Dr.Öğr.Üyesi Furkan Göz

2025

Makine Öğrenmesine Giriş

- Makine öğrenmesi, sistemlerin açıkça programlanmaksızın veri üzerinden öğrenmesini sağlayan bir yapay zeka alt alanıdır.
- Amaç: Girdilerle çıktılar arasındaki örüntüyü öğrenip, yeni girdilere uygun tahminler üretmektir.
- Uygulama alanları: e-posta filtreleme, kredi risk skorlama, sağlık teşhis sistemleri, görüntü tanıma, öneri sistemleri.
- Makine öğrenmesi istatistik, veri bilimi ve bilgisayarla öğrenme kavramlarının birleşimidir.

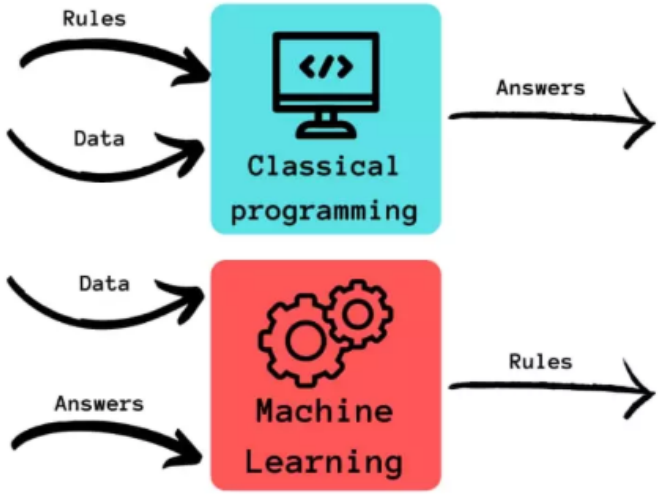
Neden Makine Öğrenmesi?

- Geleneksel programlama: ”Kuralları insan yazar, sistem çıktıyı üretir”
- Makine öğrenmesi: ”Veri ve sonuç verilir, sistem kuralları öğrenir”
- Kural yazılamayacak kadar karmaşık sistemlerde (örneğin görsel tanıma, doğal dil işleme) insan programcı yetersiz kalabilir.
- Büyük veri çağında, insan gözetiminde öğrenen sistemlere ihtiyaç vardır.
- Karar destek sistemleri, otomasyon, kişiselleştirme gibi alanlarda başarısı kanıtlanmıştır.

Not: Makine öğrenmesi veri odaklı düşünmeyi zorunlu kılar.

Klasik Programlama vs. Makine Öğrenmesi

- Klasik Programlama
  - Kurallar (kod) ve veriler bilgisayara verilir.
  - Bilgisayar bu kuralları uygular ve sonuç üretir.
- Makine Öğrenmesi
  - Veriler ve cevaplar (etiketli veri) sağlanır.
  - Bilgisayar kuralları (modeli) öğrenir.



Makine Öğrenmesinde Öğrenme Türleri

- Makine öğrenmesinde kullanılan algoritmalar, verinin etiketli olup olmamasına göre sınıflandırılır:

  1. Gözetimli Öğrenme (Supervised Learning):
     - Her veri örneği bir giriş (X) ve doğru çıkış (y) içerir.
     - Amaç: Girdi-çıktı ilişkisini öğrenerek yeni verilere doğru tahmin yapabilmektir.
     - Uygulamalar: Sınıflandırma (örneğin spam tespiti), regresyon (örneğin ev fiyatı tahmini)

2. Gözetimsiz Öğrenme (Unsupervised Learning):
   - Sadece giriş verileri (X) mevcuttur, çıkış etiketi yoktur.
   - Amaç: Veri içinde yapılar, gruplar veya dağılımlar bulmaktır.
   - Uygulamalar: Kümeleme (örneğin müşteri segmentasyonu), boyut indirgeme (örneğin PCA)

Veri ile Model Arasındaki İlişki

- Makine öğrenmesinde model, verilerdeki örüntüyü (pattern) yakalamaya çalışır.
- Her veri noktası: özellikler (features) → çıktı (etiket)
- Model, bu örneklerden yola çıkarak genelleme yapmayı öğrenir.
- Model = Matematiksel fonksiyon + öğrenilen parametreler
- Temsili form: $\hat{y} = f(x)$
- Modelin başarısı, sadece doğru algoritmaya değil, doğru veri temsiline bağlıdır.

Örnek: Bir hastanın yaş, tansiyon ve nabız bilgisi → kalp krizi riski (evet/hayır)

Modelleme Süreci ve Bileşenleri

- Bir makine öğrenmesi projesi genellikle şu adımları içerir:
  1. Veri Toplama – Gerçek dünyadan örneklerin toplanması
  2. Veri Hazırlama – Temizleme, dönüştürme, eksikleri tamamlama
  3. Özellik Belirleme – Hangi değişkenlerin kullanılacağına karar verme
  4. Model Seçimi – Problem türüne göre uygun algoritma seçimi
  5. Model Eğitimi – Eğitim verisiyle modeli besleme
  6. Değerlendirme – Test verisi ile performans ölçme
  7. İyileştirme – Hata analizi, parametre ayarı, yeni veri kullanımı
- Bu sürece bazen "veri odaklı döngü" de denir:

  Veri → Model → Tahmin → Değerlendirme → Veri...

Not: Süreç teknik olduğu kadar karar destek sürecidir.

Veri Hazırlama Adımları

- Makine öğrenmesinde veri hazırlığı, modelden daha önemlidir.
- İyi hazırlanmamış veri, en iyi algoritmayı bile başarısız kılabilir.
- Temel adımlar:
  - 1. Veri Ayrıştırma: Girdi (X) ve çıktı (y) ayrımı
  - 2. Eksik Verilerle Başa Çıkma: Silme, ortalama ile doldurma, tahmin etme
  - 3. Ölçekleme (Scaling): Özelliklerin aynı aralıkta olması gerekir (Min-Max, Z-skor)
  - 4. Etiket Kodlama (Encoding): Kategorik verilerin sayısallaştırılması
  - 5. Eğitim/Test Bölme: Modelin genelleme gücünü değerlendirmek için veri bölünür
- Bu işlemler çoğu zaman otomatik değil, bilinçli kararlarla yapılmalıdır.

Eksik Verilerle Başa Çıkma

- Gerçek veri setlerinde eksik değerler (missing values) sıkça karşılaşılır.
- Eksik veri nedenleri: sistem hatası, kullanıcı ihmalı, sensör bozulması vb.
- Baş etme yöntemleri:
  - Silme (Dropping): Eksik satırları ya da sütunları tamamen çıkarma
    - Avantaj: Kolay ve hızlı
    - Dezavantaj: Veri kaybı, örnek sayısında azalma
  - Doldurma (Imputation):
    - Sayısal değişkenler: ortalama, medyan, mod
    - Kategorik değişkenler: en sık tekrar eden değer
    - Gelişmiş: regresyonla ya da benzer örneklerle tahmin

Eksik Veri: Silmek mi, Doldurmak mı?

Duruma göre karar verilmelidir:

- 1. Veri seti küçükse & çok eksik varsa:
  - Örnek: 100 gözlem var, 30'unda yaş bilgisi eksik
  - Silme → Kayıp büyük → Mantıksız
  - Doldurma (ortalama/medyan) → Tercih edilir
- 2. Eksik veri önemsiz bir değişkende:
  - Örnek: Profil resmi yüklenme durumu eksik (sınıflandırmaya etkisiz)
  - Silme → Uygun olabilir
- 3. Eksik veri çok kritik bir değişkende:
  - Örnek: Kan şekeri düzeyi eksik → Modelin ana değişkeni
  - Gelişmiş doldurma (regresyon, k-NN imputation) önerilir
- 4. Çok az sayıda eksik gözlem varsa (örneğin %1):
  - Silme → Genellikle zararsızdır

Eksik Verilerle Başa Çıkma – Python Örneği

```python
import pandas as pd
df = pd.read_csv("veri.csv")

# Eksik degerleri say
print(df.isnull().sum())

# Sayisal alanlari ortalama ile doldur
df["yas"] = df["yas"].fillna(df["yas"].mean())
```

Özellik Ölçekleme – Python Örneği

```python
from sklearn.preprocessing import MinMaxScaler,
    StandardScaler
# Min-Max ölçekleme (0-1 araligina)
minmax = MinMaxScaler()
X_scaled_minmax = minmax.fit_transform(X)
# Z-score standardizasyonu
standard = StandardScaler()
X_scaled_zscore = standard.fit_transform(X)
```

- `fit_transform()`: Eğitim verisine göre parametreleri öğrenir ve uygular.
- Test seti varsa: `scaler.transform(X_test)` şeklinde uygulanmalıdır.

Etiket Kodlama (Encoding) Nedir?

- Makine öğrenmesi algoritmaları, genellikle metinsel (kategorik) verilerle doğrudan çalışamaz.
- Bu nedenle kategorik veriler, sayısal forma dönüştürülmelidir.
- Bu dönüştürme işlemine etiket kodlama (encoding) denir.
- Kodlama yöntemi; veri tipi, algoritma ve işlem kapasitesine göre seçilmelidir.

Not: Bazı algoritmalar (örneğin `DecisionTree`, `LightGBM`) kategorik verilerle doğrudan çalışabilir.

Label Encoding (Etiket Numaralama)

- Her kategoriye bir tam sayı atanır:
  - Örn: Cinsiyet = `Erkek → 0`, `Kadın → 1`
- Sıralı kategoriler için uygundur (örn. Düşük-Orta-Yüksek).
- Ancak sırasız kategorilerde bu sayılar yanıltıcı sıralama anlamı taşıyabilir.
- Örneğin: `Kedi=0`, `Köpek=1`, `Kuş=2` → Kuş > Köpek > Kedi mi?

Ne zaman kullanılmalı?

- Ağaç tabanlı modellerde (Decision Tree, Random Forest, XGBoost)
- Sayılar arasında sıralama değil, sadece farklılık önemliyse

Ne zaman kaçınılmalı?

- Lineer regresyon, lojistik regresyon gibi modellerde
- Kategoriler arasında anlamlı bir sıralama yoksa

Label Encoding

```python
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df["cinsiyet_encoded"] = encoder.fit_transform(df["cinsiyet
    "])
```

- `LabelEncoder`: Kategorileri sayılara dönüştürür
- Kategoriler alfabetik sıraya göre numaralandırılır
- Modelin yanlış sıralama ilişkisi kurabileceği unutulmamalı

	cinsiyet	cinsiyet_encoded
0	Kadın	1
1	Erkek	0
2	Kadın	1
3	Erkek	0

One-Hot Encoding

- Her kategori için ayrı bir sütun oluşturur.
- İlgili kategoriye ait sütun `1`, diğerleri `0` olur.
- Sayılar arası ilişki kurulmaz, bu nedenle sırasız kategorilerde daha güvenlidir.

Örnek: Renk = Kırmızı, Mavi, Yeşil

- Kırmızı $\rightarrow$ (1, 0, 0)
- Mavi $\rightarrow$ (0, 1, 0)
- Yeşil $\rightarrow$ (0, 0, 1)

Dezavantaj: Kategori sayısı çoksa, sütun sayısı hızla artar.

One-Hot Encoding – Python Örneği

```python
df = pd.DataFrame({"renk": ["Kirmizi", "Mavi", "Yesil", "Mavi"]})
onehot_df = pd.get_dummies(df, columns=["renk"])
print(onehot_df)
```

- `get_dummies()`: Her kategori için ayrı sütun üretir
- Sırasız kategoriler için uygundur (örn. renk, şehir, marka)
- Oluşan sütunlar 0 ve 1 (binary) içerir

	renk_Kırmızı	renk_Mavi	renk_Yeşil
0	1	0	0
1	0	1	0
2	0	0	1
3	0	1	0

Etiket Kodlama

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder
df = pd.DataFrame({
    "renk": ["kirmizi", "mavi", "yesil", "kirmizi"]})
# Label Encoding
le = LabelEncoder()
df["renk_encoded"] = le.fit_transform(df["renk"])
# One-Hot Encoding
df_ohe = pd.get_dummies(df["renk"], prefix="renk")
```

- LabelEncoder: ”kırmızı”=0, ”mavi”=1, ”yeşil”=2 gibi dönüşüm yapar.
- One-Hot Encoding: Her kategori için ayrı sütun, 0/1 ile temsil edilir.

Veri Setini Eğitim ve Test Olarak Ayırma

- Modelin başarısını ölçebilmek için verinin bir kısmı eğitim (train), diğer kısmı test (test) için ayrılır.
- Eğitim verisi: Modelin örüntüyü öğrenmesi için kullanılır.
- Test verisi: Öğrenilen bilginin daha önce görülmemiş verilerde ne kadar işe yaradığını gösterir.
- Yaygın bölme oranı: `train = 80%, test = 20%`
- Alternatif: validation set veya cross-validation teknikleri (ileride ele alınır)
- Amaç: Gerçek hayattaki performansı doğru tahmin edebilmek

Uyarı: Test verisini eğitim sırasında asla kullanmamalıyız!

Eğitim ve Test Verisi

- Temsil Edicilik: Eğitim ve test setleri, veri setinin genel yapısını yansıtmalıdır.
  - Tüm sınıflar her iki sette de yer almalı
  - Örn: Sadece kadınlardan oluşan bir test seti başarınızı yanıltabilir
- Dengesizlik: Eğer sınıflar dengesizse (`imbalanced data`), rastgele bölme yanlış sonuçlar doğurabilir.
  - `train_test_split(..., stratify=y)` kullanılmalı
- Zaman Bağlılığı: Zaman serilerinde önceki veriler eğitim, sonraki veriler test için ayrılmalı (rastgele bölünmemeli)
- Veri Sızıntısı (Data Leakage): Test verisinden eğitim setine bilgi geçişi varsa model başarısı yapay olarak yükselir
  - Örn: test verisindeki ortalamayı eğitimde kullanmak

`stratify=y` Neden Önemlidir?

- Hedef değişkenin sınıf dağılımı bozulmadan eğitim ve test setine bölünmesini sağlar.

```
y = [0]*90 + [1]*10  # %90 sinid 0, %10 sinif 1

# Stratify OLMADAN
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2)

# Sonuç: Test setinde sadece sinif 0 olabilir → yanlis
    degerlendirme!

# Stratify ILE
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)
```

Eğitim ve Test Seti Oluşturma

```python
from sklearn.model_selection import train_test_split

# Özellikler (X) ve etiket (y)
X = df.drop("hedef", axis=1)
y = df["hedef"]

# Egitim ve test bolme (%80 egitim, %20 test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

- `train_test_split()`: scikit-learn içinden gelir.
- `random_state`: Aynı bölmeyi tekrar elde edebilmek için kullanılır.
- Bu işlem sonrası model sadece eğitim verisiyle eğitilir.

Model Eğitimi ve Tahmin Süreci

- Bir model, eğitim verisinden örüntüleri öğrenir:
  `model.fit(X_train, y_train)`
- Öğrendikten sonra, test verisinde tahmin yapabilir:
  `model.predict(X_test)`
- Bu süreçte modelin yaptığı şey:
  - X (özellikler) ile y (etiket) arasındaki ilişkiyi bulmak
  - Gelecekte karşılaşacağı benzer X'lere karşılık gelen y'yi tahmin etmeye çalışmak
- Farklı modeller farklı öğrenme yollarına sahiptir (örnek: doğrusal, mesafe tabanlı, ağaç yapılı).
- İlk denemelerde genellikle basit bir modelle başlanır (örneğin: Lojistik Regresyon).

Not: Tahmin edilen sonuçlar, gerçek etiketlerle kıyaslanarak değerlendirilir.

Model Eğitimi ve Tahmini

```python
from sklearn.linear_model import LogisticRegression
# Modeli olustur
model = LogisticRegression()
# Egit (fit)
model.fit(X_train, y_train)
# Tahmin et (predict)
y_pred = model.predict(X_test)
```

- `fit()`: Model, eğitim verisinden öğrenme işlemini yapar.
- `predict()`: Öğrenilen yapıya göre test verileri üzerinde tahmin üretir.
- `y_pred`: Modelin tahmin ettiği sınıfları içerir.

Model Değerlendirme Metrikleri

- Bir modelin başarısı sadece doğru tahmin sayısıyla değil, tüm sınıflar üzerindeki performansla değerlendirilmelidir.
- Temel metrikler:
  - Accuracy (Doğruluk):

    Doğru tahmin sayısı / Toplam tahmin sayısı
  - Precision (Kesinlik):

    Pozitif tahminlerin ne kadarı gerçekten pozitif?
  - Recall (Duyarlılık):

    Gerçek pozitiflerin ne kadarı tahmin edilebildi?
  - F1 Skoru:

    Precision ve Recall'un dengeli ortalaması
  - Confusion Matrix:

    Gerçek vs. tahmin edilen sınıfların sayısal tablosu
- Hangi metriğin önemli olduğu, probleme göre değişir.

Precision ve Recall?

Örnek 1 – Spam E-Posta Filtresi

- FP (hatalı spam) → Önemli mail çöpe gitti
- FN (gerçek spam kaçtı) → Sadece rahatsız eder
- Precision önemli → ”Spam dediklerimin gerçekten spam olmasını isterim”

Örnek 2 – Kanser Taraması

- FN (kanseri bulamamak) → Hayati risktir!
- FP (yanlış pozitif) → Ek test yapılır, daha az sorun
- Recall önemli → ”Kanseri olanları kaçırmamalıyım”

- Precision: Yanlış alarm istemiyorsan
- Recall: Kaçırmak istemiyorsan

F1 Skoru

Örnek – Sahtekarlık Tespiti (Fraud Detection)

- Verilerin %99'u normal, %1'i sahte
- Model hep "normal" dese bile accuracy = %99
- Ama sahtekarlığı yakalayamaz → recall = 0
- Precision da düşük olabilir
- F1 score bu dengesizlikte daha anlamlı sonuç verir.

- Dengesiz veri + kritik sınıf → F1 Score kullanılır

Precision – Recall – F1 Karşılaştırması

Senaryo	TP	FP	FN	Precision	Recall
A – Dengeli	80	20	20	0.80	0.80
B – Yüksek FP	80	40	20	0.67	0.80
C – Yüksek FN	80	20	40	0.80	0.67

F1 Skorları

- A: 0.80
- B: 0.73
- C: 0.73

- Precision ya da Recall düşerse F1 de düşer
- F1 Score, dengesiz senaryolarda genel dengeyi yansıtır

Model Değerlendirme

```python
from sklearn.metrics import
    accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix, classification_report

print("Accuracy:", accuracy_score(y_test, y_pred))

print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Confusion Matrix

	Tahmin: Pozitif	Tahmin: Negatif
Gerçek: Pozitif	TP (True Positive)	FN (False Negative)
Gerçek: Negatif	FP (False Positive)	TN (True Negative)

- TP: Model doğru şekilde “ pozitif dedi.
- FP: Model yanlışlıkla pozitif dedi (tip 1 hata).
- FN: Gerçek pozitifti ama model bilemedi (tip 2 hata).
- TN: Model doğru şekilde negatif dedi.

İpuçları:

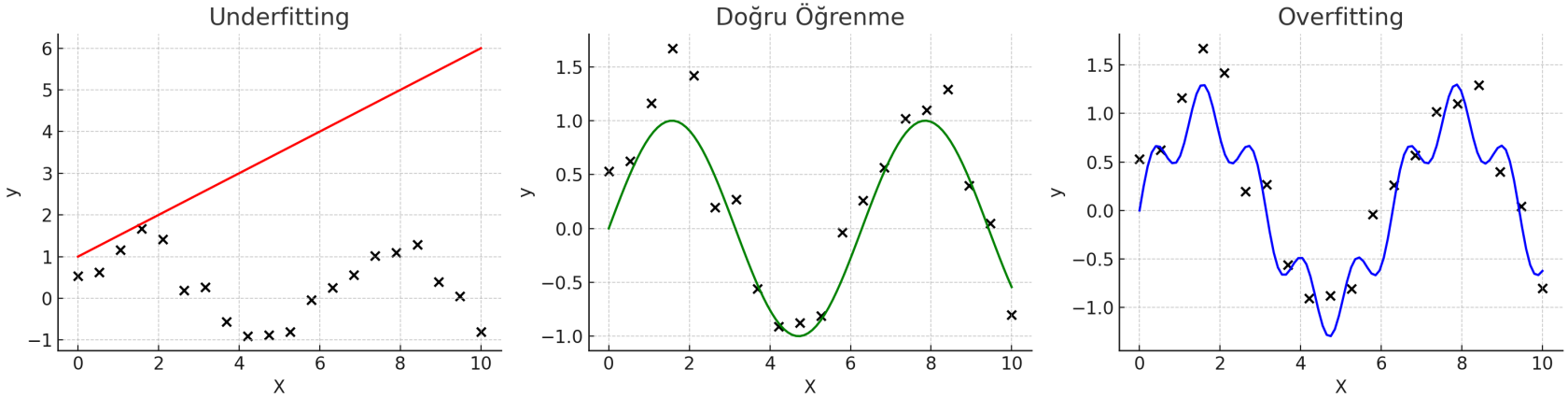
- FN → tıbbi teşhis gibi kritik konularda çok risklidir.
- FP → spam filtrelemede kullanıcıyı rahatsız edebilir.

Aşırı ve Yetersiz Öğrenme

- Makine öğrenmesinde amaç: Eğitilen modelin sadece eğitim verisinde değil, genel veri üzerinde de başarılı olmasıdır.
- Overfitting (Aşırı Öğrenme)
  - Model, eğitim verisini ezberler; karmaşık kararlar alır.
  - Test verisinde başarısı düşer — genelleme yapamaz.
  - Nedenleri: çok karmaşık model, az veri, gürültülü veri
- Underfitting (Yetersiz Öğrenme)
  - Model, eğitim verisindeki temel örüntüleri bile öğrenemez.
  - Hem eğitimde hem testte düşük başarı verir.
  - Nedenleri: çok basit model, eksik özellikler, yetersiz eğitim
- Amaç: Bu iki uç arasında dengeli bir model oluşturmak.

Not: Karmaşık modeller genelde daha fazla regularizasyon ve veri ister.

Model Performansını Artırma



- Başarısız model performansının olası nedenleri:
  - Aşırı veya yetersiz öğrenme (underfitting/overfitting)
  - Yanlış model seçimi (örnek: doğrusal model ile karmaşık ilişki)
  - Eksik/önemsiz özellikler (feature selection önemli!)
  - Yetersiz veri veya dengesiz sınıflar
- İyileştirme yolları:
  - Hiperparametre ayarlarını test etme (grid search, random search)
  - Daha fazla veya dengeli veri toplama
  - Özellik mühendisliği (feature engineering)
  - Alternatif modeller deneme (baseline modelden karmaşığa)

Model Başarısında Veri Temsili

- Güçlü algoritmalar bile yanlış temsil edilmiş veriyle başarısız olabilir!
- Veri Temsili = Özellik Mühendisliği → Modelin anlayacağı şekilde veriyi dönüştürmek
- Yaş: `age = 42` (sayısal değer — doğru)
  `age = "orta"` (kategorik metin — model için anlamsız)
- Cinsiyet: `one-hot: [1, 0]` (Kadın)
  `Erkek = 1, Kadın = 2` → Model $2 > 1$ sanabilir!
- Renk: `one-hot: kırmızı=[1,0,0], mavi=[0,1,0], yeşil=[0,0,1]`
  `kırmızı=1, mavi=2, yeşil=3` → Model "yeşil > kırmızı" gibi anlam çıkarabilir!
- Tarih: `gun = 1` (Pzt), `gun = 7` (Paz) → haftalık trend analizi için işe yarar
  `tatil = True` özelliği eklenerek daha anlamlı hale getirilebilir

Makine Öğrenmesinde Döngüsel Süreç

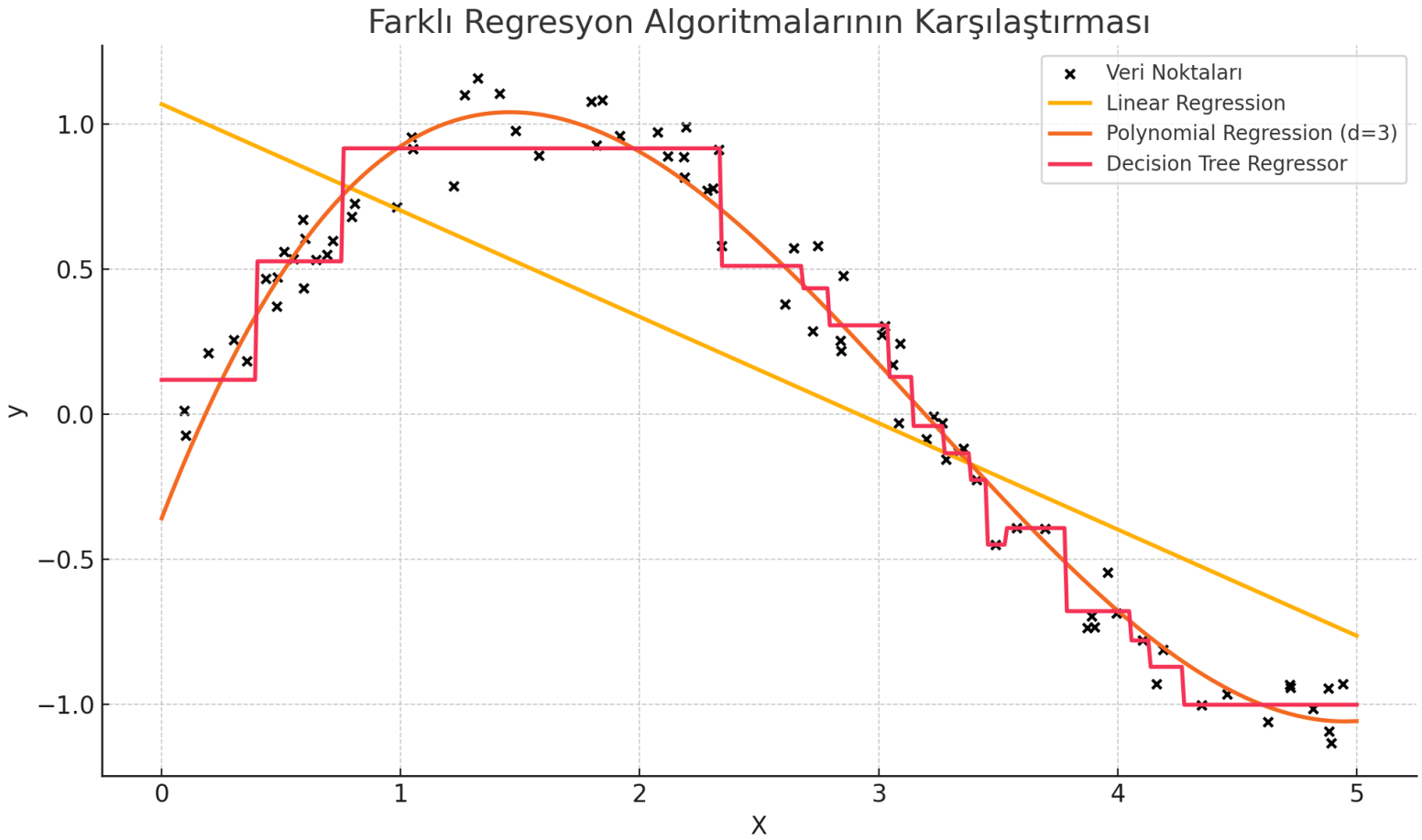
- Veri hazırlanır (temizlenir, dönüştürülür)
- → Model eğitilir (fit)
- → Tahmin yapılır (predict)
- → Değerlendirme (doğruluk, F1, confusion matrix)
- → Geri dön: iyileştirme için tekrar veri/model seçimi

Bu süreç bir defa değil, defalarca tekrarlanır.

- Regresyon: Sürekli değer tahmini (örneğin maaş, ev fiyatı)
- Sınıflandırma: Veriyi kategorilere ayırma (örneğin e-posta spam mi?)
- Kümeleme: Benzer verileri gruplayarak örüntü keşfi (örneğin müşteri segmentasyonu)

Her algoritmanın amacı farklıdır: Verinin türüne ve hedefe göre doğru yöntem seçilmelidir.

- Linear Regression En basit regresyon modeli Doğrusal ilişki:
  $y = ax + b$
  Örnek: Evin m^2'sine göre fiyat tahmini
- Polynomial Regression Doğrusal olmayan ilişkilerde tercih edilir
  $y = ax^2 + bx + c$ gibi ifadeler içerir
- Decision Tree Regressor Karar ağacı yapısıyla çalışır Karmaşık veri ilişkilerini öğrenebilir


Farklı Regresyon Algoritmalarının Karşılaştırması
Veri Noktaları
Linear Regression
Polynomial Regression (d=3)
Decision Tree Regressor
y
X
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5

Lojistik Regresyon (Logistic Regression)

- İkili sınıflandırma için kullanılır (örn: Hasta mı? Değil mi?)
- Girdi değerlerine göre sınıf olasılığı tahmin eder (0–1 arası)
- Sonuçları sigmoid fonksiyonla sınırlar:

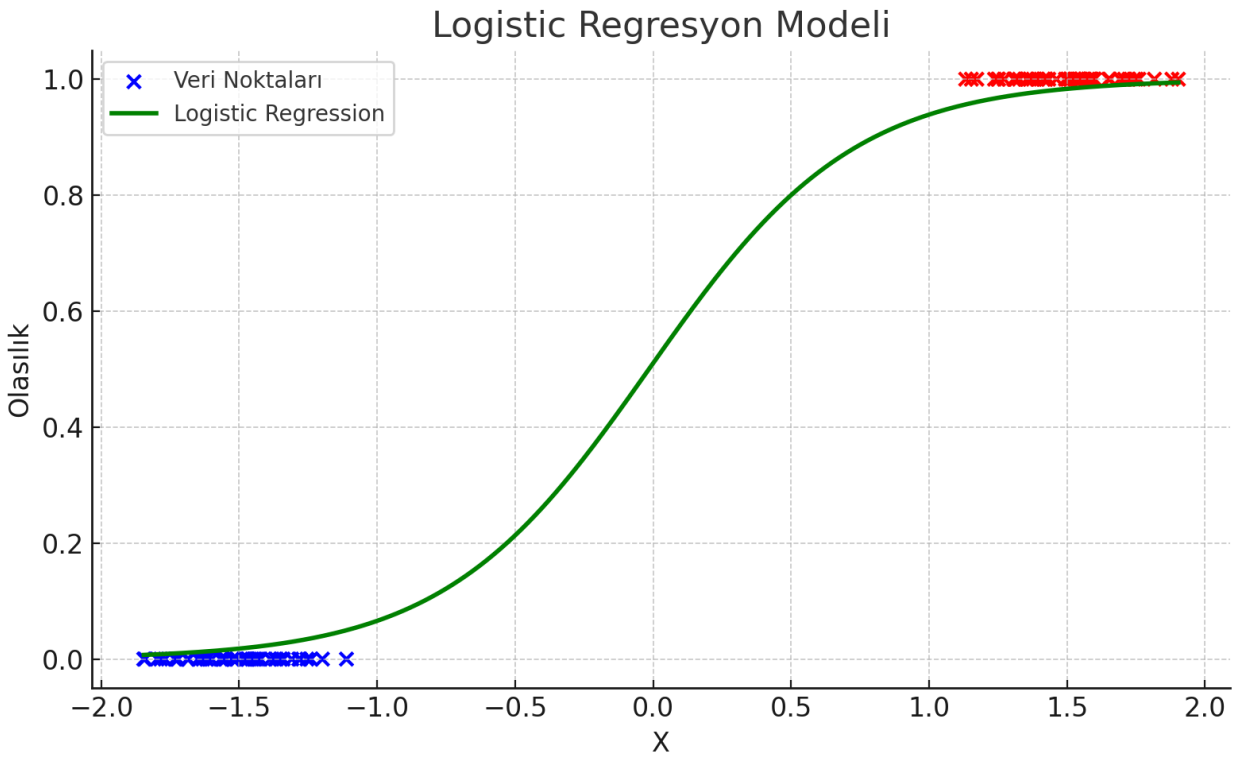
$$\hat{y} = \frac{1}{1 + e^{-(wX+b)}}$$

Veriyi doğrusal bir sınırla ayırmaya çalışır. Bu sınıra olan uzaklığa göre "Evet/Hayır" tahmini yapar.

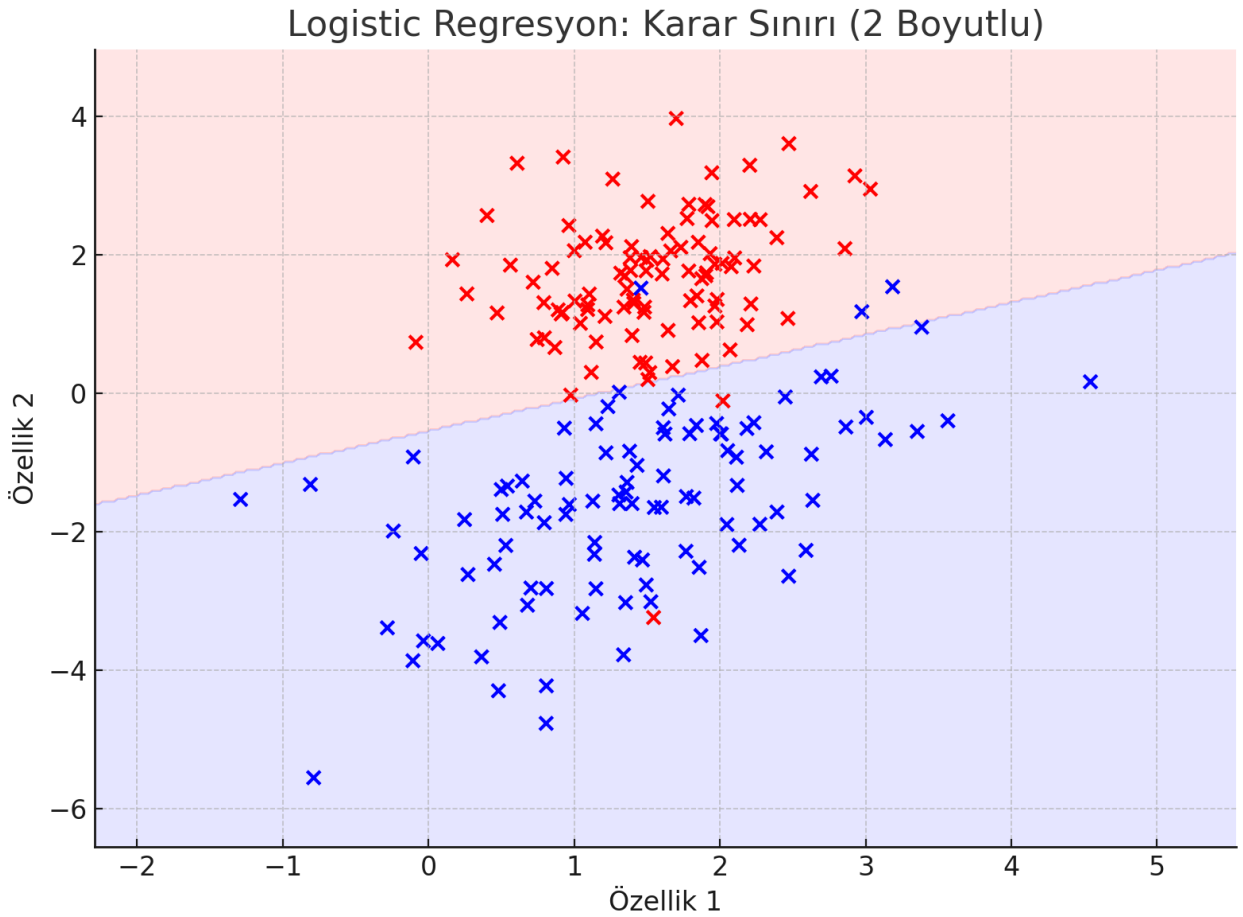
Örnek: Yaş ve tansiyona göre kalp krizi riski (Evet/Hayır)

Artıları: Hızlı, yorumlanabilir, düşük boyutlu veriler için uygundur
Eksileri: Karmaşık sınıfları ayıramaz, doğrusal sınırlara bağımlıdır

Lojistik Regresyon



Lojistik Regresyon



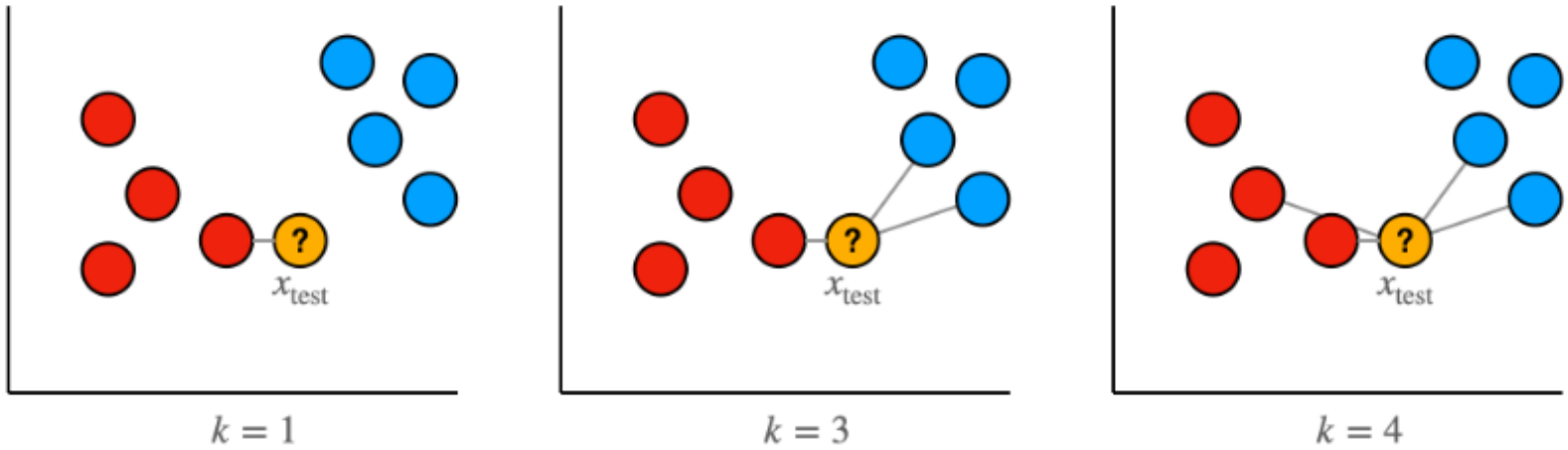
K-En Yakın Komşu (KNN)

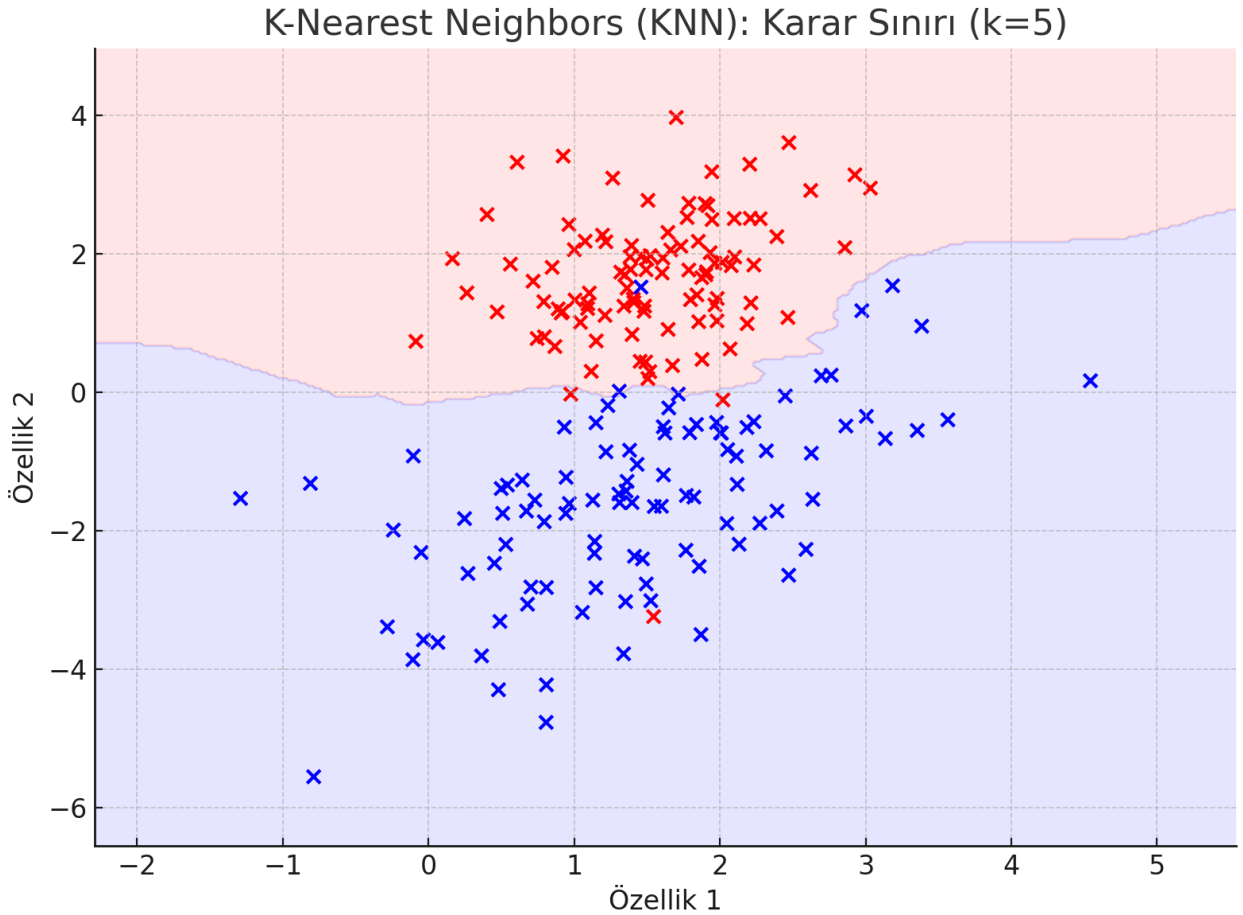
- Yeni gelen verinin sınıfını, en yakın K veri noktasına bakarak belirler
- Karar, komşuların çoğunluğuna göre verilir (oylama mantığı)
- Modelin eğitimi yoktur — hesaplama tahmin anında yapılır

Veri uzayında en yakın K nokta bulunur → Hangi sınıftan daha çok varsa, o sınıf tahmin edilir.

Örnek: Boy ve kilo bilgisiyle cinsiyet tahmini

Artıları: Basit, doğrusal olmayan yapıları da öğrenebilir
Eksileri: Büyük veri setlerinde yavaş, özelliklerin ölçeklenmesi gerekir


?
x_{test}
$k = 1$
?
x_{test}
$k = 3$
?
x_{test}
$k = 4$


K-Nearest Neighbors (KNN): Karar Sınırı (k=5)
Özellik 1
Özellik 2
4
2
0
−2
−4
−6
−2
−1
0
1
2
3
4
5

Karar Ağacı (Decision Tree)

- Veriyi if/else kurallarıyla dallara ayırır
- En iyi ayıran özelliklerle yukarıdan aşağıya dallanır
- Her yaprak düğüm, bir sınıf kararını temsil eder

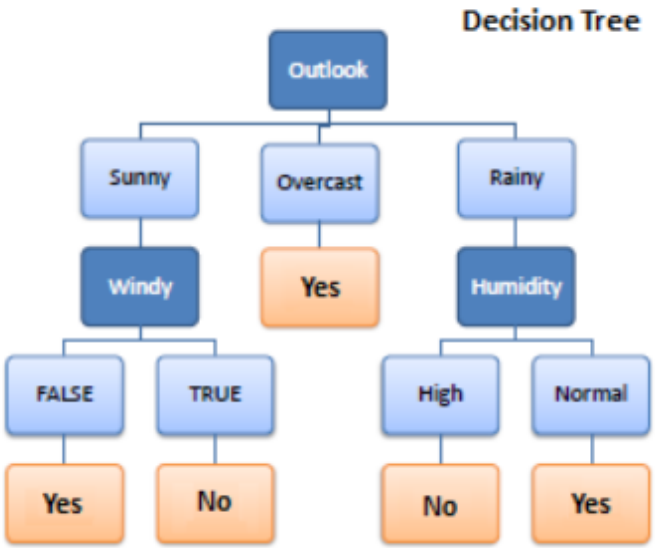
"Eğer gelir $>$ 5000 TL ise ve yaş $<$ 40 ise $\rightarrow$ onayla" gibi adım adım kurallarla karar verir.

Örnek: Gelir, yaş ve kredi skoru ile kredi başvurusu kararı

Artıları: Yorumlaması kolay, hem sayısal hem kategorik veri ile çalışır
Eksileri: Aşırı öğrenmeye açık, küçük değişikliklere duyarlı

Predictors | Target

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



Random Forest (Rastgele Orman)

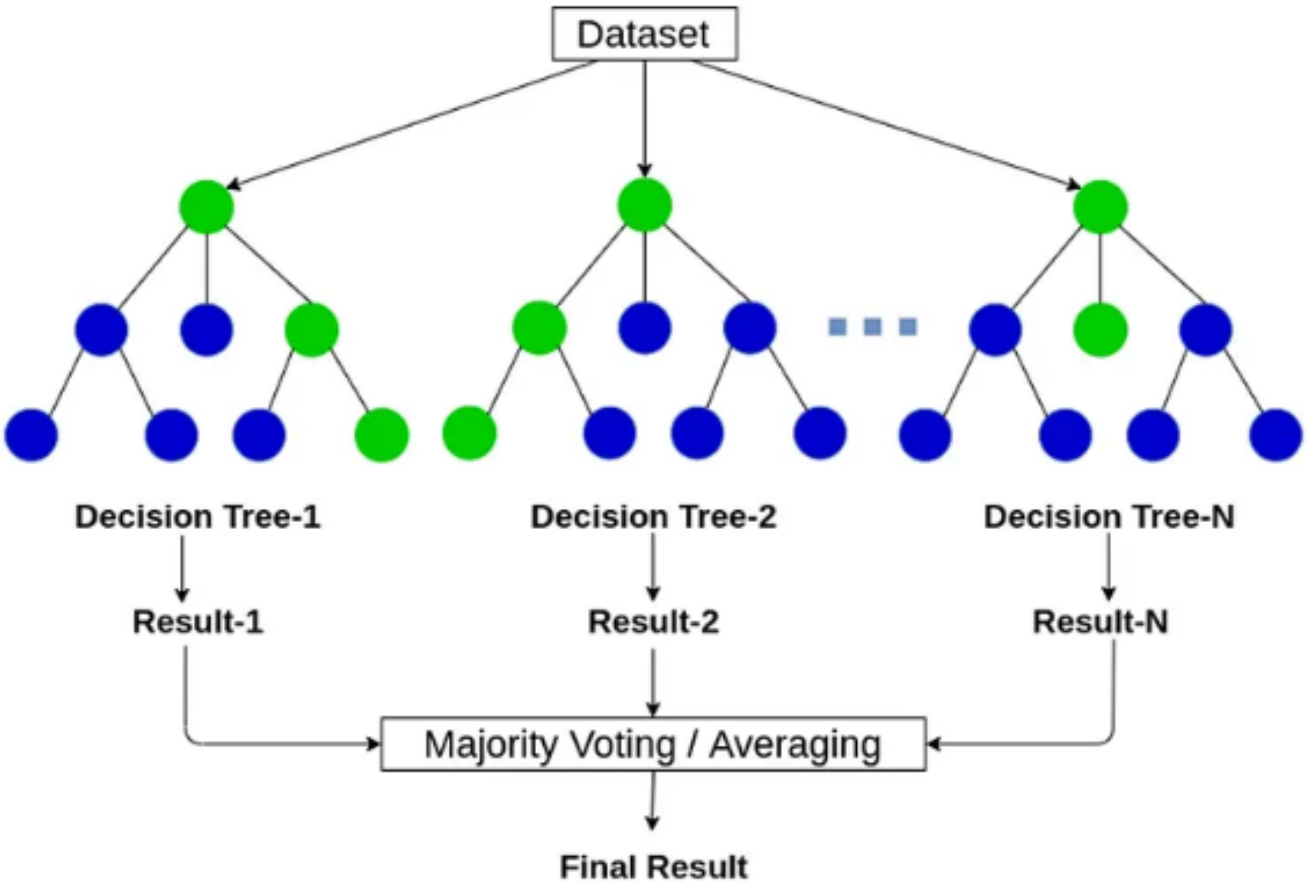
- Birçok karar ağacı eğitilir → her biri farklı veri alt kümesiyle
- Her ağaç ayrı karar verir, çoğunluk oyu ile nihai tahmin yapılır
- Bu toplu yapı aşırı öğrenmeyi azaltır

Her ağaç biraz yanılıyor olabilir ama birlikte daha doğru karar verirler.

Örnek: E-posta spam mi değil mi? (İçerik, başlık, uzunluk)

Artıları: Yüksek doğruluk, overfitting'e karşı dayanıklı
Eksileri: Yavaş çalışabilir, yorumlaması zordur

Random Forest



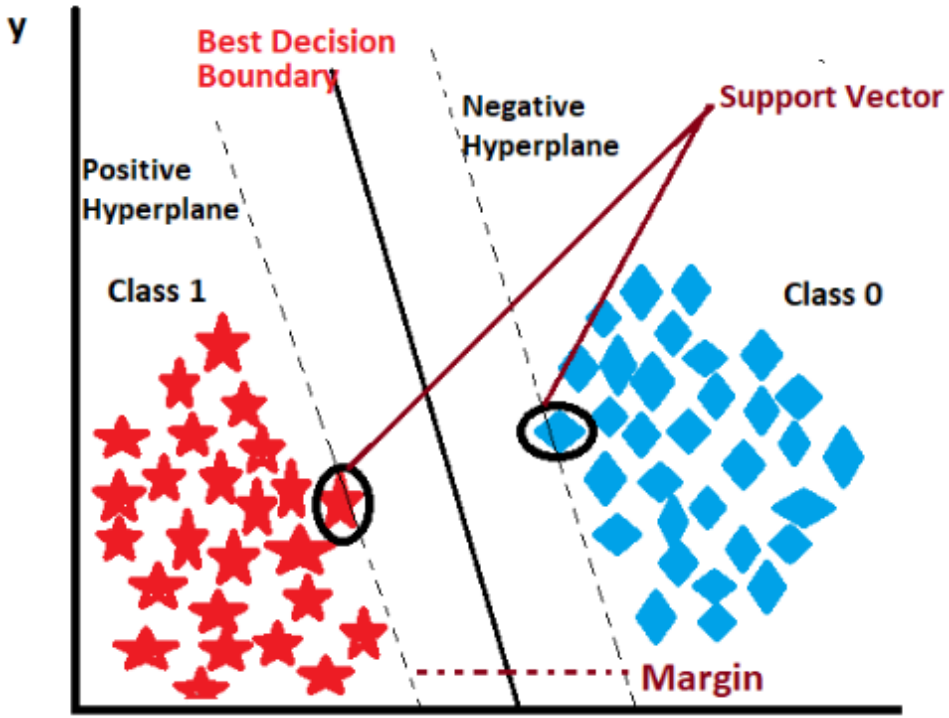
Destek Vektör Makineleri (SVM)

- Sınıfları ayıran en iyi çizgiyi/hiperdüzlemi bulur
- Amaç: sınıflar arasında maksimum boşluk bırakmak
- Karar sınırı yakınındaki örnekler (support vector) belirleyicidir
- Lineer ayrım yetmezse kernel yöntemleri kullanılır

"İki sınıfı ayıran tüm çizgiler içinde en uzakta duranı seç" — bu çizgiye en yakın birkaç nokta kararı belirler.

Örnek: Görüntüden el yazısı rakamlarını sınıflandırma

Artıları: Karmaşık sınırlarda güçlü, az veriyle de etkili
Eksileri: Eğitim süresi uzun, parametre ayarı hassas


y
Best Decision Boundary
Negative Hyperplane
Support Vector
Positive Hyperplane
Class 1
Class 0
Margin

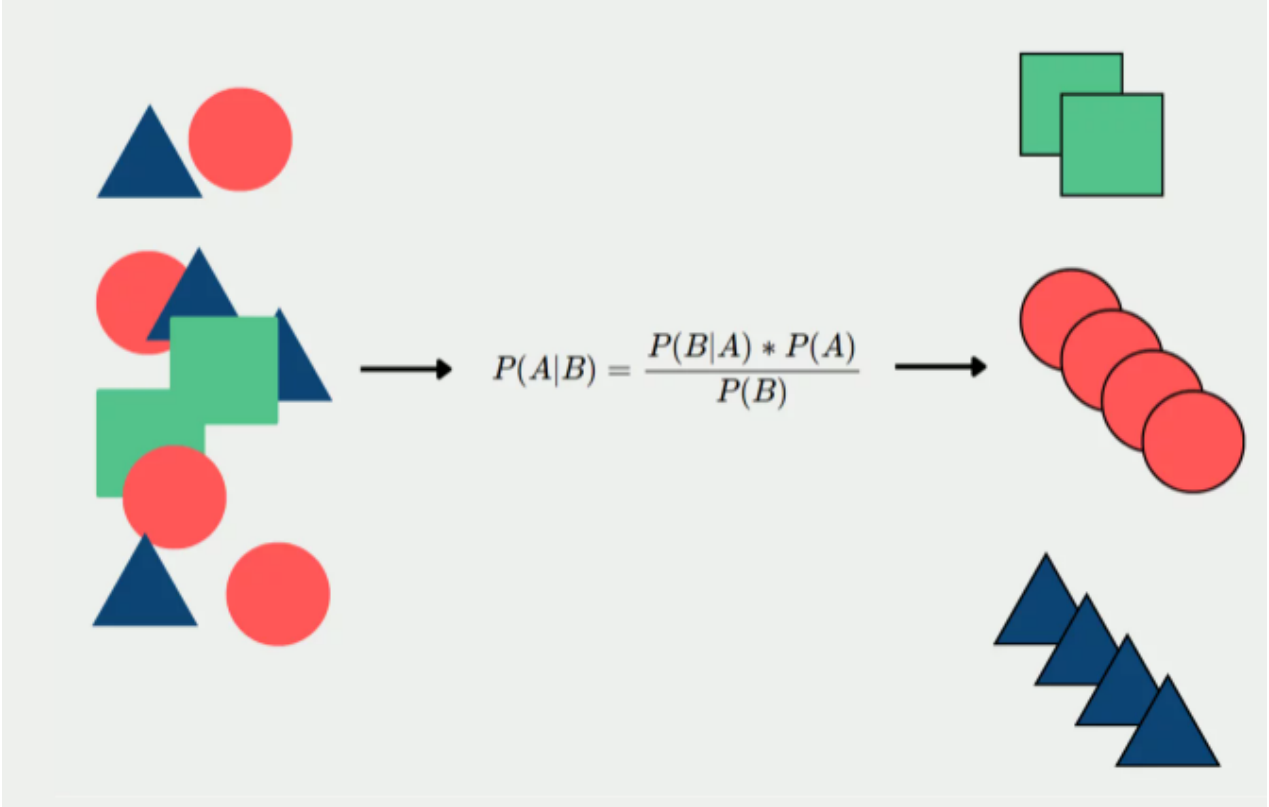
Naive Bayes

- Bayes teoremini temel alır: veriye göre olasılık hesaplar
- Her özelliğin sınıftan bağımsız olduğunu varsayar (“naive”)
- En yüksek olasılığa sahip sınıf seçilir

“Bu özelliklere sahip bir örnek, hangi sınıfa ait olma olasılığı en yüksek?”

Örnek: Bir haber metni spor, siyaset ya da ekonomi mi?

Artıları: Hızlı, metin verilerinde çok etkilidir, az veriyle de çalışır
Eksileri: Bağımlı özelliklerde performans düşer, sayısal verilerde ön işlem gerekir


$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$

Yapay Sinir Ağı (Artificial Neural Network)

- İnsan beyninden esinlenen çok katmanlı yapılar
- Giriş → Gizli Katman(lar) → Çıkış şeklinde işler
- Karmaşık örüntüleri kendi kendine öğrenebilir

Nöronlar, giriş verilerini ağırlıklarla çarpar, toplar ve aktivasyon fonksiyonu uygular. Hata, geri yayılım ile düzeltilir.

Örnek Uygulamalar: El yazısı rakam tanıma, yüz tanıma, kredi riski tahmini

Avantajlar:

- Görsel, sayısal, metin verilerinde etkili
- Çok güçlü genelleme yeteneği

Dezavantajlar:

- Çok veri ve işlem gücü ister
- "Kara kutu" gibi: Yorumlaması zordur

Kümeleme (Clustering) Nedir?

- Etiketlenmemiş verileri benzer gruplara ayırma yöntemidir
- Gözetimsiz öğrenme (unsupervised learning) türüdür
- Hedef: Benzer özelliklere sahip verileri aynı kümeye atamak

Örnek:

- Müşteri segmentasyonu (alışkanlıklara göre gruplama)
- Haberleri konu başlıklarına göre grupla (etiketsiz)

Popüler algoritmalar: K-Means, DBSCAN, Hierarchical Clustering

K-Means Kümeleme

- Belirli sayıda merkez (K) rastgele seçilir
- Her nokta en yakın merkeze atanır → kümeler oluşur
- Merkezler kümelere göre yeniden hesaplanır
- İşlem değişim kalmayana dek tekrarlanır

"Ben buna daha yakınım" diyerek verileri grupla! Her adımda merkezler güncellenir.

Örnek: Müşterileri alışveriş tutarı ve sıklığına göre segmente et

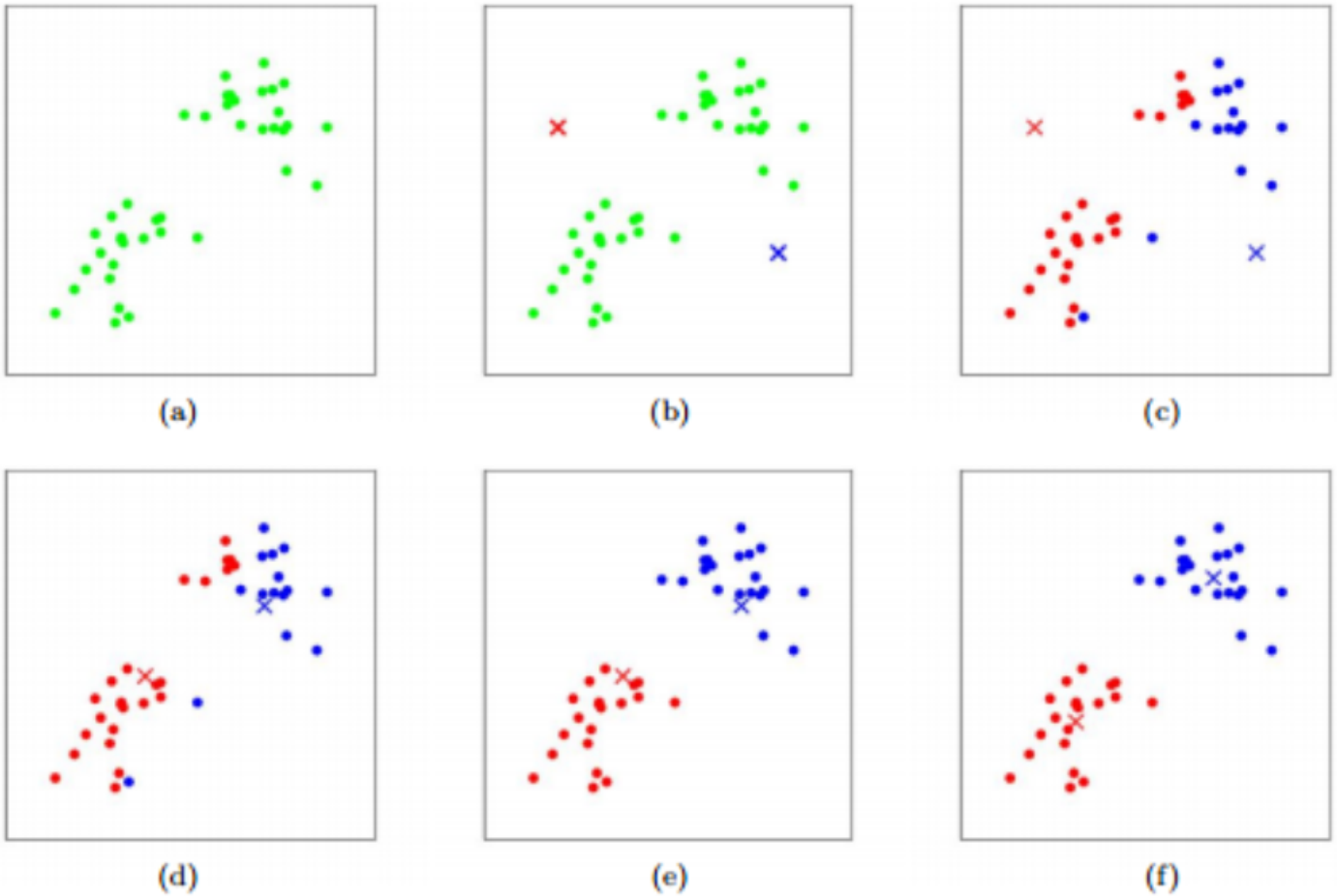
Artıları:

- Hızlı ve sezgisel

Eksileri:

- Küme sayısı önceden verilmelidir
- Karmaşık kümelerde yanıltıcı olabilir

Kmeans



DBSCAN (Density-Based Clustering)

- Yoğunluk tabanlı kümeleme: yakın ve kalabalık noktalar aynı kümeye girer
- Seyrek bölgelerdeki noktalar gürültü / dışta (outlier) sayılır

Bir noktadan başlayıp `eps` yarıçapındaki komşuları sayar; sayı `min_samples`'ı aşıyorsa küme büyütülür. Ulaşılan noktalar zincirleme eklenir; seyrekler gürültü kalır.

GPS konumlarında yoğun toplanma alanlarını (ziyaret edilen yerler) bulma

Artıları:

- Küme sayısı vermek gerekmez
- Düzensiz (daire dışı) şekillerde ve gürültülü veride etkilidir

Eksileri:

- Farklı yoğunluklu kümelerde zorlanabilir

Hiyerarşik Kümeleme (Hierarchical Clustering)

- Kümeler arası benzerlik ölçülerek küçük gruplar birleştirilir (agglomerative)
- Sonunda tüm veriler büyük bir ağaca (dendrogram) dönüşür
- İstenilen küme sayısı: ağaç yapısı kırpılarak elde edilir

İlk başta her veri bir kümedir → en yakın ikili birleşir → bu işlem tüm veriler birleşene kadar sürer

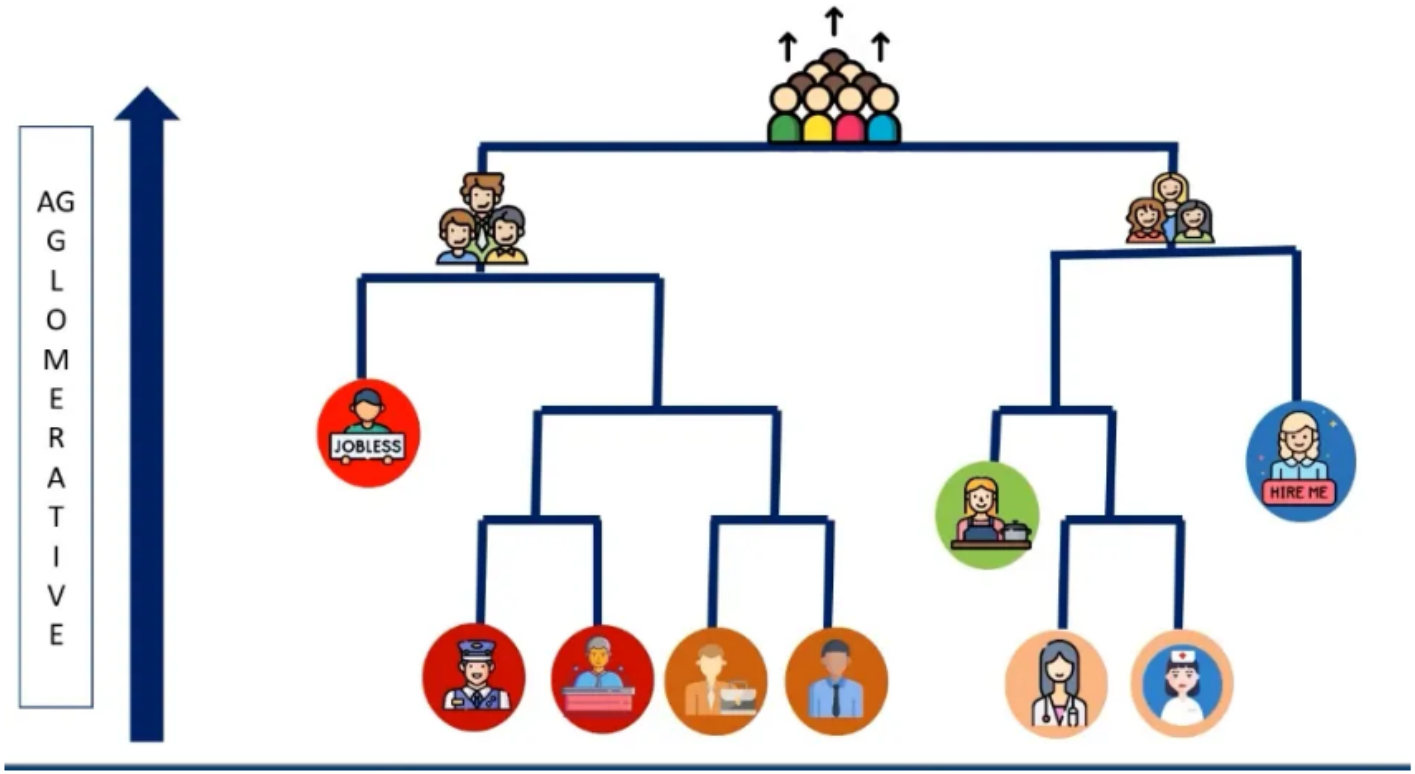
Örnek: Genetik benzerliğe göre bireyleri sınıflandırmak (soy ağacı benzeri yapı)
Artıları:

- Küme sayısını baştan belirtmeye gerek yoktur
- Dendrogram ile sezgisel analiz yapılabilir

Eksileri:

- Büyük veri setlerinde hesaplama maliyeti yüksektir
- Veri noktaları yanlış birleşirse düzeltilemez (geri dönüşsüzdür)

Hiyerarşik Kümeleme



Diabetes Veri Seti

- Kaynak: `sklearn.datasets.load_diabetes()`
- Örnek sayısı: 442 hasta
- Özellik sayısı: 10 sayısal özellik
- Hedef (target): Hastanın 1 yıl sonraki hastalık ilerleme ölçüsü (sürekli bir değer)
- Amaç: Hastaya ait ölçümlere bakarak hastalığın ne kadar ilerleyeceğini tahmin etmek

- age: yaş (normalize edilmiş)
- sex: cinsiyet (normalize edilmiş)
- bmi: vücut kitle indeksi
- bp: kan basıncı
- s1-s6: çeşitli kan testleri sonuçları

Problem türü: Regresyon problemi — Sürekli bir sayısal değer tahmin edilecek.

Veri Seti: Diyabet Hastaları

```python
from sklearn.datasets import load_diabetes

diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

print(X.shape)  # (442, 10)
print(diabetes.feature_names)
```

- 442 hasta, 10 özellik
- Amaç: hastalık ilerlemesini tahmin etmek
- Özellikler: yaş, BMI, kan basıncı, vb.

Özellik – Hedef İlişkisi

```
1 import matplotlib.pyplot as plt
2
3 plt.scatter(X[:, 2], y)
4 plt.xlabel("BMI")
5 plt.ylabel("Hedef (ıhastalk şilerleyii)")
6 plt.title("BMI vs. ıHastalk")
7 plt.show()
```

- Bazı özelliklerle hedef değişken arasında ilişki gözlenebilir.

Model Eğitimi

```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.model_selection import train_test_split
3
4 X_train, X_test, y_train, y_test = train_test_split(
5     X, y, test_size=0.2, random_state=42)
6
7 model = LinearRegression()
8 model.fit(X_train, y_train)
```

- Veri eğitim (%80) ve test (%20) olarak bölündü
- Model eğitildi

Model Başarısı

```
from sklearn.metrics import mean_squared_error,
    mean_absolute_error, r2_score
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R^2 Skoru:", r2)
```

- R2 skoru, modelin doğruluğunu gösterir (1'e yakınsa iyi)
- Gerçek ve tahmin değerleri karşılaştırılarak grafik de çizilebilir

Regresyon Değerlendirme Ölçütleri

- Mean Squared Error (MSE): Tahmin hatalarının karesinin ortalaması
  - Hataları büyütür, uç değerlere duyarlıdır
- Mean Absolute Error (MAE): Hataların mutlak değerlerinin ortalaması
  - Yorumu kolay, uç değerlere daha az duyarlı
- R^2 Skoru (Determinasyon Katsayısı): Modelin açıklayabildiği değişkenlik oranı (0–1)
  - 1'e yaklaştıkça model daha başarılı

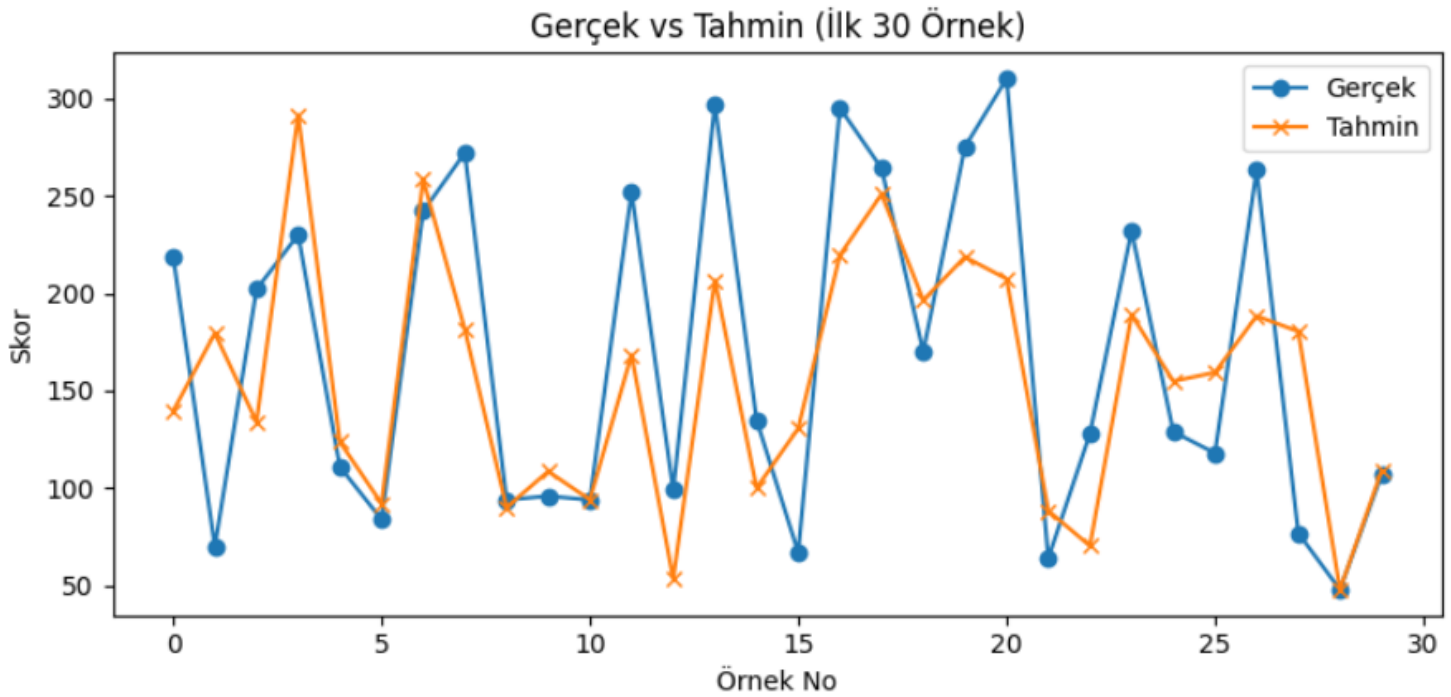
Not: Küçük MSE ve MAE $\rightarrow$ daha iyi tahmin, Yüksek R^2 $\rightarrow$ güçlü ilişki

Gerçek ve Tahmin Karşılaştırması

```
1 plt.plot(y_test, label="Gerçek")
2 plt.plot(y_pred, label="Tahmin")
3 plt.legend()
4 plt.title("Gerçek vs. Tahmin")
5 plt.show()
```

- Gerçek değer ile tahmin arasındaki fark görsel olarak incelenir

Sonuç


Gerçek vs Tahmin (İlk 30 Örnek)
Gerçek
Tahmin
Skor
Örnek No
300
250
200
150
100
50
0
5
10
15
20
25
30

Iris Veri Seti ile Kümeleme

```
1 from sklearn.datasets import load_iris
2
3 iris = load_iris()
4 X = iris.data
5 y_true = iris.target
6 feature_names = iris.feature_names
```

- 3 sınıflı, 4 özellikli etiketlenmiş veri
- `X`: Kümeleme algoritmalarına giriş
- `y_true`: Değerlendirme için gerçek sınıf etiketleri

KMeans ile Kümeleme

```
1 from sklearn.cluster import KMeans
2
3 kmeans = KMeans(n_clusters=3, random_state=0)
4 labels_kmeans = kmeans.fit_predict(X)
```

- Küme sayısı kullanıcı tarafından belirlenir (`n_clusters=3`)
- Küme merkezleri iteratif olarak güncellenir
- Sonuç: Her örnek için bir küme etiketi (`labels_kmeans`)

DBSCAN ile Kümeleme

```python
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.6, min_samples=4)
labels_dbscan = dbscan.fit_predict(X)
```

- Yoğunluk temelli: Küme sayısı önceden verilmez
- `eps` ve `min_samples` parametreleri kritik
- Gürültülü (aykırı) noktalar `-1` olarak etiketlenir

Hiyerarşik Kümeleme (Agglomerative)

```
1 from sklearn.cluster import AgglomerativeClustering
2
3 agg = AgglomerativeClustering(n_clusters=3)
4 labels_agg = agg.fit_predict(X)
```

- Küçük kümeleri birleştirerek yukarıdan aşağıya ilerler
- Küme sayısı önceden belirlenmiştir

PCA ile Görselleştirme

```
1 from sklearn.decomposition import PCA
2
3 pca = PCA(n_components=2)
4 X_pca = pca.fit_transform(X)
```

- 4 boyutlu veriler, 2 boyuta indirgenir
- Amaç: Kümeleme sonuçlarını görsel olarak karşılaştırmak

Kümeleme Sonuçlarının Görselleştirilmesi

```
1 def plot_clusters(title, labels):
2     plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap="
    Set2", s=50)
3     plt.title(title)
4     plt.show()
5
6 plot_clusters("KMeans", labels_kmeans)
7 plot_clusters("DBSCAN", labels_dbscan)
8 plot_clusters("Agglomerative", labels_agg)
```

- PCA bileşenleri (2D) üzerinde kümeler çizilir
- Renkler kümelere göre atanır

Yorum ve Uygunluk

- KMeans: Küme sayısı biliniyorsa başarılı
- DBSCAN: Gürültülü ve dağınık verilerde avantajlı
- Agglomerative: Küme yapısı belirsizken iyi çalışabilir
- Kümeleme algoritması, veri yapısına göre seçilmeli

Kümeleme Sonuçlarını Etiketlerle Kıyaslamak

- Iris veri seti sınıflandırma amaçlıdır; gerçek etiketler mevcuttur.
- Kümeleme çıktıları, bu gerçek etiketlerle doğrudan birebir eşleşmeyebilir:
  - Gerçek etiket: `[0, 1, 2, ...]`
  - Kümeleme çıktısı: `[2, 0, 1, ...]` olabilir
- Bu durumda kümeleri en iyi şekilde etiketlere eşlemek gerekir.

Amaç: Confusion Matrix ile değerlendirme yapmak ve doğruluğu ölçmek.

Veri Seti: Breast Cancer (Meme Kanseri)

- Amaç: Tümörün iyi huylu (benign) ya da kötü huylu (malignant) olup olmadığını tahmin etmek
- Özellik sayısı: 30 (örnek: hücre yoğunluğu, boyutu vs.)
- Sınıf sayısı: 2 (benign, malignant)
- Kullanılan kütüphane: `sklearn.datasets.load_breast_cancer`

Kullanılan Sınıflandırma Algoritmaları

- Logistic Regression – Basit, doğrusal model
- Karar Ağacı (Decision Tree) – Karar kurallarıyla ayrım
- Random Forest – Birden fazla ağaçtan ortalama
- Support Vector Machine (SVM) – Sınıfları ayıran çizgi/hiperdüzlem
- K-Nearest Neighbors (KNN) – Komşuluklara göre karar

Veriler eğitim/test olarak bölünür ve `StandardScaler` ile ölçeklenir.

Kütüphanelerin Yüklenmesi

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5
6 from sklearn.datasets import load_breast_cancer
7 from sklearn.model_selection import train_test_split
8 from sklearn.preprocessing import StandardScaler
9 from sklearn.metrics import accuracy_score,
      confusion_matrix, ConfusionMatrixDisplay
```

```
1 from sklearn.linear_model import LogisticRegression
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.svm import SVC
5 from sklearn.neighbors import KNeighborsClassifier
```

Veri Seti ve Hazırlık

```
1
2  data = load_breast_cancer()
3  X = data.data
4  y = data.target
5  feature_names = data.feature_names
6  target_names = data.target_names
7
8
9  X_train, X_test, y_train, y_test = train_test_split(
10     X, y, test_size=0.2, random_state=42)
11
12 scaler = StandardScaler()
13 X_train_scaled = scaler.fit_transform(X_train)
14 X_test_scaled = scaler.transform(X_test)
```

Modellerin Tanımlanması

```
1
2  models = {
3      "Lojistik Regresyon": LogisticRegression(max_iter=1000)
       ,
4      "Karar Agaci": DecisionTreeClassifier(),
5      "Random Forest": RandomForestClassifier(),
6      "SVM": SVC(),
7      "K-En Yakin Komsu": KNeighborsClassifier()
8  }
```

Model Eğitimi ve Değerlendirme (1/2)

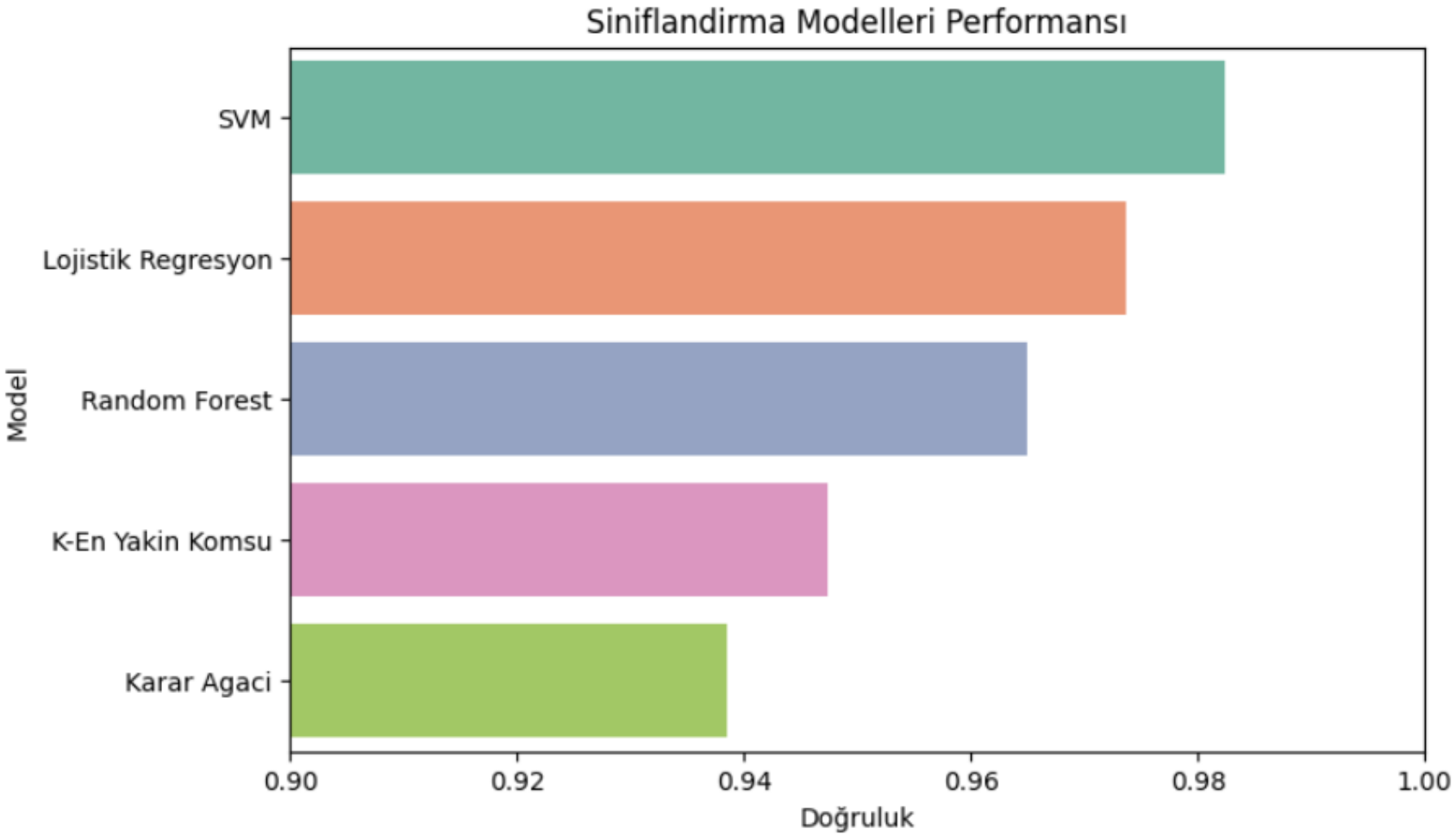
```
results = []

for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    acc = accuracy_score(y_test, y_pred)
    results.append((name, acc))
```

Model Performanslarını Karşılaştır

```
1
2  results_df = pd.DataFrame(results,
3      columns=["Model", "Dogruluk Skoru"]
4  ).sort_values(by="Dogruluk Skoru", ascending=False)
5
6  print(results_df)
7
8  plt.figure(figsize=(8,5))
9  sns.barplot(x="ğDoruluk Skoru", y="Model",
10                 data=results_df, palette="Set2")
11 plt.title("Siniflandirma Modelleri Performansi")
12 plt.xlabel("Dogruluk")
13 plt.xlim(0.9, 1.0)
14 plt.show()
```

Sonuç


Siniflandirma Modelleri Performansı
SVM
Lojistik Regresyon
Random Forest
K-En Yakin Komsu
Karar Agaci
Model
0.90
0.92
0.94
0.96
0.98
1.00
Doğruluk