SUPERVISED LEARNING

What is Supervised Learning?

Supervised learning is a type of machine learning where the model is trained using **labeled data**. That means the dataset contains input features (**X**) and a corresponding target/output (**Y**).

The algorithm learns the mapping between input and output:

f(X)=Yf(X) = Yf(X)=Y

Types of Supervised Learning :

Type	Description	Output Type
Classification	Predict discrete class labels	Categorical
Regression	Predict continuous numerical values	Continuous

What is Classification?

Classification is a **supervised learning** technique used to **predict a categorical label**. The model learns from labeled data and assigns a class label to new data.

$$f(X) \rightarrow \text{Category (Label)}$$

Real-Life Examples of Classification:

Application	Description
Email Spam Detection	Classify as "Spam" or "Not Spam"
Disease Diagnosis	Predict "Positive" or "Negative"
Image Recognition	Recognize object category (e.g., "Cat", "Dog")
Sentiment Analysis	Classify review as "Positive" or "Negative"
Credit Risk Assessment	Predict "High Risk", "Medium", or "Low"

Common Classification Algorithms

Algorithm	Description
Logistic Regression	Simple, interpretable model for binary classification
Decision Tree	Tree-based, splits data based on feature conditions
Random Forest	Ensemble of decision trees (robust and accurate)
K-Nearest Neighbors (KNN)	Predicts based on majority label of nearest neighbors
Naive Bayes	Based on Bayes' theorem, good for text classification
Support Vector Machine	Finds best boundary (hyperplane) between classes
Neural Networks (MLP)	Deep models for complex decision boundaries

Evaluation Metrics

Metric	Use Case
Accuracy	Overall correct predictions
Precision	How many predicted positives are actually positive
Recall	How many actual positives were correctly predicted
F1-Score	Harmonic mean of Precision and Recall
Confusion Matrix	Summary of prediction results
ROC-AUC	For binary classifiers, plots TPR vs FPR

Example: Iris Flower Classification (Multiclass)

We'll classify flowers into 3 species using the **Iris dataset**.

Python Code:

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification_report, confusion_matrix


# Load dataset

iris = load_iris()

X, y = iris.data, iris.target


# Train-test split

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


# Model

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)


# Predict

y_pred = clf.predict(X_test)
```

```
# Evaluation

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

print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=iris.target_names))
```

Sample Output:

```
Confusion Matrix:
 [[10  0  0]
  [ 0  8  0]
  [ 0  1 11]]

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.89      1.00      0.94         8
   virginica       1.00      0.92      0.96        12

    accuracy                           0.97        30
```

What is Regression in Machine Learning?

Regression is a **supervised learning** technique where the output (target) variable is **continuous and numeric**.

$$f(X) \rightarrow Y \text{ where } Y \in R$$

It aims to **predict a quantity**, not a class label.

Real-Life Applications of Regression

Use Case	Description
House price prediction	Predict price based on features like size, location
Sales forecasting	Estimate future sales
Temperature prediction	Estimate weather based on historical data
Stock market forecasting	Predict future share prices
Medical cost estimation	Predict hospital charges

Types of Regression

Type	Description	Output
Linear Regression	Linear relationship between X and Y	Continuous
Polynomial Regression	Non-linear (higher-degree polynomials)	Continuous
Ridge/Lasso Regression	Regularized linear regression	Continuous
Logistic Regression	Despite name, it's for classification	Categorical
SVR (Support Vector Regression)	Uses SVM for regression	Continuous
Decision Tree Regression	Non-linear regression using tree splits	Continuous

Evaluation Metrics for Regression

Metric	Formula/Interpretation
Mean Squared Error (MSE)	Average squared difference between actual & predicted
Mean Absolute Error (MAE)	Average absolute difference
Root Mean Squared Error (RMSE)	Square root of MSE
R^2 Score	Proportion of variance explained by the model
Adjusted R^2	R^2 adjusted for number of predictors

Example 1: Simple Linear Regression

Predict a student's score based on hours studied.

Python Code:

```
import numpy as np

from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt


# Input: Hours studied

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

# Output: Scores obtained

y = np.array([50, 55, 65, 70, 75])


# Model

model = LinearRegression()

model.fit(X, y)
```

```
# Predict

predicted = model.predict([[6]])

print("Predicted score for 6 hours of study:", predicted[0])


# Visualization

plt.scatter(X, y, color='blue')

plt.plot(X, model.predict(X), color='red')

plt.xlabel("Hours Studied")

plt.ylabel("Score")

plt.title("Simple Linear Regression")

plt.show()
```

Output:

Predicted score for 6 hours of study: 80.0

Example 2: Multiple Linear Regression with Dataset

Using **California Housing Dataset**.

Python Code:

```
from sklearn.datasets import fetch_california_housing

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score


# Load dataset

data = fetch_california_housing()

X = data.data
```

```
y = data.target

# 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)

# Predict
y_pred = model.predict(X_test)

# Evaluation
print("MSE:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

Output:

MSE: 0.5558

R^2 Score: 0.61

1. Regression Metrics

Regression metrics are used when the output is a **continuous value**.

A. Mean Absolute Error (MAE)

- **Definition**: Average of the absolute differences between actual and predicted values.
- **Use case**: Easy to interpret, less sensitive to outliers.

Example:

```
from sklearn.metrics import mean_absolute_error

y_true = [3, -0.5, 2, 7]

y_pred = [2.5, 0.0, 2, 8]

print("MAE:", mean_absolute_error(y_true, y_pred))

Output:

MAE: 0.5
```

B. Mean Squared Error (MSE)

- **Definition**: Average of the squared differences between actual and predicted values.
- **Use case**: Penalizes larger errors more than MAE.

Example:

```
from sklearn.metrics import mean_squared_error

print("MSE:", mean_squared_error(y_true, y_pred))

Output:

MSE: 0.375
```

C. Root Mean Squared Error (RMSE)

- **Definition**: Square root of MSE. Has the same units as the target variable.

Example:

```
import numpy as np

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

print("RMSE:", rmse)
```

Output:

RMSE: 0.612372

D. R² Score (Coefficient of Determination)

- **Definition**: Measures how well predictions approximate actual values. Closer to 1 is better.
- **Use case**: Measures the percentage of variance explained by the model.

Example:

```
from sklearn.metrics import r2_score

print("R² Score:", r2_score(y_true, y_pred))
```

Output:

R² Score: 0.9486

2. Classification Metrics

Classification metrics are used when the output is **categorical**.

A. Accuracy

- **Definition**: Proportion of correct predictions.
- **Use case**: Simple and effective when classes are balanced.

Example:

```
from sklearn.metrics import accuracy_score

y_true = [1, 0, 1, 1, 0]

y_pred = [1, 0, 1, 0, 0]
```

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

Output:

Accuracy: 0.8

B. Precision

- **Definition**: Proportion of predicted positives that are actually positive.
- **Use case**: Important in applications like spam detection or fraud detection.

Example:

from sklearn.metrics import precision_score

print("Precision:", precision_score(y_true, y_pred))

Output:

Precision: 1.0

C. Recall (Sensitivity or TPR)

- **Definition**: Proportion of actual positives that are correctly identified.
- **Use case**: Important in medical tests (don't miss real positives).

Example:

from sklearn.metrics import recall_score

print("Recall:", recall_score(y_true, y_pred))

Output:

Recall: 0.6667

D. F1 Score

- **Definition**: Harmonic mean of precision and recall.
- **Use case**: Best when you need balance between Precision and Recall.

Example:

from sklearn.metrics import f1_score

print("F1 Score:", f1_score(y_true, y_pred))

Output:

F1 Score: 0.8

E. Confusion Matrix

- **Definition**: Table that shows TP, TN, FP, FN.
- **Use case**: Gives a complete picture of prediction performance.

Example:

from sklearn.metrics import confusion_matrix

print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))

Output:

[[2 0]

 [1 2]]

F. ROC Curve and AUC (for binary classifiers)

- **ROC Curve**: Plots TPR vs. FPR at different thresholds.
- **AUC (Area Under Curve)**: Measures overall ability to rank positive examples higher.

Example:

from sklearn.metrics import roc_auc_score

y_probs = [0.8, 0.3, 0.9, 0.4, 0.2]

print("ROC AUC Score:", roc_auc_score(y_true, y_probs))

Output:

ROC AUC Score: 1.0

Summary Table

Metric	Type	Use Case
MAE	Regression	Simple average error
MSE	Regression	Penalizes large errors
RMSE	Regression	Same unit as target
R^2 Score	Regression	Variance explained
Accuracy	Classification	Balanced datasets
Precision	Classification	Important for False Positives
Recall	Classification	Important for False Negatives
F1 Score	Classification	Balance between P and R
Confusion Matrix	Classification	Visualize performance
ROC-AUC	Classification	Ranking and probability-based

What is Cross-Validation in Machine Learning?

Cross-validation is a powerful statistical method used to **evaluate the performance and generalization ability** of a machine learning model.

Instead of training and testing the model just once, cross-validation splits the dataset into multiple parts to ensure the model is **not overfitting** and performs well on **unseen data**.

Why Use Cross-Validation?

- Ensures **more reliable model evaluation**
- Reduces the risk of **overfitting/underfitting**
- Makes use of **all available data** for training and testing
- Helps in **hyperparameter tuning**

Types of Cross-Validation

Method	Description
Hold-Out Validation	Split into train and test (e.g., 80/20); simple but may be biased
K-Fold Cross-Validation	Split into `k` parts (folds); each fold is used once for testing
Stratified K-Fold	Ensures each fold maintains the same class ratio (used in classification)
Leave-One-Out (LOOCV)	Each sample is used once as test, very computationally expensive
Repeated K-Fold	K-fold run multiple times with different splits
Time Series Split	Used for time-dependent data (e.g., stock prices)

K-Fold Cross-Validation Explained

K-Fold CV splits the data into **K subsets**:

1. Train the model on K-1 folds
2. Test on the remaining fold
3. Repeat this process K times
4. Average the scores

Python Example using K-Fold

Dataset: Iris (Classification)

```
from sklearn.datasets import load_iris

from sklearn.model_selection import KFold, cross_val_score

from sklearn.tree import DecisionTreeClassifier

# Load dataset

X, y = load_iris(return_X_y=True)

# Initialize model

model = DecisionTreeClassifier()

# K-Fold CV
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=1)

scores = cross_val_score(model, X, y, cv=kf)

print("Fold-wise Accuracy:", scores)

print("Mean Accuracy:", scores.mean())
```

Output:

```
Fold-wise Accuracy: [0.9667 0.9333 0.9    1.     0.9333]

Mean Accuracy: 0.9467
```

Stratified K-Fold (for classification imbalance)

This ensures each fold has a balanced distribution of class labels.

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

scores = cross_val_score(model, X, y, cv=skf)

print("Stratified Fold Scores:", scores)

print("Average Accuracy:", scores.mean())
```

What is Feature Scaling in Machine Learning?

Feature scaling is the process of **normalizing or standardizing** the range of independent variables (features) in your dataset.

Many machine learning algorithms (especially those using distance or gradient-based calculations) **assume that all features are on the same scale**. If they're not, it can lead to:

- Poor model performance
- Slow convergence
- Biased results toward features with larger ranges

Why is Feature Scaling Important?

Without Scaling	With Scaling
Features with large values dominate	All features contribute equally
Model may converge slowly or poorly	Faster, more stable convergence
Distorted distance metrics	Accurate similarity/distance

Common Feature Scaling Techniques

1. Min-Max Scaling (Normalization)

- **Range**: [0, 1]

Use case: When you want a bounded scale (e.g., image pixel data)

Program:

```
from sklearn.preprocessing import MinMaxScaler

import numpy as np

X = np.array([[1], [5], [10]])

scaler = MinMaxScaler()

scaled = scaler.fit_transform(X)

print(scaled)
```

Output:

```
[[0.  ]

 [0.444]

 [1.  ]]
```

2. Standardization (Z-score Scaling)

- **Mean**: 0, **Standard Deviation**: 1

Use case: Suitable for most ML models (SVM, Linear Models, NN)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled = scaler.fit_transform(X)

print(scaled)

Output:

[[-1.069]

 [0.267]

 [0.802]]

3. MaxAbsScaler

- Scales data by dividing by the maximum absolute value.
- Range: [-1, 1]
- Good for sparse data (e.g., TF-IDF vectors)

Program:

from sklearn.preprocessing import MaxAbsScaler

scaler = MaxAbsScaler()

scaled = scaler.fit_transform(X)

print(scaled)

CASE STUDY:

Churn Prediction System

What Is Customer Churn?

Churn refers to the percentage of customers who stop using a service over a specific period.

- **Goal**: Predict which customers are likely to **churn** so the business can take preventive action (e.g., offering discounts, improving service).

Python Program:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix

df = pd.read_csv("Telco-Customer-Churn.csv")


# Drop customer ID

df.drop('customerID', axis=1, inplace=True)


# Convert TotalCharges to numeric

df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

df.fillna(0, inplace=True)
```

```
# Encode categorical variables

for col in df.select_dtypes('object'):

    if col != 'Churn':

        df[col] = LabelEncoder().fit_transform(df[col])


# Encode target

df['Churn'] = df['Churn'].map({'No': 0, 'Yes': 1})

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

y = df['Churn']


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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

model = RandomForestClassifier(n_estimators=100, random_state=42)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

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

print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Output:

Confusion Matrix:

```
 [[947  92]

  [165 205]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.85	0.91	0.88	1039
1	0.69	0.55	0.61	370

Data Link:

https://www.kaggle.com/datasets/blastchar/telco-customer-churn

Please download data from above link and proceed with above code.