2025

Machine Learning Lecture Notes

BEYZA KÜÇÜK

Table of Contents

1. Introduction and Fundamental Concepts

1.1 Introduction to Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that enables computers to learn from data without being explicitly programmed. In traditional programming, computers follow explicitly written instructions to perform specific tasks. In contrast, machine learning systems learn patterns and relationships from datasets and use this knowledge to make predictions or decisions on new, unseen data.

➡️ Example: On an e-commerce platform, recommending new products based on a customer's previous purchases.

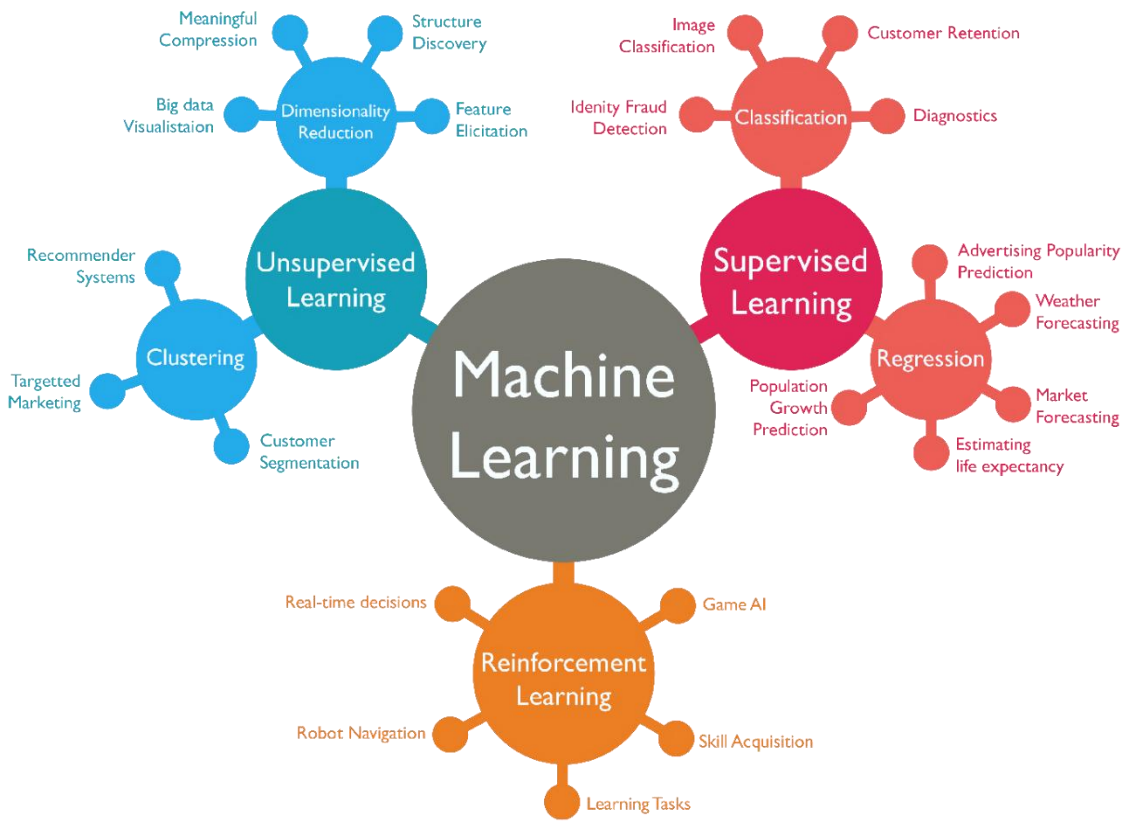
Core Components of Machine Learning:

• **Data:** The fundamental resource used to train machine learning algorithms. Datasets contain structured or unstructured information from which algorithms can learn and identify patterns.

• **Algorithms:** Mathematical models designed to learn from data and make predictions. Different machine learning algorithms are tailored for various types of data and tasks.

• **Models:** The resulting structures formed after algorithms are trained on data. Models are used to make predictions or decisions based on new input.

Types of Machine Learning:



🟦 **Supervised Learning**

This is a type of learning where models are trained using **labeled datasets**.

🎯 **Objective:** The algorithm learns the relationship between input data and output labels, enabling it to predict outputs for future inputs.

📦 **Example:** Learning to recognize which handwritten digit is written based on a dataset like MNIST.

🟧 Semi-Supervised Learning

🎯 **Objective:** To improve learning performance by using **unlabeled data** in addition to a small amount of **labeled data**, especially when labeled data is scarce.

🧠 **Advantage:** Achieves high performance with less labeled data.

📦 **Example:** Detecting cancerous tissues using a few labeled medical images and a large number of unlabeled ones.

🟩 Unsupervised Learning

A type of learning where models are trained using **unlabeled datasets**.

🔍 **Objective:** To discover hidden patterns or groupings within the data.

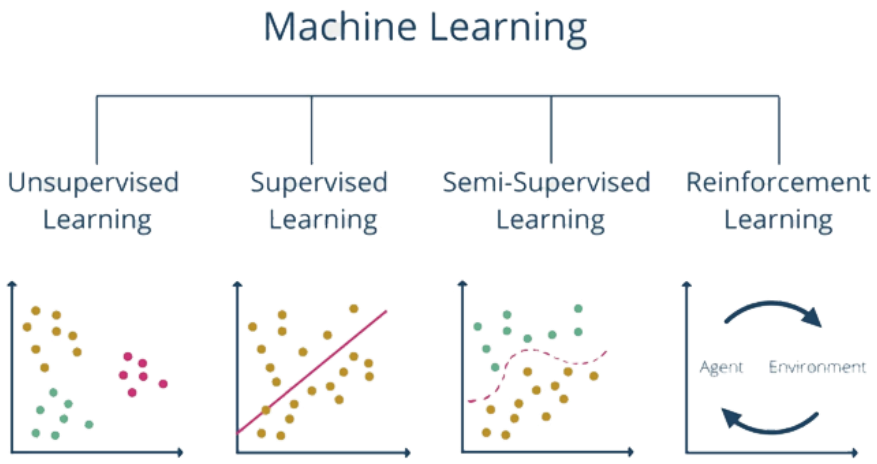
📦 **Example:** Automatically segmenting bank customers based on their spending habits.

🟥 Reinforcement Learning

This is a learning type in which an **agent** interacts with an environment and learns by receiving **rewards or penalties** for its actions.

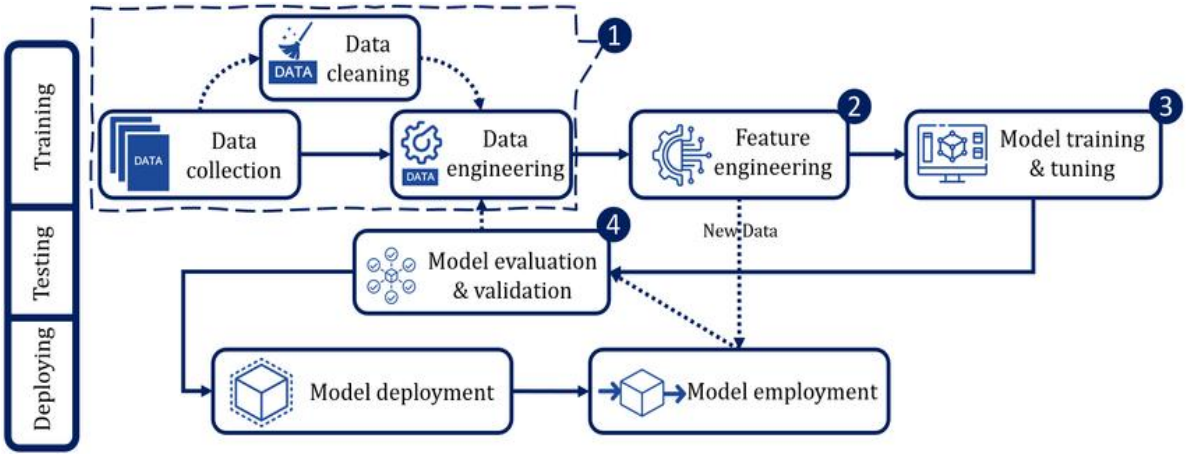
🕹️ **Objective:** To learn actions that maximize the **cumulative future reward** through trial and error.

📦 **Example:** A reinforcement learning agent learning how to win at Pac-Man by playing the game.



1.2 Machine Learning Workflow

- **Data Collection:** Gathering relevant data.
- **Veri Ön İşleme:**
  - o Handling missing values.
  - o Converting categorical variables into numerical format.
  - o Scaling the data (standardization/normalization).
- **Model Selection:** Choosing the appropriate machine learning algorithm for the problem.
- **Model Training:** Training the selected algorithm on the dataset.
- **Model Evaluation:** Measuring the model's performance using test data.
- **Model Deployment:** Making the trained model available for use.



1.3 Data Preprocessing:

- This subsection covers the process of transforming raw data into a format suitable for machine learning algorithms. The key steps in data preprocessing include:
  - **Missing Data:** Missing values can negatively impact the accuracy of a model, so they need to be handled appropriately.
    - **Deletion (dropna):** This method can be used when there are very few missing values and no systematic pattern to the missingness.

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

    - **Imputation:** This involves filling in missing (empty) values in the dataset with meaningful and reasonable estimates. Common techniques include filling with the **mean**, **median**, or **mode**.

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

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

- **Advanced Methods:** Techniques such as **KNN imputation** or **regression-based prediction** can also be used to estimate missing values more accurately.

```
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=5)

df.iloc[:, :] = imputer.fit_transform(df)
```

- **Categorical Data (Categorical Encoding):** Machine learning algorithms can only process **numerical data**. Therefore, textual or categorical variables need to be converted into numerical format.

  - **Label Encoding**

    Each category is assigned a unique numerical value.

```
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**

    Each category is expanded into a separate column. If the category is present, the value is **1**; if not, **0**.

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(sparse=False)

renk_encoded = enc.fit_transform(df[['Renk']])

print(renk_encoded)
```

  - **Ordinal Encoding**
    If the categories have a logical order (e.g., 'Low', 'Medium', 'High'), ordered numerical values are assigned accordingly.

```
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**

  Categories are encoded based on the **mean value of the target variable**. This method is commonly used in **regression problems**.

```
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:** This section explains how to standardize or normalize numerical variables that are on different scales (e.g., z-score normalization, min-max normalization).

  - **StandardScaler (Z-Score Normalization)**
    Transforms the data so that the **mean = 0** and **standard deviation = 1**.

```
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 Normalization)**
    Scales the data to a range between **0 and 1**.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df_scaled = scaler.fit_transform(df)

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

- **RobustScaler**

Resistant to **outliers**. It uses the **median** and **interquartile range (IQR)** for scaling.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df_scaled = scaler.fit_transform(df)

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

🔚 Summary Comparison

Method	**Use Case**
Label Encoding	For unordered categorical variables (use with caution!)
One-Hot Encoding	When a **separate column is needed** for each category
Ordinal Encoding	For **ordered categories** (e.g., Low, Medium, High)
Target Encoding	In **regression problems** or when the category is **strongly related to the target**
StandardScaler	**Generally recommended**, especially for **linear models**
MinMaxScaler	When **data needs to be between 0 and 1** (e.g., **neural networks**)
RobustScaler	When there are **many outliers**

💉 1.4 Model Selection and Data Splitting (Train/Test Split, Cross-Validation)

Model Selection:

Choosing the right model in machine learning depends on the type of problem, the structure of the dataset, and the nature of the target variable.

🧩 Key Factors in Model Selection:

- **Type of Problem:**
    - **Numerical prediction** → Regression models

- **Classification** → Logistic Regression, SVM, Decision Trees
- **Group discovery** → Clustering algorithms (e.g., K-Means, DBSCAN)

- **Dataset Size:**
  - **Small datasets** → Naive Bayes, Decision Trees
  - **Large datasets** → Random Forest, XGBoost
- **Feature Types:**
  - Numerical / Categorical features
  - Presence of missing or outlier values

📌 a) Train/Test Split

The dataset is typically split into **70–80% for training** and **20–30% for testing**.
The model learns from the training set and its performance is evaluated on the test set.

```
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)
```

🔹 Why Is This Important?

Testing a model on the same data it was trained on is like taking a test with the answer key.
If the model memorizes the training data (overfitting), this won't be noticed without a separate test set.The test set must be unseen by the model.

📦 Example:

In a model that predicts whether customers will make a purchase on an e-commerce platform, **80% of the data** is used for training, and **20%** for testing.

🔁 b) Cross-validation

The dataset is split into multiple subsets, and each subset is used as test data in turn.
This method evaluates the model's **ability to generalize** to unseen data.

How Does It Work?

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5)
```

Step 1: The dataset is split into equal parts (e.g., 5 parts → "5-Fold").

Step 2:

- **Round 1:** One part is used for testing, the remaining 4 for training.
- **Round 2:** Another part is used for testing, and the rest for training.
- **...** This process continues until each part has been used once as a test set.
  Result: The average performance across all rounds is calculated.

- 🔁 **K-Fold Cross Validation**
  - The dataset is divided into **K equal-sized folds**.
  - In each iteration, one fold is used as the **test set**, and the remaining **K–1 folds** as the **training set**.
  - This process is repeated **K times**, with each fold used exactly once for testing.
  - The **average of all test scores** is computed to estimate the model's overall performance.

  ✅ **Advantages of K-Fold Cross-Validation:**

  - Provides a **more stable performance estimate** compared to a single train-test split.
  - **Reduces overfitting** by validating on different subsets.
  - **No data is wasted** — each data point is used at least once for testing.

  Examlple:

```
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)**

📌 **Definition:**

Each time, only **one observation** is used as the **test set**, and the **remaining N−1 observations** are used for training.

- This process is repeated **N times**, where **N** is the total number of data points.

```
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))
```

🎯 **When to Use:**

- Ideal for **small datasets** (e.g., fewer than 100 samples).
- Maximizes usage of training data.

⚠️ **Disadvantages:**

- **Very slow** and **computationally expensive** for large datasets.
- Sensitive to outliers.

📌 **Advantage:**

Since each data point is used in both training and testing at different times, the results are generally **more reliable** and the **risk of overfitting is reduced**.

🔍 Data Processing

In machine learning, using algorithms alone is not enough to build a successful model. Properly **processing**, **selecting**, and **transforming** the data directly affects model performance.

🧩 Feature Extraction

Feature extraction is the process of generating **meaningful and numerical features** from raw data that can be processed by algorithms. This step prepares the data for machine learning and has a direct impact on the success of the model.

📦 **Examples:**

- **Text data** → Word frequency vectors (TF-IDF, Bag-of-Words)
- **Image data → Pixel intensity, edge detection, HOG (Histogram of Oriented Gradients)**
- **Time series → Mean, variance, frequency components**

🛠️ **Example Code (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())
```

Output:

```
['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]]
```

🎯 Feature Selection

Feature selection is the process of selecting the most useful variables (features) from the available ones. Irrelevant features slow down the model and increase the risk of **overfitting**.

📌 Methods:

- **Correlation Matrix:** If two features are highly correlated, one can be removed.
- **Recursive Feature Elimination (RFE):** Iteratively removes the least important features by training a model.
- **SelectKBest:** Selects the top K features based on a scoring function (e.g., ANOVA F-score).

📦 Example:

In predicting house prices, features like "Number of rooms" and "Square meters" are likely important, whereas "Street name" may not be.

```
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))
```

⚠️ Overfitting

Overfitting occurs when the model learns the training data too well but performs poorly on the test data. This means the model **fails to generalize**.

🎯 **Symptoms:**

- Very low training error, but very high test error
- More common in complex models (e.g., deep trees, high-degree polynomials).

🛡️ **Prevention Techniques:**

- **Use Cross-Validation (CV)**
- Apply **Regularization (L1/L2)** → Minimizes the difference between predicted and actual target values across the dataset
- **Reduce model complexity** (e.g., limit tree depth)
- **Use Early Stopping**
- **Train with more data**

📦 **Example (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)
```

✅ Summary Table

Step	**Purpose**	**Examples / Methods**
Feature Extraction	Generate meaningful features from data	TF-IDF, HOG
Feature Selection	Eliminate irrelevant variables	RFE, SelectKBest
Cross Validation	Evaluate model's generalization ability	K-Fold, LOOCV
Overfitting Önleme	Avoid models that memorize the training set	Regularizasyon, CV, Early Stopping

💉 Hyperparameter Tuning

Hyperparameters are settings that influence the learning process of a machine learning model and are defined **before training** begins. They are not learned from the data; instead, they are set externally by the practitioner.

📌 Characteristics:

- Defined **before** training, not during.
- Often have a **significant impact** on model performance.
- **Can be optimized through hyperparameter tuning.**

📍 Example Hyperparameters:

Model	Hyperparameters
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 tries all possible combinations of predefined hyperparameter values. The model is trained for each combination, and the one with the best performance is selected.

✅ Advantages:

- Systematic and exhaustive.
- Very effective in **small search spaces**.

⚠️ Disadvantages:

- **Computationally expensive** if there are too many combinations.
- Tries **all combinations** (no intelligent sampling).

📦 Example:

Trying different values of **C** and **gamma** for an SVM model and selecting the combination that yields the highest **F1-score**.

🔧 Code Example – SVM with GridSearchCV

```
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_)
```

Example Output:

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

🔁 Alternative: RandomizedSearchCV

- While Grid Search tests **all combinations**, **RandomizedSearchCV** tests a **random subset** of combinations.
- It is **faster** and more suitable for **large hyperparameter spaces**.

💉 1.5 Applications of Machine Learning

Machine learning is used across various sectors and applications:

- 🖼️ **Image Recognition:** Detecting theft or identifying faces via security cameras
- 📄 **Natural Language Processing (NLP):** Chatbots understanding customer queries
- 🗣️ **Speech Recognition:** Voice command detection by Siri or Google Assistant
- 📊 **Predictive Analytics:** Forecasting products likely to go out of stock
- 📺 **Recommendation Systems:** Netflix recommending shows based on your interests
- 🚗 **Autonomous Vehicles:** Detecting traffic lights and pedestrians for safe navigation
- 🛡 **Security:** Anomaly detection, facial recognition

🔮 1.6 The Future of Machine Learning

Machine learning is considered one of the foundational technologies of the future.

🧠 **Emerging Trends:**

- **Explainable AI (XAI):**
  Systems that can explain how models make decisions.
- **Federated Learning:**
  Training models on decentralized data, such as data stored on user devices.
- **Edge AI:**
  Integrating AI into **low-power devices** (e.g., IoT, smartwatches).
- **AutoML:**
  Automated model selection and hyperparameter tuning.
- **Etik AI ve Adil Modelleme:**
  (Preventing biased models and accounting for social impact in AI systems.

2. Regression Models

Regression is used to predict a **continuous (numerical)** target variable. These models are applied when the target (dependent) variable is numerical. The goal is to model the relationship between input variables (**independent variables**) and the output variable.

- **Dependent Variable – Target:** The variable to be predicted.
- **Independent Variables - Features:** The inputs used to build the model.

2.1 Simple Linear Regression:

📌 **Definition:**
Models the linear relationship between one **dependent variable (y)** and one **independent variable (x)**.

🎯 **Goal:** To make predictions based on the assumption of a **linear relationship** between the input (x) and output (y). It is the simplest form of regression.

📐 **Mathematical Model:**

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

- w: Slope
- b: 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 ✅ Multiple Linear Regression

📌 **Definition:**
Analyzes the **linear relationship** between **one dependent variable** and **multiple independent variables**.

🎯 **Objective:** To understand how multiple input features influence the output variable and to make **more complex predictions** compared to simple linear regression.

📐 **Mathematical Model:**

$$y = w_1x_1 + w_2x_2 + ... + w_nx_n + b$$

📦 **Example Code:**

```
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_)
```

Output:

```
Katsayılar: [2.5, 15.0]
Sabit: -25.0
```

2.3 ✅ Polynomial Regression

📌 **Definition:**
Polynomial Regression is applied when the relationship between input and output variables is **non-linear**. It extends linear regression by incorporating **polynomial terms** of the input variables (e.g., x^2, x^3) to capture **curved (non-linear)** relationships in the data.

• **Simple Polynomial Regression**

👉 A single variable is used, but its **powers (e.g., x^2, x^3)** are included in the model as features.

📦 **Example Code:**

```
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))
```

Output

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

• **Çoklu Polinom Regresyon**

👉 The model includes **multiple features**, along with their **polynomial powers** and **interaction terms**.

📦 **Example Code:**

```
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))
```

🔍 **Comparison:**

Model	**Use Case**
Linear Regression	When the relationship is simple and linear
Multiple Linear Regression	When there are multiple independent variables
Polynomial Regression	When the relationship is non-linear (curved)

2.4 Overfitting and Underfitting

Overfitting: 🤯 When a model learns the training data **too well**, even memorizing noise, and performs **poorly on new, unseen data**.

> 📦 **Example:** A highly complex model that perfectly predicts house prices in the training data may fail to generalize to market trends and make poor predictions on new houses.

Underfitting: 😞 When a model fails to capture the underlying patterns in the training data and thus performs **poorly on both training and test sets**.

> 📦 **Example:** A model that predicts a fixed average price regardless of house size underfits the data by ignoring the influence of important features like square footage.

2.5 Bias-Variance Tradeoff

🔁 The balance between a model's **accuracy** and its **generalization ability**.

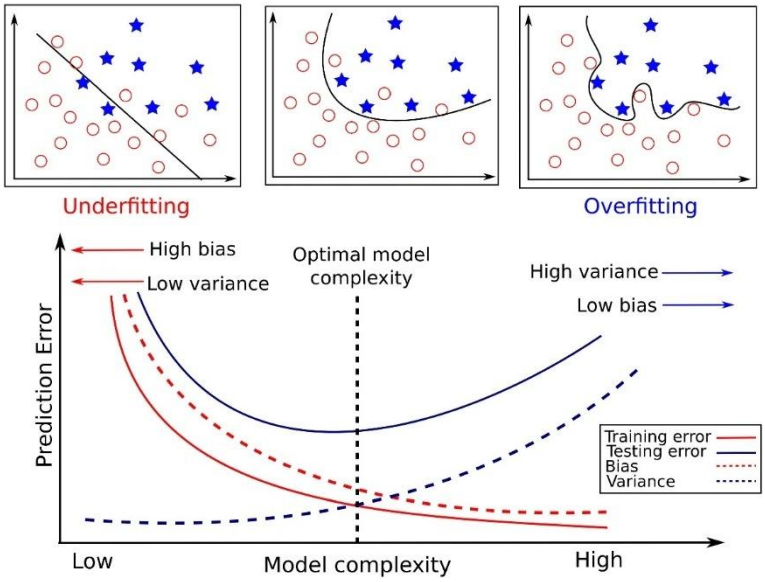
🧠 **Bias:**

Indicates how much the model's predictions deviate from the actual values.
→ **High bias = risk of underfitting**

🔁 **Variance:**

Shows how much the model's predictions change with small variations in the data.
→ **High variance = risk of overfitting**

📊 **Goal:**
Find a **balance** between bias and variance.
The best model is neither overly simplistic nor memorizing.



- Bias and variance are **inversely related**. High bias keeps the model simple but may fail to fit the data well. High variance makes the model complex, which can reduce bias but also reduce generalization.
- An **ideal model** strikes a balance and performs well on both training and unseen data.

2.6 Regularization: Ridge, Lasso, ElasticNet

Regularization is a technique to prevent overfitting by adding a penalty term to the model's weights, thus reducing complexity.

🔧 **Methods:**

🔹 **Ridge Regression (L2 Regularization):**

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

- Shrinks coefficients but does **not** set them to zero.
- Useful when **multicollinearity** exists among features.

```
from sklearn.linear_model import Ridge

model = Ridge(alpha=1.0)
```

🔹 **Lasso Regression (L1 Regularization):**

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

- Penalizes the **absolute values** of coefficients → can shrink some to **zero**, effectively performing **feature selection**.

```
from sklearn.linear_model import Lasso

model = Lasso(alpha=1.0)
```

🔹 **ElasticNet:**

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

- Combines both **L1 and L2 penalties** → performs both shrinking and zeroing out.

```
from sklearn.linear_model import ElasticNet

model = ElasticNet(alpha=1.0, l1_ratio=0.5)
```

2.7 Gradient Descent

Gradient Descent is an **iterative optimization algorithm** used to **minimize a loss function** in a model.

💡 **Core Idea:**

- Start with **random initial coefficients**
- Compute the **gradient (derivative)** of the loss function
- Move step-by-step **downhill** in the direction of the steepest descent
- **Update coefficients** in each step

🧠 **Example:** Like rolling a ball down a hill to find the **lowest point**.

🔧 **Key Parameters:**

Parameter	Desciription
Learning Rate (η)	Step size – too small = slow, too large = may overshoot the minimum
Epoch	Number of times the algorithm sees the full dataset
Batch Size	Number of samples used to update the model at each step
Momentum *(optional)*	Helps speed up learning and smooth the optimization path

Example: Use Gradient Descent to learn a simple linear relationship (e.g., **y = 3x + 2**).

✅ **Code: Linear Regression with Gradient Descent**

```
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))
```

Output:

```
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. Classification Models

3.1 Introduction to Classification

Classification is a **supervised learning** problem where data is categorized into specific **classes or categories**.
📌 The **target variable** is usually **categorical**, not numerical (e.g., "Yes/No", "Red/Blue")

🧠 **Example:**

- Email → Spam / Not Spam
- Tumor → Benign / Malignant
- Customer → Will Buy / Will Not Buy

3.2 Evaluation Metrics

Various metrics are used to measure the accuracy and performance of a classification model.

📌 **Confusion Matrix**

A 2x2 (or larger) table comparing the actual class labels to the model's predictions:

	Actual: Positive	**Actual: Negative**
Predicted: Positive	TP (True Positive)	FP (False Positive)
Predicted: Negative	FN (False Negative)	TN (True Negative)

- This matrix forms the basis for other metrics like **Accuracy**, **Precision**, **Recall**, and **F1-Score**:
  - **True Positive (TP):** Correctly predicted positive cases.
  - **False Positive (FP):** Predicted as positive, but actually negative.
  - **False Negative (FN):** Predicted as negative, but actually positive.
  - **True Negative (TN):** Correctly predicted negative cases.

📦 **Example:**

```
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)
```

Output:

```
[[2 1]

 [1 3]]
```

➡️ TN=2, FP=1, FN=1, TP=3

✅ **Accuracy**

The ratio of **correct predictions** to total predictions.

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

```
from sklearn.metrics import accuracy_score

print("Accuracy:", accuracy_score(y_true, y_pred))  # Output: 0.714
```

🎯 Precision

Measures how many of the predicted positives are **actually positive**.

$$Precision = \frac{TP}{TP + FP}$$

```
from sklearn.metrics import recall_score

print("Recall:", recall_score(y_true, y_pred))  # Output: 0.75
```

📈 Recall (Sensitivity or True Positive Rate)

Measures how many actual positives were **correctly identified**.

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

⚖️ F1-Score

The **harmonic mean** of Precision and Recall. Provides a **balanced metric** when both false positives and false negatives are important.

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

```
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)

- **The ROC Curve** plots the **True Positive Rate (Recall)** against the **False Positive Rate** at various threshold settings.
- **AUC** measures the **area under the ROC Curve**. The higher the AUC (up to 1.0), the better the model.

```
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)**

It is a more accurate performance **metric especially for imbalanced datasets.**
It focuses on the **positive class** and shows how well the model performs in identifying positive instances.

```
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()
```

🧠 Summary Table:

Metric	Meaning	When to Use
Accuracy	Overall success rate	When the dataset is **balanced**
Precision	Accuracy of positive predictions	When **reducing False Positives (FP)** is important
Recall	Proportion of actual positives correctly identified	When **reducing False Negatives (FN)** is crucial (e.g., healthcare)
F1-Score	Balance between Precision and Recall	Useful for **imbalanced datasets**
ROC-AUC	Ability of the model to distinguish between classes	Overall performance across all thresholds
PR Curve	Relationship between Precision and Recall	When performance on the **minority class** is important

📊 3.3 Additional Evaluation Metrics

🔹 Regression Metrics

📌 Regression models generate **continuous (numerical)** outputs. The following metrics are used to measure the difference between predicted and actual values:

- **MSE (Mean Squared Error):** The average of the squared errors. Penalizes **larger errors more heavily**.

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

- **RMSE (Root Mean Squared Error):** The square root of MSE. Easier to interpret as it has the **same units** as the target variable.

$$RMSE = \sqrt{MSE}$$

- **MAE (Mean Absolute Error):** The average of the **absolute differences** between predicted and actual values.

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

- **R^2 (R-Square - Determinasyon Katsayısı):** Indicates how well the model fits the data. Ranges from **0 to 1**. The **closer to 1**, the better.

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

- **Adjusted R^2 (Düzeltilmiş R-Karesi):**
  Used in **multivariate models**, it adjusts the R^2 value by penalizing the addition of irrelevant variables, providing a **more realistic** evaluation..

📦 **Example: Regression Metric Example:**

```
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)
```

Outputs:

```
MSE: 0.875
RMSE: 0.935
MAE: 0.75
R²: 0.85
```

🔹 **Unsupervised Learning / Clustering Metrics**

Since there are **no labeled targets**, clustering quality is measured using **external evaluation criteria**.

- **ARI (Adjusted Rand Index):**
  Random chance-corrected similarity metric.
- **Rand Index:**
  Measures the agreement between the **true cluster labels** and the **predicted cluster assignments**.
- **Mutual Information Score:**
  Measures the **amount of shared information** between the predicted labels and the true labels.
- **Silhouette Score:**
  The proximity of each point to its own cluster is compared with its distance to other clusters.

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

- **V-Measure:**
  The harmonic mean of homogeneity and completeness. It is suitable when the true number of clusters is unknown.

📦 **Example: Clustering Metrics in Practice**

```
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)
```

Output:

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

🔹 **NLP Metrics (Natural Language Processing)**

- **BLEU Score (Bilingual Evaluation Understudy):**
  Measures the **quality of machine translation** by comparing a generated sentence to one or more reference translations. The score ranges from **0 to 1** — the **closer to 1**, the better the translation.

📦 **Example:**
Reference: "the cat is on the mat"
Prediction: "the cat the cat on the mat"
→ The **BLEU score will be low** due to **repetition** and lack of precision.

Example Output: BLEU Score in NLP

```
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)
```

Output:

```
BLEU Score: 0.467
```

🔹 **Cross-Validation Errors**

- **CV Error (Cross-Validation Error):**
  The **average error** obtained by evaluating the model on **different data subsets** using methods like **K-Fold Cross-Validation**.
- **AMSE (Average Mean Squared Error):**
  The **mean** of the **MSE values** calculated across the folds in cross-validation.
  📌 Commonly used in **model comparison** to assess generalization performance.

📦 **Example: CV Error Calculation in Practice**

```
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)
```

Output:

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

🔹 **Choosing the Value of K (Clustering)**

Determining the **optimal number of clusters (K)** is crucial in clustering algorithms (e.g., **K-Means**).

- **Elbow Method (Dirsek Yöntemi):**
  A plot of **WCSS (Within-Cluster Sum of Squares)** is generated for different values of K.
- The **"elbow point"** — where the curve starts to flatten — indicates the **optimal K**.
- **Heuristic Methods:**
  Techniques such as **Silhouette Analysis** and **Gap Statistics** are also used to **intuitively** find the best number of clusters.

📦 **Example: Elbow Method for Determining Number of Clusters**

```
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')
```

Output:
Grafikte "dirsek" oluşan nokta ideal küme sayısını verir.

3.4 K-Nearest Neighbors (KNN)

A new data point is classified based on the **majority class** of its **K closest neighbors** in the training data.

📍 The **K value** determines how many neighbors are considered for the decision.

⚖️ It is **intuitive** and **non-complex**, but can be **slow with large datasets** due to high computational cost.

📌 **Example:**

To determine whether a person is an **athlete** or an **office worker**, we look at similar individuals based on attributes like **age, weight, and height**, and classify based on the **majority class** among them.

```
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 Logistic Regression

A **linear classification algorithm** used for **binary** or **multi-class** problems.

- **Fast** and **easy to interpret**
- Provides **probability estimates**
- 📉 **Disadvantage:**

Works well when the **decision boundary is linear**, but performs poorly with **complex class boundaries**.

📌 **Example:**
Based on a customer's **purchase history**, **age**, and **income**, the model predicts the **probability of purchasing** a product.
If the model outputs **70% probability**, the customer is classified as **"Will Buy"**.

```
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 Decision Trees

Classifies data by **splitting it into branches** using a **tree-like structure**.

📌 **Example:**
"Is age > 30?" → "Yes" → "Is income > 50k?" → "Yes" → "Premium customer"
The decision is made by following such **branching conditions**.

```
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:**
If a tree becomes too deep, it memorizes the training data (**overfitting**). Pruning simplifies the model by cutting off unnecessary branches.

```
model = DecisionTreeClassifier(max_depth=3, min_samples_leaf=5)

model.fit(X_train, y_train)
```

Pruning is done using parameters like max_depth, min_samples_split, and min_samples_leaf.

3.7 Support Vector Machines (SVM)

Finds the hyperplane (decision boundary) that best separates the classes. It can also separate non-linear classes using kernel functions.

🎯 **Kernel Trick:**
If the data is not linearly separable, kernel functions transform the data into a higher-dimensional space where it can be separated.

🧠 **Advantages:**

- High accuracy
- Strong performance on small datasets

⚖️ **Disadvantages:**

- Slow on large datasets
- Hard to interpret

📌 **Example:**
Classifying emails as "spam" or "not spam" based on features like email length or ratio of special words

```
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

A statistical classification algorithm based on Bayes' Theorem. It operates under the assumption that features are independent (hence "Naive").

📌 **Example:**
Predicts whether an email is **spam based** on the words it contains.
If words like "free", "win", "money" appear frequently → the likelihood of spam increases

🧠 **Advantages:**

- Very fast
- Especially suitable for text classification (email filtering, news categorization, etc.)

🐞 **Disadvantages:**

- If features are not truly independent, the error rate may increase

```
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 Random Forest

Random Forest is an ensemble method that builds multiple decision trees and makes predictions based on majority voting.

```
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))
```

Note: It can be customized with parameters such as `n_estimators`, `max_depth`, and `max_features`.

In advanced models, hyperparameter tuning, cross-validation, and model selection should also be applied.

4. Ensemble Learning

4.1 Introduction to Ensemble Methods

Ensemble methods combine multiple models to produce stronger and more generalizable results.It involves combining weak learners to create a stronger prediction model.

- Instead of relying on a single model, multiple models work together.
- Predictions are combined via voting (for classification) or averaging (for regression).

📦 **Example:**
Instead of relying on the diagnosis of a single doctor, a cancer diagnosis is based on the consensus of 10 independent doctors → more reliable result.

4.2 Bagging and Random Forest

📍 Bagging (Bootstrap Aggregating)

- Multiple models are trained using randomly sampled data subsets.
- Results are averaged (for regression) or voted (for classification).

Random Forest is the application of Bagging with Decision Trees.

🔧 Key Parameters:

- `n_estimators:` Number of trees
- `max_depth:` Maximum depth of each tree
- `max_features:` Number of features to consider for each tree
- `bootstrap:` Whether sampling is done with replacement

```
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

- An implementation of the Bagging method
- Each tree receives a random subset of features
- Resistant to **overfitting**

📦 **Example:**
Like evaluating a student's exam by multiple teachers, where each teacher focuses on different parts of the test.

4.3 Boosting Techniques

Boosting trains weak learners sequentially, where each model tries to correct the errors of the previous one.

⚡ AdaBoost (Adaptive Boosting)

- Each model gives more weight to the examples misclassified by the previous model
- As a result, the overall classification accuracy improves

🔧 Temel Parametreler:

- `n_estimators`: Number of sequential models
- `learning_rate`: Contribution of each model
- `base_estimator`: The type of weak learner used

📦 **Example:**
Like a teacher focusing more on the topics that the student got wrong in previous exams.

```
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

- New models are trained to correct the prediction errors (residuals) using gradient descent.
- A more flexible and customizable boosting method.

🔧 Key Parameters:

- `n_estimators`: Number of trees
- `learning_rate`: Contribution rate of the trees
- `max_depth`: Depth of each tree

```
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)**

- A faster and more efficient version of Gradient Boosting
- Offers advanced features like parallel processing and overfitting control
- Commonly used in Kaggle competitions

🔧 **Key Parameters:**

- `n_estimators`, `learning_rate`, `max_depth`
- `subsample`: The fraction of data sampled for each tree
- `colsample_bytree`: The fraction of features (columns) sampled for each tree

📦 **Example:**
Widely used in commercial applications such as sales forecasting, credit scoring, and customer churn prediction.

```
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)

The outputs of different types of algorithms are combined, and a new model (meta-learner) is trained on these outputs.

🔧 **Key Components::**

- `estimators`: Base models
- `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 Hyperparameter Tuning

Hyperparameters are settings manually defined before training the model (e.g., tree depth, learning rate, value of K).

🔍 GridSearch

- Tries all combinations of the specified parameters.
- Finds the combination that gives the best result.
- Detailed but can be slow.

```
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

- Randomly selects combinations from the parameter grid
- Works faster, but may miss the optimal result

```
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_)
```

📦 Example:

When a baker is experimenting with different ratios of flour, sugar, and butter, GridSearch tries every combination, while RandomizedSearch tries a few randomly.

Bayesian Optimization (Optional)

Can be performed using libraries like scikit-optimize or **Optuna**. It guides each new trial based on previous results to find the best solution with fewer resources.

🌟 Optuna – Smart Tuning

• Automated hyperparameter optimization
• Learns from past trials (TPE method)
• Supports early stopping (pruning)
• Fast and efficient for complex models

📌 Why Optuna?

Smarter than RandomizedSearch, faster than GridSearch.

```
# 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. Unsupervised Learning

A type of learning used when labeled data is not available — it aims to discover structure or patterns in the data. In other words, the data consists only of inputs, with no output (labels)..

Goal:

- Grouping data (clustering)
- Summarizing or simplifying data (dimensionality reduction)
- Detecting unusual (anomalous) behavior (anomaly detection)

5.1 K-Means Clustering

Groups data into *K* clusters based on similarity.

- Initially, *K* centroids are randomly selected.
- Each data point is assigned to the nearest centroid.
- Centroids are updated, and the process repeats.

📦 **Example:**
Segmenting customers of an e-commerce platform into 3 groups based on spending habits (e.g., low, medium, high spenders).

🔧 **Key Parameters:**

- n_clusters: Number of clustersNumber of clusters
- init: Method for initializing centroids (recommended: 'k-means++')
- max_iter: Maximum number of iterations
- n_init: Number of initializations

```
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 Hierarchical Clustering

Groups data points into a hierarchy based on similarity and forms a tree-like structure called a **dendrogram**.

🔧 **Parameters:**

- linkage: Clustering method ('ward', 'complete', 'average')
- affinity: Distance metric ('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)
Reduces high-dimensional features into fewer **principal components** that retain the most important information.

- Reduces dimensionality without losing essential structure
- Enables easier visualization, faster processing, and better interpretability

🔧 **Parameter:**

- `n_components`: Number of desired components

```
from sklearn.decomposition import PCA


pca = PCA(n_components=2)

X_reduced = pca.fit_transform(X)

print("Explained Variance Ratio:", pca.explained_variance_ratio_)
```

📦 **Example:**
Reducing 100-dimensional medical data to 2 components to visualize clusters of patients in a 2D graph.

5.4 Dimensionality Reduction Methods

By reducing the number of dimensions (features) in the data, we can:

- Make visualization easier
- Reduce noise
- Improve processing speed

🔹 **t-SNE (t-Distributed Stochastic Neighbor Embedding)**

- Ideal for visualization
- Places similar data points close together
- Reduces high-dimensional data to 2D or 3D

```
from sklearn.manifold import TSNE


tsne = TSNE(n_components=2, perplexity=30)

X_tsne = tsne.fit_transform(X)
```

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

- Faster and more structure-preserving than t-SEN
- Preferred for large datasets

```
import umap.umap_ as umap


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

📦 **Example:**
Reducing handwritten digit data (MNIST) from 784 dimensions to 2D to visualize cluster separation.

5.5 Anomaly Detection

The process of detecting unusual, unexpected observations that deviate from general data patterns.

📦 **Examples:**

- Detecting fraud in banking transactions
- Identifying pre-failure behavior in machine sensor data
- Spotting security attacks (e.g., DDoS) in web traffic

Methods:

- Z-score (statistical method)

```
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 (Outlier detection using density-based clustering)

```
from sklearn.cluster import DBSCAN


db = DBSCAN(eps=0.3, min_samples=5)
labels = db.fit_predict(X)
```

6. Time Series

6.1 Introduction to Time Series

Time series are datasets where data points are ordered chronologically. Each observation corresponds to a specific point in time.

📌 **Applications:**

- Economic forecasting (e.g., currency, inflation)
- Inventory and demand forecasting
- Weather prediction
- Energy consumption

6.2 Trend, Seasonality, Stationarity

🔹 **Trend**: Long-term upward or downward movement.
🔹 **Mevsimsellik (Seasonality)**: Recurring patterns at regular intervals (e.g., increased tourism in summer).
🔹 **Durağanlık (Stationarity)**: A time series with constant mean and variance over time. Most time series models require stationary data.

📌 **Stationarity Tests:**

- Augmented Dickey-Fuller (ADF)

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

📌 Transformation Methods:

• Differencing
• Log transformation

6.3 ARIMA, SARIMA Models

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

Parameters:

- p: Number of autoregressive terms (AR)
- d: Degree of differencing (for stationarity)
- q: Number of moving average terms (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)**

An extension of ARIMA that handles seasonality. Parameters:

- (p, d, q) + 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 Forecasting with Prophet

An open-source time series forecasting tool developed by Facebook. It is especially good at handling holidays and seasonality.

🔧 **Key Parameters:**

- growth: "linear" or "logistic" (trend type)
- seasonality_mode: "additive" or "multiplicative"
- holidays: Includes holiday effects in the model

```
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 Performance and Visualization

🔹 Metrics for evaluating forecast accuracy:

- 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()
```

Forecasting in time series modeling refers to predicting future values using models like ARIMA, SARIMA, or Prophet. This is considered the ultimate goal in most time series tasks.

7. Application and Advanced Topics

7.1 Feature Selection and Engineering

🔍 **What is it?**

Feature engineering is the process of transforming existing variables, creating new ones, or removing irrelevant ones to help machine learning models perform better.

🎯 **Goals:**

- Make learning easier for the model
- Reduce complexity
- Improve accuracy

```
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 Pipelines and Automation

A **pipeline** combines tasks like data preprocessing (e.g., scaling, encoding) and model training into a single structure. This provides:

- All steps applied with a single line
- Reduced code repetition
- Lower risk of errors
- Easier optimization with tools like GridSearchCV

```
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))
```

📦 **Saving Models with** joblib After training, a model can be saved to a file and reused later — avoiding the need to retrain it every time..

```
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 Data Imbalance (SMOTE, Class Weights)

Imbalanced classes can negatively impact model performance.

```
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 Real-World Project Development Steps

1. **Problem Definition:** Define the problem and success metric
2. **Data Collection:** Identify and gather data sources
3. **Data Preprocessing:** Handle missing values, outliers, and transformations
4. **Feature Engineering:** Create new features and select important ones
5. **Model Selection:** Try multiple models and choose the best
6. **Hyperparameter Optimization:** Use GridSearchCV or RandomizedSearchCV
7. **Evaluation**: Evaluate using test set and cross-validation
8. **Deployment**: Move the model to a production environment

7.5 Model Deployment

Flask for 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

Used for deploying the model in portable containers.

- Dockerfile defines the environment
- Kubernetes handles scaling and management

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

7.6 Ethics, Fairness, and Bias

Problems:

- **Data Bias**: Prejudices in the training data may be reflected in the model
- **Unfairness**: Systematic errors against certain groups
- **Privacy**: Protecting personal data

Solutions:

- Compare error rates across different groups
- Use balanced datasets
- Apply explainability tools like **LIME** and **SHAP**

📦 **Example:**
Yalnızca A model trained only on resumes from male candidates might systematically reject female applicants — an ethical concern.

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)
```