.6 Ki Kare Dağılımı

Z_i rastsal değişkenleri bağımsız birim normallerse

$$X = Z_1^2 + Z_2^2 + \ldots + Z_n^2 \tag{A.47}$$

n serbestlik dereceli ki kare bir rastsal değişkendir: $X \sim \chi_n^2$. Beklenen değeri ve değişintisi n değerine bağlıdır:

$$E[X] = n, \ \text{Var}(X) = 2n \tag{A.48}$$

$X^t \sim \mathcal{N}(\mu, \sigma^2)$ ise örneklem değişintisi

$$S^2 = \frac{\sum_t (X^t - m)^2}{N - 1} \tag{A.49}$$

olarak tanımlanır ve ki kare dağılır:

$$(N-1)\frac{S^2}{\sigma^2} \sim \chi_{N-1}^2 \tag{A.50}$$

m ve S^2 dağımsızdır.

A.3.7 t Dağılımı

$Z \sim Z$ ve $X \sim \chi_n^2$ bağımsızsa

$$T_n = \frac{Z}{\sqrt{X/n}} \tag{A.51}$$

n serbestlik dereceli t dağılmış bir rastsaldır ve beklenen değeri ve değişintisi n değerine bağlıdır.

$$E[T_n] = 0, n > 1, \ \mathrm{Var}(T_n) = \frac{n}{n-2}, n > 2 \tag{A.52}$$

t dağılımı, birim normal dağılımı gibi 0 çevresinde bakışımlıdır. n arttıkça t birim normale gittikçe daha fazla benzer. t dağılımının kuyrukları biraz daha uzundur; bu da birim normale göre daha fazla değişintisi olduğunu gösterir.

A.3.8 F Dağılımı

$X_1 \sim \mathcal{X}^2_n$ ve $X_2 \sim \mathcal{X}^2_m$, sırasıyla n ve m serbestlik dereceli bağımsız ki kare rastsal değişkenlerse

$$F_{n,m} = \frac{X_1/n}{X_2/m} \tag{A.53}$$

n ve m serbestlik dereceli F dağılmış bir rastsaldır ve beklenen değer ve değişintisi, n ve m değerlerine bağlıdır:

$$E[F_{n,m}] = \frac{m}{m-2}, m > 2, \ \mathrm{Var}(F_{n,m}) = \frac{m^2(2m+2n-4)}{n(m-2)^2(m-4)}, m > 4 \tag{A.54}$$

A.4 Kaynaklar

Casella, G., R. L. Berger. 1990. *Statistical Inference.* Belmont, CA: Duxburry.

Ross, S. M. 1987. *Introduction to Probability and Statistics for Engineers and Scientists.* New York: Wiley.

Ek B

İngilizce – Türkçe Sözlük

İngilizce	Türkçe
action	eylem
activation	etkilenim
active	etkin
acyclic	çevrimsiz
adaptation	uyarlama
additive	eklemeli
aerodynamics	hava devinim bilimi
agent	etmen
agglomerative	birleştirmeli
algorithm	algoritma
alignment	hizalama
allele	eş gen
analysis	çözümleme
analytical	çözümsel
analyzer	çözümleyici
ancestor	ata
anchor	çapa
and	ve
anneal	tavlamak
approximate	yaklaşık
approximation	yaklaşıklama
arc	ayrıt
articulator	eklemleyici
association	ilişkilendirme
attribute	öznitelik
augmented	genişletilmiş
authentication	kimlik doğrulama
autoassociator	özilişkilendirici

automaton	özdevinir
axiom	belit
backpropagation	geri yayma
backup	geriye taşıma
backward	geriye doğru
bag of words	sözcük torbası
bandit	kollu kumar makinesi
base	taban
basis	taban
basket	sepet
batch	toptan
Bayesian	Bayesçi
belief	inanç
benchmark	denek
bias	yanlılık
bias unit	ek girdi
bin	kutucuk
binary	ikili
binding	değer atama
binomial	binom
biometrics	biyometri
blocking	bölükleme
Boolean	mantıksal
bootstrap	rastgele örnekleme
branch and bound	dallan ve sınırla
breadth-first	genişlik öncelikli
bump	tümsek
canonical	asal
capacity	sığım
cascade	ardışık
case-based	örnek tabanlı
causal	nedensel
cause	neden
central limit theorem	merkezi limit kuramı
chain	zincir
chapter	bölüm
characterize	betimlemek
chi-square	ki kare
city-block distance	şehir içi uzaklık
class	sınıf
classifier	sınıflandırıcı
clique	hizip
cluster	öbek
code	gizyazı

codebook vector	gizyazı yöneyi
column	sütun
combination	birleşim
competitive	yarışmacı
complement	tümleyen
complexity	karmaşıklık
component	bileşen
compression	sıkıştırma
computation	hesaplama
concave	içbükey
condensed	seyrek
conditional independence	koşullu bağımsızlık
confidence	güven
confusion (matrix)	hata (dizeyi)
conjugate	eşlenik
connected (graph)	bağlı (çizge)
connection	bağlantı
connectivity	bağlantısallık
constant	sabit
contingency (table)	olumsallık (çizelgesi)
contour	çevrit
contrast	zıtlık
convex	dışbükey
convolution	evrişim
cooperative	işbirlikçi
coordinates	eksen değerleri
correction	düzeltme
correlation	ilinti
cost	bedel
count	sayım
coupled	bağlı
covariance	eşdeğişinti
cover	kapsamak
credit assignment	sorumluluk atama
criterion	ölçüt
criterion	kıstas
critic	eleştirmen
cross-entropy	çapraz düzensizlik
cross-validation	çapraz geçerleme
cumulative (distribution)	birikimli (dağılım)
curse of dimensionality	çok boyutluluğun laneti
curvature	eğrilik
curve	eğri
cycle	çevrim

dataset	veri kümesi
decay	sönüm
decision	karar
decode	gizçözmek
decoder	gizçözer
decomposition	ayrışım
decremental	azalımlı
dendrogram	ağaç çizit
density	yoğunluk
depth-first	derinlik öncelikli
descendant	soyundan gelen
description	betimleme
design	tasarım
detail	ayrıntı
detection	sezim
deterministic	gerekirci
deviation	sapma
diagnosis	tanı
diagnostic	tanısal
diagonal	köşegen
diagram	çizem
dichotomizer	ikili ayırıcı
diffusion	yayınım
dilemma	ikilem
dimension	boyut
directed	yönlü
discontinuity	süreksizlik
discount	indirim
discrete	kesikli
discretization	kesiklileştirme
discriminant	ayırtaç
distance	uzaklık
distributed	dağıtık
distribution	dağılım
divergence	ıraksama
diversity	çeşitlilik
divisive	bölmeli
document	belge
dot product	iç çarpım
doubt	kuşku
dual	eşlek
dummy variable	yapay değişken
dynamic	devingen
edge	ayrıt

edit distance	düzeltme uzaklığı
efficient	etkili
eigenvector	özyöney
element	öğe
eligibility	uygunluk
eliminate	elemek
embedding	gömme
emission	yayım
empirical	gözlemsel
encode	gizyazmak
encoder	gizyazar
ensemble	küme
entropy	düzensizlik
environment	ortam
episode	serüven
epoch	dönem
equation	denklem
error	hata
error bar	hata çubuğu
eşölçümsel	isometric
et al	ve ötekiler
Euclidean distance	Euclid uzaklığı
evaluation	değerlendirme
evidence	kanıt
exact	kesin
example	örnek
exercise	alıştırma
exhaustive	kapsayan
expansion	açılım
expectation-maximization	beklenti büyütme
experiment	deney
expert	uzman
explaining away	örtbas etmek
expression	bildirim
extraction	çıkarım
extrapolation	dışdeğerleme
factor	çarpan
factor	etken
factorial	etkensel
feature	öznitelik
feedforward	ileriye doğru
figure	şekil
filter	süzgeç
finite horizon	sonlu serüven

fit	oturtma, uydurma
flexible	esnek
floating	kayan
fold	kat
forward	ileriye doğru
frontal	cepheden
function	işlev
fusion	kaynaştırma
fuzzy	bulanık
generalization	genelleme
generative	üretici
generic	genel
geodesic	yerölçümsel
gradient	eğim
graph	çizge
graphical	çizgesel
greedy	açgözlü
group	öbek
hard	keskin
hash table	anahtarlı çizelge
heuristics	sezgi
hidden	saklı
hierarchical	ağaç yapılı
higher-order	üst düzey
hinge (loss)	menteşe (yitimi)
hint	ipucu
histogram	çubukçizit
hybrid	karma
hyper—	üstün—
hypothesis	denence
identical	özdeş
if-then	eğer-ise
iid	bağımsız ve özdeşçe dağılmış
ill-posed	kötü konumlanmış
impurity	katışıklık
imputation	yükleme
incremental	artımlı
independence	bağımsızlık
index	dizin
indicator	gösterge
induction	kural oluşturma
inductive bias	model varsayımı
inference	çıkarsama
infinite horizon	sonsuz serüven

influence	etki
information	bilgi
inhibition	ketleme
initial	ilk
inner product	iç çarpım
input	girdi
instance	örnek
instantiate	somutlaşmak
interaction	etkileşim
internal	iç
interpolate	ara değerlemek
interpretability	yorumlanabilirlik
interval	aralık
invariance	değişmezlik
isometric	eşölçümsel
isoprobability	eşolasılık
item	öğe
iteration	yineleme
jobshop	sipariş
joint	birleşik
jump	sıçrama
junction	kavşak
k fold	k kat
kernel	çekirdek
kernelization	çekirdekleme
k-means	k merkezli
k-nearest neighbor	en yakın k komşu
knowledge	bilgi
latent	saklı
lateral	yanal
lattice	kafes
layer	katman
leader cluster	önder öbekleme
leaf	yaprak
leave-one-out	birini dışarıda bırak
level	düzey
lift (of an association rule)	etki (ilişkilendirme kuralının)
likelihood	olabilirlik
linear	doğrusal
link	bağ
loading	yükleme
local	yerel
logistic	S biçimli
logit	ters S

loop	döngü
loss	yitim
majority	çoğunluk
map	harita
mapping	eşleme
margin	kenar payı
marginalize	tümleştirme
matching	eşleştirme
matrix	dizey
maximization	büyütme
maximum a posteriori (MAP)	en büyük sonsal (EBS)
maximum likelihood (ML)	en büyük olabilirlik (EBO)
max-product	büyütme çarpma
mean	ortalama
measure	ölçüt
mechanism	düzenek
meiosis	eşeyli bölünme
membership	üyelik
memory	bellek
message	ileti
methodology	yöntembilim
metric	ölçev
minimization	küçültme
missing	eksik
mixture	karışım
modality	kip
mode (of a density)	tepe (dağılımın)
model	model
moralization	ahlaklılaştırma
multinomial (distribution)	katlıterimli (dağılım)
multiple	çoklu
multistage	çok aşamalı
multivariate	çok değişkenli
mutually exclusive	ayrık
naive	saf
negative	eksi
network	ağ
neuron	sinir hücresi
node	düğüm
noise	gürültü
nonlinear	doğrusal olmayan
nonnegative	eksi olmayan
nonparametric	dağılımdan bağımsız
norm	büyüklük

normal	normal
normalization	normalleştirme
NP-complete	çokterimli zamanda bulunamaz
nuisance factor	zararlı etken
null hypothesis	sıfır denencesi
numeric	sayısal
oblique	yatık
observable	gözlenebilir
observation	gözlem
Occam's razor	Occam'ın usturası
occluded	örtülü
offline	çevrimdışı
off-policy	politikasız
omnivariate	tüm değişkenli
one-sided	tek yanlı
online	çevrimiçi
on-policy	politikalı
optimal	en iyi
optimization	iyileme
or	veya
order (of a polynomial)	derece (çokterimlinin)
order (statistics)	sıra (istatistikleri)
origin (of axes)	sıfır noktası (eksenlerin)
oscillate	salınım
outcome	sonuç
outlier	aykırı
overfitting	aşırı öğrenme
overtraining	aşırı eğitme
paired	eşli
pairing	eşleme
pairwise	ikili
parallel	koşut
parametric	dağılıma bağlı
parent (node)	ebeveyn (düğüm)
parity	eşlik
partial	kısmi, kısmen
passive	edilgen
path	yol
pattern	örüntü
pedigree	soy
perceptron	algılayıcı
performance	başarım
phone	sesbirimcik
phoneme	sesbirim

phylogenetic tree	soy ağacı
piecewise	parçalı
pixel	imge noktası
plasticity	yoğrukluk
plate	tabaka
plot	çizim
plurality	çoğulluk
policy	politika
polychotomizer	çoklu ayırıcı
polyhedron	çokyüzlü
polynomial	çokterimli
polytree	çoklu ağaç
positive	artı
positive definite	kesin artı
positive semidefinite	yarı kesin artı
posterior	sonsal olasılık
posthoc	artçı
postpruning	geç budama
potential function	gerilim işlevi
power	üst
power function	güç işlevi
precision	kesinlik
predicate	belirtim
prediction	öngörü
predictive	öngörücü
prepruning	erken budama
primal	asal
principal	temel
principle	ilke
prior	önsel
probabilistic	olasılıksal
probability	olasılık
probably	olası
procedure	yordam
projection	izdüşüm
propagation	yayılım
propositional (rule)	önermeli (kural)
prototype	asıl örnek
pruning	budama
pseudocode	örnek program
purity	saflık
pursuit	izleme
quadratic	karesel, ikinci dereceden
quantization	nicemleme

query	sorgu
radial	dairesel
random	rastgele
random	rastsal
randomization	rastsallaştırma
range	açıklık
rank (of a matrix)	kerte (dizeyin)
rank (test)	sıra (sınaması)
real-time	gerçek zamanlı
reasoning	akıl yürütme
recall	anma
receiver	alıcı
receptive field	algı alanı
reciprocal	ters
recognition	tanıma
reconstruction	geriçatma
recurrent	özyineli
recursive	özçağrılı
redundancy	fazlalık
reference	dayanak
references	kaynaklar
region	alan
regression	bağlanım
regressogram	bağlanım çiziti
regularization	düzenlileştirme
reinforcement	pekiştirme
reject	ret
relative	görece
relevance	ilgililik
replication	çoğaltma
repository	veri tabanı
representation	gösterim
residual	artık
response	tepki
retrieval	erişim
reverse engineering	tersine mühendislik
reward	ödül
ridge	sırt
risk	risk
robust	gürbüz
row	satır
rule	kural
running	akan
sample	örneklemek (f), örneklem (i)

scalar	sayıl
scale	ölçek
scatter	saçılım
scene	görüntü
schedule	zaman çizelgesi
scree	kayşat
search	arama
section	bölüm
segmentation	bölütleme
selection	seçim
self-organizing	özörgütlemeli
semiparametric	dağılıma yarı bağlı
sensitivity	duyarlılık
sensor	alıcı
sequence	dizi
sequential	sırayla
set	küme
shaded	gölgeli
share	paylaşmak
shatter	tuz buz etmek
short term	kısa soluklu
sigmoid	sigmoid (S işlevi)
sign	işaret
significance	anlamlılık
simulation	benzetim
simultaneous	eş zamanlı
singly (connected)	tekli (bağlı)
singular	tekil
slack (variable)	artık (değişken)
smooth	düzleştirme
smoother	düzleştirici
soft	eşiksiz
softmax (function)	eşiksiz en büyük işlevi
sort	sıralamak
spam	istenmeyen elektronik posta
span	kapsamak
sparse	seyrek
spatial	uzamsal
specific	özgül
specificity	özgüllük
spectral	izgesel
spectrum	izge
spline	eğri
split	bölme

spread	yayılım
stable	kararlı
stacking	yığma
stepsize	adım büyüklüğü
stimulation	dürtü
stochastic	rastgele
strategy	yordam
stratification	katmanlaştırma
stress	gerilim
string	dizi
structure	yapı
stump	ağaççık
subgraph	altçizge
subset	altküme
subspace	altuzay
sum-product	toplam çarpım
supervised	gözetimli
support	destek
surface	yüzey
switching	geçişli
symbol	simge
symmetric	bakışımlı
symptom	belirti
synchronization	eşzamanlama
system	dizge
table	çizelge
tangent	teğet
template	şablon
temporal	zamansal
term	terim
terminal	uç
test	sınama
theory	kuram
threshold	eşik
time delay	zaman gecikmeli
time series	zaman dizisi
topographic	yerbetimsel
topological	ilingesel
trace	iz
trade-off	ödünleşim
trajectory	gezinge
transcribe	çevriyazmak
transition	geçiş
transpose	devrik

traveling salesman	gezgin satıcı
trellis	kafes
two-sided	iki yanlı
type I	1. tür
type II	2. tür
unbiased	yansız
unconstrained	kısıtlanmamış
underfitting	eksik öğrenme
unfold	açılmış
uniform (distribution)	tekdüze (dağılım)
univariate	tek değişkenli
unobservable	gözlenemeyen
unsupervised	gözetimsiz
utility	fayda
validation	geçerleme
variance	değişinti
variational	değişimsel
vector	yöney
version	sürüm
vigilance	tetiklik
virtual	sanal
voting	oylama
weight	ağırlık
winner-take-all	kazanan hepsini alır
wrapper	dürümcü
xor	dışlamalı veya

Ek C

Türkçe –İngilizce Sözlük

Türkçe	İngilizce
1. tür	type I
2. tür	type II
açgözlü	greedy
açıklık	range
açılım	expansion
açılmış	unfold
adım büyüklüğü	stepsize
ağ	network
ağaç çizit	dendrogram
ağaç yapılı	hierarchical
ağaççık	stump
ağırlık	weight
ahlaklılaştırma	moralization
akan	running
akıl yürütme	reasoning
alan	region
algı alanı	receptive field
algılayıcı	perceptron
algoritma	algorithm
alıcı	receiver
alıcı	sensor
alıştırma	exercise
altçizge	subgraph
altküme	subset
altuzay	subspace
anahtarlı çizelge	hash table
anlamlılık	significance
anma	recall

ara değerlemek	interpolate
aralık	interval
arama	search
ardışık	cascade
artçı	posthoc
artı	positive
artık	residual
artık (değişken)	slack (variable)
artımlı	incremental
asal	canonical
asal	primal
asıl örnek	prototype
aşırı eğitme	overtraining
aşırı öğrenme	overfitting
ata	ancestor
ayırtaç	discriminant
aykırı	outlier
ayrık	mutually exclusive
ayrıntı	detail
ayrışım	decomposition
ayrıt	arc
ayrıt	edge
azalımlı	decremental
bağ	link
bağımsız ve özdeşçe dağılmış	iid
bağımsızlık	independence
bağlanım	regression
bağlanım çiziti	regressogram
bağlantı	connection
bağlantısallık	connectivity
bağlı	coupled
bağlı (çizge)	connected (graph)
bakışımlı	symmetric
başarım	performance
Bayesçi	Bayesian
bedel	cost
beklenti büyütme	expectation-maximization
belge	document
belirti	symptom
belirtim	predicate
belit	axiom
bellek	memory
benzetim	simulation
betimleme	description

betimlemek	characterize
bildirim	expression
bileşen	component
bilgi	information
bilgi	knowledge
binom	binomial
birikimli (dağılım)	cumulative (distribution)
birini dışarıda bırak	leave-one-out
birleşik	joint
birleşim	combination
birleştirmeli	agglomerative
biyometri	biometrics
boyut	dimension
bölme	split
bölmeli	divisive
bölükleme	blocking
bölüm	chapter
bölüm	section
bölütleme	segmentation
budama	pruning
bulanık	fuzzy
büyüklük	norm
büyütme	maximization
büyütme çarpma	max-product
cepheden	frontal
çapa	anchor
çapraz düzensizlik	cross-entropy
çapraz geçerleme	cross-validation
çarpan	factor
çekirdek	kernel
çekirdekleme	kernelization
çeşitlilik	diversity
çevrim	cycle
çevrimdışı	offline
çevrimiçi	online
çevrimsiz	acyclic
çevrit	contour
çevriyazmak	transcribe
çıkarım	extraction
çıkarsama	inference
çizelge	table
çizem	diagram
çizge	graph
çizgesel	graphical

çizim	plot
çoğaltma	replication
çoğulluk	plurality
çoğunluk	majority
çok aşamalı	multistage
çok boyutluluğun laneti	curse of dimensionality
çok değişkenli	multivariate
çoklu	multiple
çoklu ağaç	polytree
çoklu ayırıcı	polychotomizer
çokterimli	polynomial
çokterimli zamanda bulunamaz	NP-complete
çokyüzlü	polyhedron
çözümleme	analysis
çözümleyici	analyzer
çözümsel	analytical
çubukçizit	histogram
dağılım	distribution
dağılıma bağlı	parametric
dağılıma yarı bağlı	semiparametric
dağılımdan bağımsız	nonparametric
dağıtık	distributed
dairesel	radial
dallan ve sınırla	branch and bound
dayanak	reference
değer atama	binding
değerlendirme	evaluation
değişimsel	variational
değişinti	variance
değişmezlik	invariance
denek	benchmark
denence	hypothesis
deney	experiment
denklem	equation
derece (çokterimlinin)	order (of a polynomial)
derinlik öncelikli	depth-first
destek	support
devingen	dynamic
devrik	transpose
dışbükey	convex
dışdeğerleme	extrapolation
dışlamalı veya	xor
dizey	matrix
dizge	system

dizi	sequence
dizi	string
dizin	index
doğrusal	linear
doğrusal olmayan	nonlinear
dönem	epoch
döngü	loop
duyarlılık	sensitivity
düğüm	node
dürtü	stimulation
dürümcü	wrapper
düzeltme	correction
düzeltme uzaklığı	edit distance
düzenek	mechanism
düzenlileştirme	regularization
düzensizlik	entropy
düzey	level
düzleştirici	smoother
düzleştirme	smooth
ebeveyn (düğüm)	parent (node)
edilgen	passive
eğer-ise	if-then
eğim	gradient
eğri	curve
eğri	spline
eğrilik	curvature
ek girdi	bias unit
eklemeli	additive
eklemleyici	articulator
eksen değerleri	coordinates
eksi	negative
eksi olmayan	nonnegative
eksik	missing
eksik öğrenme	underfitting
elemek	eliminate
eleştirmen	critic
en büyük olabilirlik (EBO)	maximum likelihood (ML)
en büyük sonsal (EBS)	maximum a posteriori (MAP)
en iyi	optimal
en yakın k komşu	k-nearest neighbor
erişim	retrieval
erken budama	prepruning
esnek	flexible
eş gen	allele

eş zamanlı	simultaneous
eşdeğişinti	covariance
eşeyli bölünme	meiosis
eşik	threshold
eşiksiz	soft
eşiksiz en büyük işlevi	softmax (function)
eşlek	dual
eşleme	mapping
eşleme	pairing
eşlenik	conjugate
eşleştirme	matching
eşli	paired
eşlik	parity
eşolasılık	isoprobability
eşölçümsel	isometric
eşzamanlama	synchronization
etken	factor
etkensel	factorial
etki	influence
etki (ilişkilendirme kuralının)	lift (of an association rule)
etkilenim	activation
etkileşim	interaction
etkili	efficient
etkin	active
etmen	agent
Euclid uzaklığı	Euclidean distance
evrişim	convolution
eylem	action
fayda	utility
fazlalık	redundancy
geç budama	postpruning
geçerleme	validation
geçiş	transition
geçişli	switching
genel	generic
genelleme	generalization
genişletilmiş	augmented
genişlik öncelikli	breadth-first
gerçek zamanlı	real-time
gerekirci	deterministic
geri yayma	backpropagation
geriçatma	reconstruction
gerilim	stress
gerilim işlevi	potential function

geriye doğru	backward
geriye taşıma	backup
gezgin satıcı	traveling salesman
gezinge	trajectory
girdi	input
gizçözer	decoder
gizçözmek	decode
gizyazar	encoder
gizyazı	code
gizyazı yöneyi	codebook vector
gizyazmak	encode
gölgeli	shaded
gömme	embedding
görece	relative
görüntü	scene
gösterge	indicator
gösterim	representation
gözetimli	supervised
gözetimsiz	unsupervised
gözlem	observation
gözlemsel	empirical
gözlenebilir	observable
gözlenemeyen	unobservable
güç işlevi	power function
gürbüz	robust
gürültü	noise
güven	confidence
harita	map
hata	error
hata (dizeyi)	confusion (matrix)
hata çubuğu	error bar
hava devinim bilimi	aerodynamics
hesaplama	computation
hizalama	alignment
hizip	clique
ıraksama	divergence
iç	internal
iç çarpım	dot product
iç çarpım	inner product
içbükey	concave
iki yanlı	two-sided
ikilem	dilemma
ikili	binary
ikili	pairwise

ikili ayırıcı	dichotomizer
ileriye doğru	feedforward
ileriye doğru	forward
ileti	message
ilgililik	relevance
ilingesel	topological
ilinti	correlation
ilişkilendirme	association
ilk	initial
ilke	principle
imge noktası	pixel
inanç	belief
indirim	discount
ipucu	hint
isometric	eşölçümsel
istenmeyen elektronik posta	spam
işaret	sign
işbirlikçi	cooperative
işlev	function
iyileme	optimization
iz	trace
izdüşüm	projection
izge	spectrum
izgesel	spectral
izleme	pursuit
k kat	k fold
k merkezli	k-means
kafes	lattice
kafes	trellis
kanıt	evidence
kapsamak	cover
kapsamak	span
kapsayan	exhaustive
karar	decision
kararlı	stable
karesel, ikinci dereceden	quadratic
karışım	mixture
karma	hybrid
karmaşıklık	complexity
kat	fold
katışıklık	impurity
katlıterimli (dağılım)	multinomial (distribution)
katman	layer
katmanlaştırma	stratification

kavşak	junction
kayan	floating
kaynaklar	references
kaynaştırma	fusion
kayşat	scree
kazanan hepsini alır	winner-take-all
kenar payı	margin
kerte (dizeyin)	rank (of a matrix)
kesikli	discrete
kesiklileştirme	discretization
kesin	exact
kesin artı	positive definite
kesinlik	precision
keskin	hard
ketleme	inhibition
kısa soluklu	short term
kısıtlanmamış	unconstrained
kısmi, kısmen	partial
kıstas	criterion
ki kare	chi-square
kimlik doğrulama	authentication
kip	modality
kollu kumar makinesi	bandit
koşullu bağımsızlık	conditional independence
koşut	parallel
köşegen	diagonal
kötü konumlanmış	ill-posed
kural	rule
kural oluşturma	induction
kuram	theory
kuşku	doubt
kutucuk	bin
küçültme	minimization
küme	ensemble
küme	set
mantıksal	Boolean
menteşe (yitimi)	hinge (loss)
merkezi limit kuramı	central limit theorem
model	model
model varsayımı	inductive bias
neden	cause
nedensel	causal
nicemleme	quantization
normal	normal

normalleştirme	normalization
Occam'ın usturası	Occam's razor
olabilirlik	likelihood
olası	probably
olasılık	probability
olasılıksal	probabilistic
olumsallık (çizelgesi)	contingency (table)
ortalama	mean
ortam	environment
oturtma, uydurma	fit
oylama	voting
öbek	cluster
öbek	group
ödül	reward
ödünleşim	trade-off
öğe	item
öğe	element
ölçek	scale
ölçev	metric
ölçüt	criterion
ölçüt	measure
önder öbekleme	leader cluster
önermeli (kural)	propositional (rule)
öngörü	prediction
öngörücü	predictive
önsel	prior
örnek	example
örnek	instance
örnek program	pseudocode
örnek tabanlı	case-based
örneklemek (f), örneklem (i)	sample
örtbas etmek	explaining away
örtülü	occluded
örüntü	pattern
özçağrılı	recursive
özdeş	identical
özdevinir	automaton
özgül	specific
özgüllük	specificity
özilişkilendirici	autoassociator
öznitelik	attribute
öznitelik	feature
özörgütlemeli	self-organizing
özyineli	recurrent

özyöney	eigenvector
parçalı	piecewise
paylaşmak	share
pekiştirme	reinforcement
politika	policy
politikalı	on-policy
politikasız	off-policy
rastgele	random
rastgele	stochastic
rastgele örnekleme	bootstrap
rastsal	random
rastsallaştırma	randomization
ret	reject
risk	risk
S biçimli	logistic
sabit	constant
saçılım	scatter
saf	naive
saflık	purity
saklı	hidden
saklı	latent
salınım	oscillate
sanal	virtual
sapma	deviation
satır	row
sayıl	scalar
sayım	count
sayısal	numeric
seçim	selection
sepet	basket
serüven	episode
sesbirim	phoneme
sesbirimcik	phone
seyrek	condensed
seyrek	sparse
sezgi	heuristics
sezim	detection
sıçrama	jump
sıfır denencesi	null hypothesis
sıfır noktası (eksenlerin)	origin (of axes)
sığım	capacity
sıkıştırma	compression
sınama	test
sınıf	class

sınıflandırıcı	classifier
sıra (istatistikleri)	order (statistics)
sıra (sınaması)	rank (test)
sıralamak	sort
sırayla	sequential
sırt	ridge
sigmoid (S işlevi)	sigmoid
simge	symbol
sinir hücresi	neuron
sipariş	jobshop
somutlaşmak	instantiate
sonlu serüven	finite horizon
sonsal olasılık	posterior
sonsuz serüven	infinite horizon
sonuç	outcome
sorgu	query
sorumluluk atama	credit assignment
soy	pedigree
soy ağacı	phylogenetic tree
soyundan gelen	descendant
sönüm	decay
sözcük torbası	bag of words
süreksizlik	discontinuity
sürüm	version
sütun	column
süzgeç	filter
şablon	template
şehir içi uzaklık	city-block distance
şekil	figure
tabaka	plate
taban	base
taban	basis
tanı	diagnosis
tanıma	recognition
tanısal	diagnostic
tasarım	design
tavlamak	anneal
teğet	tangent
tek değişkenli	univariate
tek yanlı	one-sided
tekdüze (dağılım)	uniform (distribution)
tekil	singular
tekli (bağlı)	singly (connected)
temel	principal

tepe (dağılımın)	mode (of a density)
tepki	response
terim	term
ters	reciprocal
ters S	logit
tersine mühendislik	reverse engineering
tetiklik	vigilance
toplam çarpma	sum-product
toptan	batch
tuz buz etmek	shatter
tüm değişkenli	omnivariate
tümleştirme	marginalize
tümleyen	complement
tümsek	bump
uç	terminal
uyarlama	adaptation
uygunluk	eligibility
uzaklık	distance
uzamsal	spatial
uzman	expert
üretici	generative
üst	power
üst düzey	higher-order
üstün—	hyper—
üyelik	membership
ve	and
ve ötekiler	et al
veri kümesi	dataset
veri tabanı	repository
veya	or
yaklaşık	approximate
yaklaşıklama	approximation
yanal	lateral
yanlılık	bias
yansız	unbiased
yapay değişken	dummy variable
yapı	structure
yaprak	leaf
yarı kesin artı	positive semidefinite
yarışmacı	competitive
yatık	oblique
yayılım	propagation
yayılım	spread
yayım	emission

yayınım	diffusion
yerbetimsel	topographic
yerel	local
yerölçümsel	geodesic
yığma	stacking
yineleme	iteration
yitim	loss
yoğrukluk	plasticity
yoğunluk	density
yol	path
yordam	procedure
yordam	strategy
yorumlanabilirlik	interpretability
yöney	vector
yönlü	directed
yöntembilim	methodology
yükleme	imputation
yükleme	loading
yüzey	surface
zaman çizelgesi	schedule
zaman dizisi	time series
zaman gecikmeli	time delay
zamansal	temporal
zararlı etken	nuisance factor
zıtlık	contrast
zincir	chain

Dizin

0/1 yitim işlevi, 40
1. tür (I. tür) hata, 425
2. tür (II. tür) hata, 425
5×2
    çapraz geçerleme, 417
    çapraz geçerlenmiş eşli F sınaması, 430
    çapraz geçerlenmiş eşli t sınaması, 430

AdaBoost, 371
Ağırlık
    eşiksiz paylaşım, 225
    paylaşımı, 220
    sönümü, 222
    yöney, 178
Ahlaklılaştırma, 354
Akaike bilgi ölçütü, 67
Akan düzleştirici
    doğru, 147
    ortalama, 146
Algı alanı, 243
Algılayıcı, 200
Alıcı işletim özellikleri, 419
Alıcı kaynaştırma, 363
Altküme seçimi, 90
Anlamlılık düzeyi, 425
Anma, 420
Apriori algoritması, 45
Aradeğerleme, 26
Aralık kestirimi, 421
Ardışık ilinti, 223
Ardışık sınıflandırıcılar, 376
Artçı sınama, 433
Artı örnekler, 15
Artık değişken, 266
Aşırı öğrenme, 30, 65
Aşırı eğitme, 218
Ayırtaç, 5
    doğrusal, 80
    işlevi, 42
    karesel, 77
Ayırtaç tabanlı sınıflandırma, 175
Aykırılık, 7
Aykırılık sezimi, 283

Bağımsız ve özdeşçe dağılmış, 31
Bağımsızlık, 334
Bağlama, 167
Bağlanım, 7, 26
    çokterimli, 61
    çokterimli çok değişkenli, 86
    doğrusal, 61
    gürbüz, 279
Bağlanım ağacı, 159
Bağlanım belirleme katsayısı, 62
Bağlantı ağırlığı, 200
Bağlı SMM, 343
Başlangıç olasılıkları, 312
Baum-Welch algoritması, 323
Bayes kuralı, 39, 444
Bayes sınıflandırıcısı, 40
Bayes topu, 346
Bayesçi bilgi ölçütü, 67
Bayesçi kestirici, 55
Bayesçi model birleşimi, 366
Bayesçi model seçimi, 67
Bayesci ağlar, 333
Bedel duyarlı öğrenme, 409
Beklenen hata, 408

Beklenti büyütme, 123
    gözetimli, 252
Belge öbekleme, 9
Belge sınıflandırma, 84
Belirtim, 167
Bellek tabanlı öğrenme, 136
Bellman denklemi, 387
Beta dağılımı, 294
Bileşen dağılımı, 118
Bilgi çıkarma, 164, 248
Bilgi erişimi, 419
Bilginin değeri, 397, 400
Binom sınama, 427
Birim normal dağılımı, 421
Birinci derece kural, 167
Birleştirmeli öbekleme, 129
Biyometri, 379
Bonferroni düzeltmesi, 434
Boyut azaltma
    doğrusal olmayan, 227
Bölükleme, 413
Bölmeli öbekleme, 129
Budama
    erken, 160
    geç, 161
Budama kümesi, 161
Bulanık k merkezli öbekleme, 131
Bulanık üyelik işlevi, 249
Bulanık kural, 249
Büyütme çarpım algoritması, 355

C4.5, 158
C4.5Rules, 161
CART, 158, 168

Çapa, 245
Çapraz düzensizlik, 185
Çapraz geçerleme, 30, 66, 416
    5 × 2, 417
    K kat, 416
Çarpım terimleri, 177
ÇBÖ, *bkz.* Çok boyutlu ölçekleme
Çeşitlilik, 362
Çekirdek düzleştiricisi, 147
Çekirdek işlevi, 138, 271, 302
Çekirdek kestiricisi, 138
Çekirdeklemek, 271
Çekirdekli temel bileşen çözümlemesi, 286
Çevrimiçi k merkezli öbekleme, 236
Çevrimiçi öğrenme, 203
Çizgesel modeller, 333
Çoğaltma, 412
Çok boyutlu ölçekleme, 102
    doğrusal olmayan, 228
Çok boyutluluğun laneti, 141
Çok çekirdekli öğrenme, 277
Çok değişkenli ağaç, 168
Çok değişkenli çokterimli bağlanım, 86
Çok değişkenli doğrusal bağlanım, 85
Çok katmanlı algılayıcılar, 207
Çokboyutlu ölçekleme
    doğrusal olmayan, 242
Çoklu ağaç, 350
Çoklu çekirdek öğrenme, 379
Çoklu karşılaştırma, 434
Çokterimli bağlanım, 61
    çok değişkenli, 86
Çözümleme düzeyleri, 198
Çubukçizit, 137
Çubukçizit düzleştiricisi, 145

d ayrımı, 346
DAÇ, *bkz.* Doğrusal ayırtaç çözümlemesi
Dağılım kestirimi, 8
Dağılıma yarı bağlı kestirim, 118
Dağılımdan bağımsız kestirim, 135
Dağılımdan bağımsız sınama, 434
Dağıtık ya da yerel gösterim, 128, 244
Dairesel tabanlı işlev, 245
Dayanak yöneyi, 119
Değer yineleme, 388
Değişinti, 53
Değişinti oranı, 94
Değişkenlik çözümlemesi, 431
Denence, 17
    en özgül, 18
    en genel, 18

sınıf, 17
Denence sınaması, 424
Deney
etkensel, 411
Deney yordamları, 411
Destek yöney makinesi, 265
Destek, ilişkilendirme kuralı, 44
Devingen çizgesel modeller, 357
Devingen düğüm yaratma, 223
Devingen programlama, 387
Devingen sınıflandırıcı seçimi, 374
Dışdeğerleme, 26
Dirichlet dağılımı, 293
Doğrusal ayırtaç, 80, 176
Doğrusal ayırtaç çözümlemesi, 105
Doğrusal ayrılabilme, 180
Doğrusal bağlanım, 61
çok değişkenli, 85
Doğrusal devingen dizge, 343
Doğrusal parçalı yaklaşıklama, 254
Doğrusal sınıflandırıcı, 80, 180
Dönem, 211
DTİ, *bkz.* Dairesel tabanlı işlev
Duyarlılık, 420
Dürümcü, 113
Düzeltme uzaklığı, 275
Düzenlileştirilmiş ayırtaç çözümlemesi, 82
Düzenlileştirme, 66, 225
Düzensizlik, 156
Düzleştirici, 145
Düzleştirici eğri, 149
DYM, *bkz.* Destek yöney makinesi

EBO, *bkz.* En büyük olabilirlik kestirimi
EBS, *bkz.* En büyük sonsal kestirim
EÇ, *bkz.* Etken çözümlemesi
Eşölçümsel öznitelik eşleme, 110
Eşdeğişinti dizeyi, 72
Eşdeğişinti işlevi, 305
Eşik, 178
işlevi, 201
Eşiksiz ağırlık paylaşımı, 225
Eşiksiz en büyük işlevi, 188
Eşiksiz hata, 266
Eşiksiz sayım, 323
Eşlek gösterim, 287, 301
Eşleme, 413
Eşlenik önsel, 294
Eşli sınama, 429
EĞER-İSE kuralları, 161
Eğim iniş, 184
rastgele, 203
Eğim yöneyi, 184
Eğri altında kalan alan, 419
Eksi örnekler, 15
Eksik öğrenme, 30, 65
Eleştirmen, 384
Elektronik postaların süzülmesi, 85
En büyük olabilirlik kestirimi, 50
2. tür, 308
En büyük sonsal kestirim, 55, 293
En iyi ayıran üstündüzlem, 263
En iyi politika, 386
En küçük betimleme uzunluğu, 67
En küçük kare fark sınaması, 433
En küçük kare kestiricisi, 61
En yakın k komşu
düzleştirici, 147
olasılık kestirimi, 140
sınıflandırıcı, 143
En yakın komşu
seyrek, 144
En yakın komşu sınıflandırıcısı, 143
En yakın ortalama sınıflandırıcısı, 81
Erken budama, 160
Erken durma, 186, 218
Esnek ayırtaç çözümlemesi, 97
Etken çizgesi, 354
Etken çözümleme karışımları, 127
Etken çözümlemesi, 98
Etkensel SMM, 343
Etki çizemleri, 356
Etki, ilişkilendirme kuralı, 44
Etkin öğrenme, 308
Euclid uzaklığı, 80
Evrensel yaklaşıklayıcılık, 210

Fayda işlevi, 43
Fayda kuramı, 43
Fisher çekirdeği, 276
Fisher doğrusal ayırtacı, 106
Foil, 165

Gamma dağılımı, 296
Gamma işlevi, 293
Gauss önsel dağılım, 298
Geç budama, 161
Geçerleme kümesi, 30
Geçiş olasılıkları, 312
Geçişli SMM, 343
Genelleştirme, 18
Genelleme, 29
Genellenmiş doğrusal modeller, 194
Gerçek zamanlı özyineli öğrenme, 229
Geri değişkeni, 319
Geriçatma hatası, 97, 120
Gerilim işlevi, 353
Geriye doğru seçim, 90
Gezgin satıcı uygulaması, 258
Gini ölçütü, 157
Girdi, 71
Girdi çıktı SMM, 326, 342
Girdi gösterimi, 15
Gizyazı yöneyleri, 119
Görece kare hata, 62
Gösterim
    dağıtık ya da yerel, 242
Gözetimli öğrenme, 7
Gözlem, 71
Gözlem olasılığı, 314
Gözlemsel çekirdek gösterimi, 275
Gözlemsel hata, 17
Gözlenebilir değişken, 37
Gözlenebilir Markov modeli, 313
Gözlenemeyen değişkenler, 37
Gram dizeyi, 272
Güç işlevi, 425
Gürültü, 23
Gürültülü VEYA, 352
Gürbüz bağlanım, 279
Güven aralığı
    iki yanlı, 422
    tek yanlı, 423
Güven, ilişkilendirme kuralı, 44

Hamming uzaklığı, 142
Hata
    I. tür, 425
    II.tür, 425
Hata düzelten çıktı gizyazıları, 278, 368
Hata geri yayma, 210
Hebbci öğrenme, 238
Hizalama, 275
Hizip, 353

ID3, 158
Irep, 165

İki yanlı güven aralığı, 422
İki yanlı sınama, 425
İkili ayırma, 369
İkili bölme, 155
İkili sınıflandırma, 181
İkinci dereceden ayırtaç, 176
İleri değişkeni, 317
İleri-geri yöntemi, 317
İleriye doğru seçim, 90
İlişkilendirme kuralı, 3, 44
İlinti, 72
İnanç ağları, 333
İnanç durumu, 397
İndirim oranı, 386
İpucu, 220
İşaret sınaması, 435
İteleme, 371
İzgesel ayrışım, 94

K kat
    çapraz geçerleme, 416
    çapraz geçerlenmiş eşli t sınaması, 429
K kollu kumar kakinesi, 384
k merkezli öbekleme, 120
    bulanık, 131
    çevrimiçi, 236

Kalman süzgeci, 343
Kanıt, 39
Karışım bileşenleri, 118
Karışım dağılımı, 118
Karışım oranı, 118
Karışımların karışımı, 129
Karar ağacı, 153
    çok değişkenli, 168
    eşiksiz, 258
    tüm değişkenli, 170
    tek değişkenli, 155
Karar bölgesi, 42
Karar düğümü, 153
Kararlılık-yoğrukluk ikilemi, 237
Kararsız algoritma, 370
Karesel ayırtaç, 77
Karhunen-Loève açılımı, 97
Katışıklık ölçütü, 155
Katmanlaştırma, 416
Kavşak ağacı, 352
Kayşat çiziti, 95
Kayan arama, 91
Kenar payı, 19, 263, 372
Kesinlik
    bilgi erişiminde, 419
    değişintinin tersi, 296
Kısmen gözlenebilir Markov karar süreci, 396
Koşullu bağımısızlık, 334
Koşut işlem, 199
Kolmogorov karmaşıklığı, 67
Konuşma tanıma, 327
Kötü konumlanmış problem, 29
Kruskal-Wallis sınaması, 437
Kuşku, 20
Kural
    öğrenme, 165
    çıkarımı, 248
    budama, 164
Kural değer ölçevi, 165
Kural desteği, 164
Küre düğümü, 168

Laplace önseli, 299
Laplace yaklaşıklama, 303
Lasso, 301
Log olabilirlik, 50
Log olasılık oranı, 46, 182
Logit, 182

Mahalanobis uzaklığı, 74
Markov karar süreci, 386
Markov modeli, 312
    öğrenme, 314, 322
    gözlenebilir, 313
    saklı, 314
Markov rastsal alanları, 353
McNemar sınaması, 428
Melez öğrenme, 245
Menteşe kayıp işlevi, 268
Merkezi limit kuramı, 449
Model birleştirme
    koşut, 364
    sıralı, 364
Model kümesi seçimi, 376
Model seçimi, 29
Model varsayımları, 29
Momentum, 217
Müşteri bölütleme, 128
Müşteri ilişkileri yönetimi, 128

Nedensellik, 341
    nedensel çizge, 334
Nicemleme, 119

OC1, 168
Occam'ın usturası, 24
Olabilirlik, 50
Olabilirlik oranı, 46
Olabilirlik tabanlı sınıflandırma, 175
Olası yaklaşık doğru öğrenme, 21
Olasılıksal ağlar, 333
Olasılıksal TBÇ, 100
Olumsallık çizelgesi, 428
Ortak temel bileşen çözümlemesi, 97
Ortalama kare hata, 52
Ortalama yöneyi, 72
Oylama, 365

Öbek, 118
Öbek ağaçları, 129
Öbekleme, 8
    ağaç biçimli, 129
    birleştirmeli, 129
    bölmeli, 129
    çevrimiçi, 236
Öğrenen mantık programlama, 167
Öğrenen özdevinirler, 402
Öğrenen yöney nicemleme, 253
Önder öbekleme algoritması, 122
Önermeli kural, 167
Öngörü, 5
Önsel bilgi, 248
Önsel dağılım, 54
Önsel olasılık, 39
Önsel olasılık dağılımı, 291
Örnek, 71
Örnek tabanlı akıl yürütme, 150
Örnek tabanlı öğrenme, 136
Örneklem, 38
    eşdeğişinti dizeyi, 73
    ilinti katsayıları, 73
    ortalama, 72
Örtbas etmek, 339
Örüntü tanıma, 5
Özörgütlemeli harita, 241
Özgüllük, 420
Özilişkilendirici, 227
Öznitelik, 71
Öznitelik ağacı, 220
Öznitelik çıkarımı, 90
Öznitelik seçimi, 90
Özrakamlar, 96
Özyüzler, 96
Özyineli ağ, 229

Parçalı yaklaşıklama
    doğrusal, 254
    sabit, 210, 253
Parzen penceresi, 139
Pekiştirmeli öğrenme, 10
Politika, 386
Politikalı, 392
Politikasız, 392
Probit işlevi, 304

Q öğrenme, 391

Rastgele eğim inişi, 203
Rastgele özdevinir, 312
Rastsal altuzay, 363
Rastsallaştırma, 412
Renk nicemleme, 119
Ret, 26, 41
Ripper, 165
Risk işlevi, 40

S biçimli sınıflandırma, 185
S işlevi, 183
Saçılım, 106
Saf Bayes sınıflandırıcısı, 80, 341
    kesikli girdi, 84
Saf kestirici, 137
Saklı değişkenler, 45, 341
Saklı etkenler, 98
Saklı katman, 207
Saklı Markov modeli, 314, 342
    bağlı, 343
    etkensel, 343
    geçişli, 343
    girdi çıktı, 343
    soldan sağa, 327
Sammon eşlemesi, 105
    doğrusal olmayan, 228
Sammon gerilimi, 105
Sanal örnek, 221
Sarsa, 392
    Sarsa(λ), 393
Sepet çözümlemesi, 44
Serüven, 386
Sesbirimcik, 328
Seyrek en yakın komşu, 144
Sıfır denencesi, 424
Sıfır-bir yitimi, *bkz.* 0/1 yitimi
Sıkıştırma, 119
Sınıf
    hata dizeyi, 420

olabilirliği, 39
Sınıf içi saçılım dizeyi, 107
Sınıflandırma, 4
Olasılık ya da ayırtaç tabanlı, 175
Sınıflandırma ağacı, 155
Sınıflar arası saçılım dizeyi, 107
Sırayla kapsama, 165
Sırt bağlanımı, 225, 299
Sigmoid, 183
Sinir hücresi, 197
Sipariş çizelgeleme, 402
SMM, *bkz.* Saklı Markov modeli
Soldan sağa SMM, 327
Sonlu serüven, 386
Sonsal olasılık, 39
Sonsal olasılık dağılımı, 54, 291
Sonsuz serüven, 386
Sorumluluk atama, 384
Soy, 343
Soy ağacı, 342
Sözcük torbası, 84, 275
Sürüm uzayı, 19

Şablon eşleme, 81

t dağılımı, 424
t sınaması, 426
Taban işlevi
çekirdek için, 301
Taban işlevleri, 177
işbirlikçi ya da yarışmacı, 251
normalleştirme, 249
Tam bağlı öbekleme, 130
TBÇ, *bkz.* Temel bileşen çözümlemesi
TD öğrenme
TD(0), 392
Tek bağlı öbekleme, 129
Tek değişkenli ağaç, 155
Tek sınıf sınıflandırma, 283
Tek yanlı güven aralığı, 423
Tek yanlı sınama, 425
Temel bileşen çözümleme karışımı, 127
Temel bileşen çözümlemesi, 92
Temel model, 361
Tepki yüzeyi tasarımı, 412
Tetiklik, 240
Toplam çarpım algoritması, 354
Toptan öğrenme, 211
Tukey sınaması, 438
Tüm değişkenli karar ağacı, 170

Uygunluk izi, 392
Uzmanların karışım ağacı, 257
Uzmanların karışımı, 254, 373
ağaç biçiminde, 257
Markov, 326, 343
yarışan, 256
yardımlaşma, 256
Uzmanların Markov karışımı, 326

Üretici model, 292, 341
Üretici yerbetimsel eşleme, 258
Üst düzey terimler, 177

Vapnik-Chervonenkis (VC) boyutu, 20
Viterbi algoritması, 321
Voronoi mozaiği, 143

Wilcoxon işaretli sıra sınaması, 437

Yaklaşık normal sınama, 427
Yanal ketleme, 238
Yanlılık, 52
Yanlılık/değişinti ikilemi, 64
Yansız kestirici, 52
Yapısal risk küçültmesi, 67
Yapısal uyarlama, 222
Yapay sinir ağları, 197
Yaprak düğümü, 153
Yarışmacı öğrenme, 236
Yarışmacı taban işlevleri, 251
Yayımlama kümesi, 31
Yayınım çekirdeği, 275
Yerölçümsel uzaklık, 109
Yerbetimsel harita, 242
Yerel ağırlıklandırılmış akan doğru düzleştirici, 148
Yerel doğrusal gömme, 111
Yerel gösterim, 242

Yığılmış genelleme, 374
Yitim işlevi, 40
Yorumlanabilirlik, 161
Yöney nicemleme
    öğrenen, 253
Yönlü çevrimsiz çizge, 333
Yükleme, 73

z, *bkz.* Birim normal dağılımı
z normalleştirme, 75, 448
Zaman gecikmeli sinir ağ, 228
Zaman içinde açılma, 229
Zamanda hata geri yayma, 229
Zamansal fark, 389
    öğrenme, 391
    TD-Gammon, 402

Yapay öğrenme, bilgisayarların bir problemi çözmekte örnek veri ya da deneyimi kullanacak biçimde programlanmasıdır. Yüz tanıma, konuşma tanıma, istenmeyen e-postaların süzülmesi, kredi risk hesabı, müşteri bölütleme gibi uygulamalar yanında yapay zekâda, robotikte, biyoinformatikte, doğal dil işlemede son yıllarda büyük miktarda veriyi işleyerek bilgi çıkartan yapay öğrenme yöntemleri her gün daha çok öne çıkmaktadır.

Kapsamı oldukça geniş olan bu kitapta, gözetimli, gözetimsiz ve pekiştirmeli öğrenme için kullanılan tüm yaklaşımlar, olasılık tabanlı modeller, yapay sinir ağları, karar ağaçları, saklı Markov modelleri, çekirdek makineleri, Bayesçi yöntemler ve çizgesel ağlar işlenmekte, ayrıca boyut azaltma, öbekleme, model birleştirme, deney tasarımı ve çözümleme gibi konular ayrıntılı olarak tartışılmaktadır. Tüm yöntemler, okuyucunun gerektiğinde denklemleri bilgisayar programına kolayca dönüştürebilmesini sağlayacak düzeyde ele alınmaktadır.

Ethem Alpaydın (1966, Malatya), 1987 yılında Boğaziçi Üniversitesi Bilgisayar Mühendisliği Bölümü'nden lisans, 1990 yılında Ecole Polytechnique Fédérale de Lausane'dan doktora derecelerini aldı. UC Berkeley (ABD), MIT (ABD) ve Idiap'ta (İsviçre) misafir araştırmacı olarak çalıştı. 1997'de Fulbright Senior Scholar, 1998'de genç ve 2008'de kıdemli olarak Boğaziçi Üniversitesi Vakfı Araşırmada Üstün Başarı, 2001'de TÜBA Üstün Başarılı Genç Bilim İnsanı, 2002'de Tübitak Teşvik ödüllerini aldı. Boğaziçi Üniversitesi Bilgisayar Mühendisliği Bölümü'nde profesör olarak çalışmaktadır.

ISBN 978-605-4238-49-1

9 786054 238491

BOĞAZİÇİ ÜNİVERSİTESİ YAYINEVİ