BSM622 · HAFTA 8

İstatistiksel Makine Öğrenmesi

KNN · Karar Ağaçları · Bagging · Random Forest · Boosting · XGBoost

Kaynak: Bruce, Bruce & Gedeck (2026), AI-Assisted Statistics for Data Scientists, 3rd ed., Bölüm 6

Doç. Dr. Onur Polat

Hacettepe Üniversitesi · Bilişim Enstitüsü · Veri Analitiği ve İstatistik

2026-2027 Güz · Tezsiz Yüksek Lisans

Hafta 8 — İçerik

Bugün ne öğreneceğiz

01

K-En Yakın Komşu (KNN)

Benzerlik tabanlı sınıflandırma

02

Karar Ağaçları (CART)

Özyinelemeli bölme algoritması

03

Bagging ve Random Forest

Bootstrap ile topluluk öğrenmesi

04

Boosting ve XGBoost

Sıralı hata düzeltme

BÖLÜM 1

K-En Yakın Komşu (KNN)

"Benzer kayıtlar benzer sınıflara aittir" — basit, sezgisel, güçlü

KNN — Temel Fikir

Her yeni kayıt için en yakın K kaydı bul

Algoritma (3 adım):

- 1. Benzer özelliklere sahip K kaydı bul
-
- 2. Sınıflandırma: K kayıt arasında çoğunluk sınıfını yeni kayda ata
-
- 3. Regresyon: K komşunun ortalamasını tahmin et

Anahtar Kavramlar

Komşu (Neighbor): bir kayda benzer tahmin değerlerine sahip başka bir kayıt

Uzaklık ölçütü (Distance metric): iki kayıt arasındaki uzaklığı tek sayı olarak özetleyen fonksiyon

Standartlaştırma: ortalamayı çıkar, standart sapmaya böl (z-skor)

K: en yakın komşu hesabında dikkate alınan komşu sayısı

Not: KNN'de "model" yoktur — tüm eğitim verisi hafızada tutulur (lazy learner). Tahmin anında hesaplama yapılır.

LendingClub Kredi Örneği

Hedef: Yeni bir kredinin ödenip ödenmeyeceğini (paid off / default) tahmin et. Değişkenler: dti (borç/gelir), payment_inc_ratio (ödeme/gelir). K=20 ile 200 krediden tahmin — yeni kredi için 9 "default" ve 11 "paid off" komşu bulunur → tahmin: paid off.

Uzaklık Ölçütleri

İki kaydın "ne kadar uzak" olduğunu nasıl ölçeriz?

Öklid Uzaklığı

$$d(x,u) = \sqrt{[(x_1-u_1)^2 + (x_2-u_2)^2 + \ldots + (x_p-u_p)^2]}$$

"Kuş uçuşu" düz çizgi uzaklığı. Varsayılan seçim. Sürekli sayısal değişkenler için uygundur.

Manhattan Uzaklığı

$$d(x,u) = |x_1-u_1| + |x_2-u_2| + \ldots + |x_p-u_p|$$

"Şehir blokları" boyunca katedilen uzaklık. Nokta-noktaya varış süresi önemliyse uygundur.

Mahalanobis Uzaklığı

Değişkenler arası korelasyonu dikkate alır: iki değişken yüksek korelasyonlu ise esasen tek değişken gibi davranır. Temel bileşenler arasındaki Öklid uzaklığına eşdeğerdir. Kovaryans matrisi üzerinden hesaplandığı için maliyeti daha yüksektir.

⚠ Büyük ölçekli değişkenler uzaklığı domine eder. Örn. gelir/kredi tutarı (binlerce) oran değişkenlerini (1-100) ezer. → Standartlaştırma gerekli.

One-Hot Encoding ve Standartlaştırma

Değişkenleri KNN'ye uygun hale getirmek

One-Hot Encoding

Kategorik (string) değişkenleri ikili (binary) kukla değişkenlere dönüştürür. "home_ownership" → 4 sütun (mortgage/own/rent/other). Dijital devre terminolojisinden: tek bitin pozitif olmasına izin verilir.

Standartlaştırma (z-skor)

$$z = (x - \bar{x}) / s$$

"Standart sapma cinsinden ortalamadan uzaklık". Tüm değişkenleri benzer ölçeğe getirir — böylece hiçbir değişken sırf büyük ölçeği nedeniyle modeli domine etmez.

Standartlaştırmanın etkisi — LendingClub örneği

Standartlaştırılmamış veride revol_bal (dolar, binlerce) uzaklığı domine eder — en yakın 5 komşunun revol_bal değerleri yeni kayda çok yakın, diğer değişkenler neredeyse rol oynamaz. Standartlaştırma sonrası tüm değişkenler katkıda bulunur.

🔑 **KNN, PCA ve kümeleme algoritmaları için standartlaştırma zorunludur.**

K Seçimi — Bias-Variance Dengesi

Küçük K: overfitting · Büyük K: oversmoothing

K çok küçükse (örn. K=1)

⚠ **Overfitting (aşırı uyum)**

- Verideki gürültüye kadar her şey ezberlenir
- Yeni veride performans düşer
- Variance yüksek: eğitim seti değişince model çok değişir

K çok büyükse

⚠ **Oversmoothing (aşırı yumuşatma)**

- Yerel yapı kaybolur
- KNN'nin en büyük avantajı (lokal esneklik) yiter
- Bias yüksek: model basit, gerçek ilişkiyi yakalayamaz

En iyi K'yı nasıl buluruz?

- Holdout/validation seti üzerindeki doğruluk ile belirlenir (cross-validation)
- Tipik aralık: K = 1 ile 20 arası
- Berabere kalmamak için **tek sayı** seçilir
- Yüksek sinyal/gürültü oranlı veri (el yazısı tanıma): küçük K uygun
- Düşük sinyal/gürültü oranlı veri (kredi riski): büyük K uygun

KNN Bir "Özellik Motoru" Olarak

KNN tek başına değil, başka modellere yardımcı olarak

Aşamalı kullanım (staged KNN):

1. Veri üzerinde KNN çalıştır → her kayıt için sınıf ya da olasılık türet
2. Bu tahmini **yeni bir özellik** olarak veriye ekle
3. İkinci aşamada başka bir sınıflandırıcı (ağaç, lojistik regresyon) çalıştır

Emlak örneği — "comps" (karşılaştırılabilir satışlar)

Emlakçılar bir evin fiyatını tahmin ederken benzer evlerin son satış fiyatlarına bakar — bu aslında manuel bir KNN. Konum, metrekare, yapı tipi, arsa büyüklüğü, oda sayısı üzerinden "K en benzer ev"in fiyat ortalaması. Bu değer, daha büyük bir modele yeni özellik olarak eklenebilir.

⚠ *Multicollinearity sorun mu? Hayır — KNN'den gelen bilgi yüksek derecede lokaldir (yalnızca birkaç yakın kayıttan türetilir), bu yüzden yeni bilgi sayılır ve yedekli değildir.*

BÖLÜM 2

Karar Ağaçları (CART)

Breiman (1984) — "Eğer-ise-değilse" kurallarıyla sınıflandırma ve regresyon

Karar Ağaçları — Temel Fikir

Sınıflandırma ve Regresyon Ağaçları (CART)

CART / Decision Tree özellikleri:

- "Eğer-ise-değilse" kuralları: yoruma açık, uygulanması kolay
- Doğrusal olmayan ilişkileri ve karmaşık etkileşimleri yakalar
- KNN'nin aksine, kurallar anlaşılır biçimde ifade edilir
- Breiman, Friedman, Olshen, Stone (1984) tarafından geliştirildi
- **Random Forest ve Boosted Trees'in temeli**

Anahtar Kavramlar

Özyinelemeli bölme (Recursive partitioning): veriyi tekrar tekrar homojen alt gruplara böl
Bölme değeri (Split value): kayıtları ikiye ayıran tahmin değeri
Düğüm (Node): bölme değerinin ağaçtaki gösterimi
Yaprak (Leaf): kuralların sonu — sınıflandırma kuralı
Impurity (Safsızlık): alt bölümdeki sınıf karışımının derecesi
Pruning (Budama): overfitting'i azaltmak için dalları kesme

Basit Örnek — LendingClub Kredi Ağacı

"Eğer borrower_score ≥ 0.575 → paid off (%70 güven); değilse borrower_score ≥ 0.375 ve payment_inc_ratio < 10.42 → paid off..."

Ağaç yukarıdan aşağıya (root → leaf) traversed. Her düğüm bir kuraldır. Her yaprak, o yola uyan kayıtlar için tahmindir.

Özyinelemeli Bölme Algoritması

Ağaç nasıl büyür?

Her A bölümü için en iyi iki alt bölümü bul:

```
1. Her Xj tahmin değişkeni için:
   a. Xj'nin her sj değeri için:
       i. Veriyi Xj < sj ve Xj ≥ sj olarak ikiye böl
      ii. Her alt bölümdeki sınıf homojenliğini ölç
   b. Maksimum homojenlik veren sj'yi seç
2. Tüm değişkenler arasından en iyi (Xj, sj) çiftini seç
```

Özyineleme (recursion):

1. A ← tüm veri seti
2. A'yı bölme algoritmasıyla A_1 ve A_2'ye ayır
3. Aynı prosedürü A_1 ve A_2 üzerinde tekrarla
4. Homojenliği yeterince artıran bir bölme kalmadığında dur

Homojenliği / Safsızlığı Ölçmek

Gini impurity ve Entropy

Doğruluk (accuracy) kötü bir ölçüttür. Yaygın iki impurity ölçüsü:

Gini Impurity

$$I(A) = p(1 - p)$$

- p = sınıf 1'in bölümdeki oranı
- Maksimum: p=0.5 iken I=0.25 (en karışık)
- Minimum: p=0 veya p=1 iken I=0 (saf)
- Hesaplaması hızlı

Entropy

$$I(A) = -p \cdot \log_2(p) - (1-p) \cdot \log_2(1-p)$$

- Bilgi teorisinden gelir
- Orta ve yüksek doğruluk oranlarında Gini'den daha yüksek değer
- Logaritma hesabı → biraz daha yavaş
- Benzer davranış gösterir

Algoritmada kullanım

Önerilen her bölme için her iki alt bölümün impurity'si ölçülür → bir ağırlıklı ortalama hesaplanır → en düşük ağırlıklı ortalamayı veren bölme seçilir. Bu, "bilgi kazanımı"nı (information gain) maksimize etmek demektir.

Ağacın Büyümesini Durdurmak

Overfitting'den kaçınmak için üç strateji

1. Minimum Yaprak Boyutu

`min_samples_leaf (varsayılan: 1)`
`min_samples_split (varsayılan: 2)`

Yaprağa düşen kayıt sayısı eşiğin altına düşerse bölünmeyi bırak. Küçük, gürültü-içeren yaprakları önler.

2. Maksimum Derinlik

`max_depth (varsayılan: None)`

Ağacın belirli bir derinliğe kadar büyümesine izin ver. Sabit sınır konusu — çoğu zaman 5-15 arası yeterli.

3. Karmaşıklık Cezası

`ccp_alpha (varsayılan: 0)`

Impurity düşüşü yeterli değilse bölme. Cost-complexity pruning: ağaca karmaşıklık penaltısı eklenir — büyük α → basit ağaç.

💡 *Optimum değer tipik olarak cross-validation ile belirlenir (GridSearchCV).*

Regresyon ve Ağaçların Kullanımı

Sürekli hedef için ağaçlar + güçlü yönleri

Sürekli değer tahmini (regresyon)

Aynı algoritma, farklı ölçüt: impurity yerine alt bölümlerdeki karedevik ortalama farklar (MSE) kullanılır. Performans RMSE ile değerlendirilir. scikit-learn'de: DecisionTreeRegressor.

Ağaçların çekici yönleri:

✓ Veri keşif aracı

Hangi değişkenlerin önemli olduğu, nasıl ilişkilendikleri görselleştirilebilir. Doğrusal olmayan ilişkileri yakalar.

✓ İletişim gücü

Kurallar teknik olmayan kişilere kolayca anlatılabilir. Bir "veri madenciliği" projesini yöneticilere "satmak" için idealdir.

⚠ Tek ağacın zaafı

Tek karar ağacı tahmin gücü olarak genellikle zayıftır. Random Forest ve Boosted Trees ağaçların gücünü korurken tahmin doğruluğunu çok artırır — ancak tek ağacın kural-tabanlı iletişim gücü kaybolur.

BÖLÜM 3

Bagging ve Random Forest

"Kalabalığın bilgeliği" — birçok modelin ortalaması tek modelden daha iyidir

Topluluk Öğrenmesi (Ensemble Learning)

Galton'ın öküzü ve kalabalığın bilgeliği

1906 — Sir Francis Galton, İngiltere'deki bir panayırda

Panayırda sergilenen bir öküzün ağırlığını tahmin etmek için yarışma düzenlenmiş. 800 kişi tahminde bulunmuş. Bireysel tahminler çok farklı olsa da, tüm tahminlerin ortalaması ve medyanı öküzün gerçek ağırlığının %1'i kadar yakın çıkmış. James Surowiecki bu gözlemi "The Wisdom of Crowds" (2004) kitabında incelemiştir.

Ensemble Modelleme — Temel Algoritma

1. Verilen verilerle bir temel modeli oluştur
2. Bir ek model geliştir (farklı algoritma ya da örneklem)
3. Her kayıt için tahminleri **ortala / çoğunluk oyu** al
4. *"Kalabalığın bilgeliği" → tek bir modelden daha iyi tahmin!*

Bagging (Bootstrap Aggregation)

Breiman (1996) — Ensemble için genel algoritma

Algoritma:

1. M (kurulacak model sayısı) ve n (her örneklemde seçilecek kayıt, n < N) belirle. m = 1.
2. Eğitim verisinden bootstrap örneklemi (replacement ile) al → alt-örneklem (Y_m, X_m) = "bag"
3. (Y_m, X_m) ile model $f_m(X)$ eğit
4. m = m + 1. Eğer m ≤ M ise 2'ye git.

Bagged tahmin:

$$f(X) = (1/M) \cdot [f_1(X) + f_2(X) + \ldots + f_M(X)]$$

Variance'ı azaltır (overfitting'i düşürür). Yüksek-variance algoritmalar (karar ağaçları) için özellikle etkilidir.

Random Forest

Bagging + random feature selection = Breiman (2001)

Standart bagging'e ek bir "oynarlık" ekler:

1. Kayıtlardan bootstrap alt-örneklemi al
2. İlk bölme için P değişken içinden p < P değişkeni rastgele (replacement'sız) seç
3. Örneklenen p değişken içinden en iyi (X_j, s_j) çiftini bul — tüm olası bölünme noktalarını değerlendir
4. Maksimum homojenlik veren (X_j, s_j)'yi seç
5. Bir sonraki bölme için 2. adıma geri dön (her bölme için **yeniden** p değişken örneklenir)
6. Ağaç tamamen büyüyene kadar devam et
7. Baştaki 1. adıma dön → yeni bootstrap örneklemi, yeni ağaç

💡 **p (her bölmede örneklenen değişken sayısı) için pratik kural: $p \approx \sqrt{P}$**

```
Python'da scikit-learn: RandomForestClassifier(n_estimators=500, max_features="sqrt", n_jobs=-1, oob_score=True, random_state=42)
```

Out-of-Bag (OOB) Hatası ve Değişken Önemi

RF'nin iki güçlü teşhis aracı

Out-of-Bag Hatası

- Bootstrap örneklemi ~%63'ünü kapsar
- Her ağaç için %37 veri "out-of-bag"
- OOB veriler üzerinde hata hesapla
- ≈ cross-validation ama ücretsiz!
- oob_score=True ile aktive edilir

Değişken Önemi

Hangi değişkenin modeli ne kadar etkilediğini sayısal olarak sıralamak. Tek bir ağaçta görsel olarak okunabilir; RF'de kaybolur — ancak iki yöntemle geri kazanılır.

İki ölçüm yöntemi:

① Accuracy Decrease (Permutation)

Bir değişkenin değerleri rastgele karıştırılırsa modelin doğruluğu ne kadar düşer? Doğruluk OOB verisinden hesaplanır → çapraz-doğrulanmış tahmin. Daha güvenilir ama hesaplama maliyeti yüksek.

② Gini Decrease

O değişken üzerinde yapılan tüm bölmelerden kaynaklanan ortalama Gini impurity düşüşü. Algoritmanın yan ürünü → hızlı. feature_importances_ ile erişilir. Training veri üzerinden → daha az güvenilir.

Random Forest Hiperparametreleri

Düğmeler (knobs) — overfitting'i önlemek için kritik

`n_estimators`

Varsayılan: 500

Ağaç sayısı. Daha fazla ağaç → daha stabil ama daha yavaş. 100-500 tipik. OOB plateauya ulaşana kadar artırılır.

`max_features`

Varsayılan: "sqrt"

Her bölmede değerlendirilecek değişken sayısı. Varsayılan: sqrt(n_features) sınıflandırma için. Düşük değerler ağaçları de-correlate eder.

`min_samples_leaf`

Varsayılan: 1 → 10

Terminal düğümdeki minimum kayıt sayısı. Python'da varsayılan 1 (overfitting riski!). 5-20 arası tavsiye edilir.

`max_leaf_nodes`

Varsayılan: None

Her ağaçtaki maksimum yaprak sayısı. Sınırsız varsayılan → büyük ağaç. Düşürüldüğünde daha basit model.

BÖLÜM 4

Boosting ve XGBoost

"Sıradaki her model önceki modelin hatasını azaltmaya çalışır"

Boosting — Temel Fikir

Hatayı azaltan modellerin dizisi

Bagging = Honda Accord

🚗 Güvenilir ve istikrarlı
- Paralel ağaç eğitimi
- Az tuning gerekir
- OOB skoruyla teşhis
- Overfitting riski düşük

Boosting = Porsche

🏎️ Güçlü ama dikkat ister
- Sıralı ağaç eğitimi
- Çok hiperparametre
- Cross-validation şart
- Overfitting riski yüksek — regularization gerekli

Boosting çeşitleri:

- **AdaBoost (Freund & Schapire, 1997):** yanlış sınıflanan kayıtların ağırlığını artır
- **Gradient Boosting (Friedman, 2001):** bir maliyet fonksiyonunu minimize et; pseudo-residuals'a model fit et
- **Stochastic Gradient Boosting:** her turda hem kayıt hem değişken örnekleme — en yaygın kullanılan tür
- **XGBoost:** stochastic gradient boosting'in en yaygın ve hızlı implementasyonu (Chen & Guestrin, 2016)

AdaBoost Algoritması

Freund & Schapire — en sezgisel boosting versiyonu

1. M (max model sayısı) belirle, m = 1. Her gözlem ağırlığı: $w_i = 1/N$. Ensemble $F_0 = 0$.
2. Ağırlıkları kullanarak ağırlıklı hatayı e_m minimize eden f_m modelini eğit.
3. Modele ensemble'a ekle: $F_m = F_{m-1} + \alpha_m \cdot f_m$, burada $\alpha_m = ½ \cdot \log((1-e_m)/e_m)$
4. Ağırlıkları güncelle: yanlış sınıflanan gözlemlerin ağırlığı ↑ (α_m kadar).
5. m = m + 1. m ≤ M ise 2'ye git.

Boosted tahmin: $F = \alpha_1 \cdot f_1 + \alpha_2 \cdot f_2 + \ldots + \alpha_M \cdot f_M$

Yanlış-sınıflanan gözlemlerin ağırlığının artması → algoritma zayıf tarafına "daha çok antrenman yapar". Düşük hatalı modellerin α_m'si daha büyük → oyları daha ağır.

Gradient Boosting ve XGBoost

AdaBoost'un optimizasyon formüle edilmesi

Gradient Boosting (Friedman, 2001)

Ağırlık güncellemek yerine, problemi maliyet fonksiyonunun optimizasyonu olarak kurar. Her tur, önceki modellerin pseudo-residuals'ına bir model fit eder — büyük residuals'ı olan kayıtlarda "daha çok eğitir". Stochastic Gradient Boosting her turda hem kayıt hem değişken örneklemesi de ekler.

XGBoost — eXtreme Gradient Boosting (Chen & Guestrin, 2016)

- **Public domain, en yaygın boosting yazılımı**
- Hesaplama açısından verimli, çoklu dil desteği (Python, R, Scala)
- Kritik parametreler:
  - `subsample:` her turda örneklenecek kayıt oranı (random forest-benzeri)
  - `eta / learning_rate:` α_m'ye uygulanan shrinkage — küçük (0.1) ise overfitting'i düşürür
  - `max_depth:` ağaç derinliği (varsayılan 3-6 — RF'den sığ)
  - `reg_lambda, reg_alpha:` L2 ve L1 regularization

Overfitting ve Regularization

XGBoost'un kronik hastalığı: aşırı uyum

⚠ XGBoost gözü kapalı overfit eder

LendingClub örneği: tüm değişkenlerle default XGBoost → eğitim hatası %13.3, test hatası %35.3! Büyük fark → model eğitim verisini ezberlemiş, genelleme başarısız. Random Forest'tan farklı olarak XGBoost default ayarlarla güvenilir değildir.

Regularization — maliyet fonksiyonuna penaltı ekle

L2 (Ridge) — reg_lambda

Katsayıların karedevik toplamına ceza. $\lambda \cdot (b_1^2 + \ldots + b_p^2)$. Büyük katsayıları küçültür. Öklid uzaklığı tabanlı.

L1 (Lasso) — reg_alpha

Katsayıların mutlak değer toplamına ceza. $\alpha \cdot (|b_1| + \ldots + |b_p|)$. Bazı katsayıları tam sıfıra çeker → feature selection. Manhattan uzaklığı tabanlı.

💡 **Örnek: lambda=1000 ile LendingClub → eğitim hatası %31, test hatası %33. Eğitim ile test farkı neredeyse kapandı.**

Hiperparametreler ve Cross-Validation

XGBoost'u ehlileştirmek için grid search + CV

`eta / learning_rate`	Shrinkage faktörü (0-1). Varsayılan: 0.3 (R) / 0.1 (Python). Gürültülü veri için 0.05-0.1 önerilir. Daha düşük → daha çok tur gerekli.
`n_estimators / nrounds`	Boosting turu sayısı. eta küçükse artırılmalı. Overfitting'i önleyen parametreler varsa fazla tur zarar vermez.
`max_depth`	Ağaç derinliği. Varsayılan 6 (R) / 3 (Python). RF'den sığ ağaçlar tercih edilir — gürültülü veriyle karmaşık etkileşimlerden kaçınmak için.
`subsample, colsample_bytree`	Her tur için örneklenen kayıt ve değişken oranı. Varsayılan: 1.0 (hepsi). 0.5-0.8 — overfitting'i düşürür.
`reg_lambda, reg_alpha`	L2 ve L1 regularization. Overfitting kontrolü. Python varsayılanı: reg_lambda=1, reg_alpha=0. R'de her ikisi de 0.

K-fold Cross-Validation ile Hiperparametre Seçimi

LendingClub örneği

Prosedür:

1. Veriyi K fold'a (örn. 5) rastgele böl
2. Her fold için: diğer K-1 fold ile eğit, kalan fold üzerinde test et → fold hatası
3. K fold'un ortalama hatası → o hiperparametre setinin "örneklem dışı" doğruluğu
4. Tüm parametre kombinasyonlarını dene → en düşük ortalama hatayı veren seti seç
5. Nihai modeli tüm veri üzerinde bu optimum parametrelerle yeniden eğit

Grid search sonucu (LendingClub, 5-fold):

```
tree_depth=3, learn_rate=0.5 → %33.34 (en iyi)
tree_depth=6, learn_rate=0.1 → %33.51
tree_depth=3, learn_rate=0.9 → %33.67
tree_depth=12, learn_rate=0.9 → %38.52 (en kötü)
```

→ Sığ ağaçlar + düşük learning rate daha doğru + daha stabil sonuç veriyor.

Dört Yöntemin Karşılaştırması

Hangi metod ne zaman?

Yöntem	Hız	Yorumlanabilirlik	Overfit Riski	Tuning Zorluğu	Tipik Kullanım
KNN	Yavaş (tahmin)	Düşük	Yüksek (K küçükse)	Düşük (K + scale)	Küçük veri, feature engine
Karar Ağacı	Hızlı	YÜKSEK ✓	Çok yüksek	Orta	Keşif + iletişim
Random Forest	Orta	Orta	Düşük	Düşük	Üretim, dengeli
XGBoost	Orta (paralel)	Düşük	Yüksek — reg. gerekli	YÜKSEK	En yüksek doğruluk

Pratik öneri: Prototip için Random Forest ile başla (robust, az tuning). Son %1-2'lik iyileştirme için XGBoost + CV. Yorumlanabilirlik şartsa tek karar ağacı veya SHAP değerleri.

Hafta 8 — Özet

İstatistiksel Makine Öğrenmesi'nin dört yapıtaşı

KNN

- Benzer kayıtların sınıfını ata
- Standartlaştırma şart
- Feature engine olarak güçlü

Karar Ağaçları

- Özyinelemeli bölme, Gini/Entropy
- Yorumlanabilir kurallar
- Pruning ile overfitting kontrolü

Random Forest

- Bagging + random features
- OOB hatası = ücretsiz CV
- Permutation vs Gini importance

XGBoost

- Stochastic gradient boosting
- L1/L2 regularization kritik
- CV ile hiperparametre arama

🎯 Bir sonraki haftaya hazırlık

- Bu hafta işlenen dört yöntemi LendingClub (veya kendi verinizle) sklearn ile uygulayacak ödev yayınlanacak.
- Haftaya: Unsupervised Learning (Kümeleme, PCA) — denetimsiz öğrenme yöntemleri.
- Okuma: Bruce, Bruce & Gedeck (2026), Bölüm 6 — sayfa 249-297.