Yöntem ile Makine Öğrenmesi

ÖNDER ÖRNEK

Yöntem ile Makine Öğrenmesi

20 Algoritma

BİRİNCİ BASKI
2026

Yöntem ile Makine Öğrenmesi: 20 Algoritma
Önder Örnek

Telif hakkı © 2026 Önder Örnek.

Bu eser Creative Commons Atıf 4.0 Uluslararası Lisansı (CC BY 4.0) ile lisanslanmıştır. Yazarın adı belirtilerek, lisansa bağlantı verilerek ve yapılan değişiklikler gösterilerek ticari amaçlar dahil her ortamda ve biçimde kopyalanabilir, dağıtılabilir ve uyarlanabilir. Lisans metni:
`https://creativecommons.org/licenses/by/4.0/deed.tr`

Eserdeki örnek programlar MIT lisansıyla kullanılabilir. Yöntem yazılımı da MIT lisanslıdır.

Birinci baskı: Kasım 2026

Dizgi: Metin TeX Gyre Pagella, program kodları DejaVu Sans Mono yazı tipiyle, XeLaTeX kullanılarak dizilmiştir.

İçindekiler

Önsöz vii

Giriş 1
Başlarken 1
Kodları çalıştırmak 1
Kullanılan araçlar 2

1 Denetimli öğrenmenin temelleri 3
1. Doğrusal regresyon 3
2. Lojistik regresyon 5
3. K-en yakın komşu (k-NN) 7
4. Naif Bayes 8
5. Karar ağaçları 10

2 Topluluk yöntemleri ve destek vektör makinesi 15
6. Rastgele orman 15
7. Gradyan artırma 19
8. Destek vektör makinesi (SVM) 21
14. AdaBoost 23
15. XGBoost 25

3 Denetimsiz öğrenme 29
9. K-ortalamalar 29
10. Hiyerarşik kümeleme 31
11. DBSCAN 33
12. Temel bileşenler analizi (PCA) 35
13. Gauss karışım modelleri (GMM) 36

4 Derin öğrenme 39
16. Çok katmanlı algılayıcı (sinir ağı) 39
17. Evrişimli sinir ağı (CNN) 41
18. Yinelemeli sinir ağı (RNN) 44
19. Otokodlayıcı 47
20. Transformer ve dikkat mekanizması 48

A Hangi algoritma ne zaman? 53

Önsöz

Bu kılavuz, makine öğrenmesinin en çok kullanılan 20 algoritmasını Yöntem ile, kodu çalıştırarak öğretir. Her algoritma birkaç düzine satırlık, baştan sona okunabilen bir programla anlatılır; hazır bir kütüphaneyi çağırmak yerine fikrin kendisini kodlarsınız.

Giriş

Başlarken

Kılavuz, Yöntem'in temellerini (değişkenler, döngüler, fonksiyonlar, listeler) bilen ama makine öğrenmesine yeni başlayan okurlar içindir. Matematik bilgisi olarak toplama, çarpma ve ortalama yeterlidir; gereken her formül yanında Türkçe açıklamasıyla verilir.

Her bölüm aynı sırayı izler:

1. **Fikir**: algoritma neyi, hangi sezgiyle yapar.
2. **Formül**: algoritmanın kalbindeki tek denklem.
3. **Yöntem kodu**: tam, çalışan bir program.
4. **Çıktı**: programın gerçekten ürettiği sonuç.
5. **Ne öğrendik**: sonucun yorumu ve sık yapılan hata.
6. **Alıştırma**: kodu değiştirerek deneyebileceğiniz küçük bir görev.

Kodları çalıştırmak

Programların hepsi `ml_kilavuz/` klasöründedir ve oyun alanında "Makine öğrenmesi kılavuzu" grubunda hazır bekler. Bilgisayarınızda şöyle çalıştırılır:

```
node js/dist/cli.js ml_kilavuz/01_dogrusal_regresyon.y
```

Çıktılar `Tohum(...)` ile sabitlenmiş rastgele sayılar kullandığı için her çalıştırmada aynıdır; kendi bilgisayarınızda da buradaki sonuçları birebir görmelisiniz.

Kullanılan araçlar

Araç	Ne için	Nasıl gelir
Listeler ve döngüler	Veri kümeleri, uzaklıklar, sayaçlar	Dilin kendisi
`Matematik` birimi	`Üstel`, `Ln`, `Kök`, `Sigmoid` gibi işlevler	`kullan Matematik`
`YZ` birimi	Tensörler, otomatik türev, `Yoğun` katman, `Adam`	`kullan YZ`
`MakineOgrenmesi` birimi	Hazır k-NN, k-ortalamalar ve regresyonlar	`kullan MakineOgrenmesi`

İlk bölümlerde algoritmaları sıfırdan yazıyoruz; hazır birimleri yalnızca sonucu karşılaştırmak için kullanıyoruz. Derin öğrenme bölümünde ise `YZ` biriminin otomatik türevi, gradyan hesabını bizim yerimize yapar.

BÖLÜM

1

Denetimli öğrenmenin temelleri

Denetimli öğrenmede elimizde doğru cevabı bilinen örnekler vardır; model bu örneklerden genel bir kural çıkarır. Cevap bir sayıysa (fiyat) **regresyon**, bir sınıfsa (geçti/kaldı) **sınıflandırma** yapıyoruz demektir.

1. Doğrusal regresyon

Fikir. Veriye en iyi uyan düz çizgiyi buluruz. "En iyi", tahminlerle gerçek değerler arasındaki farkların karelerinin ortalamasının (OKH, ortalama kare hata) en küçük olduğu çizgidir. Eğimi ve kesişimi gradyan inişiyle adım adım düzeltiriz: hatayı artıran yönün tersine küçük bir adım.

$$\hat{y} = ax + b, \qquad \text{OKH} = \frac{1}{n}\sum_{i}(\hat{y}_i - y_i)^2$$

```
program DoğrusalRegresyon
{ Ev büyüklüğünden (m²) fiyatı (bin TL) tahmin eden doğrusal model: ŷ = a·x + b
  Parametreler gradyan inişiyle öğrenilir, sonra kapalı form çözümüyle
      ↪ karşılaştırılır. }

değişken
  X, Y: ondalık dizisi
  a, b, ga, gb, hata, okh, oran: ondalık
  i, adım, n: tamsayı
  ox, oy, pay, payda: ondalık

fonksiyon Ölçekle(m2: ondalık): ondalık
başla
  sonuç := m2 / 100      { 100 m² = 1 birim: sayılar küçük kalınca iniş kararlı olur
      ↪ }
bitir

başla
  X := [50, 65, 80, 95, 110, 130, 150]
  Y := [1450, 1900, 2250, 2700, 3100, 3600, 4150]
  n := Uzunluk(X)
  a := 0; b := 0; oran := 0.1

  1'den 3000'e kadar her adım için
  başla
    ga := 0; gb := 0; okh := 0
    0'dan n - 1'e kadar her i için
    başla
      hata := a * Ölçekle(X[i]) + b - Y[i]
      ga := ga + 2 * hata * Ölçekle(X[i]) / n
      gb := gb + 2 * hata / n
      okh := okh + hata * hata / n
    bitir
    a := a - oran * ga
    b := b - oran * gb
    eğer adım = 1 veya adım mod 1000 = 0 ise
      yaz(Biçimle('Adım {:5}  OKH = {:12.2}', adım, okh), 1)
  bitir
  yaz(Biçimle('Gradyan inişi: fiyat ≈ {:.2} × m² + {:.2}', a / 100, b), 1)

  { Kapalı form (en küçük kareler): a = Σ(x-x̄)(y-ȳ) / Σ(x-x̄)² }
  ox := Toplam(X) / n; oy := Toplam(Y) / n
  pay := 0; payda := 0
  0'dan n - 1'e kadar her i için
  başla
    pay := pay + (X[i] - ox) * (Y[i] - oy)
    payda := payda + (X[i] - ox) ** 2
  bitir
```

```
  yaz(Biçimle('Kapalı form:   fiyat ≈ {:.2} × m² + {:.2}', pay / payda, oy - pay /
      ↪ payda * ox), 1)
  yaz(Biçimle('120 m² için tahmin: {:.0} bin TL', a * Ölçekle(120) + b), 1)
bitir.
```

Programın çıktısı şöyledir:

```
Adım     1  OKH =   8265357.14
Adım  1000  OKH =       424.67
Adım  2000  OKH =       424.67
Adım  3000  OKH =       424.67
Gradyan inişi: fiyat ≈ 26.83 × m² + 129.40
Kapalı form:   fiyat ≈ 26.83 × m² + 129.40
120 m² için tahmin: 3349 bin TL
```

Ne öğrendik. Gradyan inişi, formülün doğrudan verdiği çözüme 1000 adımda ulaştı. OKH sıfıra inmez, çünkü gerçek fiyatlar tam bir doğru üzerinde durmaz. Sık yapılan hata şudur: girdileri ölçeklemeden (100'e bölmeden) aynı öğrenme oranını kullanmak; adımlar çok büyük olur ve hata patlar.

Alıştırma. `Ölçekle` fonksiyonunu `sonuç := m2` yapıp programı çalıştırınız. OKH ne oluyor? Hangi `oran` değeriyle yeniden yakınsıyor?

2. Lojistik regresyon

Fikir. Doğrusal bir puanı sigmoid fonksiyonundan geçirip 0 ile 1 arasında bir olasılığa çevirir iriz. Olasılık 0,5'i geçerse sınıf 1 deriz. Kayıp olarak çapraz entropi kullanılır: emin olduğu bir tahminde yanılan modeli çok ağır cezalandırır.

$$P(y = 1 \mid x) = \sigma(wx + b) = \frac{1}{1 + e^{-(wx+b)}}$$

```
program LojistikRegresyon
{ Çalışma saatinden sınavı geçme olasılığını tahmin eder: P(geçer) = σ(w·x + b)
  σ(z) = 1 / (1 + e^(-z)) her sayıyı 0 ile 1 arasına sıkıştırır. Kayıp: çapraz
      ↪ entropi. }
kullan Matematik

değişken
  saat: ondalık dizisi
  geçti: tamsayı dizisi
  w, b, p, gw, gb, kayıp: ondalık
  i, adım, n, isabet: tamsayı

başla
  saat := [0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0]
```

```
  geçti := [0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1]
  n := Uzunluk(saat)
  w := 0; b := 0

  1'den 2000'e kadar her adım için
  başla
    gw := 0; gb := 0; kayıp := 0
    0'dan n - 1'e kadar her i için
    başla
      p := Sigmoid(w * saat[i] + b)
      kayıp := kayıp - (geçti[i] * Ln(p) + (1 - geçti[i]) * Ln(1 - p)) / n
      gw := gw + (p - geçti[i]) * saat[i] / n     { çapraz entropinin türevi
          ↪ şaşırtıcı biçimde basittir }
      gb := gb + (p - geçti[i]) / n
    bitir
    w := w - 0.5 * gw
    b := b - 0.5 * gb
    eğer adım mod 500 = 0 ise yaz(Biçimle('Adım {:4}  kayıp = {:.4}', adım, kayıp),
        ↪ 1)
  bitir

  yaz(Biçimle('Model: P(geçer) = σ({:.3} × saat {} {:.3})', w, eğer b < 0 ise '-'
      ↪ değilse '+', Mutlak(b)), 1)
  yaz(Biçimle('Karar sınırı (P = 0,5): {:.2} saat', -b / w), 1)
  isabet := 0
  0'dan n - 1'e kadar her i için
  başla
    p := Sigmoid(w * saat[i] + b)
    eğer (p > = 0.5) = (geçti[i] = 1) ise isabet := isabet + 1
  bitir
  yaz('Eğitim doğruluğu: ' + isabet + '/' + n, 1)
  yaz(Biçimle('2 saat çalışan için olasılık: {:.2}', Sigmoid(w * 2 + b)), 1)
  yaz(Biçimle('5 saat çalışan için olasılık: {:.2}', Sigmoid(w * 5 + b)), 1)
bitir.
```

Programın çıktısı şöyledir:

```
Adım  500  kayıp = 0.3650
Adım 1000  kayıp = 0.3649
Adım 1500  kayıp = 0.3649
Adım 2000  kayıp = 0.3649
Model: P(geçer) = σ(1.424 × saat - 4.630)
Karar sınırı (P = 0,5): 3.25 saat
Eğitim doğruluğu: 10/12
2 saat çalışan için olasılık: 0.14
5 saat çalışan için olasılık: 0.92
```

Ne öğrendik. Model 3,25 saati bir eşik olarak öğrendi. 12 öğrenciden 2'si (2,5 saatte geçen, 4 saatte kalan) bu eşiğe uymuyor; hiçbir düz sınır onları doğru ayıramaz. Adı "regresyon" olsa da bu bir **sınıflandırma** yöntemidir.

Alıştırma. Veriye ikinci bir öznitelik ekleyiniz (örneğin uyku saati) ve modeli `w1·saat + w2·uyku + b` biçimine genişletiniz.

3. K-en yakın komşu (k-NN)

Fikir. "Bana arkadaşını söyle, sana kim olduğunu söyleyeyim." Yeni bir örneğin sınıfını, ona en yakın k eğitim örneğinin oy çokluğu belirler. Model eğitilmez; tüm iş tahmin anında yapılır.

$$d(p, q) = \sqrt{\sum_{j} (p_j - q_j)^2}$$

```
program KEnYakınKomşu
{ Meyveyi ağırlığına (g) ve kabuk pürüzlülüğüne (0-10) bakarak sınıflandırır.
  Yeni bir meyvenin sınıfı, ona en yakın k eğitim örneğinin oy çokluğuyla belirlenir. }
kullan Matematik

değişken
  veri: ondalık dizisi dizisi
  etiket: metin dizisi
  yeniler: ondalık dizisi dizisi
  i: tamsayı
  s: ondalık dizisi

fonksiyon Uzaklık(p, q: ondalık dizisi): ondalık
başla
  { Öklid uzaklığı; ağırlık 100 g ölçeğinde, pürüzlülük 1 ölçeğinde olsun diye
    ağırlığı 20'ye bölüyoruz }
  sonuç := Kök(((p[0] - q[0]) / 20) ** 2 + (p[1] - q[1]) ** 2)
bitir

fonksiyon Sınıflandır(x: ondalık dizisi; k: tamsayı): metin
değişken
  sıra: tamsayı dizisi
  oyElma, oyPortakal, j: tamsayı
başla
  sıra := Sırala(Aralık(Uzunluk(veri)), fonksiyon(n: tamsayı) => Uzaklık(veri[n], x))
  oyElma := 0; oyPortakal := 0
  0'dan k - 1'e kadar her j için
    eğer etiket[sıra[j]] = 'elma' ise oyElma := oyElma + 1 değilse oyPortakal :=
      oyPortakal + 1
  sonuç := eğer oyElma > oyPortakal ise 'elma' değilse 'portakal'
```

```
bitir

başla
  { [186, 7] alışılmadık, portakala benzeyen bir elma: k =1 bu tek örneğe kanar }
  veri := [[150, 2], [170, 1], [140, 3], [130, 2], [180, 2], [186, 7],
           [160, 7], [200, 8], [190, 6], [210, 7], [175, 9]]
  etiket := ['elma', 'elma', 'elma', 'elma', 'elma', 'elma',
             'portakal', 'portakal', 'portakal', 'portakal', 'portakal']
  yeniler := [[155, 3], [185, 7], [165, 5], [200, 3]]
  0'dan Uzunluk(yeniler) - 1'e kadar her i için
  başla
    s := yeniler[i]
    yaz(Biçimle('{} g, pürüzlülük {} → k =1: {:9}  k =3: {:9}  k =5: {}',
      s[0], s[1], Sınıflandır(s, 1), Sınıflandır(s, 3), Sınıflandır(s, 5)), 1)
  bitir
bitir.
```

Programın çıktısı şöyledir:

```
155 g, pürüzlülük 3 → k =1:      elma  k =3:      elma  k =5: elma
185 g, pürüzlülük 7 → k =1:      elma  k =3:  portakal  k =5: portakal
165 g, pürüzlülük 5 → k =1:  portakal  k =3:  portakal  k =5: portakal
200 g, pürüzlülük 3 → k =1:      elma  k =3:      elma  k =5: elma
```

Ne öğrendik. k =1 iken model, 185 g'lık meyveyi tek bir tuhaf örnek yüzünden elma sandı; k =3'te çoğunluk bu gürültüyü bastırdı. Özniteliklerin ölçeği çok önemlidir: ağırlığı 20'ye bölmeseydik uzaklığı neredeyse yalnızca gramlar belirlerdi.

Alıştırma. `Uzaklık` içindeki `/ 20` bölümünü kaldırınız. Hangi tahminler değişiyor, neden?

4. Naif Bayes

Fikir. Her sınıfta hangi sözcüklerin ne sıklıkta geçtiğini sayarız. Yeni bir metin geldiğinde Bayes kuralıyla "bu sözcükleri hangi sınıf üretmiş olabilir?" diye sorarız. "Naif" denmesinin nedeni, sözcüklerin birbirinden bağımsız olduğunu varsaymasıdır; bu varsayım yanlış olsa da yöntem şaşırtıcı derecede iyi çalışır.

$$P(c \mid w_1 \dots w_n) \propto P(c) \prod_i P(w_i \mid c)$$

```
program NaifBayes
{ E-postaları 'istenmeyen' ya da 'normal' diye sınıflandırır.
  Bayes kuralı: P(sınıf | sözcükler) ∝ P(sınıf) × Π P(sözcük | sınıf)
```

```
  "Naif" varsayım: sözcükler, sınıf bilindiğinde birbirinden bağımsızdır. }
kullan Matematik

değişken
  metinler, sınıflar, sınıfAdları, denemeler: metin dizisi
  sayım: sözlük           { sınıf → (sözcük → kaç kez) }
  belgeSayısı, toplamSözcük: sözlük
  sözlükçe: sözlük        { görülen tüm farklı sözcükler }
  i: tamsayı
  s, c: metin

fonksiyon Sözcükler(m: metin): metin dizisi
başla
  sonuç := KüçükHarf(m).Ayır
bitir

fonksiyon Puan(m: metin; sınıfAdı: metin): ondalık
değişken
  w: metin
  p: ondalık
  sayaç: sözlük
başla
  { Olasılıkları çarpmak yerine logaritmalarını topluyoruz: küçük sayılar sıfıra
    ↪ yuvarlanmasın }
  p := Ln(belgeSayısı[sınıfAdı] / Uzunluk(metinler))
  sayaç := sayım[sınıfAdı]
  Sözcükler(m) içindeki her w için
    { Laplace düzeltmesi (+1): hiç görülmemiş sözcük olasılığı sıfırlamasın }
    p := p + Ln((sayaç.Al(w, 0) + 1) / (toplamSözcük[sınıfAdı] + Uzunluk(sözlükçe)))
  sonuç := p
bitir

başla
  metinler := ['bedava para kazan hemen tıkla', 'kazan büyük ödül bedava',
               'hemen tıkla indirim kampanya', 'toplantı yarın saat onda',
               'rapor ekte toplantı notları', 'yarın öğle yemeği var mı',
               'proje raporu yarın teslim']
  sınıflar := ['istenmeyen', 'istenmeyen', 'istenmeyen', 'normal', 'normal',
      ↪ 'normal', 'normal']
  sınıfAdları := ['istenmeyen', 'normal']
  sayım := [:]; belgeSayısı := [:]; toplamSözcük := [:]; sözlükçe := [:]
  sınıfAdları içindeki her c için
  başla
    sayım[c] := [:]; belgeSayısı[c] := 0; toplamSözcük[c] := 0
  bitir

  { Eğitim: her sınıfta her sözcüğün kaç kez geçtiğini say }
  0'dan Uzunluk(metinler) - 1'e kadar her i için
  başla
```

```
    c := sınıflar[i]
    belgeSayısı[c] := belgeSayısı[c] + 1
    Sözcükler(metinler[i]) içindeki her s için
    başla
      sayım[c][s] := sayım[c].Al(s, 0) + 1
      toplamSözcük[c] := toplamSözcük[c] + 1
      sözlükçe[s] := 1
    bitir
  bitir
  yaz('Sözlükçe: ' + Uzunluk(sözlükçe) + ' farklı sözcük', 1)

  denemeler := ['bedava ödül kazan', 'yarın toplantı var', 'hemen rapor teslim',
      ↪ 'indirim kampanya yarın']
  denemeler içindeki her s için
    yaz(Biçimle('{:24} → {:10} (log puanlar: istenmeyen {:.2}, normal {:.2})', '"' +
        ↪ s + '"',
      eğer Puan(s, 'istenmeyen') > Puan(s, 'normal') ise 'istenmeyen' değilse
          ↪ 'normal',
      Puan(s, 'istenmeyen'), Puan(s, 'normal')), 1)
bitir.
```

Programın çıktısı şöyledir:

```
Sözlükçe: 23 farklı sözcük
     "bedava ödül kazan" → istenmeyen (log puanlar: istenmeyen -8.71, normal -11.63)
    "yarın toplantı var" →     normal (log puanlar: istenmeyen -11.60, normal -8.45)
    "hemen rapor teslim" →     normal (log puanlar: istenmeyen -10.50, normal -10.24)
"indirim kampanya yarın" → istenmeyen (log puanlar: istenmeyen -10.21, normal -10.24)
```

Ne öğrendik. Son örnekte "indirim kampanya" istenmeyene, "yarın" normale çekti; puanlar çok yakın çıktı. Logaritma kullanmak şarttır: yüzlerce küçük olasılığı çarpmak bilgisayarda sıfıra yuvarlanır. `+1` eklenmezse eğitimde görülmemiş tek bir sözcük tüm olasılığı sıfır yapardı.

Alıştırma. Eğitim verisine iki istenmeyen ileti daha ekleyiniz ve son örneğin kararının nasıl değiştiğine bakınız. Hazır sürüm için `kullan Dil` ile gelen `NaifBayes` birimine göz atınız.

5. Karar ağaçları

Fikir. Veriyi evet/hayır sorularıyla tekrar tekrar ikiye (ya da daha fazla parçaya) böleriz. Her adımda, oluşan grupları en "saf" yapan soru seçilir. Saflığı Gini ölçüsü verir: tek sınıflı bir grupta 0, yarı yarıya karışık bir grupta 0,5.

$$\text{Gini} = 1 - \sum_{k} p_k^2$$

```
program KararAğacı
{ Hava durumuna bakarak "tenis oynanır mı?" sorusunu yanıtlayan bir karar ağacı
    ↪ kurar.
  Her düğümde, veriyi en "saf" gruplara ayıran öznitelik seçilir (Gini safsızlığı en
      ↪ düşük olan).
  Gini = 1 − Σ p², p: gruptaki her sınıfın oranı. Tek sınıflı grupta Gini = 0. }

değişken
  satırlar: metin dizisi dizisi      { son sütun: sınıf (evet / hayır) }
  öznitelikler: metin dizisi
  tümü: tamsayı dizisi
  i: tamsayı

fonksiyon Gini(sıra: tamsayı dizisi): ondalık
değişken
  evet, j: tamsayı
  p: ondalık
başla
  evet := 0
  sıra içindeki her j için eğer satırlar[j][4] = 'evet' ise evet := evet + 1
  p := evet / Uzunluk(sıra)
  sonuç := 1 - p * p - (1 - p) * (1 - p)
bitir

fonksiyon Değerler(sıra: tamsayı dizisi; sütun: tamsayı): metin dizisi
değişken j: tamsayı
başla
  sonuç := []
  sıra içindeki her j için
    eğer değil (satırlar[j][sütun] içinde sonuç) ise sonuç.Ekle(satırlar[j][sütun])
bitir

fonksiyon Süzgeç(sıra: tamsayı dizisi; sütun: tamsayı; d: metin): tamsayı dizisi
başla
  sonuç := Süz(sıra, fonksiyon(j: tamsayı) => satırlar[j][sütun] = d)
bitir

{ Ağırlıklı Gini: bölünmeden sonra çocukların Gini değerlerinin, büyüklükleriyle
    ↪ ağırlıklı ortalaması }
fonksiyon BölünmeGinisi(sıra: tamsayı dizisi; sütun: tamsayı): ondalık
değişken d: metin
başla
  sonuç := 0
  Değerler(sıra, sütun) içindeki her d için
    sonuç := sonuç + Uzunluk(Süzgeç(sıra, sütun, d)) / Uzunluk(sıra) *
        ↪ Gini(Süzgeç(sıra, sütun, d))
bitir

prosedür Kur(sıra: tamsayı dizisi; kullanılmış: tamsayı dizisi; girinti: metin)
```

```
değişken
  en, s: tamsayı
  enIyi: ondalık
  d: metin
başla
  eğer Gini(sıra) = 0 veya Uzunluk(kullanılmış) = 4 ise
  başla
    yaz(girinti + '→ ' + satırlar[sıra[0]][4] + ' (' + Uzunluk(sıra) + ' örnek)', 1)
    dön
  bitir
  en := -1; enIyi := 2
  0'dan 3'e kadar her s için
    eğer değil (s içinde kullanılmış) ve BölünmeGinisi(sıra, s) < enIyi ise
    başla
      enIyi := BölünmeGinisi(sıra, s); en := s
    bitir
  Değerler(sıra, en) içindeki her d için
  başla
    yaz(Biçimle('{}{} = {}?  (Gini {:.3} → {:.3})', girinti, öznitelikler[en], d,
        ↪  Gini(sıra), enIyi), 1)
    Kur(Süzgeç(sıra, en, d), kullanılmış + [en], girinti + '    ')
  bitir
bitir

başla
  öznitelikler := ['Hava', 'Sıcaklık', 'Nem', 'Rüzgâr']
  satırlar := [
    ['güneşli', 'sıcak', 'yüksek', 'yok', 'hayır'], ['güneşli', 'sıcak', 'yüksek',
        ↪  'var', 'hayır'],
    ['bulutlu', 'sıcak', 'yüksek', 'yok', 'evet'],  ['yağmurlu', 'ılık', 'yüksek',
        ↪  'yok', 'evet'],
    ['yağmurlu', 'serin', 'normal', 'yok', 'evet'], ['yağmurlu', 'serin', 'normal',
        ↪  'var', 'hayır'],
    ['bulutlu', 'serin', 'normal', 'var', 'evet'],  ['güneşli', 'ılık', 'yüksek',
        ↪  'yok', 'hayır'],
    ['güneşli', 'serin', 'normal', 'yok', 'evet'],  ['yağmurlu', 'ılık', 'normal',
        ↪  'yok', 'evet'],
    ['güneşli', 'ılık', 'normal', 'var', 'evet'],   ['bulutlu', 'ılık', 'yüksek',
        ↪  'var', 'evet'],
    ['bulutlu', 'sıcak', 'normal', 'yok', 'evet'],  ['yağmurlu', 'ılık', 'yüksek',
        ↪  'var', 'hayır']]
  tümü := Aralık(Uzunluk(satırlar))
  yaz(Biçimle('Kökün Gini değeri: {:.3}', Gini(tümü)), 2)
  0'dan 3'e kadar her i için
    yaz(Biçimle('{:9} ile bölünürse ağırlıklı Gini: {:.3}', öznitelikler[i],
        ↪  BölünmeGinisi(tümü, i)), 1)
  yaz('', 1)
  Kur(tümü, [], '')
bitir.
```

Programın çıktısı şöyledir:

```
Kökün Gini değeri: 0.459

     Hava ile bölünürse ağırlıklı Gini: 0.343
 Sıcaklık ile bölünürse ağırlıklı Gini: 0.440
      Nem ile bölünürse ağırlıklı Gini: 0.367
   Rüzgâr ile bölünürse ağırlıklı Gini: 0.429

Hava = güneşli?  (Gini 0.459 → 0.343)
    Nem = yüksek?  (Gini 0.480 → 0.000)
        → hayır (3 örnek)
    Nem = normal?  (Gini 0.480 → 0.000)
        → evet (2 örnek)
Hava = bulutlu?  (Gini 0.459 → 0.343)
    → evet (4 örnek)
Hava = yağmurlu?  (Gini 0.459 → 0.343)
    Rüzgâr = yok?  (Gini 0.480 → 0.000)
        → evet (3 örnek)
    Rüzgâr = var?  (Gini 0.480 → 0.000)
        → hayır (2 örnek)
```

Ne öğrendik. Ağaç, 14 günü yalnızca üç öznitelikle hatasız ayırdı ve Sıcaklığı hiç kullanmadı. Sonuç bir insanın okuyabileceği kurallardır: "güneşliyse neme bak, yağmurluysa rüzgâra bak". Tehlikesi şudur: derin ağaçlar eğitim verisini ezberler (aşırı uyum). Bir sonraki bölümdeki orman ve artırma yöntemleri bu sorunu çözer.

Alıştırma. `Kur` prosedürüne bir derinlik sınırı ekleyiniz: derinlik 1'de durunca yapraktaki çoğunluk sınıfını yazdırınız. Bu tek katlı ağaca "karar kütüğü" denir ve AdaBoost'ta tekrar karşımıza çıkacak.

BÖLÜM

2

Topluluk yöntemleri ve destek vektör makinesi

Tek bir ağaç ya ezberler ya da çok basit kalır. Topluluk yöntemleri birçok küçük modeli birleştirir: **torbalama** (rastgele orman) birbirinden bağımsız ağaçların oyunu alarak dalgalanmayı azaltır; **artırma** (gradyan artırma, AdaBoost, XGBoost) modelleri sırayla ekler, her biri öncekilerin hatasını düzeltir. Destek vektör makinesi ise başka bir yoldan gider: sınıflar arasına olabildiğince geniş bir boşluk bırakan sınırı arar.

6. Rastgele orman

Fikir. Aynı veriden birçok farklı ağaç kurarız. Farklılığı iki rastgelelik sağlar: her ağaç verinin geri koyarak çekilmiş bir kopyasını (bootstrap) görür ve her düğümde özniteliklerin yalnızca bir kısmı denenir. Tek tek ağaçlar zayıftır ama hataları farklı yerlerdedir; oy çokluğu hataları birbirine söndürür.

$$\hat{y}(x) = \text{çoğunluk}\{\, h_1(x), h_2(x), \ldots, h_T(x) \,\}$$

```
program RastgeleOrman
{ Tek bir derin karar ağacı ile 25 ağaçlık rastgele ormanı karşılaştırır.
  Her ağaç, verinin geri koyarak çekilmiş bir örneğiyle (bootstrap) ve her düğümde
  rastgele seçilen özniteliklerle kurulur; tahmin, ağaçların oy çokluğudur. }
kullan Matematik

tip
  Düğüm = sınıf
    yaprak: mantıksal
    sınıfı, öznitelik: tamsayı
    eşik: ondalık
    sol, sağ: Düğüm
  bitir

değişken
  X, Xt: ondalık dizisi dizisi
  Y, Yt: tamsayı dizisi
  orman: Düğüm dizisi
  tek: Düğüm
  i, t: tamsayı

{ Veri: bir çemberin içi (1) ve dışı (0); etiketlerin %10'u bilerek yanlış (gürültü)
    ↪ }
prosedür Üret(n: tamsayı; X: ondalık dizisi dizisi; Y: tamsayı dizisi)
değişken a, b: ondalık; k, j: tamsayı
başla
  1'den n'e kadar her j için
  başla
    a := Rastgele * 2 - 1; b := Rastgele * 2 - 1
    k := eğer a * a + b * b < 0.5 ise 1 değilse 0
    eğer Rastgele < 0.1 ise k := 1 - k
    X.Ekle([a, b]); Y.Ekle(k)
  bitir
bitir

fonksiyon Gini(sıra: tamsayı dizisi): ondalık
değişken p: ondalık; j: tamsayı
başla
  eğer Uzunluk(sıra) = 0 ise dön 0
  p := 0
  sıra içindeki her j için p := p + Y[j]
  p := p / Uzunluk(sıra)
  sonuç := 2 * p * (1 - p)
bitir

fonksiyon Çoğunluk(sıra: tamsayı dizisi): tamsayı
değişken s, j: tamsayı
başla
  s := 0
```

```
  sıra içindeki her j için s := s + Y[j]
  sonuç := eğer 2 * s >= Uzunluk(sıra) ise 1 değilse 0
bitir

{ öznitelikSayısı = 1 ise her düğümde tek bir rastgele öznitelik denenir (ormanın
    ↪ "rastgele" kısmı) }
fonksiyon Ağaç(sıra: tamsayı dizisi; derinlik, öznitelikSayısı: tamsayı): Düğüm
değişken
  d: Düğüm
  f, k, j, enF: tamsayı
  e, puan, enPuan, enE: ondalık
  sol, sağ, adaylar: tamsayı dizisi
başla
  d := Düğüm.Oluştur
  eğer derinlik = 0 veya Gini(sıra) = 0 ise
  başla
    d.yaprak := doğru; d.sınıfı := Çoğunluk(sıra); dön d
  bitir
  adaylar := eğer öznitelikSayısı = 2 ise [0, 1] değilse [Rastgele(2)]
  enPuan := 9; enF := 0; enE := 0
  adaylar içindeki her f için
    sıra içindeki her k için
    başla
      e := X[k][f]
      sol := Süz(sıra, fonksiyon(j: tamsayı) => X[j][f] < e)
      sağ := Süz(sıra, fonksiyon(j: tamsayı) => X[j][f] >= e)
      puan := (Uzunluk(sol) * Gini(sol) + Uzunluk(sağ) * Gini(sağ)) / Uzunluk(sıra)
      eğer puan < enPuan ise başla enPuan := puan; enF := f; enE := e bitir
    bitir
  d.öznitelik := enF; d.eşik := enE
  sol := Süz(sıra, fonksiyon(j: tamsayı) => X[j][enF] < enE)
  sağ := Süz(sıra, fonksiyon(j: tamsayı) => X[j][enF] >= enE)
  eğer Uzunluk(sol) = 0 veya Uzunluk(sağ) = 0 ise
  başla
    d.yaprak := doğru; d.sınıfı := Çoğunluk(sıra); dön d
  bitir
  d.sol := Ağaç(sol, derinlik - 1, öznitelikSayısı)
  d.sağ := Ağaç(sağ, derinlik - 1, öznitelikSayısı)
  sonuç := d
bitir

fonksiyon Tahmin(d: Düğüm; x: ondalık dizisi): tamsayı
başla
  eğer d.yaprak ise dön d.sınıfı
  eğer x[d.öznitelik] < d.eşik ise dön Tahmin(d.sol, x)
  sonuç := Tahmin(d.sağ, x)
bitir

fonksiyon OrmanTahmini(x: ondalık dizisi): tamsayı
```

```
değişken oy: tamsayı; a: Düğüm
başla
  oy := 0
  orman içindeki her a için oy := oy + Tahmin(a, x)
  sonuç := eğer 2 * oy > Uzunluk(orman) ise 1 değilse 0
bitir

fonksiyon Doğruluk(tekMi: mantıksal): ondalık
değişken j, isabet, p: tamsayı
başla
  isabet := 0
  0'dan Uzunluk(Xt) - 1'e kadar her j için
  başla
    p := eğer tekMi ise Tahmin(tek, Xt[j]) değilse OrmanTahmini(Xt[j])
    eğer p = Yt[j] ise isabet := isabet + 1
  bitir
  sonuç := isabet / Uzunluk(Xt)
bitir

başla
  Tohum(3)
  X := []; Y := []; Xt := []; Yt := []
  Üret(150, X, Y)
  Üret(300, Xt, Yt)

  tek := Ağaç(Aralık(Uzunluk(X)), 8, 2)
  yaz(Biçimle('Tek derin ağaç (derinlik 8)  → test doğruluğu: {:.3}',
      ↪  Doğruluk(doğru)), 1)

  orman := []
  1'den 25'e kadar her t için
  başla
    { bootstrap: n örneği geri koyarak çek }
    orman.Ekle(Ağaç(Eşle(Aralık(Uzunluk(X)), fonksiyon(j: tamsayı) =>
        ↪  Rastgele(Uzunluk(X))), 8, 1))
    eğer t = 1 veya t = 5 veya t = 25 ise
      yaz(Biçimle('Rastgele orman, {:2} ağaç       → test doğruluğu: {:.3}', t,
          ↪  Doğruluk(yanlış)), 1)
  bitir
bitir.
```

Programın çıktısı şöyledir:

```
Tek derin ağaç (derinlik 8)  → test doğruluğu: 0.803
Rastgele orman,  1 ağaç       → test doğruluğu: 0.720
Rastgele orman,  5 ağaç       → test doğruluğu: 0.783
Rastgele orman, 25 ağaç       → test doğruluğu: 0.840
```

Ne öğrendik. Ormanın tek ağacı (%72), tüm veriyle kurulan derin ağaçtan (%80) kötüdür; 25 ağaç bir araya gelince %84'e çıkıyor. Etiketlerin %10'u gürültü olduğu için ulaşılabilecek en iyi doğruluk %90 civarıdır. Doğruluğu her zaman eğitimde görülmemiş bir test kümesinde ölçmek gerekir: derin ağaç eğitim verisinde %100'e yakın görünür.

Alıştırma. Ağaç sayısını 100'e çıkarınız ve doğruluğun nerede durduğuna bakınız. Bootstrap'ı kaldırıp her ağacı tüm veriyle kurarsanız ne olur?

7. Gradyan artırma

Fikir. Basit bir tahminle başlarız (ortalama). Sonra sırayla küçük ağaçlar ekleriz; her yeni ağaç o ana kadarki toplamın artıklarını (gerçek − tahmin) öğrenir. Her katkı bir öğrenme oranıyla (η) küçültülür, böylece hiçbir ağaç tek başına fazla söz sahibi olmaz.

$$F_m(x) = F_{m-1}(x) + \eta\, h_m(x)$$

```
program GradyanArtırma
{ Gürültülü bir sinüs eğrisini, art arda eklenen küçük "karar kütükleri"yle öğrenir.
  Her yeni kütük, o ana kadarki toplam modelin hatalarını (artıkları) tahmin etmeye
      ↪ çalışır:
  F_m(x) = F_(m-1)(x) + η · h_m(x)    η: öğrenme oranı }
kullan Matematik

tip
  Kütük = kayıt eşik, sol, sağ: ondalık; bitir      { x < eşik ise sol, değilse sağ
      ↪ değerini verir }

değişken
  X, Y, F, artık: ondalık dizisi
  kütükler: Kütük dizisi
  i, m, n: tamsayı
  eta, okh: ondalık
  k: Kütük

fonksiyon Ortalama(L: ondalık dizisi): ondalık
başla
  sonuç := eğer Uzunluk(L) = 0 ise 0.0 değilse Toplam(L) / Uzunluk(L)
bitir

{ Artıklara en iyi uyan tek bölmeli ağacı bulur: her olası eşiği dener, kare hatası
    ↪ en küçük olanı seçer }
fonksiyon KütükUydur(r: ondalık dizisi): Kütük
değişken
  j, q: tamsayı
  sol, sağ: ondalık dizisi
```

```
  hata, enHata: ondalık
başla
  enHata := 1e9
  1'den n - 1'e kadar her j için
  başla
    sol := []; sağ := []
    0'dan n - 1'e kadar her q için eğer X[q] < X[j] ise sol.Ekle(r[q]) değilse
        ↪  sağ.Ekle(r[q])
    hata := 0
    sol içindeki her e için hata := hata + (e - Ortalama(sol)) ** 2
    sağ içindeki her e için hata := hata + (e - Ortalama(sağ)) ** 2
    eğer hata < enHata ise
    başla
      enHata := hata
      sonuç := Kütük.Oluştur(X[j], Ortalama(sol), Ortalama(sağ))
    bitir
  bitir
bitir

başla
  Tohum(1)
  n := 40
  X := []; Y := []
  0'dan n - 1'e kadar her i için
  başla
    X.Ekle(i * 6.28 / n)
    Y.Ekle(Sin(i * 6.28 / n) + Normal(0, 0.15))
  bitir
  eta := 0.3
  F := Eşle(Y, fonksiyon(v: ondalık) => Ortalama(Y))   { başlangıç: herkese
      ↪  ortalamayı tahmin et }
  kütükler := []
  1'den 60'a kadar her m için
  başla
    artık := []
    0'dan n - 1'e kadar her i için artık.Ekle(Y[i] - F[i])
    k := KütükUydur(artık)
    kütükler.Ekle(k)
    okh := 0
    0'dan n - 1'e kadar her i için
    başla
      F[i] := F[i] + eta * (eğer X[i] < k.eşik ise k.sol değilse k.sağ)
      okh := okh + (Y[i] - F[i]) ** 2 / n
    bitir
    eğer m < = 3 veya m mod 20 = 0 ise
      yaz(Biçimle('{:2}. kütük: x < {:.2} ise {:6.3}, değilse {:6.3}   →  OKH =
          ↪  {:.4}', m, k.eşik, k.sol, k.sağ, okh), 1)
  bitir
```

```
  yaz(Biçimle('x = 1.57 için tahmin {:.3} (gerçek sin = 1.000)', F[10]), 1)
bitir.
```

Programın çıktısı şöyledir:

```
 1. kütük: x < 3.30 ise  0.599, değilse -0.662   →  OKH = 0.2739
 2. kütük: x < 3.61 ise  0.386, değilse -0.522   →  OKH = 0.1713
 3. kütük: x < 2.98 ise  0.351, değilse -0.318   →  OKH = 0.1144
20. kütük: x < 5.97 ise -0.011, değilse  0.200   →  OKH = 0.0229
40. kütük: x < 4.24 ise  0.019, değilse -0.039   →  OKH = 0.0096
60. kütük: x < 2.04 ise  0.022, değilse -0.010   →  OKH = 0.0055
x = 1.57 için tahmin 0.888 (gerçek sin = 1.000)
```

Ne öğrendik. İlk kütük eğriyi kabaca ikiye böldü; sonrakiler giderek küçülüp ince ayar yaptı. 20. kütükte OKH, verideki gürültünün varyansına ($0{,}15^2 \approx 0{,}0225$) ulaştı. Ondan sonraki düşüş, model artık gürültüyü ezberlediği için yanıltıcıdır. Gerçek projelerde kaç ağaç ekleneceğine ayrı bir doğrulama kümesine bakarak karar verilir.

Alıştırma. `eta` değerini 1,0 ve 0,05 yapınız. Aynı OKH'ye kaç kütükte ulaşılıyor?

8. Destek vektör makinesi (SVM)

Fikir. İki sınıfı ayıran pek çok doğru vardır; SVM, iki taraftaki en yakın noktalara en uzak olanı seçer. Bu en yakın noktalara **destek vektörleri** denir: sınır yalnızca onlara bağlıdır, diğer noktaları silseniz sonuç değişmez. Doğrusal olmayan sınırlar için "çekirdek hilesi" kullanılır.

$$\min_{w,b} \; \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i} \max\bigl(0,\; 1 - y_i(w^\top x_i + b)\bigr)$$

```
program DestekVektörMakinesi
{ İki sınıfı ayıran ve iki sınıfa da en uzak duran doğruyu (en geniş marjı) bulur.
  Kayıp (menteşe kaybı): λ/2·‖w‖² + ortalama(max(0, 1 − y·(w·x + b)))
  Doğru tarafta ve marjın dışında kalan noktalar kayba hiç katkı yapmaz. }
kullan Matematik

değişken
  X: ondalık dizisi dizisi
  Y: tamsayı dizisi      { +1 ya da −1 }
  w0, w1, b, lam, eta, m, g0, g1, gb: ondalık
  i, t, n: tamsayı

fonksiyon Puan(x: ondalık dizisi): ondalık
başla
```

```
  sonuç := w0 * x[0] + w1 * x[1] + b
bitir

başla
  X := [[1, 2], [2, 3], [2, 1], [3, 2], [1.5, 1.5],       { −1 sınıfı }
        [5, 5], [6, 4], [5, 6], [7, 5], [6, 6.5], [4.5, 4]] { +1 sınıfı }
  Y := [-1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1]
  n := Uzunluk(X)
  w0 := 0; w1 := 0; b := 0; lam := 0.01

  1'den 5000'e kadar her t için
  başla
    eta := 1 / (lam * t + 100)        { zamanla küçülen adım }
    g0 := lam * w0; g1 := lam * w1; gb := 0
    0'dan n - 1'e kadar her i için
      eğer Y[i] * Puan(X[i]) < 1 ise    { marjın içinde ya da yanlış tarafta: düzelt
          ↪ }
      başla
        g0 := g0 - Y[i] * X[i][0] / n
        g1 := g1 - Y[i] * X[i][1] / n
        gb := gb - Y[i] / n
      bitir
    w0 := w0 - eta * g0; w1 := w1 - eta * g1; b := b - eta * gb
  bitir

  yaz(Biçimle('Karar sınırı: {:.3}·x + {:.3}·y {} {:.3} = 0', w0, w1, eğer b < 0 ise
      ↪  '-' değilse '+', Mutlak(b)), 1)
  yaz(Biçimle('Marj genişliği: 2/‖w‖ = {:.3}', 2 / Kök(w0 * w0 + w1 * w1)), 1)
  yaz('Noktalar ve y·(w·x + b) değerleri:', 1)
  0'dan n - 1'e kadar her i için
  başla
    m := Y[i] * Puan(X[i])
    yaz(Biçimle('  ({:3}, {:3}) sınıf {:2}  → {:6.3}{}', X[i][0], X[i][1], Y[i], m,
      eğer m < 1.05 ise '   ← destek vektörü' değilse ''), 1)
  bitir
  yaz(Biçimle('Yeni nokta (3.5, 3.5) → sınıf {}', eğer Puan([3.5, 3.5]) > = 0 ise
      ↪  '+1' değilse '-1'), 1)
bitir.
```

Programın çıktısı şöyledir:

```
Karar sınırı: 0.581·x + 0.536·y - 3.780 = 0
Marj genişliği: 2/‖w‖ = 2.529
Noktalar ve y·(w·x + b) değerleri:
  (  1,   2) sınıf -1  →  2.126
  (  2,   3) sınıf -1  →  1.008   ← destek vektörü
  (  2,   1) sınıf -1  →  2.081
  (  3,   2) sınıf -1  →  0.963   ← destek vektörü
  (1.5, 1.5) sınıf -1  →  2.103
```

```
  (  5,   5) sınıf  1  →  1.808
  (  6,   4) sınıf  1  →  1.853
  (  5,   6) sınıf  1  →  2.345
  (  7,   5) sınıf  1  →  2.971
  (  6, 6.5) sınıf  1  →  3.194
  (4.5,   4) sınıf  1  →  0.981   ← destek vektörü
Yeni nokta (3.5, 3.5) → sınıf +1
```

Ne öğrendik. 11 noktadan yalnızca 3'ü marjın kenarındadır (değeri 1'e yakındır): sınırı onlar belirler. Diğerlerinin değeri 1'den büyüktür, yani kayba katkıları sıfırdır. Özniteliklerin ölçeği burada da önemlidir; farklı birimlerdeki öznitelikler önce standartlaştırılmalıdır.

Alıştırma. `lam` değerini 1 yapınız. Marj genişliği ve destek vektörü sayısı nasıl değişiyor? (Büyük λ, geniş marj için birkaç hataya göz yumar.)

14. AdaBoost

Fikir. Her tur, yalnızca tek bir eşik kullanan çok zayıf bir sınıflandırıcı (karar kütüğü) eğitir. Yanlış sınıflanan örneklerin ağırlığı büyütülür; sonraki kütük zorunlu olarak onlara odaklanır. Sonunda her kütük, isabet oranına göre belirlenen bir ağırlıkla (α) oy verir.

$$\alpha_t = \tfrac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}, \qquad F(x) = \text{işaret}\Big(\sum_t \alpha_t h_t(x)\Big)$$

```
program AdaBoost
{ Zayıf öğrenici (tek eşikli "karar kütükleri") ağırlıklı oylamayla güçlü bir
    ↪ sınıflandırıcıya dönüşür.
  Her turda yanlış sınıflanan örneklerin ağırlığı artar; sonraki kütük onlara
      ↪ odaklanır.
  α_t = ½ · ln((1 − hata) / hata)     F(x) = işaret(Σ α_t · h_t(x)) }
kullan Matematik

tip
  Kütük = kayıt eşik: ondalık; yön: tamsayı; alfa: ondalık; bitir   { h(x) = yön, x <
      ↪ eşik ise; −yön değilse }

değişken
  X, D: ondalık dizisi
  Y: tamsayı dizisi
  kütükler: Kütük dizisi
  i, t, n, isabet: tamsayı
  k, enK: Kütük
  hata, enHata, z: ondalık
```

```
fonksiyon H(k: Kütük; x: ondalık): tamsayı
başla
  sonuç := eğer x < k.eşik ise k.yön değilse -k.yön
bitir

fonksiyon Güçlü(x: ondalık): tamsayı
değişken s: ondalık; j: tamsayı
başla
  s := 0
  0'dan Uzunluk(kütükler) - 1'e kadar her j için s := s + kütükler[j].alfa *
      ↪ H(kütükler[j], x)
  sonuç := eğer s > = 0 ise 1 değilse -1
bitir

başla
  { Tek bir eşik bu veriyi ayıramaz: + + + − − − − + + + }
  X := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  Y := [1, 1, 1, -1, -1, -1, -1, 1, 1, 1]
  n := Uzunluk(X)
  D := Eşle(X, fonksiyon(x: ondalık) => 1.0 / n)       { başlangıçta tüm örnekler
      ↪ eşit ağırlıkta }
  kütükler := []
  1'den 4'e kadar her t için
  başla
    enHata := 2
    { en düşük ağırlıklı hatayı veren eşik ve yönü ara }
    0'dan n'e kadar her i için
      [1, -1] içindeki her yön için
      başla
        k := Kütük.Oluştur(i + 0.5, yön, 0)
        hata := 0
        0'dan n - 1'e kadar her j için eğer H(k, X[j]) <> Y[j] ise hata := hata +
            ↪ D[j]
        eğer hata < enHata ise başla enHata := hata; enK := k bitir
      bitir
    k := enK
    k.alfa := 0.5 * Ln((1 - enHata) / enHata)
    kütükler.Ekle(k)
    { ağırlıkları güncelle: yanlışlar ×e^α, doğrular ×e^−α, sonra toplam 1 olacak
        ↪ şekilde ölçekle }
    0'dan n - 1'e kadar her j için D[j] := D[j] * Üstel(-k.alfa * Y[j] * H(k, X[j]))
    z := Toplam(D)
    0'dan n - 1'e kadar her j için D[j] := D[j] / z
    isabet := 0
    0'dan n - 1'e kadar her j için eğer Güçlü(X[j]) = Y[j] ise isabet := isabet + 1
    yaz(Biçimle('Tur {}: x < {:4} ise {:2}   hata = {:.3}  α = {:.3}   topluluk
        ↪ doğruluğu: {}/{}',
      t, k.eşik, k.yön, enHata, k.alfa, isabet, n), 1)
    yaz('        ağırlıklar: ' + Eşle(D, fonksiyon(d: ondalık) => Yuvarla(d, 2)), 1)
```

```
  bitir
bitir.
```

Programın çıktısı şöyledir:

```
Tur 1: x <  3.5 ise  1   hata = 0.300  α = 0.424   topluluk doğruluğu: 7/10
       ağırlıklar: [0.07, 0.07, 0.07, 0.07, 0.07, 0.07, 0.07, 0.17, 0.17, 0.17]
Tur 2: x <  7.5 ise -1   hata = 0.214  α = 0.650   topluluk doğruluğu: 7/10
       ağırlıklar: [0.17, 0.17, 0.17, 0.05, 0.05, 0.05, 0.05, 0.11, 0.11, 0.11]
Tur 3: x <  0.5 ise -1   hata = 0.182  α = 0.752   topluluk doğruluğu: 10/10
       ağırlıklar: [0.1, 0.1, 0.1, 0.12, 0.12, 0.12, 0.12, 0.06, 0.06, 0.06]
Tur 4: x <  3.5 ise  1   hata = 0.194  α = 0.711   topluluk doğruluğu: 10/10
       ağırlıklar: [0.06, 0.06, 0.06, 0.08, 0.08, 0.08, 0.08, 0.17, 0.17, 0.17]
```

Ne öğrendik. İlk kütük sağdaki üç noktayı kaçırdı; ağırlıkları 0,07'den 0,17'ye çıktı ve ikinci kütük tam onları doğru sınıflandırdı. Tek başına en fazla %70 isabet eden üç kütük, birlikte 10 noktanın hepsini doğru sınıflandırıyor. Gürültülü etiketler AdaBoost'un zayıf noktasıdır: yanlış etiketli bir örneğin ağırlığı her turda büyür.

Alıştırma. `Y` listesindeki bir etiketi bilerek ters çeviriniz ve 10 tur çalıştırınız. O örneğin ağırlığını izleyiniz.

15. XGBoost

Fikir. XGBoost, gradyan artırmanın hızlı ve düzenlileştirilmiş bir sürümüdür. Kaybın yalnızca eğimini (g) değil eğriselliğini de (h, ikinci türev) kullanır; böylece her yaprağın en iyi değeri tek bir formülle bulunur. λ yaprak değerlerini küçülterek aşırı uyumu engeller; γ yeterince kazanç getirmeyen bölmeleri yasaklar.

$$w^{*} = -\frac{G}{H+\lambda}, \qquad \text{kazanç} = \tfrac{1}{2}\Big[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\Big] - \gamma$$

```
program XGBoost
{ XGBoost'un çekirdeği: kaybın birinci (g) ve ikinci (h) türevini kullanan,
    ↪ düzenlileştirilmiş artırma.
  Lojistik kayıpta p = σ(F) için:  g = p − y,  h = p·(1 − p)  (kodda grad ve hess
      ↪ dizileri)
  Yaprak ağırlığı: w = −G / (H + λ)          Bölmenin kazancı:
  kazanç = ½ [G_L²/(H_L+λ) + G_R²/(H_R+λ) − G²/(H+λ)] − γ   (kazanç ≤ 0 ise bölme
      ↪ yapılmaz) }
kullan Matematik

değişken
  X: ondalık dizisi
```

```
  Y: tamsayı dizisi
  i: tamsayı

prosedür Eğit(lam, gama: ondalık)
değişken
  F, grad, hess: ondalık dizisi
  t, j, n, enEşik: tamsayı
  GL, HL, GR, HR, G, H, kazanç, enKazanç, wL, wR, kayıp, p: ondalık
başla
  n := Uzunluk(X)
  F := Eşle(X, fonksiyon(x: ondalık) => 0.0)
  yaz(Biçimle('λ = {}, γ = {}', lam, gama), 1)
  1'den 4'e kadar her t için
  başla
    grad := []; hess := []
    0'dan n - 1'e kadar her j için
    başla
      p := Sigmoid(F[j]); grad.Ekle(p - Y[j]); hess.Ekle(p * (1 - p))
    bitir
    G := Toplam(grad); H := Toplam(hess)
    enKazanç := 0; enEşik := 0
    1'den n - 1'e kadar her i için    { eşik: X[i-1] ile X[i] arası }
    başla
      GL := 0; HL := 0
      0'dan i - 1'e kadar her j için başla GL := GL + grad[j]; HL := HL + hess[j]
          ↪ bitir
      GR := G - GL; HR := H - HL
      kazanç := 0.5 * (GL * GL / (HL + lam) + GR * GR / (HR + lam) - G * G / (H +
          ↪ lam)) - gama
      eğer kazanç > enKazanç ise başla enKazanç := kazanç; enEşik := i bitir
    bitir
    eğer enKazanç < = 0 ise
    başla
      yaz('  Tur ' + t + ': hiçbir bölme γ cezasını karşılamıyor, ağaç büyümüyor', 1)
      kır
    bitir
    GL := 0; HL := 0
    0'dan enEşik - 1'e kadar her j için başla GL := GL + grad[j]; HL := HL + hess[j]
        ↪ bitir
    wL := -GL / (HL + lam); wR := -(G - GL) / (H - HL + lam)
    kayıp := 0
    0'dan n - 1'e kadar her j için
    başla
      F[j] := F[j] + 0.5 * (eğer j < enEşik ise wL değilse wR)       { 0.5: küçültme
          ↪ (shrinkage) }
      p := Sigmoid(F[j])
      kayıp := kayıp - (Y[j] * Ln(p) + (1 - Y[j]) * Ln(1 - p)) / n
    bitir
```

```
    yaz(Biçimle('  Tur {}: x < {:4.1} | kazanç {:.3} | yapraklar {:6.3} / {:6.3} |
        ↪  kayıp {:.4}',
      t, (X[enEşik - 1] + X[enEşik]) / 2, enKazanç, wL, wR, kayıp), 1)
  bitir
bitir

başla
  { Müşterinin yaşı (sıralı) ve ürünü satın alıp almadığı }
  X := [18, 22, 25, 28, 31, 35, 38, 42, 46, 50, 55, 60]
  Y := [0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1]
  Eğit(0, 0)
  Eğit(5, 0)
  Eğit(5, 0.8)
bitir.
```

Programın çıktısı şöyledir:

```
λ = 0.0, γ = 0.0
  Tur 1: x < 26.5 | kazanç 2.722 | yapraklar -2.000 /  1.111 | kayıp 0.5110
  Tur 2: x < 33.0 | kazanç 1.252 | yapraklar -1.023 /  0.957 | kayıp 0.4290
  Tur 3: x < 26.5 | kazanç 0.493 | yapraklar -1.221 /  0.439 | kayıp 0.3962
  Tur 4: x < 52.5 | kazanç 0.318 | yapraklar -0.208 /  1.285 | kayıp 0.3755
λ = 5.0, γ = 0.0
  Tur 1: x < 33.0 | kazanç 0.580 | yapraklar -0.240 /  0.370 | kayıp 0.6428
  Tur 2: x < 26.5 | kazanç 0.476 | yapraklar -0.245 /  0.309 | kayıp 0.6023
  Tur 3: x < 33.0 | kazanç 0.395 | yapraklar -0.214 /  0.285 | kayıp 0.5696
  Tur 4: x < 26.5 | kazanç 0.324 | yapraklar -0.217 /  0.239 | kayıp 0.5431
λ = 5.0, γ = 0.8
  Tur 1: hiçbir bölme γ cezasını karşılamıyor, ağaç büyümüyor
```

Ne öğrendik. $\lambda = 0$ iken yaprak değerleri ± 2'ye kadar çıktı ve 4. turda model 50 yaşındaki tek "hayır" için ayrı bir bölme açtı: gürültüyü kovalamaya başladı. $\lambda = 5$ yaprakları $\pm 0{,}3$ civarında tuttu ve model yalnızca iki anlamlı eşiği (26,5 ve 33) kullandı. $\gamma = 0{,}8$ ise hiçbir bölmenin bu kadar kazanç getirmediğine karar verdi. Gerçek XGBoost bunlara ek olarak derin ağaçlar, özellik örnekleme ve paralel hesap kullanır; fikir aynıdır.

Alıştırma. `Eğit(1, 0.1)` ekleyiniz ve sonucu üç çalıştırmayla karşılaştırınız. Hangi ayar, verideki gerçek deseni (yaş ~30'dan sonra satın alım) en iyi yakalıyor?

BÖLÜM

3

Denetimsiz öğrenme

Denetimsiz öğrenmede doğru cevaplar yoktur; algoritma verinin kendi yapısını bulur. Kümeleme benzer noktaları gruplar, boyut azaltma ise veriyi daha az sayıyla özetler. Bu yöntemler genellikle bir projenin ilk adımıdır: veriyi tanımak, görselleştirmek, aykırıları bulmak.

9. K-ortalamalar

Fikir. k merkez seçeriz. Her nokta en yakın merkeze atanır, sonra her merkez kendi noktalarının ortalamasına taşınır. Bu iki adım, atamalar değişmeyene kadar tekrarlanır.

$$\min_{\mu_1 \dots \mu_k} \sum_i \| x_i - \mu_{c(i)} \|^2$$

```
program KOrtalamalar
{ Etiketsiz noktaları k kümeye ayırır. İki adım sırayla tekrarlanır:
  1) Atama: her nokta en yakın merkeze atanır.   2) Güncelleme: her merkez, kümesinin
      ↪  ortalamasına taşınır.
  Amaç (eylemsizlik): Σ ‖x − merkez‖²  her turda küçülür ya da aynı kalır. }
kullan Matematik

değişken
  P, merkez: ondalık dizisi dizisi
  küme: tamsayı dizisi
  i, c, tur, k: tamsayı
  üyeler: ondalık dizisi dizisi
  bulut: ondalık dizisi

fonksiyon U2(a, b: ondalık dizisi): ondalık
başla
  sonuç := (a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2
bitir

başla
  Tohum(5)
  P := []
  { üç bulut: (0,0), (5,1), (2,5) çevresinde 15'er nokta }
  [[0, 0], [5, 1], [2, 5]] içindeki her bulut için
    1'den 15'e kadar her i için P.Ekle([bulut[0] + Normal(0, 0.8), bulut[1] +
        ↪  Normal(0, 0.8)])
  k := 3
  merkez := [P[0], P[1], P[2]]       { kötü başlangıç: hepsi aynı buluttan }
  küme := Eşle(P, fonksiyon(p: ondalık dizisi) => 0)
  1'den 6'ya kadar her tur için
  başla
    0'dan Uzunluk(P) - 1'e kadar her i için
      küme[i] := Sırala([0, 1, 2], fonksiyon(j: tamsayı) => U2(P[i], merkez[j]))[0]
    0'dan k - 1'e kadar her c için
    başla
      üyeler := []
      0'dan Uzunluk(P) - 1'e kadar her i için eğer küme[i] = c ise üyeler.Ekle(P[i])
      merkez[c] := [Toplam(Eşle(üyeler, fonksiyon(p: ondalık dizisi) => p[0])) /
          ↪  Uzunluk(üyeler),
                  Toplam(Eşle(üyeler, fonksiyon(p: ondalık dizisi) => p[1])) /
                      ↪  Uzunluk(üyeler)]
    bitir
    yaz(Biçimle('Tur {}: eylemsizlik {:7.2}  merkezler ', tur,
      Toplam(Eşle(Aralık(Uzunluk(P)), fonksiyon(j: tamsayı) => U2(P[j],
          ↪  merkez[küme[j]])))), 0)
    yaz(Eşle(merkez, fonksiyon(m: ondalık dizisi) => [Yuvarla(m[0], 1), Yuvarla(m[1],
        ↪  1)]), 1)
  bitir
bitir.
```

Programın çıktısı şöyledir:

```
Tur 1: eylemsizlik  324.04  merkezler [[1.2, 3.1], [4.2, 1.0], [-0.1, -1.8]]
Tur 2: eylemsizlik   83.05  merkezler [[2.1, 5.0], [5.0, 1.2], [0.2, -0.4]]
Tur 3: eylemsizlik   70.41  merkezler [[2.3, 5.2], [5.0, 1.2], [0.1, -0.3]]
Tur 4: eylemsizlik   70.41  merkezler [[2.3, 5.2], [5.0, 1.2], [0.1, -0.3]]
Tur 5: eylemsizlik   70.41  merkezler [[2.3, 5.2], [5.0, 1.2], [0.1, -0.3]]
Tur 6: eylemsizlik   70.41  merkezler [[2.3, 5.2], [5.0, 1.2], [0.1, -0.3]]
```

Ne öğrendik. Üç merkez de aynı buluttan başladığı halde algoritma 3 turda gerçek merkezleri (0,0), (5,1), (2,5) yaklaşık olarak buldu. k-ortalamalar k'yi sizin seçmenizi ister ve yuvarlak, benzer büyüklükte kümeler varsayar; halka gibi biçimlerde başarısız olur (bkz. DBSCAN). Hazır sürümü `MakineOgrenmesi` birimindeki `KOrtalamalar` fonksiyonudur.

Alıştırma. `k := 2` ve `k := 4` ile çalıştırınız (merkez listesini de buna göre ayarlayınız). Eylemsizlik k ile nasıl değişiyor? "Dirsek yöntemi" bu eğriye bakarak k'yi seçer.

10. Hiyerarşik kümeleme

Fikir. Her nokta kendi kümesiyle başlar; her adımda birbirine en yakın iki küme birleşir. Birleşmelerin sırası ve yüksekliği bir ağaç (dendrogram) oluşturur. Ağacı hangi yükseklikten keserseniz o kadar küme elde edersiniz; k'yi önceden seçmek gerekmez.

$$d(A,B) = \frac{1}{|A|\,|B|} \sum_{a \in A} \sum_{b \in B} \|a - b\|$$

```
program HiyerarşikKümeleme
{ Birleştirici (aşağıdan yukarı) kümeleme: her nokta kendi kümesiyle başlar; her
    ↪ adımda en yakın
  iki küme birleşir. Birleşme sırası bir ağaç (dendrogram) oluşturur; ağacı
      ↪ istediğiniz yükseklikte
  keserek istediğiniz sayıda küme elde edersiniz. Kümeler arası uzaklık: ortalama
      ↪ bağlantı. }
kullan Matematik

değişken
  P: ondalık dizisi dizisi
  adlar: metin dizisi
  kümeler: tamsayı dizisi dizisi
  a, b, i, j, enA, enB: tamsayı
  d, enD: ondalık

fonksiyon Uzaklık(A, B: tamsayı dizisi): ondalık
```

```
değişken x, y: tamsayı
başla
  sonuç := 0
  A içindeki her x için
    B içindeki her y için
      sonuç := sonuç + Kök((P[x][0] - P[y][0]) ** 2 + (P[x][1] - P[y][1]) ** 2)
  sonuç := sonuç / (Uzunluk(A) * Uzunluk(B))
bitir

fonksiyon Ad(K: tamsayı dizisi): metin
başla
  sonuç := '{' + Birleştir(Eşle(K, fonksiyon(x: tamsayı) => adlar[x]), ' ') + '}'
bitir

başla
  { Altı şehrin kabaca konumu (yüz km cinsinden, doğu ve kuzey) }
  adlar := ['İstanbul', 'Bursa', 'İzmir', 'Ankara', 'Konya', 'Kayseri']
  P := [[2.9, 4.1], [2.9, 3.4], [0.9, 2.6], [5.9, 3.1], [5.3, 1.6], [7.7, 2.3]]
  kümeler := Eşle(Aralık(Uzunluk(P)), fonksiyon(x: tamsayı) => [x])
  i := 1
  Uzunluk(kümeler) > 1 iken
  başla
    enD := 1e9
    0'dan Uzunluk(kümeler) - 2'ye kadar her a için
      a + 1'den Uzunluk(kümeler) - 1'e kadar her b için
      başla
        d := Uzaklık(kümeler[a], kümeler[b])
        eğer d < enD ise başla enD := d; enA := a; enB := b bitir
      bitir
    yaz(Biçimle('Adım {}: {:5.2} yükseklikte  {} + {}', i, enD, Ad(kümeler[enA]),
        ↪ Ad(kümeler[enB])), 1)
    kümeler[enA] := kümeler[enA] + kümeler[enB]
    kümeler.Sil(enB)
    eğer Uzunluk(kümeler) = 3 ise yaz('        → 3 kümede kesilirse: ' +
        ↪ Birleştir(Eşle(kümeler, @Ad), ', '), 1)
    i := i + 1
  bitir
bitir.
```

Programın çıktısı şöyledir:

```
Adım 1:  0.70 yükseklikte  {İstanbul} + {Bursa}
Adım 2:  1.62 yükseklikte  {Ankara} + {Konya}
Adım 3:  2.23 yükseklikte  {Ankara Konya} + {Kayseri}
        → 3 kümede kesilirse: {İstanbul Bursa}, {İzmir}, {Ankara Konya Kayseri}
Adım 4:  2.33 yükseklikte  {İstanbul Bursa} + {İzmir}
Adım 5:  4.34 yükseklikte  {İstanbul Bursa İzmir} + {Ankara Konya Kayseri}
```

Ne öğrendik. En son birleşme $(4{,}34)$ öncekilerden çok daha yüksektedir: veri doğal olarak iki gruba (batı ve iç Anadolu) ayrılır. Birleşme yüksekliklerindeki büyük sıçrama, kaç küme seçileceğine dair iyi bir ipucudur. Yöntem tüm çiftlerin uzaklığını hesapladığı için binlerce noktada yavaşlar.

Alıştırma. `Uzaklık` fonksiyonunu ortalama yerine en kısa uzaklığı (tek bağlantı) döndürecek şekilde değiştiriniz. Birleşme sırası değişiyor mu?

11. DBSCAN

Fikir. Kümeleri yoğunluktan tanımlarız. ε yarıçapında en az `minNokta` komşusu olan nokta bir "çekirdek"tir; birbirine yakın çekirdekler zincir gibi aynı kümeye bağlanır. Hiçbir çekirdeğe ulaşamayan noktalar gürültü olarak işaretlenir. Küme sayısı önceden verilmez ve kümeler her biçimde olabilir.

$$N_\varepsilon(p) = \{\, q : \|p - q\| \le \varepsilon \,\}, \qquad p \text{ çekirdektir} \iff |N_\varepsilon(p)| \ge \text{minNokta}$$

```
program DBSCAN
{ Yoğunluğa dayalı kümeleme. ε yarıçaplı komşuluğunda en az minNokta nokta bulunan
    ↪ noktalar "çekirdek"tir.
  Birbirine ε'dan yakın çekirdekler aynı kümeye girer; hiçbir çekirdeğe ulaşamayan
      ↪ noktalar gürültüdür.
  Küme sayısını önceden vermek gerekmez ve kümeler her biçimde olabilir (burada bir
      ↪ halka). }
kullan Matematik

değişken
  P: ondalık dizisi dizisi
  etiket: tamsayı dizisi       { −2: ziyaret edilmedi, −1: gürültü, 0, 1, …: küme }
  i, j, c, kume: tamsayı
  açı, eps: ondalık
  kuyruk, kom: tamsayı dizisi
  minNokta: tamsayı

fonksiyon Komşular(i: tamsayı): tamsayı dizisi
başla
  sonuç := Süz(Aralık(Uzunluk(P)), fonksiyon(j: tamsayı) =>
    Kök((P[i][0] - P[j][0]) ** 2 + (P[i][1] - P[j][1]) ** 2) < = eps)
bitir

başla
  Tohum(2)
  P := []
  0'dan 39'a kadar her i için       { yarıçapı 4 olan halka }
  başla
    açı := i * 2 * Matematik.pi / 40
    P.Ekle([4 * Cos(açı) + Normal(0, 0.2), 4 * Sin(açı) + Normal(0, 0.2)])
```

```
  bitir
  1'den 15'e kadar her i için P.Ekle([Normal(0, 0.4), Normal(0, 0.4)])   { ortada
      ↪ yoğun bir bulut }
  P := P + [[7, 7], [-6, 5], [6, -6]]                                    { üç aykırı
      ↪ nokta }

  eps := 1.5; minNokta := 3
  etiket := Eşle(P, fonksiyon(p: ondalık dizisi) => -2)
  kume := 0
  0'dan Uzunluk(P) - 1'e kadar her i için
  başla
    eğer etiket[i] <> -2 ise devam
    kom := Komşular(i)
    eğer Uzunluk(kom) < minNokta ise başla etiket[i] := -1; devam bitir
    { yeni küme: çekirdekten başlayıp ulaşılabilen herkesi genişleyerek ekle }
    etiket[i] := kume
    kuyruk := kom
    Uzunluk(kuyruk) > 0 iken
    başla
      j := kuyruk.Çıkar(0)
      eğer etiket[j] = -1 ise etiket[j] := kume      { gürültü sanılan bir sınır
          ↪ noktası }
      eğer etiket[j] <> -2 ise devam
      etiket[j] := kume
      kom := Komşular(j)
      eğer Uzunluk(kom) > = minNokta ise kuyruk := kuyruk + kom
    bitir
    kume := kume + 1
  bitir

  yaz('Bulunan küme sayısı: ' + kume, 1)
  0'dan kume - 1'e kadar her c için
    yaz(Biçimle('  Küme {}: {} nokta', c, Uzunluk(Süz(etiket, fonksiyon(e: tamsayı)
        ↪ => e = c))), 1)
  yaz('  Gürültü: ' + Eşle(Süz(Aralık(Uzunluk(P)), fonksiyon(j: tamsayı) => etiket[j]
      ↪ = -1), fonksiyon(j: tamsayı) => P[j]), 1)
bitir.
```

Programın çıktısı şöyledir:

```
Bulunan küme sayısı: 2
  Küme 0: 40 nokta
  Küme 1: 15 nokta
  Gürültü: [[7, 7], [-6, 5], [6, -6]]
```

Ne öğrendik. DBSCAN halkayı (40 nokta) ve ortadaki bulutu (15 nokta) iki ayrı küme olarak buldu, üç aykırı noktayı da gürültü diye ayırdı. k-ortalamalar

aynı veride halkayı dilimlere bölerdi. Bunun bedeli, ε seçimine duyarlılıktır. $\varepsilon = 1{,}2$ ile halka, noktalar arasındaki küçük boşluklardan üç parçaya kopar.

Alıştırma. `eps := 1.2` ve `eps := 5` deneyiniz. İkincisinde ne oluyor, neden?

12. Temel bileşenler analizi (PCA)

Fikir. Veri bulutunun en çok yayıldığı yönü buluruz; bu yön birinci temel bileşendir. Ona dik ve en çok yayılan ikinci yön ikinci bileşendir; böyle devam eder. Matematikte bu yönler kovaryans matrisinin özvektörleridir. Az sayıda bileşeni tutarak veriyi bilgi kaybetmeden küçültebiliriz.

$$C = \frac{1}{n-1} X^{\top} X, \qquad Cv = \lambda v$$

```
program TemelBileşenler
{ Temel Bileşenler Analizi (PCA): veriyi en çok yayıldığı yönlere döndürür.
  1) Veriyi ortala.  2) Kovaryans matrisini kur: C = XᵀX / (n − 1).
  3) C'nin özvektörleri yönleri, özdeğerleri o yöndeki varyansı verir. En büyükleri
     ↪ tutup boyut azaltılır.
  Özvektörü "kuvvet yinelemesi"yle buluyoruz: v ← C·v / ‖C·v‖ işlemi en büyük
     ↪ özvektöre yakınsar. }
kullan YZ, Matematik

değişken
  X, C, v, Cv, v2: tensör
  boy, kilo: ondalık dizisi
  i: tamsayı
  l1, l2: ondalık

başla
  { 8 kişinin boyu (cm) ve kilosu (kg): iki öznitelik güçlü biçimde ilişkili }
  boy := [160, 165, 170, 172, 175, 180, 185, 190]
  kilo := [55, 60, 63, 70, 72, 78, 85, 90]
  X := Tensör([boy, kilo]).T                                  { 8 × 2 matris }
  X := X - X.Ortalama(0)                                       { her sütunu ortala }
  C := X.T @ X / (8 - 1)
  yaz('Kovaryans matrisi:', 1); yaz(C, 1)

  v := Tensör([1.0, 0.0])
  1'den 50'ye kadar her i için
  başla
    Cv := C @ v
    v := Cv / Kök((Cv * Cv).Toplam.Değer)
  bitir
  l1 := (v @ C @ v).Değer
  v2 := Tensör([-v[1], v[0]])            { 2 boyutta ikinci bileşen birinciye diktir
     ↪ }
```

```
  l2 := (v2 @ C @ v2).Değer
  yaz(Biçimle('1. bileşen: ({:.3}, {:.3})  varyans {:.1}  → açıklanan oran %{:.1}',
      ↪ v[0], v[1], l1, 100 * l1 / (l1 + l2)), 1)
  yaz(Biçimle('2. bileşen: ({:.3}, {:.3})  varyans {:.1}  → açıklanan oran %{:.1}',
      ↪ v2[0], v2[1], l2, 100 * l2 / (l1 + l2)), 1)
  yaz('Tek boyuta indirgenmiş veri (1. bileşen üzerindeki izdüşüm):', 1)
  yaz(X @ v, 1)
bitir.
```

Programın çıktısı şöyledir:

```
Kovaryans matrisi:
tensör([[101.125, 122.125],
        [122.125, 149.4107]])
1. bileşen: (0.635, 0.773)  varyans 249.8  → açıklanan oran %99.7
2. bileşen: (-0.773, 0.635)  varyans 0.8  → açıklanan oran %0.3
Tek boyuta indirgenmiş veri (1. bileşen üzerindeki izdüşüm):
tensör([-22.1297, -15.0923, -9.6002, -2.922, 0.5278, 8.3379, 16.9206, 23.958])
```

Ne öğrendik. Boy ve kilo neredeyse aynı bilgiyi taşıyor: birinci bileşen varyansın %99,7'sini açıklıyor. Bu bileşen kabaca "iri yapılılık" anlamına gelir; iki sayı yerine tek sayıyla hemen hemen hiç bilgi kaybetmeden çalışabiliriz. Birimleri farklı öznitelikleri (örneğin cm ve TL) PCA'dan önce standartlaştırmak gerekir; yoksa büyük sayılı öznitelik baskın çıkar.

Alıştırma. Üçüncü bir öznitelik olarak ayakkabı numarasını ekleyiniz (3×3 kovaryans). Kuvvet yinelemesiyle en büyük bileşeni bulunuz.

13. Gauss karışım modelleri (GMM)

Fikir. Verinin birkaç çan eğrisinin (normal dağılımın) üst üste gelmesinden oluştuğunu varsayarız. EM algoritması iki adımı tekrarlar: E adımında her noktanın hangi eğriden gelmiş olabileceği olasılıkla tahmin edilir; M adımında eğrilerin ortalaması, genişliği ve ağırlığı bu olasılıklarla güncellenir. k-ortalamaların "yumuşak" sürümüdür.

$$p(x) = \sum_{k} \pi_k \, \mathcal{N}(x \mid \mu_k, \sigma_k^2)$$

```
program GaussKarışım
{ Veriyi iki Gauss (normal) dağılımın karışımı olarak modeller ve parametreleri EM
    ↪ algoritmasıyla bulur.
  E adımı: her nokta için "hangi dağılımdan gelmiş olabilir?" olasılıkları
      ↪ (sorumluluklar) hesaplanır.
```

```
  M adımı: her dağılımın ortalaması, sapması ve ağırlığı bu olasılıklarla yeniden
      ↪ hesaplanır.
  k-ortalamalardan farkı: atamalar kesin değil, olasılıklıdır. }
kullan Matematik

değişken
  X, r: ondalık dizisi               { r[i]: i. noktanın 1. dağılıma ait olma
      ↪ olasılığı }
  m1, m2, s1, s2, w1, p1, p2, n1, logOlabilirlik: ondalık
  i, tur: tamsayı

fonksiyon Normal1(x, m, s: ondalık): ondalık
başla
  sonuç := Üstel(-0.5 * ((x - m) / s) ** 2) / (s * Kök(2 * Matematik.pi))
bitir

başla
  Tohum(4)
  X := []
  1'den 60'a kadar her i için X.Ekle(Normal(165, 6))     { 1. grup }
  1'den 40'a kadar her i için X.Ekle(Normal(180, 5))     { 2. grup }
  m1 := 160; m2 := 190; s1 := 10; s2 := 10; w1 := 0.5    { kaba bir başlangıç tahmini
      ↪ }
  r := Eşle(X, fonksiyon(x: ondalık) => 0.0)
  1'den 30'a kadar her tur için
  başla
    { E adımı }
    logOlabilirlik := 0
    0'dan Uzunluk(X) - 1'e kadar her i için
    başla
      p1 := w1 * Normal1(X[i], m1, s1)
      p2 := (1 - w1) * Normal1(X[i], m2, s2)
      r[i] := p1 / (p1 + p2)
      logOlabilirlik := logOlabilirlik + Ln(p1 + p2)
    bitir
    { M adımı }
    n1 := Toplam(r)
    m1 := 0; m2 := 0
    0'dan Uzunluk(X) - 1'e kadar her i için başla m1 := m1 + r[i] * X[i]; m2 := m2 +
        ↪ (1 - r[i]) * X[i] bitir
    m1 := m1 / n1; m2 := m2 / (Uzunluk(X) - n1)
    s1 := 0; s2 := 0
    0'dan Uzunluk(X) - 1'e kadar her i için
    başla
      s1 := s1 + r[i] * (X[i] - m1) ** 2
      s2 := s2 + (1 - r[i]) * (X[i] - m2) ** 2
    bitir
    s1 := Kök(s1 / n1); s2 := Kök(s2 / (Uzunluk(X) - n1))
    w1 := n1 / Uzunluk(X)
```

```
    eğer tur < = 3 veya tur mod 10 = 0 ise
      yaz(Biçimle('Tur {:2}: log-olabilirlik {:8.2} | 1: ort {:6.1} sapma {:4.1}
          ↪ ağırlık {:.2} | 2: ort {:6.1} sapma {:4.1}',
        tur, logOlabilirlik, m1, s1, w1, m2, s2), 1)
  bitir
  yaz(Biçimle('172 cm boyundaki biri 1. gruptan olasılıkla {:.2}', w1 * Normal1(172,
      ↪ m1, s1) /
    (w1 * Normal1(172, m1, s1) + (1 - w1) * Normal1(172, m2, s2))), 1)
bitir.
```

Programın çıktısı şöyledir:

```
Tur  1: log-olabilirlik  -408.17 | 1: ort  165.9 sapma  6.1 ağırlık 0.64 | 2: ort  180.1
    ↪ sapma  5.9
Tur  2: log-olabilirlik  -355.73 | 1: ort  165.4 sapma  5.6 ağırlık 0.63 | 2: ort  180.6
    ↪ sapma  5.3
Tur  3: log-olabilirlik  -353.54 | 1: ort  165.2 sapma  5.2 ağırlık 0.63 | 2: ort  180.9
    ↪ sapma  4.8
Tur 10: log-olabilirlik  -352.10 | 1: ort  164.9 sapma  4.9 ağırlık 0.62 | 2: ort  181.0
    ↪ sapma  4.5
Tur 20: log-olabilirlik  -352.10 | 1: ort  164.8 sapma  4.8 ağırlık 0.61 | 2: ort  180.9
    ↪ sapma  4.6
Tur 30: log-olabilirlik  -352.09 | 1: ort  164.8 sapma  4.8 ağırlık 0.61 | 2: ort  180.8
    ↪ sapma  4.6
172 cm boyundaki biri 1. gruptan olasılıkla 0.75
```

Ne öğrendik. Veri gerçekte ortalaması 165 ve 180 olan 60 + 40 kişiden üretildi; EM, etiketleri hiç görmeden ortalamaları 164,8 ve 180,8, ağırlığı 0,61 (gerçek 0,60) buldu. Log-olabilirlik her turda artar; artış durunca algoritma yakınsamıştır. GMM'nin k-ortalamalara göre kazancı son satırda görülür: 172 cm'lik biri için kesin bir etiket değil, %75'lik bir olasılık verir.

Alıştırma. Başlangıç tahminini `m1 := 170; m2 := 171` yapınız. Algoritma yine aynı çözüme ulaşıyor mu, kaç turda?

BÖLÜM

4

Derin öğrenme

Şimdiye kadarki modellerde kuralı biz tasarladık (bir eşik, bir mesafe, bir ağaç). Sinir ağlarında ise yalnızca yapıyı kurarız; milyonlarca küçük ayarı (ağırlığı) veri belirler. Bu bölümde `YZ` biriminin iki özelliğini kullanacağız: tensörler (sayı tabloları) ve otomatik türev. `kayıp.GeriYay` çağrısı, kaybın her ağırlığa göre türevini kendisi hesaplar. Optimize edici (`Adam`) de her ağırlığı kaybı azaltacak yöne biraz kaydırır. Beş programın hepsi bu döngüyü kullanır:

```
opt.SıfırlaGrad          { eski türevleri sil }
kayıp := ...             { ileri hesap: tahmin et, hatayı ölç }
kayıp.GeriYay            { geri yayılım: her ağırlığın türevi }
opt.Adım                 { ağırlıkları güncelle }
```

16. Çok katmanlı algılayıcı (sinir ağı)

Fikir. Bir nöron girdilerinin ağırlıklı toplamını alır ve sonucu doğrusal olmayan bir fonksiyondan (aktivasyon) geçirir. Nöronlar katmanlara dizilir; her katmanın

çıktısı bir sonrakinin girdisidir. Lojistik regresyon tek bir nörondur ve yalnızca düz bir sınır çizebilir. Araya gizli bir katman eklenince ağ çember gibi eğri sınırlar da öğrenebilir.

$$h = \mathrm{ReLU}(xW_1 + b_1), \qquad \hat{y} = \sigma(hW_2 + b_2)$$

```
program ÇokKatmanlıAlgılayıcı
{ Çok katmanlı algılayıcı (MLP): katman katman dizilmiş "nöronlar".
  Her katman  çıktı = aktivasyon(girdi @ W + b)  hesaplar. Doğrusal olmayan
      ↪ aktivasyon (ReLU, tanh)
  sayesinde ağ eğri karar sınırları öğrenebilir. Ağırlıklar geri yayılım + Adam ile
      ↪ güncellenir.
  Görev: bir noktanın yarıçapı 1 olan çemberin içinde mi dışında mı olduğunu
      ↪ öğrenmek. }
kullan YZ

değişken
  noktalar, etiket, deneme: liste
  X, Y, tahmin: tensör
  model, opt: herhangi
  geçmiş: liste
  i, isabet: tamsayı
  px, py: ondalık

başla
  Tohum(7)
  noktalar := []; etiket := []
  1'den 200'e kadar her i için
  başla
    px := Rastgele(4000) / 1000 - 2       { [-2, 2) aralığında }
    py := Rastgele(4000) / 1000 - 2
    noktalar.Ekle([px, py])
    eğer px * px + py * py < 1 ise etiket.Ekle([1]) değilse etiket.Ekle([0])
  bitir
  X := Tensör(noktalar)
  Y := Tensör(etiket)

  { 2 girdi → 16 gizli nöron (ReLU) → 1 çıktı (sigmoid: "içeride olma olasılığı") }
  model := Ardışık([Yoğun(2, 16, 'relu'), Yoğun(16, 1, 'sigmoid')])
  opt := Adam(model.Parametreler, 0.05)
  geçmiş := Eğit(model, X, Y, 'ikiliÇaprazEntropi', opt, 400, 100)

  tahmin := model.Tahmin(X)
  isabet := 0
  0'dan 199'a kadar her i için
    eğer (tahmin[i, 0] > 0.5) = (etiket[i][0] = 1) ise isabet := isabet + 1
  yaz('Eğitim doğruluğu: %' + isabet / 2, 1)
  yaz('Parametre sayısı: ' + model.ParametreSayısı, 1)
```

```
  yaz('Yeni noktalar:', 1)
  deneme := [[0, 0], [0.5, 0.5], [0.9, 0], [1.2, 0], [1.5, 1.5]]
  tahmin := model.Tahmin(Tensör(deneme))
  0'dan 4'e kadar her i için
    yaz(Biçimle('  {}  içeride olma olasılığı {:.2}', deneme[i], tahmin[i, 0]), 1)
bitir.
```

Programın çıktısı şöyledir:

```
  Epok     1/400   kayıp = 0.564626
  Epok   100/400   kayıp = 0.021568
  Epok   200/400   kayıp = 0.006376
  Epok   300/400   kayıp = 0.003043
  Epok   400/400   kayıp = 0.001746
Eğitim doğruluğu: %100.0
Parametre sayısı: 65
Yeni noktalar:
  [0, 0]  içeride olma olasılığı 1.00
  [0.5, 0.5]  içeride olma olasılığı 1.00
  [0.9, 0]  içeride olma olasılığı 0.31
  [1.2, 0]  içeride olma olasılığı 0.00
  [1.5, 1.5]  içeride olma olasılığı 0.00
```

Ne öğrendik. 65 ağırlıklı küçük bir ağ, kimse ona çember formülünü söylemeden eğitim noktalarının hepsini doğru sınıflandırdı. Ama `[0.9, 0]` noktasına dikkat etmekte yarar vardır: nokta aslında içeridedir, ağ ise %31 der. Eğitim doğruluğunun %100 olması modelin kusursuz olduğu anlamına gelmez. Sınıra yakın bölgede az örnek vardı ve ağ sınırı biraz içeriden çizdi. Gerçek başarıyı her zaman ayrı tutulmuş bir test kümesiyle ölçmek gerekir.

Alıştırma. Gizli katmanı `Yoğun(2, 2, 'relu')` yapınız. İki nöron bir çember çizmeye yetiyor mu? Sonra 200 yerine 1000 nokta kullanıp `[0.9, 0]` tahmininin nasıl değiştiğine bakınız.

17. Evrişimli sinir ağı (CNN)

Fikir. Bir resimde "kenar" nerede olursa olsun kenardır. CNN bu yüzden her yerde aynı küçük filtreyi kullanır: filtre resmin üzerinde kaydırılır ve her konumda altındaki yamayla çarpılıp toplanır. Sonuç bir öznitelik haritasıdır: filtrenin aradığı desen nerede varsa orada büyük değer çıkar. Havuzlama bu haritayı küçültür. Gerçek ağlarda filtreler elle seçilmez, veriden öğrenilir.

$$(R * F)[i, j] = \sum_{a} \sum_{b} R[i + a,\ j + b] \cdot F[a, b]$$

```
program EvrişimliAğ
{ Evrişimli sinir ağı (CNN) iki temel işlem üzerine kurulur:
  1) Evrişim: küçük bir filtre (çekirdek) görüntünün üzerinde kaydırılır; her konumda
      ↪ filtre ile altındaki
     yama çarpılıp toplanır. Aynı filtre her yerde kullanıldığı için desen nerede
        ↪ olursa olsun bulunur.
  2) Havuzlama: harita küçük bloklara bölünür, her bloktan en büyük değer alınır.
  Bölüm A elle seçilmiş bir kenar filtresini uygular. Bölüm B bir filtreyi veriden
      ↪ öğrenir:
  yamaları bir matrisin satırlarına dizersek evrişim tek bir matris çarpımına dönüşür
      ↪ (yama @ filtre). }
kullan YZ

değişken
  resim, harita, havuz, filtre: ondalık dizisi dizisi
  sinyal, hedef, yama: ondalık dizisi
  yamalar: liste
  P, H, w: tensör
  opt: herhangi
  kayıp: tensör
  i, adım: tamsayı

fonksiyon Evrişim(R, F: ondalık dizisi dizisi): ondalık dizisi dizisi
değişken i, j, a, b: tamsayı; satır: ondalık dizisi; t: ondalık
başla
  sonuç := []
  0'dan Uzunluk(R) - Uzunluk(F)'e kadar her i için
  başla
    satır := []
    0'dan Uzunluk(R[0]) - Uzunluk(F[0])'a kadar her j için
    başla
      t := 0
      0'dan Uzunluk(F) - 1'e kadar her a için
        0'dan Uzunluk(F[0]) - 1'e kadar her b için
          t := t + R[i + a][j + b] * F[a][b]
      satır.Ekle(t)
    bitir
    sonuç.Ekle(satır)
  bitir
bitir

fonksiyon Göster(M: ondalık dizisi dizisi)
değişken s: ondalık dizisi
başla
  M içindeki her s için
    yaz('  ' + Birleştir(Eşle(s, fonksiyon(v: ondalık) => Biçimle('{:4.0}', v)), '
        ↪ '), 1)
bitir
```

```
başla
  yaz('--- A) Hazır filtreyle dikey kenar bulma ---', 1)
  { 6×6 gri tonlu resim: solu koyu (0), sağı açık (9) }
  resim := []
  1'den 6'ya kadar her i için resim.Ekle([0, 0, 9, 9, 9, 9])
  filtre := [[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]]     { soldan sağa parlaklık artışına
      ↪ tepki verir }
  harita := Evrişim(resim, filtre)
  yaz('Öznitelik haritası (4×4):', 1); Göster(harita)
  { 2×2 en büyük değer havuzlaması }
  havuz := []
  0'dan 1'e kadar her i için
    havuz.Ekle([Maks([harita[2*i][0], harita[2*i][1], harita[2*i+1][0],
        ↪ harita[2*i+1][1]]),
              Maks([harita[2*i][2], harita[2*i][3], harita[2*i+1][2],
                  ↪ harita[2*i+1][3]])])
  yaz('Havuzlamadan sonra (2×2):', 1); Göster(havuz)

  yaz('', 1)
  yaz('--- B) Filtreyi veriden öğrenmek ---', 1)
  Tohum(3)
  sinyal := []
  1'den 30'a kadar her i için sinyal.Ekle(Rastgele(10))
  { Gizli kural: hedef = sinyalin [1, -2, 1] filtresiyle evrişimi. Ağ bu filtreyi
      ↪ bilmiyor. }
  hedef := []; yamalar := []
  0'dan Uzunluk(sinyal) - 3'e kadar her i için
  başla
    yama := [sinyal[i], sinyal[i + 1], sinyal[i + 2]]
    yamalar.Ekle(yama)
    hedef.Ekle(1 * yama[0] - 2 * yama[1] + 1 * yama[2])
  bitir
  P := Tensör(yamalar)                          { 28 × 3: her satır bir yama }
  H := Tensör(hedef)
  w := RastgeleTensör(3, boş, 0.5, doğru)     { öğrenilecek filtre }
  opt := Adam([w], 0.1)
  1'den 300'e kadar her adım için
  başla
    opt.SıfırlaGrad
    kayıp := OKH(P @ w, H)
    kayıp.GeriYay
    opt.Adım
    eğer adım = 1 veya adım mod 100 = 0 ise
      yaz(Biçimle('Adım {:3}: kayıp {:9.4}  filtre ({:.3}, {:.3}, {:.3})', adım,
          ↪ kayıp.Değer, w[0], w[1], w[2]), 1)
  bitir
bitir.
```

Programın çıktısı şöyledir:

```
--- A) Hazır filtreyle dikey kenar bulma ---
Öznitelik haritası (4×4):
    27   27    0    0
    27   27    0    0
    27   27    0    0
    27   27    0    0
Havuzlamadan sonra (2×2):
    27    0
    27    0

--- B) Filtreyi veriden öğrenmek ---
Adım   1: kayıp   98.3248  filtre (0.147, 0.525, -0.366)
Adım 100: kayıp    0.0003  filtre (0.999, -1.997, 0.998)
Adım 200: kayıp    0.0000  filtre (1.000, -2.000, 1.000)
Adım 300: kayıp    0.0000  filtre (1.000, -2.000, 1.000)
```

Ne öğrendik. A bölümünde filtre, koyudan açığa geçilen sütunlarda 27, düz bölgelerde 0 verdi; havuzlamadan sonra bile "kenar solda" bilgisi korunuyor. B bölümü CNN'lerin asıl gücünü gösteriyor: rastgele başlayan filtre, yalnızca girdi–çıktı örneklerine bakarak gizli kuralı `[1, -2, 1]` tam olarak buldu. Gerçek bir CNN aynı şeyi yüzlerce filtre ve birkaç katmanla yapar. İlk katmanlar kenarları, sonrakiler gözler ve tekerlekler gibi daha karmaşık desenleri öğrenir.

Alıştırma. A bölümünde filtreyi yatay kenar bulacak biçimde (`[[-1, -1, -1], [0, 0, 0], [1, 1, 1]]`) değiştiriniz. Bu resimde ne çıkar, neden? Sonra üst yarısı koyu, alt yarısı açık bir resim kurup tekrar deneyiniz.

18. Yinelemeli sinir ağı (RNN)

Fikir. Cümleler, ses ve fiyat grafikleri birer dizidir; anlamları sıraya bağlıdır. RNN diziyi eleman eleman okur ve her adımda bir "gizli durum" günceller. Bu durum, ağın o ana kadar okuduklarından aklında tuttuğu özettir. Her adımda aynı ağırlıklar kullanılır, bu yüzden ağ her uzunlukta diziyi okuyabilir.

$$h_t = \tanh(x_t W_x + h_{t-1} W_h + b), \qquad \hat{y} = h_T W_c + b_c$$

```
program YinelemeliAğ
{ Yinelemeli sinir ağı (RNN): bir diziyi eleman eleman okur ve bir "gizli durum" h
    ↪ taşır.
  Her adımda   h ← tanh(x_t · Wx + h @ Wh + b)   hesaplanır; h, o ana kadar
      ↪ okunanların özetidir.
  Dizinin sonunda çıktı h'den üretilir:   ŷ = h @ Wç + bç.
  Görev: 5 sayılık bir diziyi oku, sonunda İLK sayıyı söyle. Bunu yapabilmesi için
      ↪ ağın
  ilk sayıyı 4 adım boyunca gizli durumunda saklamayı öğrenmesi gerekir. }
kullan YZ
```

```
sabit
  N: tamsayı = 64          { örnek sayısı }
  UZUNLUK: tamsayı = 5     { dizi uzunluğu }
  GİZLİ: tamsayı = 8       { gizli durum boyutu }

değişken
  diziler: ondalık dizisi dizisi
  sütun: liste
  Xt: liste                        { Xt[t]: tüm örneklerin t. öğesi, N × 1 tensör }
  hedef: tensör
  Wx, Wh, b, Wç, bç: tensör
  opt: herhangi
  kayıp, tahmin: tensör
  i, t, adım: tamsayı

fonksiyon İleri(girdiler: liste; n: tamsayı): tensör
değişken h: tensör; t: tamsayı
başla
  h := Sıfırlar(n, GİZLİ)
  0'dan UZUNLUK - 1'e kadar her t için
    h := (girdiler[t] @ Wx + h @ Wh + b).Tanh
  dön h @ Wç + bç
bitir

fonksiyon Sütunlar(D: ondalık dizisi dizisi): liste
değişken t: tamsayı
başla
  sonuç := []
  0'dan UZUNLUK - 1'e kadar her t için
    sonuç.Ekle(Tensör(Eşle(D, fonksiyon(d: ondalık dizisi) => [d[t]])))
bitir

başla
  Tohum(11)
  diziler := []
  1'den N'ye kadar her i için
  başla
    sütun := []
    1'den UZUNLUK'a kadar her t için sütun.Ekle(Rastgele(200) / 100 - 1)   { [-1, 1)
        ↪ }
    diziler.Ekle(sütun)
  bitir
  Xt := Sütunlar(diziler)
  hedef := Tensör(Eşle(diziler, fonksiyon(d: ondalık dizisi) => [d[0]]))

  Wx := RastgeleTensör(1, GİZLİ, 0.5, doğru)
  Wh := RastgeleTensör(GİZLİ, GİZLİ, 0.3, doğru)
  b  := Sıfırlar(GİZLİ, boş, doğru)
```

```
  Wç := RastgeleTensör(GİZLİ, 1, 0.5, doğru)
  bç := Sıfırlar(1, boş, doğru)
  opt := Adam([Wx, Wh, b, Wç, bç], 0.02)

  1'den 600'e kadar her adım için
  başla
    opt.SıfırlaGrad
    kayıp := OKH(İleri(Xt, N), hedef)
    kayıp.GeriYay          { zaman içinde geri yayılım: hata 5 adımın hepsine geri
        ↪ dağıtılır }
    opt.Adım
    eğer adım = 1 veya adım mod 150 = 0 ise
      yaz(Biçimle('Adım {:3}: kayıp {:.4}', adım, kayıp.Değer), 1)
  bitir

  yaz('Hiç görülmemiş diziler:', 1)
  diziler := [[0.8, -0.3, 0.5, -0.9, 0.1], [-0.6, 0.9, 0.9, 0.2, -0.4], [0.1, 0.7,
      ↪ -0.8, 0.6, 0.9]]
  tahmin := İleri(Sütunlar(diziler), 3)
  0'dan 2'ye kadar her i için
    yaz(Biçimle('  {}  →  ağın cevabı {:.2}  (doğrusu {})', diziler[i], tahmin[i, 0],
        ↪ diziler[i][0]), 1)
bitir.
```

Programın çıktısı şöyledir:

```
Adım   1: kayıp 0.2896
Adım 150: kayıp 0.0004
Adım 300: kayıp 0.0012
Adım 450: kayıp 0.0006
Adım 600: kayıp 0.0001
Hiç görülmemiş diziler:
  [0.8, -0.3, 0.5, -0.9, 0.1]  →  ağın cevabı 0.79  (doğrusu 0.8)
  [-0.6, 0.9, 0.9, 0.2, -0.4]  →  ağın cevabı -0.59  (doğrusu -0.6)
  [0.1, 0.7, -0.8, 0.6, 0.9]  →  ağın cevabı 0.11  (doğrusu 0.1)
```

Ne öğrendik. Ağ ilk sayıyı dört adım boyunca "hafızasında" taşımayı öğrendi ve hiç görmediği dizilerde de doğru cevap verdi. Kaybın 300. adımda kısa süre yükselmesi normaldir: Adam bazen bir vadiden taşar, sonra geri döner. Düz RNN'lerin bilinen zayıflığı uzun dizilerdir. Hata geriye doğru yayılırken her adımda küçülür (kaybolan gradyan) ve ağ uzak geçmişi unutur. LSTM ve GRU hücreleri bu sorun için geliştirildi.

Alıştırma. `UZUNLUK` sabitini 15 yapınız. Kayıp aynı hızla düşüyor mu? Sonra görevi "dizinin toplamını söyle" olarak değiştiriniz; hangisi RNN için daha kolay, neden?

19. Otokodlayıcı

Fikir. Otokodlayıcı girdiyi çıktıda yeniden üretmeyi öğrenir; hedef girdinin kendisi olduğu için etiket gerekmez. İşin püf noktası ortadaki dar boğazdır. Kodlayıcı girdiyi birkaç sayılık bir koda sıkıştırır, çözücü bu koddan girdiyi geri kurar. Boğazdan sığmak için ağ verinin özünü bulmak zorundadır. Bu yüzden otokodlayıcılar sıkıştırmada, gürültü temizlemede ve "bu örnek tuhaf" demek gereken anomali tespitinde kullanılır.

$$z = f(x), \qquad \hat{x} = g(z), \qquad \min_{f,g} \; L(x, \hat{x})$$

```
program Otokodlayıcı
{ Otokodlayıcı: girdiyi kendisine kopyalamayı öğrenen bir ağ — ama ortada dar bir
    ↪ boğaz vardır.
  Kodlayıcı girdiyi küçük bir koda sıkıştırır, çözücü koddan girdiyi geri kurar.
  Etiket gerekmez: hedef girdinin kendisidir. Boğazdan geçebilmek için ağ verinin
      ↪ özünü yakalamak zorundadır.
  Klasik deney: 8 farklı "tek-sıcak" vektörü 3 sayıya sıkıştırmak (8 = 2³, yani ikili
      ↪ kod yeterli). }
kullan YZ

değişken
  X, kod, geri: tensör
  kodlayıcı, çözücü, opt: herhangi
  kayıp: tensör
  i, adım, isabet: tamsayı

başla
  Tohum(21)
  X := BirSıcak([0, 1, 2, 3, 4, 5, 6, 7], 8)       { 8 × 8 birim matris }
  kodlayıcı := Ardışık([Yoğun(8, 3, 'sigmoid')])
  çözücü := Ardışık([Yoğun(3, 8)])
  opt := Adam(kodlayıcı.Parametreler + çözücü.Parametreler, 0.1)
  1'den 1500'e kadar her adım için
  başla
    opt.SıfırlaGrad
    kayıp := ÇaprazEntropi(çözücü.İleri(kodlayıcı.İleri(X)), X)   { ÇaprazEntropi
        ↪ softmaksı kendisi uygular }
    kayıp.GeriYay
    opt.Adım
    eğer adım = 1 veya adım mod 500 = 0 ise
      yaz(Biçimle('Adım {:4}: yeniden kurma kaybı {:.4}', adım, kayıp.Değer), 1)
  bitir

  kod := kodlayıcı.İleri(X)
  geri := Softmaks(çözücü.İleri(kod))
  yaz('Girdi  →  3 sayılık kod        →  geri kurulan', 1)
```

```
  isabet := 0
  0'dan 7'ye kadar her i için
  başla
    yaz(Biçimle('  {}    →  ({:.2}, {:.2}, {:.2})  →  {}', Birleştir(Eşle(Aralık(8),
        ↪  fonksiyon(k: tamsayı) => eğer k = i ise '1' değilse '0'), ''),
      kod[i, 0], kod[i, 1], kod[i, 2], ArgMaks(geri)[i]), 1)
    eğer ArgMaks(geri)[i] = i ise isabet := isabet + 1
  bitir
  yaz('Doğru geri kurulan: ' + isabet + ' / 8', 1)
bitir.
```

Programın çıktısı şöyledir:

```
Adım    1: yeniden kurma kaybı 2.1617
Adım  500: yeniden kurma kaybı 0.0050
Adım 1000: yeniden kurma kaybı 0.0015
Adım 1500: yeniden kurma kaybı 0.0007
Girdi  →  3 sayılık kod        →  geri kurulan
  10000000    →  (0.01, 0.35, 0.34)  →  0
  01000000    →  (0.01, 1.00, 1.00)  →  1
  00100000    →  (0.00, 0.00, 1.00)  →  2
  00010000    →  (0.01, 1.00, 0.00)  →  3
  00001000    →  (1.00, 0.01, 0.99)  →  4
  00000100    →  (0.99, 0.99, 0.01)  →  5
  00000010    →  (0.99, 0.99, 0.98)  →  6
  00000001    →  (0.89, 0.01, 0.01)  →  7
Doğru geri kurulan: 8 / 8
```

Ne öğrendik. Ağ, kimse ona öğretmeden 3 bitlik ikili koda çok benzeyen bir sıkıştırma icat etti: kodların çoğu 0 ve 1'lerden oluşuyor. Sekizinci köşe $(0, 0, 0)$ yerine ilk vektör için $(0{,}01,\ 0{,}35,\ 0{,}34)$ gibi bir ara nokta seçti; çözücü bunu da ayırt edebildiği için bir sorun yoktur. Bu örnekte kodlayıcı ve çözücü iki ayrı `Ardışık` modeldir. Böylece eğitimden sonra kodlayıcıyı tek başına kullanıp her girdinin kodunu alabiliyoruz.

Alıştırma. Boğazı 2 nöron yapınız (`Yoğun(8, 2, 'sigmoid')` ve `Yoğun(2, 8)`). 8 vektörün hepsi yine geri kurulabiliyor mu? Kodlar nasıl yerleşiyor?

20. Transformer ve dikkat mekanizması

Fikir. RNN bir cümleyi sırayla okur ve uzak kelimeleri unutabilir. Dikkat mekanizmasında ise her kelime cümledeki bütün kelimelere aynı anda bakar ve hangisine ne kadar önem vereceğine kendisi karar verir. Her kelime üç vektör üretir: sorgu ("ne arıyorum?"), anahtar ("bende ne var?") ve değer ("ne aktarırım?"). Sorgusu bir anahtara ne kadar benzerse o kelimenin değerinden o kadar çok pay alır. ChatGPT ve Claude gibi dil modellerinin temelindeki Transformer, bu işlemi katman katman yığarak kurulur.

$$\text{Dikkat}(Q, K, V) = \text{softmaks}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

```
program DikkatMekanizması
{ Transformer'ların kalbi: ölçeklenmiş nokta çarpımı dikkati.
  Her kelime üç vektör üretir: Sorgu (Q: "ne arıyorum?"), Anahtar (K: "bende ne
      ↪ var?"), Değer (V: "ne aktarırım?").
     ağırlıklar = softmaks(Q Kᵀ / √d)      çıktı = ağırlıklar @ V
  Bir kelimenin sorgusu hangi kelimenin anahtarına benziyorsa, o kelimenin değerinden
      ↪ daha çok pay alır.
  Böylece her kelime cümlenin geri kalanından bağlamını toplar.
  Bölüm A: elle seçilmiş küçük vektörlerle "o" zamirinin kime baktığını görelim.
  Bölüm B: Q ve K izdüşümlerini veriden öğrenerek dikkatin bir görevi çözmesini
      ↪ sağlayalım. }
kullan YZ, Matematik

değişken
  kelimeler: metin dizisi
  E, Wq, Wk, Q, K, V, A: tensör
  i, adım: tamsayı
  kayıp: tensör
  opt: herhangi
  girdi, hedef, çıkış: tensör

fonksiyon Dikkat(Q, K: tensör): tensör
başla
  dön Softmaks(Q @ K.T / Kök(K.Şekil[1]))
bitir

başla
  yaz('--- A) "kedi sütü içti çünkü o acıkmıştı" ---', 1)
  kelimeler := ['kedi', 'sütü', 'içti', 'çünkü', 'o', 'acıkmıştı']
  { 4 boyutlu temsiller: [canlı mı, nesne mi, eylem mi, zamir mi] }
  E := Tensör([[1.0, 0.0, 0.0, 0.0],
               [0.0, 1.0, 0.0, 0.0],
               [0.0, 0.2, 1.0, 0.0],
               [0.0, 0.0, 0.3, 0.0],
               [0.3, 0.0, 0.0, 1.0],
               [0.4, 0.0, 1.0, 0.0]])
  { Elle kurulmuş izdüşümler: zamir özelliği olan kelimeler "canlı" bir şey arasın }
  Wq := Tensör([[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 1, 0], [3, 0, 0, 0]])
  Wk := Tensör([[3, 0, 0, 0], [0, 0, 0, 0], [0, 0, 1, 0], [0, 0, 0, 0]])
  Q := E @ Wq
  K := E @ Wk
  A := Dikkat(Q, K)
  yaz('Dikkat ağırlıkları (satır: bakan kelime, sütun: bakılan kelime)', 1)
  yaz('            ' + Birleştir(Eşle(kelimeler, fonksiyon(k: metin) =>
      ↪ Biçimle('{:10}', k)), ''), 1)
```

```
  0'dan 5'e kadar her i için
    yaz(Biçimle('{:11} ', kelimeler[i]) + Birleştir(Eşle(Aralık(6), fonksiyon(k:
        ↪ tamsayı) => Biçimle('{:10.2}', A[i, k])), ''), 1)
  yaz('"o" en çok şuna bakıyor: ' + kelimeler[ArgMaks(A.Satır(4))], 1)

  yaz('', 1)
  yaz('--- B) Dikkati öğrenmek: "her konum bir sonraki konuma baksın" ---', 1)
  { 4 konum; her konumun temsili kendi konumunun tek-sıcak vektörü. Hedef: her konum,
    bir sonraki konumun değerini getirsin (sonuncu başa dönsün). Wq ve Wk rastgele
        ↪ başlar. }
  Tohum(8)
  girdi := BirSıcak([0, 1, 2, 3], 4)
  { V'nin her sütunu ayrı bir eğitim örneği (aynı konumlar, farklı değerler).
    Tek örnekle ağ "hileli" karışımlar bulabilir; çeşitli örnekler gerçek kuralı
        ↪ zorlar. }
  V := Tensör([[5, 2, 0], [1, 8, 4], [7, 3, 9], [3, 6, 1]])
  hedef := Tensör([[1, 8, 4], [7, 3, 9], [3, 6, 1], [5, 2, 0]])
  Wq := RastgeleTensör(4, 4, 0.5, doğru)
  Wk := RastgeleTensör(4, 4, 0.5, doğru)
  opt := Adam([Wq, Wk], 0.1)
  1'den 300'e kadar her adım için
  başla
    opt.SıfırlaGrad
    çıkış := Dikkat(girdi @ Wq, girdi @ Wk) @ V
    kayıp := OKH(çıkış, hedef)
    kayıp.GeriYay
    opt.Adım
    eğer adım = 1 veya adım mod 100 = 0 ise
      yaz(Biçimle('Adım {:3}: kayıp {:.4}  1. örneğin çıktıları {}', adım,
          ↪ kayıp.Değer,
        Eşle(Aralık(4), fonksiyon(k: tamsayı) => Yuvarla(çıkış[k, 0], 2))), 1)
  bitir
  A := Dikkat(girdi @ Wq, girdi @ Wk)
  yaz('1. örneğin hedefi [1, 7, 3, 5]. Öğrenilen dikkat ağırlıkları:', 1)
  0'dan 3'e kadar her i için
    yaz(Biçimle('  konum {} → ', i) + Birleştir(Eşle(Aralık(4), fonksiyon(k: tamsayı)
        ↪ => Biçimle('{:6.2}', A[i, k])), ''), 1)
  { Öğrenilen şey değerler değil bakılacak yer: yepyeni değerlerle de çalışır }
  V := Tensör([[10.0], [20.0], [30.0], [40.0]])
  çıkış := A @ V
  yaz('Yeni değerler [10, 20, 30, 40] → ' + Eşle(Aralık(4), fonksiyon(k: tamsayı) =>
      ↪ Yuvarla(çıkış[k, 0], 1)), 1)
bitir.
```

Programın çıktısı şöyledir:

```
--- A) "kedi sütü içti çünkü o acıkmıştı" ---
Dikkat ağırlıkları (satır: bakan kelime, sütun: bakılan kelime)
                 kedi      sütü      içti     çünkü         o acıkmıştı
```

```
       kedi       0.17      0.17      0.17      0.17      0.17      0.17
       sütü       0.17      0.17      0.17      0.17      0.17      0.17
       içti       0.13      0.13      0.22      0.16      0.13      0.22
      çünkü       0.16      0.16      0.18      0.16      0.16      0.18
          o       0.87      0.01      0.01      0.01      0.04      0.06
  acıkmıştı       0.13      0.13      0.22      0.16      0.13      0.22
"o" en çok şuna bakıyor: kedi

--- B) Dikkati öğrenmek: "her konum bir sonraki konuma baksın" ---
Adım   1: kayıp 7.1553  1. örneğin çıktıları [3.73, 4.06, 3.93, 3.99]
Adım 100: kayıp 0.0001  1. örneğin çıktıları [1.0, 7.0, 3.0, 5.0]
Adım 200: kayıp 0.0000  1. örneğin çıktıları [1.0, 7.0, 3.0, 5.0]
Adım 300: kayıp 0.0000  1. örneğin çıktıları [1.0, 7.0, 3.0, 5.0]
1. örneğin hedefi [1, 7, 3, 5]. Öğrenilen dikkat ağırlıkları:
  konum 0 →   0.00  1.00  0.00  0.00
  konum 1 →   0.00  0.00  1.00  0.00
  konum 2 →   0.01  0.01  0.00  0.99
  konum 3 →   1.00  0.00  0.00  0.00
Yeni değerler [10, 20, 30, 40] → [20.0, 30.0, 39.7, 10.0]
```

Ne öğrendik. A bölümünde "o" zamiri ağırlığının %87'sini "kedi"ye verdi, yani kimin acıktığını bağlamdan çıkardı. Sorgusu boş olan kelimeler (kedi, sütü) herkese eşit ($1/6 \approx 0{,}17$) baktı. B bölümünde ağ, rastgele başlayan Wq ve Wk'den "bir sonrakine bak" desenini öğrendi. Öğrenilen şey değerler değil nereye bakılacağı olduğu için desen yepyeni değerlerle de çalışıyor. Programdaki yoruma da dikkat etmekte yarar vardır: tek bir örnekle eğitilen ağ, hedefi tutturan ama genellemeyen "hileli" karışımlar bulabilir. V'yi tek sütuna indirip deneyerek bunu kendiniz görebilirsiniz. Gerçek bir Transformer bu işlemi birden fazla "kafa" ile paralel yapar, araya Yoğun katmanlar koyar ve onlarca kez üst üste yığar.

Alıştırma. B bölümünde hedefi "bir önceki konuma bak" olarak değiştiriniz (hedef satırlarını bir aşağı kaydırınız). Ardından V'yi tek sütun `[[5], [1], [7], [3]]` yapıp eğitiniz. Öğrenilen ağırlıklar ve yeni değerlerle sonuç nasıl değişiyor?

EK

A

Hangi algoritma ne zaman?

Karşılaştırma tablosu

#	Algoritma	Tür	En iyi olduğu durum	Dikkat edilecek nokta
1	Doğrusal regresyon	Denetimli, sayı tahmini	İlişki kabaca düz bir çizgiyse; ilk deneme modeli	Aykırı değerlere duyarlı
2	Lojistik regresyon	Denetimli, sınıflandırma	İki sınıf, olasılık gerekli, yorumlanabilirlik önemli	Yalnızca düz sınır çizer
3	K-en yakın komşu	Denetimli	Küçük veri, karmaşık sınırlar	Ölçeklendirme şart; büyük veride yavaş

#	Algoritma	Tür	En iyi olduğu durum	Dikkat edilecek nokta
4	Naif Bayes	Denetimli, sınıflandırma	Metin, spam, çok hızlı bir temel model	Öznitelikleri bağımsız varsayar
5	Karar ağacı	Denetimli	Kuralların okunabilir olması gerektiğinde	Derin ağaç ezberler
6	Rastgele orman	Denetimli	Tablo verisinde güvenli varsayılan	Tek ağaçtan daha zor yorumlanır
7	Gradyan artırma	Denetimli	Tablo verisinde en yüksek doğruluk	Öğrenme oranı ve tur sayısı ayar ister
8	Destek vektör makinesi	Denetimli	Orta boy veri, net bir sınır payı	Ölçeklendirme şart; büyük veride yavaş
9	K-ortalamalar	Denetimsiz, kümeleme	Yuvarlak, benzer boyutlu gruplar	k'yi siz seçersiniz; başlangıca duyarlı
10	Hiyerarşik kümeleme	Denetimsiz	Küme sayısı belirsiz, ilişki ağacı isteniyor	Binlerce noktada yavaş
11	DBSCAN	Denetimsiz	Tuhaf biçimli kümeler, aykırı bulma	ε seçimi kritik
12	PCA	Denetimsiz, boyut azaltma	Görselleştirme, öznitelik sıkıştırma	Yalnızca doğrusal yapıyı yakalar
13	Gauss karışımı	Denetimsiz	Yumuşak (olasılıklı) küme üyeliği	Yerel en iyiye takılabilir
14	AdaBoost	Denetimli	Zayıf kurallardan güçlü sınıflandırıcı	Gürültülü etiketlere duyarlı
15	XGBoost	Denetimli	Yarışmalar, büyük tablo verisi	Birçok ayar parametresi

#	Algoritma	Tür	En iyi olduğu durum	Dikkat edilecek nokta
16	Sinir ağı (MLP)	Denetimli	Çok veri, karmaşık ilişkiler	Veri aç, aşırı uyuma yatkın
17	CNN	Derin	Resim, ses, sinyal	Bol veri ve hesap gücü
18	RNN	Derin	Kısa diziler, zaman serileri	Uzun dizilerde unutur
19	Otokodlayıcı	Derin, denetimsiz	Sıkıştırma, anomali tespiti, gürültü temizleme	Boğaz boyutu ayar ister
20	Transformer / dikkat	Derin	Dil, uzun diziler, çok modlu veri	Çok veri ve hesap gücü

Kabaca bir yol haritası da şöyledir:

- **Etiketiniz var ve sayı tahmin ediyorsanız:** doğrusal regresyonla başlayabilir, yetmezse gradyan artırmaya geçebilirsiniz.
- **Etiketiniz var ve sınıf tahmin ediyorsanız:** lojistik regresyon veya naif Bayes ile bir temel çizgi kurabilir, sonra rastgele orman veya gradyan artırmayı deneyebilirsiniz.
- **Etiketiniz yoksa:** veriyi PCA ile iki boyuta indirip inceleyebilir, ardından k-ortalamalar veya DBSCAN deneyebilirsiniz.
- **Veri resim, ses veya metinse:** derin öğrenme (CNN, RNN, Transformer) uygun olacaktır.

Sık yapılan hatalar

1. **Modeli eğittiği veride değerlendirmek.** Eğitim doğruluğu iyimserdir (bkz. 16. bölümdeki `[0.9, 0]` noktası). Veriyi en başta eğitim ve test diye ayırmanız, test kümesine de yalnızca en sonda bakmanız gerekir.
2. **Ölçeklendirmeyi unutmak.** KNN, SVM, k-ortalamalar ve PCA mesafeyle çalışır. Biri santimetre, öteki lira cinsinden iki öznitelikte büyük sayılı olan her şeye hükmeder. Ağaç tabanlı yöntemler ölçekten etkilenmez.
3. **Temel çizgi kurmamak.** "%90 doğruluk" kulağa iyi gelir. Ama örneklerin %90'ı tek bir sınıftansa her şeye o sınıf diyen bir model de %90 alır. Bu yüzden önce en basit modeli çalıştırmak gerekir.
4. **Rastgeleliği sabitlememek.** Programlardaki `Tohum(…)` satırları her çalıştırmada aynı sonucu verir. Tohum olmadan iki deneyi karşılaştırdığınızda farkın değişiklikten mi yoksa şanstan mı geldiğini bilemezsiniz.

5. **Tek bir sonuca güvenmek.** Rastgele orman, k-ortalamalar ve sinir ağları başlangıca göre farklı sonuç verebilir. Birkaç farklı tohumla denemeniz iyi olur.
6. **Gelecekten bilgi sızdırmak.** Tahmin anında elinizde olmayacak bir sütunu (örneğin "kredi geri ödendi mi" tahmininde "son ödeme tarihi") girdi yapmak, gerçekte çalışmayan harika bir model üretir.
7. **Kaybı izlememek.** Eğitim kaybı düşmüyorsa öğrenme oranı çok küçük veya çok büyüktür. Eğitim kaybı düşerken test kaybı artıyorsa model ezberliyordur.
8. **Yanlış kayıp ve aktivasyon eşleşmesi.** `ÇaprazEntropi` softmaksı kendisi uygular, bu yüzden ona ham çıktılar (logit) verilmelidir. Araya bir `Softmaks` daha koymak eğitimi sessizce bozar: program çalışır ama kayıp yüksek kalır. `İkiliÇaprazEntropi` ise sigmoid sonrası olasılık bekler.
9. **Yöntem'e özgü: harf büyüklüğü çakışması.** Yöntem `x` ile `X`'i ve `s` ile `ş`'yi ayırt etmez. Matematikteki gibi veri matrisine `X`, tek noktaya `x` derseniz "zaten tanımlı" hatası alırsınız. Bu kitaptaki programlar `px`, `nokta`, `v` gibi adlar kullanır.

Sonraki adımlar

- **Alıştırmaları yapabilirsiniz.** Her bölümün sonundaki alıştırma, o algoritmanın en önemli ayarını elle değiştirmenizi ister. Bir algoritmayı anlamanın en hızlı yolu onu bozmaktır.
- **Kendi verinizle çalışabilirsiniz.** `oku` ile bir CSV dosyasını okuyup 1–8 arasındaki algoritmalardan birini kendi probleminize uygulayabilirsiniz: harcamalarınız, spor kayıtlarınız, bir sensörün ölçümleri.
- **Hazır fonksiyonları deneyebilirsiniz.** `MakineOgrenmesi` birimi, burada elle yazdığımız algoritmaların bir kısmının hazır sürümlerini sunar. Kendi yazdığınızla aynı sonucu verip vermediklerini karşılaştırabilirsiniz.
- **Modelinizi bir cihaza taşıyabilirsiniz.** `YZ` ile eğittiğiniz `Yoğun` katmanlı bir ağı `ModeliDışaAktar` ile bir Yöntem birimine çevirip ESP32 üzerinde çalıştırabilirsiniz (Yöntem kitabı, gömülü sistemler bölümü).
- **Büyük kütüphanelere geçebilirsiniz.** Gerçek projelerde scikit-learn (1–15) ve PyTorch (16–20) kullanılır. Bu kitaptaki kavramlar oraya birebir taşınır. `Yoğun` katmanın PyTorch'taki karşılığı `nn.Linear`, `Ardışık` modelinki `nn.Sequential`, `kayıp.GeriYay` çağrısınınki `loss.backward()` olur.
- **Okuma önerileri.** Matematik tarafı için Gareth James ve arkadaşlarının *An Introduction to Statistical Learning* kitabı ücretsiz ve yeni başlayanlara uygundur. Derin öğrenme için Michael Nielsen'in çevrim içi kitabı *Neural Networks and Deep Learning*, sinir ağlarını bu kitaptaki gibi sıfırdan kurar.