

2025

# Makine Öğrenmesi Çalışma Notları

BEYZA KÜÇÜK

# İçindekiler Tablosu

## İçindekiler

	<b>1. Giriş ve Temel Kavramlar.....</b>	<b>4</b>
<b>1.1</b>	Makine Öğrenmesine Giriş.....	4
<b>1.2</b>	Makine Öğrenmesi Adımları.....	6
<b>1.3</b>	Veri Ön İşleme.....	6
•	Eksik veriler	
•	Kategorik veriler	
•	Standardizasyon / Normalizasyon	
<b>1.4</b>	Model Seçimi ve Veri Bölme (Train/Test Split, Cross-validation).....	9
<b>1.5</b>	Makine Öğreniminin uygulama Alanları.....	18
<b>1.6</b>	Makine Öğreniminin Geleceği.....	18
	<b>2. Regresyon Modelleri.....</b>	<b>18</b>
<b>2.1</b>	Doğrusal Regresyon (Linear Regression).....	19
<b>2.2</b>	Çoklu Doğrusal Regresyon (Multi Linear Regression).....	20
<b>2.3</b>	Polinom Regresyon (Polynomial Regression).....	21
•	Basit Polinom Regresyon	
•	Çoklu Polinom Regresyon	
<b>2.4</b>	Aşırı Uyum ve Yetersiz Uyum (Overfitting vs Underfitting).....	22
<b>2.5</b>	Sapma - Varyans Takası (Bias-Variance Tradeoff).....	23
<b>2.6</b>	Düzenlileştirme (Ridge, Lasso, ElasticNet, Regularization).....	24
<b>2.7</b>	Gradyan İnişi (Gradient Descent).....	24
	<b>3. Sınıflandırma Modelleri.....</b>	<b>26</b>
<b>3.1</b>	Sınıflandırmaya Giriş.....	26
<b>3.2</b>	Değerlendirme Metrikleri.....	27
•	Confusion Matrix	
•	Accuracy, Precision, Recall, F1	
•	ROC, AUC, PR Curve	
<b>3.3</b>	Ek Değerlendirme Metrikleri.....	30
<b>3.4</b>	K-En Yakın Komşu (KNN- K-Nearest Neighbors).....	35
<b>3.5</b>	Lojistik Regresyon (Logistic Regression).....	35
<b>3.6</b>	Karar Ağaçları(Decision Tree).....	36
•	Görselleştirme	
•	Overfitting - Pruning	
<b>3.7</b>	Destek Vektör Makineleri (SVM- Support Vector Machine).....	37
<b>3.8</b>	Naive Bayes .....	37
<b>3.9</b>	Rastgele Ormanlar (Random Forest) (10_Random Forest).....	38

 4. Topluluk Öğrenimi (Ensemble Learning).....	38
4.1 Topluluk Yöntemlerine Giriş (ML Overview).....	38
4.2 Bagging Yöntemi ve Random Forest.....	39
4.3 Boosting Teknikleri (Boosting Methods).....	40
• AdaBoost	
• Gradient Boosting	
• XGBoost	
4.4 Stacking (Stacking Method).....	42
4.5 Hiperparametre Ayarlama.....	42
• GridSearch	
• RandomizedSearch	
• Bayesian Optimization	
 5. Denetimsiz Öğrenme.....	44
5.1 K-Means Kümeleme (K-Means Clustering).....	44
5.2 Hiyerarşik Kümeleme (Hierarchical Clustering).....	45
5.3 Ana Bileşenler Analizi - PCA (Principal Component Analysis).....	45
5.4 Boyut İndirgeme Yöntemleri.....	46
• t-SNE	
• UMAP	
5.5 Anomali Tespiti (Isolation Forest, DBSCAN, Z-Score).....	47
 6. Zaman Serileri (Time Series).....	48
6.1 Zaman Serilerine Giriş (16_Time Series).....	48
6.2 Trend, Mevsimsellik, Durağanlık.....	48
6.3 ARIMA, SARIMA Modelleri.....	48
6.4 Prophet ile Tahminleme.....	49
6.5 Model Performansı ve Görselleştirme.....	50
 7. Uygulama ve İleri Konular.....	50
7.1 Özellik Seçimi ve Mühendisliği (Feature Engineering).....	50
7.2 Pipeline ve Otomasyon.....	52
• Scikit-learn Pipeline	
• joblib ile Model Kaydetme	
7.3 Veri Dengesizliği (SMOTE, Class Weights).....	53
7.4 Gerçek Hayat Projesi Oluşturma Adımları.....	53
7.5 Model Dağılımı.....	54
• Flask / FastAPI ile API	
• Docker, Kubernetes	
7.6 Etik, Adalet ve Yanlılık.....	55
• SHAP, LIME, Fairlearn	

## 1. Giriş ve Temel Kavramlar

### 1.1 Makine Öğrenmesine Giriş

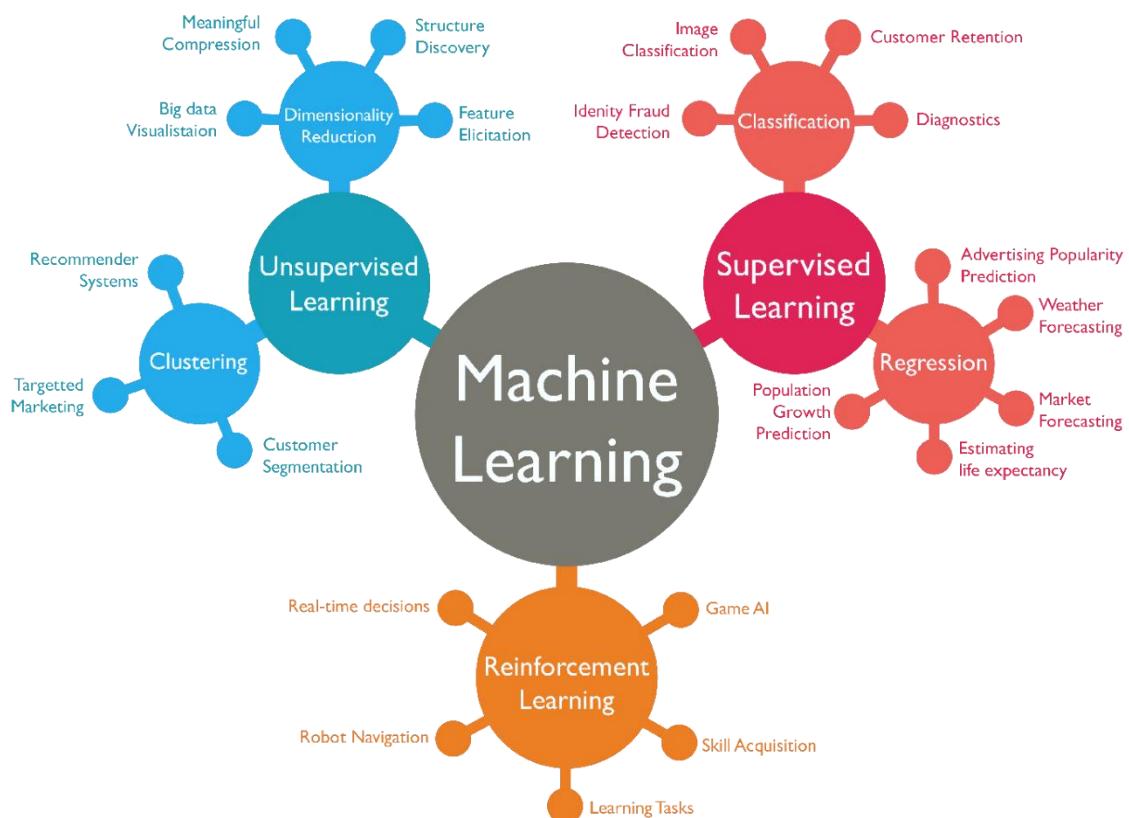
Makine Öğrenmesi (ML), bilgisayarların açıkça programlanmadan verilerden öğrenmesini sağlayan bir Yapay Zeka (AI) alt dalıdır. Geleneksel programlamada bilgisayarlar belirli görevleri yerine getirmek için açıkça kodlanırken, makine öğrenmesinde bilgisayarlar veri kümelerinden kalıpları ve ilişkileri öğrenir, bu öğrenmeyi gelecekteki veriler hakkında tahminler yapmak veya kararlar almak için kullanır.

→ Örnek: Bir e-ticaret sitesinde, müşterilerin daha önce satın aldığı ürünlere göre yeni ürün önerileri gösterilmesi.

#### Makine Öğrenmesinin Temel Bileşenleri:

- **Veri (Data):** Makine öğrenimi algoritmalarının eğitildiği temel kaynaktır. Veri kümeleri, algoritmaların öğrenebileceği ve kalıpları tanımlayabileceği yapılandırılmış veya yapılandırılmamış bilgiler içerir.
- **Algoritmalar (Algorithms):** Verilerden öğrenmek ve tahminler yapmak için kullanılan matematiksel modellerdir. Farklı makine öğrenimi algoritmaları, farklı türde veriler ve görevler için tasarlanmıştır.
- **Modeller (Models):** Algoritmaların veri üzerinde eğitilmesiyle oluşturulan ve gelecekteki veriler hakkında tahminler yapmak için kullanılan yapay zeka yapılarıdır.

#### Makine Öğrenimi Türleri:



## ■ Denetimli Öğrenme (Supervised Learning)

Etiketlenmiş (labeled) veri kümeleri kullanılarak modellerin eğitildiği bir öğrenme türüdür.

⌚ **Amaç:** Algoritma, girdi verileri ile çıktı etiketleri arasındaki ilişkiyi öğrenir ve gelecekteki girdiler için çıktıları tahmin eder.

📦 **Örnek:** El yazısı rakamları içeren bir veri setinden hangi rakamın yazıldığını tanımayır öğrenmek (örneğin: MNIST veri seti).

## ■ Yarı Denetimli Öğrenme (Semi-Supervised Learning)

⌚ **Amaç:** Etiketli verilerin yetersiz olduğu durumlarda, etiketsiz verileri de kullanarak öğrenme başarımını artırmak.

🧠 **Avantajı:** Daha az etiketli veriyle yüksek performans elde etmek.

📦 **Örnek:** Etiketli birkaç tıbbi görüntü ve çok sayıda etiketsiz görüntü ile kanserli dokuların tespitı.

## ■ Denetimsiz Öğrenme (Unsupervised Learning)

Etiketlenmemiş (unlabeled) veri kümeleri kullanılarak modellerin eğitildiği bir öğrenme türüdür.

🔍 **Amaç:** Gizli desenleri veya grupları bulmak

📦 **Örnek:** Banka müşterilerini harcama alışkanlıklarına göre otomatik olarak segmentlere ayırmak

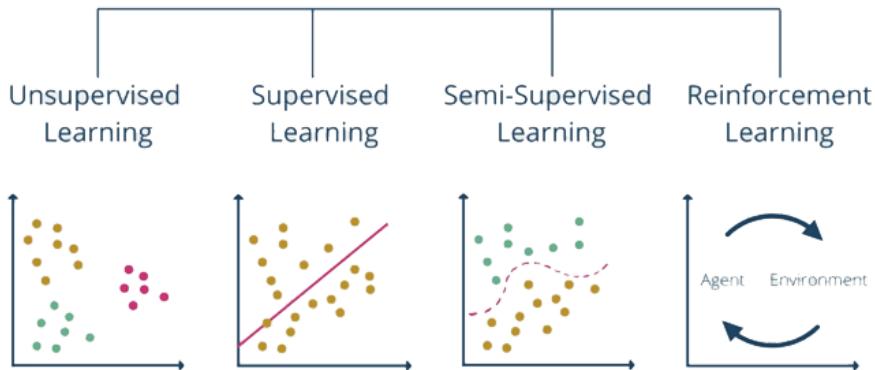
## ■ Pekiştirmeli Öğrenme (Reinforcement Learning)

Bir aracının (agent) bir ortamda etkileşimde bulunarak ve ödüller veya cezalar alarak öğrendiği bir öğrenme türüdür.

🎰 **Amaç:** Deneme-yanılma yoluyla, gelecekteki toplam ödülü maksimize eden eylemleri öğrenmektir.

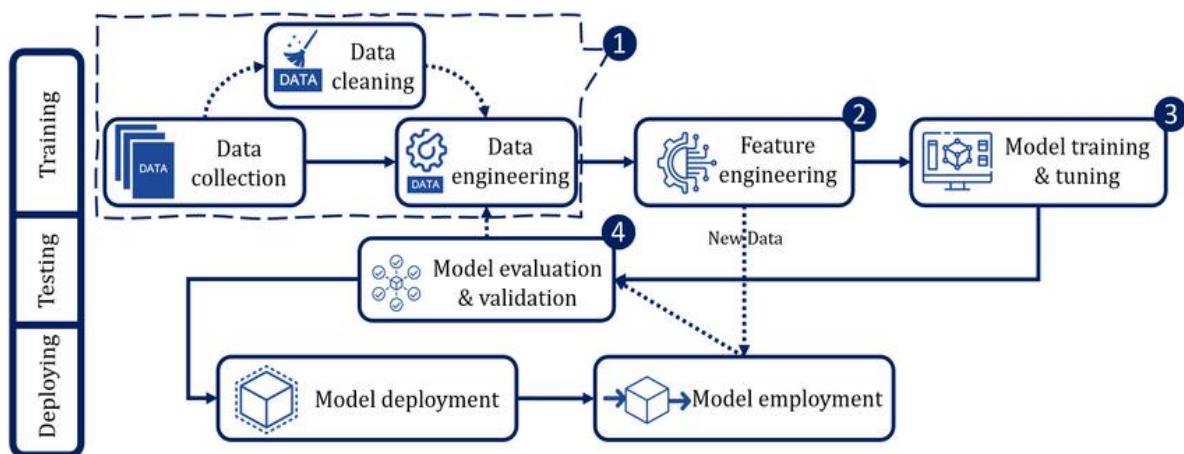
📦 **Örnek:** Bir yapay zekâ ajanının Pac-Man oyununu oynayarak kazanma stratejisini öğrenmesi

# Machine Learning



## 1.2 Makine Öğrenmesi Adımları

- **Veri Toplama:** İlgili verilerin toplanması.
- **Veri Ön İşleme:**
  - Eksik verilerin giderilmesi.
  - Kategorik verilerin sayısal verilere dönüştürülmesi.
  - Verilerin ölçeklendirilmesi (standardizasyon/normalizasyon).
- **Model Seçimi:** Probleme uygun makine öğrenimi algoritmasının seçilmesi.
- **Model Eğitimi:** Seçilen algoritmanın veri üzerinde eğitilmesi.
- **Model Değerlendirmesi:** Modelin performansının test verileri ile ölçülmesi.
- **Model Dağıtıımı:** Eğitilen modelin kullanıma sunulması.



## 1.3 Veri Ön İşleme:

- Bu alt başlık, ham verileri makine öğrenimi algoritmaları için uygun bir formata dönüştürme sürecini ele alır. Veri ön işlemenin temel adımları şunlardır:
  - **Eksik veriler:** Eksik değerler modelin doğruluğunu bozabilir, bu yüzden uygun yöntemle doldurulması gereklidir.
    - **Silme (Dropna):** Çok az sayıda eksik veri varsa ve sistematik bir eksiklik yoksa kullanılabilir.
    - **Doldurma (Imputation):** veri setlerinde eksik (boş) değerlerin olduğu durumlarda, bu boşlukları anlamlı ve makul değerlerle **doldurma işlemidir**. (Ortalama, medyan, mod ile doldurma.)

```
df.dropna(inplace=True)
```

- **Doldurma (Imputation):** veri setlerinde eksik (boş) değerlerin olduğu durumlarda, bu boşlukları anlamlı ve makul değerlerle **doldurma işlemidir**. (Ortalama, medyan, mod ile doldurma.)

```
df['yas'].fillna(df['yas'].mean(), inplace=True)
```

```
df['cinsiyet'].fillna(df['cinsiyet'].mode()[0], inplace=True)
```

- **Gelişmiş Yöntemler:** KNN imputation, regresyonla tahmin.

```
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5)
df.iloc[:, :] = imputer.fit_transform(df)
```

- **Kategorik veriler (Categorical Encoding):** Makine öğrenimi algoritmaları, yalnızca sayısal verilerle çalışır. Bu nedenle, metin/kategorik veri sayısal forma dönüştürülmelidir.

- **Label Encoding**

Her kategoriye bir sayı atanır.

```
from sklearn.preprocessing import LabelEncoder
data = {'Renk': ['Kırmızı', 'Mavi', 'Yeşil', 'Kırmızı', 'Yeşil']}
import pandas as pd
df = pd.DataFrame(data)
le = LabelEncoder()
df['Renk_encoded'] = le.fit_transform(df['Renk'])
print(df)
```

- **One-Hot Encoding**

Her kategori, ayrı bir sütun olarak açılır. O kategori varsa 1, yoksa 0.

```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(sparse=False)
renk_encoded = enc.fit_transform(df[['Renk']])
print(renk_encoded)
```

- **Ordinal Encoding**

Kategoriler **mantıksal bir sıralama içeriyorsa** (ör: 'Düşük', 'Orta', 'Yüksek'), sıralı sayılar atanır.

```
from sklearn.preprocessing import OrdinalEncoder
df = pd.DataFrame({'Seviye': ['Düşük', 'Orta', 'Yüksek', 'Düşük']})
oe = OrdinalEncoder(categories=[[['Düşük', 'Orta', 'Yüksek']]])
df['Seviye_encoded'] = oe.fit_transform(df[['Seviye']])
print(df)
```

- **Target Encoding**

Kategoriler, hedef değişkene göre ortalama değer ile kodlanır. Genelde regresyon problemlerinde kullanılır.

```
import pandas as pd

df = pd.DataFrame({
    'Meyve': ['Elma', 'Armut', 'Elma', 'Muz', 'Armut'],
    'Fiyat': [3, 4, 2.5, 5, 4.5]
})

# Meyve türüne göre ortalama fiyat
ortalamalar = df.groupby('Meyve')['Fiyat'].mean()

df['Meyve_encoded'] = df['Meyve'].map(ortalamalar)

print(df)
```

- **Standardizasyon / Normalizasyon:** Farklı ölçeklerdeki sayısal değişkenlerin nasıl standartlaştırılacağını veya normalleştirileceğini (örneğin, z-skoru normalizasyonu, min-max normalizasyonu) açıklar.

- **StandardScaler (Z-Score Normalizasyonu)**

Ortalama = 0, Standart sapma = 1 olacak şekilde dönüştürür.

```
from sklearn.preprocessing import StandardScaler

df = pd.DataFrame({'Yaş': [18, 25, 30, 35, 50], 'Gelir': [1000, 2000, 4000,
3500, 8000]})

scaler = StandardScaler()

df_scaled = scaler.fit_transform(df)

print(pd.DataFrame(df_scaled, columns=df.columns))
```

- **MinMaxScaler (0-1 Normalizasyonu)**

Veriyi 0 ile 1 arasında sıkıştırır.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df_scaled = scaler.fit_transform(df)

print(pd.DataFrame(df_scaled, columns=df.columns))
```

- **RobustScaler**

Aykırı değerlere karşı dayanıklıdır. Ortanca (median) ve IQR (çeyrekler arası mesafe) kullanır.

```
from sklearn.preprocessing import MinMaxScaler  
  
scaler = MinMaxScaler()  
  
df_scaled = scaler.fit_transform(df)  
  
print(pd.DataFrame(df_scaled, columns=df.columns))
```

 **Özet Karşılaştırma**

Yöntem	Kullanım Durumu
Label Encoding	Sırasız kategorik veriler (dikkatli ol!)
One-Hot Encoding	Her kategori için ayrı sütun gerektiğinde
Ordinal Encoding	Sıralı kategoriler (ör. düşük, orta, yüksek)
Target Encoding	Regresyon & kategorinin hedefle ilişkili olduğu durumlar
StandardScaler	Genelde önerilir, özellikle lineer modellerde
MinMaxScaler	0–1 arası veri gerektiğinde (ör. sinir ağı)
RobustScaler	Aykırı değer çoksa

 **1.4 Model Seçimi ve Veri Bölme (Train/Test Split, Cross-validation)**

**Model Seçimi:**

Makine öğreniminde doğru modeli seçmek, problemin türüne, veri setinin yapısına ve hedef değişkenin tipine bağlıdır.

 **Model Seçerken Dikkat Edilen Faktörler:**

- **Problem tipi:**
  - Sayısal tahmin → Regresyon modelleri
  - Sınıflandırma → Lojistik Regresyon, SVM, Karar Ağaçları
  - Grup bulma → Kümeleme algoritmaları (K-Means, DBSCAN)

- **Veri seti boyutu:**
  - Küçük veri → Naive Bayes, Karar Ağaçları
  - Büyük veri → Random Forest, XGBoost
- **Özellik türü:**
  - Sayısal/kategorik veriler
  - Eksik/aykırı verilerin durumu

### a) Train/Test Split

Veri seti, genellikle %70-80 eğitim (%train), %20-30 test olarak ikiye ayrılır. Eğitim verisiyle model öğrenir, test verisiyle performansı ölçülür.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

#### ◆ Neden Önemli?

Modeli **eğittiğiniz veri ile test etmek, aynı sorularla sınav yapmak** gibidir. Model "ezber yaptığında" (**overfitting**) bunu fark edemezsiniz. Test verisi modele "görülmemiş" olmalı.

### Örnek:

Bir e-ticaret sitesinin verilerini kullanarak müşterilerin alışveriş yapıp yapmayacağı tahmin edecek bir modelde, verinin %80'i eğitim, %20'si test için ayrılr.

### b) Çapraz Doğrulama (Cross-validation)

Veri setini birden fazla parçağa bölgerek her bir parça test verisi olarak kullanılır. Böylece modelin genelleme kabiliyeti ölçülür.

### Nasıl Çalışır?

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=5)
```

**1. Adım:** Veri seti **eşit parçalara** bölünür (örneğin 5 parça → "5-Fold").

## 2. Adım:

- **1. Tur:** 1 parça **test**, diğer 4'ü **eğitim** için kullanılır.
- **2. Tur:** Başka 1 parça **test**, kalanlar **eğitim** olur.
- ... Tüm parçalar sırayla test edilene kadar devam eder.  
**Sonuç:** Her turun performansı ortalaması alınır.
-  **K-Fold Cross Validation (K-Katlı Çapraz Doğrulama)**
  - Veri seti, eşit büyüklükte **K parçaya (fold)** bölünür.
  - Her seferinde bir fold test verisi olarak ayrılır, kalan K-1 fold eğitim verisi olur.
  - Bu işlem **K kez tekrarlanır**, her bir fold bir kere test seti olur.
  - Tüm test sonuçlarının ortalaması alınarak modelin genel performansı elde edilir.

### Avantajları:

- Eğitim-test ayrimına göre **daha kararlı** performans tahmini sağlar.
- Overfitting'i azaltır.
- Veri kaybı yoktur; her veri noktası en az bir kez testte kullanılır.

### Örnek:

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.array([[1], [2], [3], [4], [5]])
y = np.array([1, 2, 1.3, 3.75, 2.25])

kf = KFold(n_splits=5)
model = LinearRegression()
scores = cross_val_score(model, X, y, cv=kf)

print(scores)
print("Ortalama skor:", scores.mean())
```

```
Fold 1: [Test] 1 | [Train] 2 3 4 5  
Fold 2: [Test] 2 | [Train] 1 3 4 5  
Fold 3: [Test] 3 | [Train] 1 2 4 5  
Fold 4: [Test] 4 | [Train] 1 2 3 5  
Fold 5: [Test] 5 | [Train] 1 2 3 4
```

-  **Leave-One-Out Cross Validation (LOOCV)**

 **Tanım:**

Veri setinde her seferde yalnızca 1 örnek test, geri kalan tüm örnekler eğitim olarak kullanılır.

- Bu işlem, veri sayısı N olan bir sette N kez tekrarlanır.

```
from sklearn.model_selection import LeaveOneOut  
  
from sklearn.linear_model import LinearRegression  
  
from sklearn.metrics import mean_squared_error  
  
import numpy as np  
  
  
X = np.array([[1], [2], [3], [4]])  
y = np.array([1.1, 1.9, 3.0, 3.9])  
  
  
loo = LeaveOneOut()  
model = LinearRegression()  
  
  
errors = []  
for train_idx, test_idx in loo.split(X):  
    X_train, X_test = X[train_idx], X[test_idx]  
    y_train, y_test = y[train_idx], y[test_idx]  
    model.fit(X_train, y_train)  
    y_pred = model.predict(X_test)  
    errors.append(mean_squared_error(y_test, y_pred))  
  
  
print("Ortalama MSE:", np.mean(errors))
```

### Ne zaman kullanılır?

- Küçük veri setlerinde idealdir (örnek: < 100 gözlem).
- Eğitim verisinden maksimum fayda sağlar.

### Dezavantaj:

- Büyük veri setlerinde çok yavaş ve hesaplama maliyeti yüksektir.
- Aykırı verilere karşı daha hassastır.

### Avantajı:

Model, tüm veriyi farklı zamanlarda hem eğitim hem testte kullandığı için sonuçlar daha güvenilirdir ve aşırı öğrenme (overfitting) riski azalır.

## Veri İşlemleri (Data Processing)

Makine öğrenmesinde başarılı bir model elde etmek için yalnızca algoritmaları kullanmak yeterli değildir. Veriyi doğru **işlemek, seçmek ve dönüştürmek** model performansını doğrudan etkiler.

### Özellik Çıkarımı (Feature Extraction)

Özellik çıkarımı, ham veriden algoritmaların işleyebileceğii **anlamlı ve sayısal özellikler (features)** üretme sürecidir. Bu adım, veriyi makine öğrenmesi modelleri için uygun hâle getirir ve modelin başarısını doğrudan etkiler.

### Örnekler:

- **Metin verisi** → Kelime sıklığı vektörü (TF-IDF, Bag-of-Words)
- **Görüntü verisi** → Piksel yoğunluğu, kenar tespiti, HOG (Histogram of Oriented Gradients)
- **Zaman serisi** → Ortalama, varyans, frekans bileşenleri

### Örnek Kod (TF-IDF):

```
from sklearn.feature_extraction.text import TfidfVectorizer

metinler = ["Bu bir örnek cümledir", "Bir başka örnek daha"]

vectorizer = TfidfVectorizer()

X = vectorizer.fit_transform(metinler)

print(vectorizer.get_feature_names_out())

print(X.toarray())
```

Çıktı:

```
['başka', 'bir', 'bu', 'cümledir', 'daha', 'örnek']  
[[0.      , 0.4099, 0.5761, 0.5761, 0.      , 0.4099],  
 [0.5761, 0.4099, 0.      , 0.      , 0.5761, 0.4099]]
```

## 🎯 Özellik Seçimi (Feature Selection)

Elimizdeki tüm değişkenlerden, **modele en çok katkı sağlayanları** seçme işlemidir. Gereksiz özellikler hem modeli yavaşlatır hem de aşırı öğrenme (**overfitting**) riskini artırır.

### 📌 Yöntemler:

- **Korelasyon Matrisi:** Özellikler arasında yüksek korelasyon varsa, birini çıkar.
- **Recursive Feature Elimination (RFE):** Model kurarak, en az katkı sağlayanları iteratif olarak çıkarır.
- **SelectKBest:** Belirli bir skor fonksiyonuna göre en iyi K özelliği seçer (örneğin ANOVA F-score).

### 📦 Örnek:

Ev fiyatı tahmini için "Oda sayısı", "Metrekare" gibi özellikler önemliyken, "Sokak adı" belki değildir.

```
from sklearn.feature_selection import SelectKBest, f_regression  
from sklearn.datasets import make_regression  
  
X, y = make_regression(n_samples=100, n_features=10, noise=0.1)  
selector = SelectKBest(score_func=f_regression, k=5)  
X_selected = selector.fit_transform(X, y)  
  
print("Seçilen özellikler:", selector.get_support(indices=True))
```

## ⚠️ Aşırı Uyum (Overfitting)

Model, eğitim verisini çok iyi öğrenir ama test verisinde kötü performans gösterir. Bu, modelin genellemeye yapamadığı anlamına gelir.

### Belirtileri:

- Eğitim hatası çok düşük, test hatası çok yüksek.
- Karmaşık modellerde daha sık görülür (örneğin: derin ağaçlar, yüksek derece polinomlar).

### Önleme Yöntemleri:

- **Çapraz Doğrulama(Cross-Validation)** kullanmak
- **Regularizasyon (L1/L2)** uygulamak -> Tüm verisetindeki tahmin edilen target değer ve gerçek değerleri arasındaki farkı minimize ediyor.
- **Model karmaşıklığını** azaltmak (örneğin ağaç derinliğini sınırlamak)
- **Erken durdurma (Early Stopping)**
- **Daha fazla veri ile eğitmek**

### Örnek (Early Stopping - XGBoost):

```
import xgboost as xgb

from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)

model = xgb.XGBRegressor()
model.fit(X_train, y_train,
          eval_set=[(X_val, y_val)],
          early_stopping_rounds=10,
          verbose=False)
```

Adım	Amaç	Kullanım
Feature Extraction	Veriden anlamlı özellik üretmek	TF-IDF, HOG
Feature Selection	Gereksiz değişkenleri eleme	RFE, SelectKBest
Cross Validation	Modelin genelleme yeteneğini test etmek	K-Fold, LOOCV
Overfitting Önleme	Ezberleyen modelin önüne geçmek	Regularizasyon, CV, Early Stopping

## Hiperparametre Ayarı (Hyperparameter Tuning)

Hiperparametreler, bir makine öğrenmesi modelinin **öğrenme sürecini etkileyen**, model eğitilmeden önce belirlenen ayarlardır. Modelin içinden öğrenilmmezler; **biz dışarıdan belirleriz.**

### Özellikleri:

- Eğitim sırasında değil, **öncesinde** belirlenir.
- Genellikle modelin **performansını önemli ölçüde etkiler**.
- **Hiperparametre ayarlama (tuning)** ile bu değerler optimize edilir.

### Örnek Hiperparametreler:

Model	Hiperparametreler
KNN	n_neighbors, metric
Decision Tree	max_depth, min_samples_split
SVM	C, gamma, kernel
Random Forest	n_estimators, max_features
XGBoost	learning_rate, max_depth, n_estimators

## Grid Search

Grid Search, belirli hiperparametrelerin önceden belirlenmiş kombinasyonlarını dener. Her kombinasyon için model eğitilir ve en iyi performansı veren ayar seçilir.

### Avantajları:

- Sistematisk ve kapsamlıdır.
- Küçük arama alanlarında çok etkilidir.

### Dezavantajları:

- Kombinasyon sayısı çoxsa zaman maliyeti yüksektir.
- Tüm kombinasyonlar denenir (akıllı seçim yapılmaz).

### Örnek:

SVM algoritması için C ve gamma parametrelerinin farklı değerleri denenerek en yüksek F1 skoru veren kombinasyon seçilir.

### Kod Örneği – SVM ile Grid SearchCV

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)

# Model ve parametreler
model = SVC()
param_grid = {
    'C': [0.1, 1, 10],
    'gamma': [1, 0.1, 0.01],
    'kernel': ['rbf']
}
# Grid Search
grid = GridSearchCV(model, param_grid, cv=5, scoring='f1_macro')
grid.fit(X, y)

print("En iyi parametreler:", grid.best_params_)
print("En iyi F1 skoru:", grid.best_score_)
```

Örnek Çıktı:

```
En iyi parametreler: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
En iyi F1 skoru: 0.96
```

### Alternatif: RandomizedSearchCV

- Grid Search tüm kombinasyonları denerken, RandomizedSearch sadece belirli sayıda rastgele kombinasyon dener.
- Daha hızlı, büyük hiperparametre alanlarında daha uygundur.

## 1.5 Makine Öğreniminin Uygulama Alanları:

Makine öğrenimi, çeşitli sektörlerde ve uygulamalarda kullanılmaktadır:

- **Görüntü Tanıma:** Güvenlik kameralarının hırsızlık veya yüz tanıma yapması
- **Doğal Dil İşleme (NLP):** Chatbot'ların gelen soruları anlaması
- **Konuşma Tanıma:** Siri veya Google Assistant'ın sesli komutları algılaması
- **Tahmine Dayalı Analitik:** Stokta bitecek ürünlerin önceden tahmin edilmesi
- **Öneri Sistemleri:** Netflix'in ilgini çekebilecek dizileri önermesi
- **Otonom Araçlar:** Trafik ışıklarını ve yayaları algılayarak güvenli şekilde ilerleme
- **Güvenlik:** Anomali tespiti, yüz tanıma

## 1.6 Makine Öğreniminin Geleceği:

Makine öğrenimi, **geleceğin omurgası** olacak teknolojilerden biri olarak görülüyor.

### Yükselen Trendler:

- **Explainable AI (XAI):**  
Modellerin kararlarını açıklayabilen sistemler.
- **Federated Learning:**  
Veriyi merkezileştirmeden, cihazlar üzerinde öğrenme.
- **Edge AI:**  
AI'nın düşük güçlü cihazlara entegre edilmesi (ör: IoT, akıllı saatler).
- **AutoML:**  
Otomatik model seçimi ve hiperparametre ayarlaması.
- **Etik AI ve Adil Modelleme:**  
Yanlı (biased) modellerin önlenmesi, toplumsal etkilerin dikkate alınması.

## 2. Regresyon Modelleri(Regression Models)

Regresyon, **sürekli (sayısal) bir hedef değişkeni** tahmin etmek için kullanılır. Tahmin etmeye çalıştığımız hedef değişkenin (**dependent variable** - bağımlı değişken) sayısal olduğu durumlarda kullanılan modellerdir. Amaç, girdi değişkenleri (**independent variables** - bağımsız değişkenler) ile çıktı değişkeni arasındaki ilişkiyi modellemektir.

- **Bağımlı Değişken - Hedef (Dependent Variable - Target):** Tahmin edilmek istenen değişkendir.
- **Bağımsız Değişkenler - Özellikler (Independent Variables - Features):** Model için kullanılan girdilerdir.

## 2.1 Basit Doğrusal Regresyon ((Simple Linear Regression):)

### 📌 Tanım:

Bir bağımlı değişken (y) ile bir bağımsız değişken (x) arasındaki **doğrusal ilişkiyi** modellemeye çalışır.

🎯 Amaç: Girdi değişkenleri (**x-independent variables**) ile çıktı değişkeni (**y-dependent variable**) arasında doğrusal bir ilişki olduğunu varsayıarak tahmin yapmak. En basit regresyon modelidir.

### 📐 Matematiksel Model:

$$y = w \cdot x + b$$

- w: Eğim (slope)
- b: Y-eksenini kesen nokta (intercept)

### 📦 Örnek Kod:

```
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.array([[1], [2], [3], [4]])
y = np.array([2, 4, 5, 7])

model = LinearRegression()
model.fit(X, y)

print("Eğim (w):", model.coef_)
print("Kesişim (b):", model.intercept_)
```

Çıktı:

Eğim (w): **[1.7]**

Kesişim (b): **0.5**

## 2.2 Çoklu Doğrusal Regresyon (Multiple Linear Regression)

### Tanım:

Bir bağımlı değişken (**one dependent variable**) ile birden fazla bağımsız değişkeni (**multiple independent variables**) arasındaki doğrusal ilişkiyi analiz eder.

 **Amaç :** Birden fazla girdi değişkeninin çıktı değişkenini nasıl etkilediğini anlamak ve daha karmaşık tahminler yapmak.

### Matematiksel Model:

$$y = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

### Örnek Kod:

```
import pandas as pd

from sklearn.linear_model import LinearRegression

# Örnek veri
df = pd.DataFrame({
    'Metrekare': [50, 60, 70, 80],
    'Oda': [1, 2, 2, 3],
    'Fiyat': [100, 150, 180, 210]
})

X = df[['Metrekare', 'Oda']]

y = df['Fiyat']

model = LinearRegression()
model.fit(X, y)

print("Katsayılar:", model.coef_)
print("Sabit:", model.intercept_)
```

### Çıktı:

Katsayılar: [2.5, 15.0]

Sabit: -25.0

## 2.3 Polinom Regresyon (Polynomial Regression)

### Tanım:

Girdi (**input**) ve çıktı (**output**) değişkenleri arasındaki ilişkinin doğrusal olmadığı durumlarda, girdi değişkenlerinin polinomlarını kullanarak doğrusal regresyonu genişletmek için uygulanır. Veri ile doğru arasında **doğrusal olmayan (eğrisel)** bir ilişki varsa,  $x^2$ ,  $x^3$  gibi polinom terimleri eklenerek bu ilişki modellenir.

### • Basit Polinom Regresyon

 Tek bir değişken kullanılır ama bu değişkenin **kuvvetleri** modele katılır.

### Örnek Kod:

```
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.linear_model import LinearRegression  
from sklearn.pipeline import make_pipeline  
  
X = np.array([[1], [2], [3], [4]])  
y = np.array([3, 5, 10, 20])  
  
# 2. dereceden polinom regresyon  
model = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())  
model.fit(X, y)  
  
print("Tahminler:", model.predict(X))
```

### Çıktı:

```
Tahminler: [ 2.8  4.6  9.4 18.2]
```

- **Çoklu Polinom Regresyon**

👉 Birden fazla özelliğin hem kendisi hem de etkileşimli kuvvetli terimleri modele dahil edilir.

📦 **Örnek Kod:**

```
X = df[['Metrekare', 'Oda']]  
y = df['Fiyat']  
  
poly_model = make_pipeline(PolynomialFeatures(degree=2, include_bias=False),  
                           LinearRegression())  
  
poly_model.fit(X, y)  
  
print("Tahminler:", poly_model.predict(X))
```

🔍 **Karşılaştırma:**

Model	Kullanım Durumu
Doğrusal Regresyon	Basit, doğrusal ilişki varsa
Çoklu Doğrusal Regresyon	Birden çok bağımsız değişken varsa
Polinom Regresyon	Eğrisel (non-linear) ilişki varsa

## 2.4 Aşırı Uyum ve Yetersiz Uyum (Overfitting and Underfitting)

**Aşırı Uyum (Overfitting):** 🧐 Modelin eğitim verisini çok iyi öğrenmesi, hatta gürültüyü bile ezberlemesi ve bu nedenle yeni, görülmemiş verilerde kötü performans göstermesi.

📦 **Örnek:** Eğitim verisindeki her bir evin fiyatını mükemmel tahmin eden çok karmaşık bir model, piyasadaki genel eğilimleri yakalayamayabilir ve yeni bir evin fiyatını yanlış tahmin edebilir.

**Yetersiz Uyum (Underfitting):** 😞 Modelin eğitim verisindeki temel ilişkileri bile yakalayamaması ve hem eğitim hem de yeni verilerde kötü performans göstermesi.

💡 **Örnek:** Evin büyülüğu ile fiyatı arasında sadece sabit bir değer (örneğin, ortalama fiyat) tahmin eden basit bir model, büyülüğun etkisini göz ardı ettiği için yetersiz uyum gösterir.

## 2.5 Sapma - Varyans Takası (Bias-Variance Tradeoff)

⌚ Modelin doğruluğu ile genelleme yeteneği arasındaki dengeyi ifade eder.

🧠 **Sapma (Bias):**

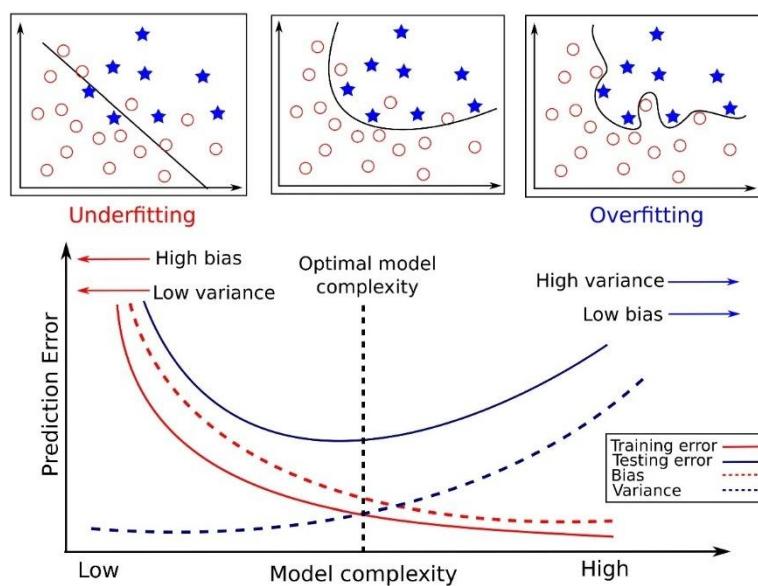
Modelin **ne kadar yanlış (doğruluğu düşük)** tahminler yaptığı gösterir.  
→ Yüksek sapma = **Underfitting** riski

⌚ **Varyans (Variance):**

Modelin **verideki küçük değişikliklere ne kadar duyarlı** olduğunu gösterir.  
→ Yüksek varyans = **Overfitting** riski

📊 **Amaç:**

Sapma ile varyans arasında denge kurmak.  
En iyi model ne çok ezberler, ne de yüzeysel öğrenir.



- Bias ve varyans arasında ters bir ilişki vardır. Yüksek bias, modeli basit tutarak varyansı düşürür, ancak modeli veriyle yeterince uyumlu hale getiremez. Yüksek varyans ise modeli karmaşıklaştırarak bias'ı düşürür, ancak genelleme başarısını azaltır.
- İdeal bir model, bias ve varyans arasında bir denge kurar ve hem eğitim hem de test verilerinde iyi performans gösterir.

## 2.6 Düzenlileştirme (Regularization: Ridge, Lasso, ElasticNet)

Modelin **aşırı uyum yapmasını (overfitting)** engellemek için ağırlıklara ceza (penalty) eklenerek karmaşıklık azaltılır.



### Yöntemler:

- ◆ **Ridge Regresyon (L2 Regularization):**

$$\text{Hata} + \lambda \sum w^2$$

- Katsayıları küçültür ama sıfıra indirmez
- Çoklu doğrusal bağlantı (multicollinearity) varsa işe yarar

```
from sklearn.linear_model import Ridge  
model = Ridge(alpha=1.0)
```

- ◆ **Lasso Regresyon (L1 Regularization):**

$$\text{Hata} + \lambda \sum |w|$$

- Ağırlıkların mutlak değerlerini cezalandırır → bazı ağırlıkları sıfıra indirir → **özellik seçimi (feature selection)** sağlar.

```
from sklearn.linear_model import Lasso  
model = Lasso(alpha=1.0)
```

- ◆ **ElasticNet:**

$$\text{Hata} + \alpha \left( r \cdot \sum |w| + (1 - r) \cdot \sum w^2 \right)$$

- Hem L1 hem L2'yi birlikte kullanır. → hem küçültme hem sıfırlama yapar.

```
from sklearn.linear_model import ElasticNet  
model = ElasticNet(alpha=1.0, l1_ratio=0.5)
```

## 2.7 Gradyan İnişi (Gradient Descent)

Bir modelin **hata fonksiyonunu (loss function)** en aza indirmek için kullanılan **iteratif optimizasyon algoritmasıdır**.

### Temel Fikir:

- İlk olarak rastgele katsayılar verilir
- Kayıp fonksiyonunun eğimi (türevi) alınarak, **en dik aşağı yönde** adım adım ilerlenir
- Her adımda katsayılar güncellenir

 **Örnek:** Bir topu tepenin yamacından yuvarlayarak en dip noktaya ulaştırmak gibi düşünebilirsin.

### Parametreler:

Parametre	Açıklama
Learning Rate ( $\eta$ )	Adım boyu – çok küçükse yavaş, çok büyükse sapar
Epoch	Tüm veri üzerinden geçen tekrar sayısı
Batch Size	Kaç örnekle bir seferde güncelleme yapılacağı
Momentum (isteğe bağlı)	Öğrenmeye hız kazandırır

**Örnek:** Basit doğrusal bir ilişkiyi (örneğin,  $y = 3x + 2$ ) gradyan inişiyle tahmin edelim.

### KOD: Gradyan Inişi ile Doğrusal Regresyon

```
import numpy as np

# Örnek veri (x: giriş, y: hedef)

X = np.array([1, 2, 3, 4, 5])

y = np.array([5, 8, 11, 14, 17]) # Gerçek denklem: y = 3x + 2

# Ağırlıklar (slope=w, bias=b) rastgele başlatılır

w = 0.0 # eğim

b = 0.0 # kesişim

# Hiperparametreler

learning_rate = 0.01

epochs = 1000

n = len(X)

# Gradyan iniş algoritması

for epoch in range(epochs):

    y_pred = w * X + b

    error = y_pred - y
```

```

# Gradyan hesaplama (Mean Squared Error türevine göre)

dw = (2/n) * np.dot(error, X)

db = (2/n) * np.sum(error)

# Ağırlıkları güncelle

w -= learning_rate * dw

b -= learning_rate * db

# Her 100 adımda durumu yazdır

if epoch % 100 == 0:

    loss = np.mean(error ** 2)

    print(f"Epoch {epoch}: Loss={loss:.4f}, w={w:.4f}, b={b:.4f}")

# Final sonuç

print("\n🔊 Öğrenilen Denklem: y =", round(w, 2), "* x +", round(b, 2))

```

**Çıktı:**

```

Epoch 0: Loss=182.6000, w=1.0400, b=0.2400
Epoch 100: Loss=0.2027, w=2.9363, b=2.2346
Epoch 200: Loss=0.0701, w=2.9745, b=2.1034
...
Epoch 900: Loss=0.0033, w=2.9989, b=2.0063

🔊 Öğrenilen Denklem: y = 3.0 * x + 2.01

```

### 3. Sınıflandırma Modelleri (Classification Models)

#### 3.1 Sınıflandırmaya Giriş (Introduction to Classification)

Sınıflandırma, verileri belirli **kategori veya sınıflara** ayıran **denetimli öğrenme (supervised learning)** problemidir.

📌 Hedef değişken (target) **sayısal değil**, genellikle kategoriktir (örneğin: "Evet/Hayır", "Kırmızı/Mavi").

#### 🧠 Örnekler:

- E-posta → Spam / Değil
- Tümör → İyi Huylu / Kötü Huylu
- Müşteri → Satın alır / Almaz

## 3.2 Değerlendirme Metrikleri (Evaluation Metrics)

Bir sınıflandırma modelinin doğruluğunu ölçmek için çeşitli metrikler kullanılır.

### 💡 Confusion Matrix (Karmaşıklık Matrisi)

Gerçek sınıflarla modelin tahmin etiği sınıfları karşılaştırılan  $2 \times 2$  ya da daha büyük bir tablodur.

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

- Bu matris, diğer metriklerin (Accuracy, Precision, Recall, F1) temelini oluşturur:
  - **True Positive (TP):** Gerçek pozitif ve doğru tahmin.
  - **False Positive (FP):** Gerçek negatif ama pozitif tahmin edilmiş.
  - **False Negative (FN):** Gerçek pozitif ama negatif tahmin edilmiş.
  - **True Negative (TN):** Gerçek negatif ve doğru tahmin.

Örnek:

```
from sklearn.metrics import confusion_matrix  
  
y_true = [1, 0, 1, 1, 0, 0, 1]  
  
y_pred = [1, 0, 1, 0, 0, 1, 1]  
  
cm = confusion_matrix(y_true, y_pred)  
  
print(cm)
```

Çıktı:

```
[[2 1]  
 [1 3]]
```

→ TN=2, FP=1, FN=1, TP=3

### ✓ Accuracy (Doğruluk)

Doğru tahmin edilen gözlemlerin oranıdır.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

```
from sklearn.metrics import accuracy_score
print("Accuracy:", accuracy_score(y_true, y_pred)) # Output: 0.714
```

### Precision (Kesinlik)

Pozitif tahminlerin ne kadarının doğru olduğunu ölçer.

$$\text{Recall} = \frac{TP}{TP + FN}$$

```
from sklearn.metrics import recall_score
print("Recall:", recall_score(y_true, y_pred)) # Output: 0.75
```

### F1-Score

Precision ve Recall'un harmonik ortalamasıdır. Dengeli bir değerlendirme sunar.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

```
from sklearn.metrics import f1_score
print("F1 Score:", f1_score(y_true, y_pred)) # Output: 0.75
```

### ROC (Receiver Operating Characteristic) & AUC (Area Under Curve)

- **ROC Eğrisi**, farklı eşik değerlerinde modelin **True Positive Rate (Recall)** ve **False Positive Rate**'ini çizer.

- AUC, bu eğrinin altında kalan alandır. AUC ne kadar büyükse model o kadar iyidir (maksimum 1.0).

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

y_prob = [0.9, 0.2, 0.8, 0.4, 0.3, 0.6, 0.85]

fpr, tpr, thresholds = roc_curve(y_true, y_prob)

roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.title("ROC Curve")
plt.grid()
plt.show()
```

## PR Curve (Precision-Recall Eğrisi)

Özellikle **dengesiz veri setlerinde** daha doğru bir performans ölçüsüdür. Pozitif sınıfı odaklanarak modelin başarısını gösterir.

```
from sklearn.metrics import precision_recall_curve

precision, recall, _ = precision_recall_curve(y_true, y_prob)

plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title("Precision-Recall Curve")
plt.grid()
plt.show()
```

## Özet Tablo:

Metrik	Anlamı	Ne Zaman Kullanılır?
Accuracy	Genel başarı oranı	Dengeli veri setlerinde
Precision	Pozitif tahminlerin doğruluğu	FP'yi azaltmak önemliyse
Recall	Gerçek pozitifleri yakalama oranı	FN'yi azaltmak önemliyse (ör. sağlık)
F1-Score	Precision ve Recall'un dengesi	Dengesiz veri setlerinde
ROC-AUC	Modelin sınıflar arası ayırma gücü	Tüm eşiklerde genel performans
PR Curve	Precision ve Recall arasındaki ilişki	Azınlık sınıf performansı gerektiğinde

### 3.3 Ek Değerlendirme Metrikleri

#### ◆ Regresyon Metrikleri (Regression Metrics)

 Regresyon modelleri sürekli (sayısal) çıktılar üretir, bu çıktılar ile gerçek değerler arasındaki farkı ölçmek için aşağıdaki metrikler kullanılır:

- **MSE (Mean Squared Error - Ortalama Kare Hata):** Hataların karesinin ortalamasıdır. Büyük hatalar daha çok cezalandırılır.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- **RMSE (Root Mean Squared Error - Karekök Ortalama Kare Hata):** MSE'nin kareköküdür. Yorumlaması daha kolaydır çünkü aynı birime sahiptir.

$$RMSE = \sqrt{MSE}$$

- **MAE (Mean Absolute Error - Ortalama Mutlak Hata):** Tahmin ile gerçek değer arasındaki farkların mutlak ortalaması.

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- **R<sup>2</sup> (R-Square - Determinasyon Katsayısı):** Modelin veriye ne kadar iyi uyduğunu gösterir. 0–1 arasında değer alır. 1'e ne kadar yakınsa o kadar iyi.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

- **Adjusted R<sup>2</sup> (Düzeltilmiş R-Karesi):** Özellikle çok değişkenli modellerde R<sup>2</sup> değerini cezalandırarak daha gerçekçi sonuç verir.

### Regresyon Metrikleri Örnek:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Gerçek ve tahmin değerleri
y_true = [3, 5, 2.5, 7]
y_pred = [2.5, 5, 4, 8]

mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)

mae = mean_absolute_error(y_true, y_pred)

r2 = r2_score(y_true, y_pred)

print("MSE:", mse)
print("RMSE:", rmse)
print("MAE:", mae)
print("R2:", r2)
```

Çıktı:

MSE: 0.875

RMSE: 0.935

MAE: 0.75

R<sup>2</sup>: 0.85

### ◆ Denetimsiz Öğrenme Metrikleri (Unsupervised Learning / Clustering)

Etiketli veri olmadığı için kümeleme kalitesi dışsal kriterlerle ölçülür.

- **ARI (Adjusted Rand Index):**  
Rastgele şansa göre düzeltilmiş benzerlik metriği.
- **Rand Index:**  
Gerçek küme ve tahmin edilen küme gruplarının uyumluluğunu ölçer.
- **Mutual Information Score (Karşılıklı Bilgi):**  
Gerçek etiketlerle tahmin edilen etiketler arasındaki ortak bilgi miktarını ölçer.
- **Silhouette Score:**  
Her noktanın kendi kümesine olan yakınlığı ile diğer kümelere olan uzaklıği karşılaştırılır.

$$\text{Silhouette Score} \in [-1, 1]$$

- **V-Measure:**  
Homojenlik ve bütünlüğün harmonik ortalaması. Etiket sayısı bilinmediğinde değerlendirme için uygundur.

#### Kümeleme Metrikleri (Clustering) Örnek:

```
from sklearn.metrics import adjusted_rand_score, v_measure_score, silhouette_score
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Veri oluştur
X, y_true = make_blobs(n_samples=100, centers=3, cluster_std=0.6, random_state=0)

# Kümeleme modeli
kmeans = KMeans(n_clusters=3, random_state=0)
y_pred = kmeans.fit_predict(X)
ari = adjusted_rand_score(y_true, y_pred)
v_score = v_measure_score(y_true, y_pred)
silhouette = silhouette_score(X, y_pred)

print("ARI:", ari)
print("V-Measure:", v_score)
print("Silhouette Score:", silhouette)
```

Çıktı:

```
ARI: 1.0
V-Measure: 1.0
Silhouette Score: 0.65
```

#### ◆ NLP Metrikleri (Doğal Dil İşleme)

- **BLEU Score (Bilingual Evaluation Understudy):**  
Makine çevirisi kalitesini ölçer. 0 ile 1 arasında değer alır. 1'e ne kadar yakınsa o kadar iyidir.



Örnek:  
Referans: "the cat is on the mat"  
Tahmin: "the cat the cat on the mat"  
→ BLEU skoru düşük olur çünkü tekrar var.

#### BLEU Score (NLP) Örnek:

```
from nltk.translate.bleu_score import sentence_bleu

reference = [['the', 'cat', 'is', 'on', 'the', 'mat']]
candidate = ['the', 'cat', 'the', 'cat', 'on', 'the', 'mat']

bleu = sentence_bleu(reference, candidate)

print("BLEU Score:", bleu)
```

Çıktı:

```
BLEU Score: 0.467
```

#### ◆ Çapraz Doğrulama Hataları (Cross-Validation Errors)

- **CV Error:**  
K-Fold gibi yöntemlerle modelin farklı veri alt kümelerinde aldığı ortalama hata.
- **AMSE (Average Mean Squared Error):**  
MSE değerlerinin farklı alt kümelerdeki ortalamasıdır.  
👉 Özellikle model karşılaştırmalarında kullanılır.

#### Çapraz Doğrulama Hatası (CV Error) Örnek:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_regression

# Veri oluştur
X, y = make_regression(n_samples=100, n_features=1, noise=10)

model = LinearRegression()

scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=5)
amse = -scores.mean()

print("AMSE (Average MSE):", amse)
```

**Çıktı:**

AMSE (Average MSE): 104.23 (değer örnektir, farklı çalıştırımda değişebilir)

◆ **K Sayısı Belirleme (Kümeleme)**

Kümeleme algoritmalarında (ör. K-Means) ideal **küme sayısını** belirlemek önemlidir.

- **Elbow Method (Dirsek Yöntemi):**  
Küme sayısına göre WCSS (within-cluster sum of squares) grafiği çizilir. Eğrinin "dirsek" yaptığı nokta ideal K'dır.
- **Heuristic Methods:**  
Silhouette analizi, Gap Statistics gibi yöntemler de sezgisel olarak uygun K sayısını bulmak için kullanılır.

**Elbow Yöntemi (Küme Sayısı Belirleme) Örnek:**

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

X, _ = make_blobs(n_samples=300, centers=4, random_state=42)
wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_) # inertia = WCSS

plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Yöntemi')
plt.xlabel('Küme Sayısı (K)')
plt.ylabel('WCSS')
```

**Çıktı:**

Grafikte "dirsek" oluşan nokta ideal küme sayısını verir.

### 3.4 K-En Yakın Komşu (KNN - K-Nearest Neighbors)

Yeni bir veri noktası, eğitim verisindeki **en yakın K komşunun sınıfına göre sınıflandırılır**.

- 📍 K değeri, kaç komşuya bakılacağını belirler.
- ⚖️ Karmaşık olmayan, sezgisel ama büyük veri kümelerinde yavaş olabilir.

#### 📌 Örnek:

Bir kişinin yaş, kilo, boy gibi değerlerine bakarak onun **sporcu mu yoksa ofis çalışanı mı** olduğunu belirlemek için, benzer değerlere sahip kişilerin çoğunluğuna göre karar verilir.

```
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.datasets import load_iris  
from sklearn.model_selection import train_test_split  
from sklearn.metrics import accuracy_score  
  
X, y = load_iris(return_X_y=True)  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
  
model = KNeighborsClassifier(n_neighbors=3)  
model.fit(X_train, y_train)  
y_pred = model.predict(X_test)  
  
print("Accuracy:", accuracy_score(y_test, y_pred))
```

### 3.5 Lojistik Regresyon (Logistic Regression)

İkili (binary) veya çok sınıflı problemler için kullanılan doğrusal bir sınıflandırma yöntemidir.

- Hızlı ve yorumlanabilir
  - Olasılık verir
- 📉 **Dezavantaj:**
- Karar sınırı lineerse iyi çalışır, karmaşık ayrımlarda zayıftır.

### 📌 Örnek:

Bir müşterinin satın alma geçmişsi, yaşı ve gelirine bakarak, bir ürünü **satın alma olasılığı** tahmin edilir. %70 olasılıkla "Evet" diyorsa, bu "Alır" sınıfına atanır.

```
from sklearn.linear_model import LogisticRegression  
from sklearn.datasets import load_iris  
  
X, y = load_iris(return_X_y=True)  
model = LogisticRegression(max_iter=200)  
model.fit(X, y)  
y_pred = model.predict(X)  
print("Doğruluk:", accuracy_score(y, y_pred))
```

## 3.6 Karar Ağaçları (Decision Trees)

Veriyi ağaç yapısında dallandırarak sınıflandırma yapar.

### 📌 Örnek:

"Yaş > 30 mu?" → "Evet" → "Geliri > 50k mı?" → "Evet" → "Premium müşteri" Bu şekilde dallanarak karar verir.

```
from sklearn.tree import DecisionTreeClassifier, plot_tree  
import matplotlib.pyplot as plt  
  
model = DecisionTreeClassifier(max_depth=3)  
model.fit(X, y)  
plt.figure(figsize=(10, 6))  
plot_tree(model, feature_names=load_iris().feature_names,  
          class_names=load_iris().target_names, filled=True)  
plt.show()
```

### 🔗 Overfitting - Pruning (Budama):

Ağaç çok derinleşirse, eğitim verisine ezberleme yapar (**overfitting**). **Pruning** ile gereksiz dallar kesilerek model sadeleştirilir.

```
model = DecisionTreeClassifier(max_depth=3, min_samples_leaf=5)  
model.fit(X_train, y_train)
```

max\_depth, min\_samples\_split, min\_samples\_leaf gibi parametrelerle budama (pruning) yapılır.

### 3.7 Destek Vektör Makineleri (SVM - Support Vector Machine)

Sınıfları en iyi ayıran hiper düzlemi (decision boundary) bulur. Kernel fonksiyonları ile doğrusal olmayan sınıfları da ayıracaktır.

#### 🎯 Kernel Trick:

Veri doğrusal olarak ayrılamıyorsa, kernel fonksiyonları ile veri **yüksek boyuta** dönüştürülerek ayrılabilir.

#### 🧠 Avantaj:

- Yüksek başarı
- Küçük veri setlerinde güçlü

#### ⚖️ Dezavantaj:

- Büyük veri setlerinde yavaş
- Yorumlanması zordur

#### 📌 Örnek:

E-postaların uzunluk, özel kelime oranı gibi özelliklerine bakarak "spam" veya "değil" sınıfına ayırmak.

```
from sklearn.svm import SVC

model = SVC(kernel='rbf', C=1.0, gamma='scale')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("SVM Doğruluk:", accuracy_score(y_test, y_pred))
```

### 3.8 Naive Bayes

İstatistiksel sınıflandırma algoritmasıdır, **Bayes Teoremi'ne** dayanır. Özelliklerin **bağımsız** olduğu varsayıımı ile çalışır (bu yüzden "Naive").

#### 📌 Örnek:

E-postadaki kelimelere göre mesajın **spam olup olmadığını** tahmin eder. Eğer "free", "win", "money" gibi kelimeler çoxsa → spam olma ihtimali artar.

### Avantaj:

- Çok hızlıdır.
- Özellikle **metin sınıflandırma** için çok uygundur (e-posta filtreleme, haber kategorisi vs.)

### Dezavantaj:

- Özellikler gerçekten bağımsız değilse, hata artabilir

```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Naive Bayes Doğruluk:", accuracy_score(y_test, y_pred))
```

## 3.9 Rastgele Ormanlar (Random Forest)

Çok sayıda karar ağacının oluşturulup oy çokluğuyla tahmin yapılmasını sağlayan topluluk yöntemidir.

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, max_depth=4)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Random Forest Doğruluk:", accuracy_score(y_test, y_pred))
```

**Not:** n\_estimators, max\_depth, max\_features gibi parametrelerle özelleştirilebilir.

Gelişmiş modellerde hiperparametre ayarları, çapraz doğrulama ve model seçimi süreci de uygulanmalıdır.

## 4. Topluluk Öğrenimi (Ensemble Learning)

Topluluk yöntemleri (ensemble methods), birden fazla modelin birleştirilerek daha güçlü ve genellenebilir sonuçlar elde edilmesini sağlar. Zayıf öğrenicilerin bir araya getirilerek daha güçlü bir tahmin elde edilmesidir.

- Tek bir model yerine, **birden çok modelin** birlikte çalışması sağlanır.
- Tahminler **oylama (classification)** ya da **ortalama (regression)** ile birleştirilir.

 **Örnek:**

Aynı ayrı çalışan 10 doktorun kanser teşhisi yerine, hepsinin ortak kararı alınır → daha güvenilir sonuç.

## 4.2 Bagging ve Random Forest

 **Bagging (Bootstrap Aggregating)**

- Veriden **rastgele örnekler** alarak birden fazla model eğitilir.
- Sonuçlar ortalanır (regresyon) ya da oylama yapılır (sınıflandırma).

**Random Forest**, Bagging'in Karar Ağaçları ile uygulanmış halidir.

 **Temel Parametreler:**

- `n_estimators`: Ağaç sayısı
- `max_depth`: Her ağacın maksimum derinliği
- `max_features`: Her ağacın görmesi gereken öznitelik sayısı
- `bootstrap`: Örneklem alırken tekrar seçime izin verilip verilmemiği

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = load_iris(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestClassifier(n_estimators=100, max_depth=4)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy (Random Forest):", accuracy_score(y_test, y_pred))
```

### Random Forest

- Bagging yönteminin bir uygulamasıdır.
- Her ağaca sadece rastgele seçilmiş **ozellik alt kümeleri** verilir.
- **Aşırı öğrenmeye (overfitting)** karşı dayanıklıdır.

### Örnek:

Bir öğrencinin sınav sonucunu, farklı öğretmenlerin değerlendirmesine benzetebiliriz. Her öğretmen, soruların farklı bölümlerine odaklanır.

## 4.3 Boosting Teknikleri (Boosting Techniques)

Boosting, zayıf öğreniciler sırayla eğitilir. Her bir model, bir öncekinde yapılan hataları düzeltmeye çalışır.

### AdaBoost (Adaptive Boosting)

- Her model, bir önceki modelin **yanlış sınıflandırdığı örneklerde** daha fazla ağırlık verir.
- Sonuçta doğru sınıflama oranı artar.

### Temel Parametreler:

- `n_estimators`: Aşamalı model sayısı
- `learning_rate`: Her modelin katkı oranı
- `base_estimator`: Kullanılan temel zayıf öğrenici

### Örnek:

Öğretmenin her sınavda öğrencinin yanlış yaptığı konulara daha çok yoğunlaşması gibi.

```
from sklearn.ensemble import AdaBoostClassifier  
  
from sklearn.tree import DecisionTreeClassifier  
  
model = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),  
n_estimators=50)  
  
model.fit(X_train, y_train)  
  
y_pred = model.predict(X_test)  
  
print("Accuracy (AdaBoost):", accuracy_score(y_test, y_pred))
```

### Gradient Boosting

- Tahmin hatalarına (residual) karşı **gradyan inişi** kullanılarak yeni modeller eğitilir.
- Daha esnek ve özelleştirilebilir bir boosting yöntemidir.

### Temel Parametreler:

- `n_estimators`: Ağaç sayısı
- `learning_rate`: Ağaçların katkı oranı
- `max_depth`: Her ağacın derinliği

```
from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy (Gradient Boosting):", accuracy_score(y_test, y_pred))
```

### XGBoost (Extreme Gradient Boosting)

- Gradient Boosting'in daha **hızlı ve verimli** halidir.
- **Paralel işlem, overfitting kontrolü** gibi gelişmiş özelliklere sahiptir.
- Kaggle yarışmalarında sıkça kullanılır.

### Temel Parametreler:

- `n_estimators`, `learning_rate`, `max_depth`
- `subsample`: Her ağaca örneklenen veri oranı
- `colsample_bytree`: Her ağaca örneklenen sütun oranı

### Örnek:

Satış tahmini, kredi skorlama, müşteri terk tahmini gibi ticari uygulamalarda çok kullanılır.

```
from xgboost import XGBClassifier

model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

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

#### 4.4 Stacking (Stacking Method)

Farklı türdeki algoritmaların çıktıları birleştirilir ve bu çıktılar üzerine yeni bir model (meta-learner) eğitilir.

##### 🔧 Temel Bileşenler:

- `estimators`: Alt modeller
- `final_estimator`: Meta model

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
base_models = [
    ('knn', KNeighborsClassifier(n_neighbors=3)),
    ('svm', SVC(probability=True))
]
meta_model = LogisticRegression()
model = StackingClassifier(estimators=base_models, final_estimator=meta_model)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy (Stacking):", accuracy_score(y_test, y_pred))
```

#### 4.5 Hiperparametre Ayarlama (Hyperparameter Tuning)

Hiperparametreler, model eğitimi öncesinde elle belirlenen ayarlardır (örneğin, ağaç derinliği, öğrenme oranı, K değeri gibi).

##### 🔍 GridSearch

- Belirlenen parametrelerin **tüm kombinasyonlarını** dener.
- En iyi sonucu veren kombinasyonu bulur.
- Detaylıdır ama yavaş olabilir.

```

from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators': [50, 100], 'max_depth': [2, 4, 6]}

grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print("Best Score:", grid.best_score_)

```

## RandomizedSearch

- Parametre kombinasyonlarından **rastgele seçimler** yapar.
- Daha hızlı çalışır, ancak en iyi sonucu kaçırabilir.

```

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_dist = {'n_estimators': randint(10, 200), 'max_depth': randint(1, 10)}

random_search = RandomizedSearchCV(RandomForestClassifier(),
param_distributions=param_dist, n_iter=10, cv=5, random_state=42)
random_search.fit(X_train, y_train)

print("Best Parameters:", random_search.best_params_)

```

## Örnek:

Bir fırınçı kek yaparken, un, şeker ve yağ oranlarını farklı şekillerde dener. GridSearch hepsini dener, RandomizedSearch bazılarını rastgele dener.

## Bayesian Optimization (Opsiyonel)

scikit-optimize veya optuna gibi kütüphanelerle yapılır. Her yeni denemeyi önceki sonuçlara göre yönlendirerek daha az kaynakla en iyi sonucu bulur.

```

# Not: Bayesian Optimization örneği için optuna kurulmalıdır.

import optuna

def objective(trial):
    n_estimators = trial.suggest_int('n_estimators', 10, 200)
    max_depth = trial.suggest_int('max_depth', 1, 10)
    clf = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth)
    clf.fit(X_train, y_train)
    return clf.score(X_test, y_test)

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=20)
print("Best trial:", study.best_trial.params)

```

## 5. Denetimsiz Öğrenme (Unsupervised Learning)

Etiketli verilerin bulunmadığı durumlarda, verideki yapıyı veya desenleri bulmak için kullanılan öğrenme türündür. Yani veri sadece girdilerden oluşur, çıkış (etiket) yoktur.

### Amaç:

- Verileri gruplamak (clustering)
- Veriyi özetlemek veya basitleştirmek (boyut indirgeme)
- Sıra dışı (anormal) durumları tespit etmek (anomali tespiti)

### 5.1 K-Means Kümeleme (K-Means Clustering)

Verileri, birbirine benzerliklerine göre **K adet kümeye** (gruba) ayırır.

- Başlangıçta K merkez (centroid) rastgele seçilir.
- Her veri noktası, en yakın merkeze atanır.
- Merkezler güncellenir ve işlem tekrar edilir.

### Örnek:

Bir e-ticaret sitesindeki müşterileri harcama alışkanlıklarına göre 3 gruba ayırmak (örneğin: düşük, orta, yüksek harcayanlar).

### Temel Parametreler:

- `n_clusters`: Küme sayısı
- `init`: Başlangıç küme merkezi yöntemi ('k-means++' önerilir)
- `max_iter`: Maksimum iterasyon sayısı
- `n_init`: Başlatma denemesi sayısı

```
from sklearn.cluster import KMeans  
  
import numpy as np  
  
  
X = np.random.rand(100, 2)  
  
kmeans = KMeans(n_clusters=3, init='k-means++', n_init=10, max_iter=300)  
  
kmeans.fit(X)  
  
print("Cluster Centers:", kmeans.cluster_centers_)
```

## 5.2 Hiyerarşik Kümeleme (Hierarchical Clustering)

Veri noktalarını benzerliğe göre bir hiyerarşi içinde gruplayarak ağaç yapısı oluşturur (dendrogram).

### Parametreler:

- `linkage`: Kümeleme türü ('ward', 'complete', 'average')
- `affinity`: Uzaklık ölçüsü ('euclidean', 'manhattan')

```
from sklearn.cluster import AgglomerativeClustering  
  
  
model = AgglomerativeClustering(n_clusters=3, linkage='ward')  
  
y_pred = model.fit_predict(X)  
  
print("Labels:", y_pred)
```

## 5.3 PCA (Principal Component Analysis – Temel Bileşen Analizi)

Verideki yüksek boyutlu özellikler, en fazla bilgiyi taşıyacak şekilde daha az sayıda **ana bileşene** indirger.

- Verinin özünü bozmadan boyut sayısı azaltılır.
- Görselleştirme, hız ve yorumlama kolaylığı sağlar.

### Parametreler:

- `n_components`: İstenen bileşen sayısı

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)
print("Explained Variance Ratio:", pca.explained_variance_ratio_)
```

### 💡 Örnek:

100 özellikli tıbbi veriyi 2 bileşene indirip 2D grafikte göstererek hasta kümeleri arasında ayırm görmek.

## 5.4 Boyut İndirgeme Yöntemleri (Dimensionality Reduction)

Verideki boyut sayısını (özellik sayısını) azaltarak:

- Görselleştirme kolaylığı sağlar
  - Gürültüyü azaltır
  - İşlem hızını artırır
- ◆ **t-SNE (t-Distributed Stochastic Neighbor Embedding)**
- Özellikle **görselleştirme** için uygundur.
  - Benzer veri noktalarını yakın yerleştirir.
  - Yüksek boyutlu veriyi 2D ya da 3D'ye indirir.

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, perplexity=30)
X_tsne = tsne.fit_transform(X)
```

◆ **UMAP (Uniform Manifold Approximation and Projection)**

- t-SNE'ye göre daha **hızlı ve anlam koruyucu** bir yöntemdir.
- Büyük veri setlerinde tercih edilir.

```
import umap.umap_ as umap

reducer = umap.UMAP(n_neighbors=15, min_dist=0.1)
X_umap = reducer.fit_transform(X)
```

### Örnek:

El yazısı rakam verilerini (MNIST) 784 boyuttan 2 boyuta indirip, kümelenmeleri 2D grafikte göstermek.

## 5.5 Anomali Tespiti (Anomaly Detection)

Genel veri deseninden sapma gösteren, sıra dışı veya beklenmedik gözlemleri bulma işlemidir.

### Örnekler:

- Banka işlemlerinde dolandırıcılık tespiti
- Makine sensör verilerinde arıza öncesi davranış
- Web trafiğinde güvenlik saldırısı (DDoS gibi)

### Yöntemler:

- Z-score (istatistiksel yöntem)

```
from scipy.stats import zscore

z_scores = np.abs(zscore(X))

outliers = (z_scores > 3).any(axis=1)
```

- Isolation Forest

```
from sklearn.ensemble import IsolationForest

clf = IsolationForest(contamination=0.1)

clf.fit(X)

outliers = clf.predict(X)
```

- DBSCAN (yoğunluk tabanlı kümeleme ile aykırıları bulma)

```
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=0.3, min_samples=5)

labels = db.fit_predict(X)
```

## 6. Zaman Serileri (Time Series)

### 6.1 Zaman Serilerine Giriş (16\_Time Series)

Zaman serileri, verilerin zamana bağlı olarak sıralandığı veri kümeleridir. Her gözlem belirli bir zaman noktasına karşılık gelir.

#### 💡 Kullanım Alanları:

- Ekonomik tahminler (döviz, enflasyon)
- Stok ve talep tahminleri
- Hava durumu
- Enerji tüketimi

### 6.2 Trend, Mevsimsellik, Durağanlık

◆ **Trend:** Uzun dönemli artış veya azalış eğilimi. ◆ **Mevsimsellik (Seasonality):** Belirli aralıklarla tekrar eden kalıplar (örneğin, yazın artan turizm). ◆ **Durağanlık (Stationarity):** Ortalama ve varyansı zamanla değişmeyen serilerdir. Zaman serisi modelleme için genellikle durağan veri gereklidir.

#### 💡 Durağanlık Testleri:

- Augmented Dickey-Fuller (ADF)

```
from statsmodels.tsa.stattools import adfuller  
  
result = adfuller(series)  
  
print("ADF Statistic:", result[0])  
  
print("p-value:", result[1])
```

#### 💡 Dönüşüm Yöntemleri:

- Fark alma (diff)
- Log dönüşümü

### 6.3 ARIMA, SARIMA Modelleri

#### 🔧 ARIMA (AutoRegressive Integrated Moving Average)

Parametreler:

- p: Otoregresif terim sayısı (AR)
- d: Durağanlaştmak için kaç kere fark alındığı
- q: Hareketli ortalama terimi sayısı (MA)

```
from statsmodels.tsa.arima.model import ARIMA  
  
model = ARIMA(series, order=(2, 1, 2))  
  
model_fit = model.fit()  
  
forecast = model_fit.forecast(steps=10)
```

### 🔧 SARIMA (Seasonal ARIMA)

ARIMA'nın mevsimsel veriye uygun versiyonudur. Parametreler:

- $(p, d, q) + \text{seasonal\_order}=(P, D, Q, s)$

```
from statsmodels.tsa.statespace.sarimax import SARIMAX  
  
model = SARIMAX(series, order=(1,1,1), seasonal_order=(1,1,1,12))  
  
model_fit = model.fit()  
  
forecast = model_fit.forecast(steps=12)print
```

## 6.4 Prophet ile Tahminleme

Facebook tarafından geliştirilmiş açık kaynak zaman serisi tahminleme aracıdır. Özellikle tatil ve mevsimsel etkileri ele alabilir.

### 🔧 Temel Parametreler:

- `growth`: "linear" veya "logistic" (trend tipi)
- `seasonality_mode`: "additive" veya "multiplicative"
- `holidays`: Tatil etkilerini hesaba katmak için

```
from prophet import Prophet  
  
import pandas as pd  
  
# Prophet, 'ds' ve 'y' adında iki sütun ister  
  
df = pd.DataFrame({"ds": dates, "y": values})  
  
model = Prophet()  
  
model.fit(df)  
  
future = model.make_future_dataframe(periods=30)  
  
forecast = model.predict(future)  
  
model.plot(forecast)
```

## 6.5 Model Performansı ve Görselleştirme

- ◆ Tahminlerin doğruluğunu ölçmek için kullanılan metrikler:

- MAE (Mean Absolute Error)
- RMSE (Root Mean Squared Error)
- MAPE (Mean Absolute Percentage Error)

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np
mae = mean_absolute_error(y_true, y_pred)
rmse = np.sqrt(mean_squared_error(y_true, y_pred))
print("MAE:", mae)
print("RMSE:", rmse)
# Görselleştirme:
import matplotlib.pyplot as plt
plt.plot(y_true, label='Gerçek')
plt.plot(y_pred, label='Tahmin')
plt.legend()
plt.show()
```

Zaman serisi modellemede forecast (tahminleme) işlemi, ARIMA/SARIMA veya Prophet gibi modellerin çıktısı olarak gelecekteki değerleri öngörmeyi sağlar. Bu, tüm zaman serisi çalışmalarının **nihai hedefi** olarak değerlendirilir.

## 7. Uygulama ve İleri Konular

### 7.1 Özellik Seçimi ve Mühendisliği (Feature Engineering)



Nedir?

**Feature Engineering**, makine öğrenmesi modellerinin daha başarılı çalışılabilmesi için veri setindeki mevcut özelliklerini (değişkenleri) dönüştürme, yeni özellikler üretme veya gereksiz olanları çıkarma işlemidir.



Amaç:

- Modelin öğrenmesini kolaylaştırmak
- Karmaşıklığı azaltmak
- Doğruluğu artırmak

```

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.feature_selection import SelectKBest, f_classif


# Örnek veri

data = {

    'yaş': [25, 30, 35, 40, 45],  

    'gelir': [50000, 60000, 80000, 110000, 150000],  

    'eğitim': ['Lisans', 'Yüksek Lisans', 'Lisans', 'Doktora', 'Yüksek Lisans'],  

    'hedef': [0, 1, 0, 1, 1]
}

df = pd.DataFrame(data)

# Kategorik ve sayısal özelliklerin ayrılması

numeric_features = ['yaş', 'gelir']

categorical_features = ['eğitim']

# Özellik dönüşümleri

preprocessor = ColumnTransformer(  

    transformers=[  

        ('num', StandardScaler(), numeric_features),  

        ('cat', OneHotEncoder(), categorical_features)
    ])
# Özellik seçimi

X = df.drop('hedef', axis=1)

y = df['hedef']

X_transformed = preprocessor.fit_transform(X)

# En iyi 3 özelliği seçme

selector = SelectKBest(score_func=f_classif, k=3)

X_selected = selector.fit_transform(X_transformed, y)

print("Seçilen özelliklerin şekli:", X_selected.shape)

```

## 7.2 Pipeline ve Otomasyon

Pipeline, veri ön işleme (scaling, encoding) ve model eğitimi gibi işlemleri tek bir yapı içinde birleştirir. Bu sayede:

- Tek satırda tüm işlemler uygulanır.
- Kod tekrarı azalır.
- Hata riski düşer.
- GridSearch gibi optimizasyonlar kolaylaşır.

```
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
# Pipeline oluşturma
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('feature_selector', selector),
    ('classifier', RandomForestClassifier(random_state=42))
])
# Veriyi bölme
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Modeli eğitme
pipeline.fit(X_train, y_train)
# Skor
print("Test skoru:", pipeline.score(X_test, y_test))
```

**joblib ile Model Kaydetme :** Eğitilen modeli dosyaya kaydedip sonra tekrar kullanmak. Böylece her seferinde yeniden eğitmeye gerek kalmaz.

```
import joblib
# Modeli kaydet
joblib.dump(pipeline, 'model_pipeline.joblib')
# Modeli yükle
loaded_model = joblib.load('model_pipeline.joblib')
# Yüklenen modelle tahmin yapma
print(loaded_model.predict(X_test))
```

### 7.3 Veri Dengesizliği (SMOTE, Class Weights)

Dengesiz sınıflar model başarısını düşürebilir.

```
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Dengesiz veri örneği
X_imb = np.array([[1, 1], [2, 2], [3, 3], [4, 4], [5, 5], [6, 6]])
y_imb = np.array([0, 0, 0, 0, 1, 1]) # 4:2 oranında dengesiz

# SMOTE uygulama
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X_imb, y_imb)

# Class weights kullanma
model = LogisticRegression(class_weight='balanced')
model.fit(X_imb, y_imb)

print("SMOTE sonrası sınıf dağılımı:", np.bincount(y_res))
print("Classification Report:\n", classification_report(y_imb, model.predict(X_imb)))
```

### 7.4 Gerçek Hayat Projesi Oluşturma Adımları

1. **Problem Tanımı:** Çözülecek problemi ve başarı metriğini tanımla
2. **Veri Toplama:** Veri kaynaklarını belirle ve topla
3. **Veri Ön İşleme:** Eksik veri, aykırı değerler, dönüşümler
4. **Özellik Mühendisliği:** Yeni özellikler oluştur, önemli özellikleri seç
5. **Model Seçimi:** Birden fazla model deneyerek en iyisini seç
6. **Hiperparametre Optimizasyonu:** GridSearchCV veya RandomizedSearchCV
7. **Değerlendirme:** Test seti ve çapraz doğrulama ile modeli değerlendir
8. **Dağıtım:** Modeli üretim ortamına taşı

## 7.5 Model Dağıtıımı

### Flask ile API

```
# app.py

from flask import Flask, request, jsonify
import joblib
import pandas as pd

app = Flask(__name__)
model = joblib.load('model_pipeline.joblib')

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    df = pd.DataFrame(data)
    predictions = model.predict(df)
    return jsonify({'predictions': predictions.tolist()})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

- Docker, Kubernetes

Modelin taşınabilir konteynerlerde dağıtılması için kullanılır.

- Dockerfile ile ortam hazırlanır
- Kubernetes ile ölçekte ve yönetim yapılır

```
FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY ..
CMD ["python", "app.py"]
```

## 7.6 Etik, Adalet ve Yanlılık

### Problemler:

- **Veri yanlılığı (Bias):** Eğitim verisindeki önyargılar modele de yansır.
- **Adaletsizlik:** Bazı gruplara karşı sistematik hata yapılması
- **Veri gizliliği:** Kişisel verilerin korunması

### Çözümler:

- Farklı gruplar üzerindeki hata oranlarını karşılaştırmak
- Veri dengeleme
- Açıklanabilirlik araçları (LIME, SHAP)

#### 💡 Örnek:

Yalnızca erkek adaylardan oluşan bir CV verisiyle eğitilen model, kadın adayları dışlayabilir. Bu etik bir problemdir.

### SHAP :

```
import shap  
  
explainer = shap.Explainer(model, X)  
  
shap_values = explainer(X)  
  
shap.plots.waterfall(shap_values[0])
```

### LIME:

```
from lime.lime_tabular import LimeTabularExplainer  
  
explainer = LimeTabularExplainer(X_train.values, feature_names=feature_names)  
  
explanation = explainer.explain_instance(X_test[0], model.predict_proba)  
  
explanation.show_in_notebook()
```

### Fairlearn:

```
from fairlearn.metrics import MetricFrame  
  
from sklearn.metrics import accuracy_score  
  
frame = MetricFrame(metrics=accuracy_score, y_true=y_test, y_pred=y_pred,  
sensitive_features=X_test['gender'])  
  
print(frame.by_group)
```