**WEEK 6(B)**

**Types of Regression Metrics**

Some common regression metrics in scikit-learn with examples

* Mean Absolute Error (MAE)
* Mean Squared Error (MSE)
* R-squared (R²) Score
* Root Mean Squared Error (RMSE)

### ****Mean Absolute Error (MAE)****

In the fields of statistics and machine learning, the [Mean Absolute Error (MAE)](https://www.geeksforgeeks.org/python/how-to-calculate-mean-absolute-error-in-python/) is a frequently employed metric. It's a measurement of the typical absolute discrepancies between a dataset's actual values and projected values.

#### Mathematical Formula

The formula to calculate MAE for a data with "n" data points is: *MAE*=*n*1​∑*i*=1*n*​∣*xi*​−*yi*​∣

Where:

* xi represents the actual or observed values for the i-th data point.
* yi represents the predicted value for the i-th data point.

Example:

from sklearn.metrics import mean\_absolute\_error

true\_values = [2.5, 3.7, 1.8, 4.0, 5.2]

predicted\_values = [2.1, 3.9, 1.7, 3.8, 5.0]

​

mae = mean\_absolute\_error(true\_values, predicted\_values)

print(&quot;Mean Absolute Error:&quot;, mae)

Output:

Mean Absolute Error: 0.22000000000000003

### ****Mean Squared Error (MSE)****

A popular metric in statistics and machine learning is the [Mean Squared Error](https://www.geeksforgeeks.org/python/python-mean-squared-error/) (MSE). It measures the square root of the average discrepancies between a dataset's actual values and projected values. MSE is frequently utilized in regression issues and is used to assess how well predictive models work.

#### Mathematical Formula

For a dataset containing 'n' data points, the MSE calculation formula is:

*MSE*=*n*1​∑*i*=1*n*​(*xi*​−*yi*​)2

Example:

from sklearn.metrics import mean\_squared\_error

​

true\_values = [2.5, 3.7, 1.8, 4.0, 5.2]

predicted\_values = [2.1, 3.9, 1.7, 3.8, 5.0]

​

mse = mean\_squared\_error(true\_values, predicted\_values)

print(&quot;Mean Squared Error:&quot;, mse)

Output:

Mean Squared Error: 0.057999999999999996

R-squared (R²) Score

A statistical metric frequently used to assess the goodness of fit of a regression model is the R-squared (R2) score, also referred to as the coefficient of determination. It quantifies the percentage of the dependent variable's variation that the model's independent variables contribute to. R2 is a useful statistic for evaluating the overall effectiveness and explanatory power of a regression model.

Mathematical Formula

The formula to calculate the R-squared score is as follows:

Evaluating a regression model is crucial to understand how well it predicts continuous outcomes. Here’s a structured approach:

## ****Split Data****

Before evaluating, split your dataset:

python

CopyEdit

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)

This helps test model performance on unseen data.

**WHAT IS MULTI LINEAR REGRESSION:**

**Multilinear Regression** (commonly called **Multiple Linear Regression**) is a statistical technique used to model the relationship between one **dependent variable (target)** and **two or more independent variables (features)**.

### ****. Concept****

* **Linear regression**:  
  Models relationship with one independent variable:

y=b0+b1x+ϵy = b\_0 + b\_1x + \epsilony=b0​+b1​x+ϵ

* **Multilinear regression**:  
  Extends to multiple independent variables:

y=b0+b1x1+b2x2+⋯+bnxn+ϵy = b\_0 + b\_1x\_1 + b\_2x\_2 + \dots + b\_nx\_n + \epsilony=b0​+b1​x1​+b2​x2​+⋯+bn​xn​+ϵ

Where:

* yyy = target variable
* b0b\_0b0​ = intercept
* b1,b2,...,bnb\_1, b\_2, ..., b\_nb1​,b2​,...,bn​ = coefficients for each feature
* x1,x2,...,xnx\_1, x\_2, ..., x\_nx1​,x2​,...,xn​ = independent variables
* ϵ\epsilonϵ = error term

### ****When to Use****

* When you want to **predict a continuous value** using **multiple factors**.
* Examples:
  + Predicting **house price** using size, location, number of rooms.
  + Estimating **salary** using experience, education, age.

### ****3. Assumptions****

* Linear relationship between features and target
* No or little multicollinearity
* Homoscedasticity (constant variance of errors)
* Normally distributed residuals.

### ****4. Python Example****

python

CopyEdit

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

# Sample dataset

data = {

'Size': [1000, 1500, 2000, 2500, 3000],

'Bedrooms': [2, 3, 4, 4, 5],

'Price': [200000, 250000, 300000, 350000, 400000]

}

df = pd.DataFrame(data)

# Features and target

X = df[['Size', 'Bedrooms']]

y = df['Price']

# Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Coefficients

print("Intercept:", model.intercept\_)

print("Coefficients:", model.coef\_)

# Predictions

y\_pred = model.predict(X\_test)

print("Predicted Prices:", y\_pred)

### ✅ ****Interpretation****

* Each coefficient shows how much the target changes when that feature increases by 1 unit, keeping others constant.
* Helps understand **individual impact** of each variable.

## ****2. Common Regression Metrics****

### ****a. Mean Absolute Error (MAE)****

* Measures average magnitude of errors without considering direction.
* Less sensitive to outliers.  
  Formula:

MAE=1n∑∣ytrue−ypred∣MAE = \frac{1}{n} \sum |y\_{true} - y\_{pred}|MAE=n1​∑∣ytrue​−ypred​∣

python

CopyEdit

from sklearn.metrics import mean\_absolute\_error

mae = mean\_absolute\_error(y\_test, y\_pred)

### ****b. Mean Squared Error (MSE)****

* Squares the errors, giving more weight to large errors.
* Useful if you want to penalize big mistakes.  
  Formula:

MSE=1n∑(ytrue−ypred)2MSE = \frac{1}{n} \sum (y\_{true} - y\_{pred})^2MSE=n1​∑(ytrue​−ypred​)2

python

CopyEdit

from sklearn.metrics import mean\_squared\_error

mse = mean\_squared\_error(y\_test, y\_pred)

### ****c. Root Mean Squared Error (RMSE)****

* Square root of MSE.
* Easier to interpret since it’s in the same unit as the target.  
  Formula:

RMSE=MSERMSE = \sqrt{MSE}RMSE=MSE​

python

CopyEdit

import numpy as np

rmse = np.sqrt(mse)

### ****d. R² Score (Coefficient of Determination)****

* Measures how much variance in the target is explained by the model.
* Range: 0 → no fit, 1 → perfect fit.  
  Formula:

R2=1−SSresSStotR^2 = 1 - \frac{SS\_{res}}{SS\_{tot}}R2=1−SStot​SSres​​

python

CopyEdit

from sklearn.metrics import r2\_score

r2 = r2\_score(y\_test, y\_pred)

## ****3. Residual Analysis****

* **Residuals = y\_true - y\_pred**
* Plot residuals to check for patterns.

python

CopyEdit

import matplotlib.pyplot as plt

residuals = y\_test - y\_pred

plt.scatter(y\_pred, residuals)

plt.axhline(0, color='red', linestyle='--')

plt.xlabel("Predicted")

plt.ylabel("Residuals")

plt.title("Residual Plot")

plt.show()

A good model will have residuals randomly scattered around zero.

## ****4. Cross-Validation****

* Helps evaluate model stability.

python

CopyEdit

from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(model, X, y, scoring='r2', cv=5)

print(scores.mean())

## ✅ ****Example Output****

python

CopyEdit

print("MAE:", mae)

print("MSE:", mse)

print("RMSE:", rmse)

print("R² Score:", r2)

Output:

yaml

CopyEdit

MAE: 2.1

MSE: 7.5

RMSE: 2.74

R² Score: 0.89

**COMPLETE EXAMPLE** OF **DATA EXPLORATION → PREPROCESSING → TRAIN/TEST SPLITTING** ON A **CRICKET PLAYER PERFORMANCE DATASET** IN PYTHON.

**1. Sample Dataset**

Assume we have a dataset cricket\_player.csv with these columns:

* Player → Name of player
* Matches → Number of matches played
* Runs → Total runs scored
* Batting\_Avg → Batting average
* Strike\_Rate → Strike rate
* Wickets → Number of wickets taken
* Bowling\_Avg → Bowling average
* Performance\_Score → Overall score (target variable for regression)
* **Create Dataset in Python**
* python
* CopyEdit
* **import pandas as pd**
* **import numpy as np**
* **# Sample data**
* **data = {**
* **'Player': ['Virat', 'Rohit', 'Dhoni', 'Kohli', 'Sachin', 'Sehwag', 'Yuvraj', 'Rahul', 'Jadeja', 'Bumrah'],**
* **'Matches': np.random.randint(50, 250, 10),**
* **'Runs': np.random.randint(1000, 12000, 10),**
* **'Batting\_Avg': np.random.uniform(25, 60, 10),**
* **'Strike\_Rate': np.random.uniform(70, 150, 10),**
* **'Wickets': np.random.randint(0, 150, 10),**
* **'Bowling\_Avg': np.random.uniform(20, 50, 10),**
* **}**
* **# Calculate a synthetic "Performance\_Score"**
* **data['Performance\_Score'] = (**
* **0.4 \* (data['Runs'] / data['Matches']) +**
* **0.3 \* data['Batting\_Avg'] +**
* **0.2 \* data['Strike\_Rate'] / 2 -**
* **0.1 \* data['Bowling\_Avg']**
* **)**
* **df = pd.DataFrame(data)**
* **# Show dataset**
* **print(df)**
* **# Save to CSV**
* **df.to\_csv("cricket\_player.csv", index=False)**
* **print("CSV file 'cricket\_player.csv' created successfully!")**
* **print(df.head())**
* **# Basic info**
* **print(df.info())**
* **# Summary statistics**
* **print(df.describe())**
* **# Check missing values**
* **print(df.isnull().sum())**
* **# Check data types**
* **print(df.dtypes)**
* **# Correlation between features**
* **print(df.corr())**
* **# Visualize distribution of runs**
* **import matplotlib.pyplot as plt**
* **import seaborn as sns**
* **sns.histplot(df['Runs'], bins=20, kde=True)**
* **plt.title("Runs Distribution")**
* **plt.show()**
* **# Pairplot to see feature relationships**
* **sns.pairplot(df[['Matches', 'Runs', 'Batting\_Avg', 'Strike\_Rate', 'Performance\_Score']])**
* **plt.show()**
* **3. Data Preprocessing**
* **Handle missing values**
* **Encode categorical columns**
* **Remove outliers**
* **Normalize or scale features**
* **python**
* **Copy**
* **Edit**
* **# Fill missing values (example: replace with mean)**
* **df['Batting\_Avg'].fillna(df['Batting\_Avg'].mean(), inplace=True)**
* **# Encode player names (if needed)**
* **from sklearn.preprocessing import LabelEncoder**
* **df['Player'] = LabelEncoder().fit\_transform(df['Player'])**
* **# Remove outliers using IQR**
* **Q1 = df['Runs'].quantile(0.25)**
* **Q3 = df['Runs'].quantile(0.75)**
* **IQR = Q3 - Q1**
* **df = df[~((df['Runs'] < (Q1 - 1.5 \* IQR)) | (df['Runs'] > (Q3 + 1.5 \* IQR)))]**
* **# Feature scaling**
* **from sklearn.preprocessing import StandardScaler**
* **scaler = StandardScaler()**
* **numeric\_cols = ['Matches', 'Runs', 'Batting\_Avg', 'Strike\_Rate', 'Wickets', 'Bowling\_Avg']**
* **df[numeric\_cols] = scaler.fit\_transform(df[numeric\_cols])**
* **4. Splitting Data for Modeling**
* **python**
* **Copy**
* **Edit**
* **from sklearn.model\_selection import train\_test\_split**
* **# Features and target**
* **X = df[['Matches', 'Runs', 'Batting\_Avg', 'Strike\_Rate', 'Wickets', 'Bowling\_Avg']]**
* **y = df['Performance\_Score']**
* **# Train-test split**
* **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**
* **print("Training shape:", X\_train.shape)**

**print("Testing shape:", X\_test.shape)**

## ****3. Non-Linear Regression Algorithms****

### ✅ ****a. Decision Tree Regression****

* Splits data into regions and fits constant values.
* Handles non-linear relationships well.

### ✅ ****b. Random Forest Regression****

* Ensemble of decision trees (bagging).
* Reduces overfitting and improves accuracy.

### ✅ ****c. Gradient Boosting Regression****

* Builds trees sequentially to correct previous errors.
* Variants: XGBoost, LightGBM, CatBoost.

### ✅ ****d. Support Vector Regression (SVR)****

* Uses Support Vector Machines for regression.
* Can model complex non-linear relationships with kernel tricks.

Machine Learning, classification is a supervised learning technique used to predict categorical labels (discrete outcomes) for given input data based on prior observations.

1. Definition

Classification algorithms learn patterns from a labeled dataset where the output variable (target) consists of categories (e.g., "spam" or "not spam").

The model then classifies new, unseen data into one of these predefined categories.

2. Examples

Email spam detection → Spam / Not Spam

Medical diagnosis → Healthy / Sick

Image recognition → Cat / Dog / Bird

Sentiment analysis → Positive / Negative / Neutral

3. Types of Classification

✅ a. Binary Classification

Only two possible classes.

Example: Fraudulent transaction (Yes/No).

✅ b. Multiclass Classification

More than two classes.

Example: Classifying fruits as apple, orange, banana.

✅ c. Multilabel Classification

Each instance can belong to multiple labels simultaneously.

Example: A movie can be both Comedy and Romance.

4. Common Classification Algorithms

Logistic Regression

K-Nearest Neighbors (KNN)

Support Vector Machine (SVM)

Decision Trees

Random Forest

Naive Bayes

**Gradient Boosting (XGBoost, LightGBM, CatBoost)**

**Neural Networks**

**5. Evaluation Metrics**

**Accuracy → Overall correctness**

**Precision & Recall → Performance on specific classes**

**F1-score → Balance between precision and recall**

**Confusion Matrix → Detailed class predictions**

**ROC-AUC → Performance on binary classification**

**6. Simple Python Example**

**python**

**Copy**

**Edit**

**from sklearn.datasets import load\_iris**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.metrics import accuracy\_score**

**# Load dataset**

**X, y = load\_iris(return\_X\_y=True)**

**# Split data**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**# Train classifier**

**model = LogisticRegression(max\_iter=200)**

**model.fit(X\_train, y\_train)**

**# Predict**

**y\_pred = model.predict(X\_test)**

**# Accuracy**

**print("Accuracy:", accuracy\_score(y\_test, y\_pred))**

**ChatGPT said:**

**Classification models are supervised machine learning algorithms used to categorize data into predefined classes. Here’s a structured list of the most common classification models:**

**1. Linear Models**

**✅ a. Logistic Regression**

**Uses a sigmoid function to output probabilities between 0 and 1.**

**Best for binary classification.**

**Example: Predicting if a transaction is fraudulent.**

**✅ b. Linear Discriminant Analysis (LDA)**

**Finds a linear combination of features that separates classes.**

**Works well when data follows a normal distribution.**

**2. Instance-Based Models**

**✅ a. K-Nearest Neighbors (KNN)**

**Classifies a point based on the majority class of its k nearest neighbors.**

**Non-parametric and simple to implement.**

**3. Tree-Based Models**

**✅ a. Decision Tree Classifier**

**Splits data into branches based on feature thresholds.**

**Produces interpretable if-else rules.**

**✅ b. Random Forest Classifier**

**Ensemble of multiple decision trees (bagging).**

**Reduces overfitting, improves accuracy.**

**✅ c. Gradient Boosting Classifiers**

**Builds trees sequentially to fix previous errors.**

**Variants: XGBoost, LightGBM, CatBoost.**

**4. Support Vector Machines (SVM)**

**Finds the optimal hyperplane that maximizes margin between classes.**

**Supports linear and non-linear classification using kernels.**

**5. Probabilistic Models**

**✅ a. Naive Bayes**

**Based on Bayes’ theorem with assumption of feature independence.**

**Works well for text classification (e.g., spam detection).**

**6. Neural Network Models**

**✅ a. Artificial Neural Networks (ANN)**

**Layers of interconnected neurons used for complex classification tasks.**

**Can handle large, high-dimensional datasets.**

**✅ b. Convolutional Neural Networks (CNN)**

**Specialized for image classification.**

**✅ c. Recurrent Neural Networks (RNN, LSTM)**

**Used for sequential classification tasks, like sentiment analysis.**

**7. Ensemble Methods**

**Combine multiple classifiers to improve performance:**

**Bagging (Bootstrap Aggregating)**

**Boosting (e.g., AdaBoost, Gradient Boosting)**

**Stacking (meta-model that combines predictions of multiple classifiers)**

**What is a Decision Tree Model?**

A tree-like structure that makes decisions based on feature values.

Splits the dataset into subsets using conditions (rules).

Each internal node represents a feature test, each branch represents the outcome, and each leaf node represents a final prediction (class label or value).

Example (Classification Tree)

If we want to classify whether a cricket player is “Good” or “Average” based on:

Runs scored

Strike rate

The tree might look like:

yaml

Copy

Edit

Runs > 5000?

/ \

Yes No

/ \

Strike rate > 100? "Average"

/ \

Yes No

"Good" "Average"

2. Decision Tree Algorithm

Goal: Select the best feature and threshold at each node to split data such that resulting subsets are as pure as possible (contain mostly one class).

Common Algorithms:

CART (Classification and Regression Trees) → Used in sklearn

ID3 (Iterative Dichotomiser 3) → Uses Information Gain

C4.5 / C5.0 → Improvement over ID3 (uses Gain Ratio)

CHAID → Uses Chi-square statistics.

## ****How to Stop Overfitting in Decision Tree Classifier****

Overfitting happens when a decision tree becomes **too complex**, memorizing the training data instead of generalizing to new data. To **prevent overfitting**, you can:

### ****(a) Set Tree Depth Limits****

* **Parameter:** max\_depth
* This limits how deep the tree can grow, preventing very complex branches.

python

CopyEdit

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(max\_depth=5)

### ****(b) Minimum Samples for Splitting or Leaf Nodes****

* **Parameters:**
  + min\_samples\_split: Minimum samples required to split a node
  + min\_samples\_leaf: Minimum samples required to be at a leaf node

python

CopyEdit

clf = DecisionTreeClassifier(min\_samples\_split=10, min\_samples\_leaf=5)

This avoids creating branches for very small data subsets (which usually leads to overfitting).

### ****(c) Limit Number of Features Used****

* **Parameter:** max\_features
* Reduces the number of features considered at each split, making the tree less complex.

python

CopyEdit

clf = DecisionTreeClassifier(max\_features='sqrt')

### ****(d) Control Node Impurity****

* **Parameter:** max\_leaf\_nodes
* Limits the number of leaf nodes to avoid too many small splits.

### ****(e) Use Cross-Validation****

* Evaluate performance on unseen data to choose the best hyperparameters and avoid overfitting.

## ****2️⃣ What is Pruning in Decision Trees?****

Pruning is a **technique to reduce overfitting** by **removing branches** that have little importance or do not improve model accuracy significantly.

### ****Types of Pruning****

1. **Pre-pruning (Early Stopping):**
   * Stop the tree from growing beyond a certain point using parameters like:
     + max\_depth
     + min\_samples\_split
     + min\_samples\_leaf  
       ✅ This is commonly used in sklearn.
2. **Post-pruning (Cost Complexity Pruning):**
   * Build the full tree first, then **remove weak branches** that add little value.
   * In scikit-learn, this is done using the **ccp\_alpha** parameter (Cost Complexity Pruning).

python

CopyEdit

clf = DecisionTreeClassifier(ccp\_alpha=0.01)

* Larger ccp\_alpha → more pruning → simpler tree.

3. Key Concepts

Entropy (Information Theory): Measures impurity in a dataset.

𝐸𝑛𝑡𝑟𝑜𝑝y=−∑𝑝𝑖log2(𝑝𝑖)Entropy=−∑p i log 2(pi)

Information Gain: Reduction in entropy after a split.

Gini Impurity: Measures how often a randomly chosen element would be incorrectly classified.

𝐺𝑖𝑛𝑖=1−∑𝑝𝑖2Gini=1−∑pi2

​

Splitting: Choosing the best feature and threshold.

Stopping Criteria: Tree stops growing if:

Maximum depth reached

Minimum samples per leaf reached

No further gain from splitting

Pruning: Removing unnecessary branches to reduce overfitting.

4. Python Example

python

Copy

Edit

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

import matplotlib.pyplot as plt

# Load dataset

X, y = load\_iris(return\_X\_y=True)

# Create Decision Tree

model = DecisionTreeClassifier(criterion='gini', max\_depth=3, random\_state=42)

model.fit(X, y)

# Plot tree

plt.figure(figsize=(12, 6))

tree.plot\_tree(model, filled=True, feature\_names=load\_iris().feature\_names, class\_names=load\_iris().target\_names)

plt.show()

✅ Advantages

Easy to understand and visualize

Handles both numerical and categorical data

Requires little data preprocessing

⚠️ Disadvantages

Prone to overfitting

Unstable with small changes in data

Greedy splits may not always lead to the best tree globally