Introduction to Classification

Classification is the process of categorizing data or objects into **predefined classes** or **categories** based on their features or attributes.

In **Machine Learning**, classification is a type of **supervised learning** where an algorithm is trained on a labeled dataset to predict the class or category of new, unseen data.

Objective of Classification

The main goal of classification is to build a model that can accurately assign a label or category to a new observation based on its features.

Example

- **Problem**: Classify images as either "dogs" or "cats."
- **Dataset**: Labeled images of dogs and cats.
- **Features**: Attributes such as color, texture, and shape.
- **Goal**: Predict whether a new, unseen image belongs to the "dog" or "cat" class.

Types of Classification

1. Binary Classification

In binary classification, the task is to classify the input into one of **two possible classes**.

Example

- **Scenario**: Predict if a person has a disease based on their health data.
- **Classes**:
  - Positive Class: The person has the disease.
  - Negative Class: The person does not have the disease.
- **Features**: Blood pressure, cholesterol level, age, etc.

Patient ID	Blood Pressure (mmHg)	Cholesterol (mg/dL)	Disease (Yes/No)
1	120	200	Yes
2	130	180	No
3	140	250	Yes
4	115	170	No

2. Multiclass Classification

In multiclass classification, the task is to classify the input into one of **several possible classes**.

Example

- **Scenario**: Classify flower species based on petal and sepal dimensions.
- **Classes**: Iris Setosa, Iris Versicolor, Iris Virginica.
- **Features**: Petal length, petal width, sepal length, sepal width.

Sample ID	Petal Length (cm)	Petal Width (cm)	Species
1	1.4	0.2	Iris Setosa
2	4.7	1.4	Iris Versicolor
3	5.5	2.1	Iris Virginica
4	1.5	0.3	Iris Setosa

Key Takeaways

1. **Classification** involves predicting predefined categories or labels based on input features.

2. **Binary Classification** deals with two possible classes, while **Multiclass Classification** handles multiple categories.

3. Real-world applications include **spam detection**, **medical diagnosis**, and **image classification**.

By understanding the types and examples of classification, you can better design and implement models to solve a wide range of problems in machine learning.

Classification Algorithms

Classification algorithms are used to predict the category or class of a given observation based on its features. These algorithms can be broadly categorized into **Linear** and **Non-linear** classifiers, with a special mention for **Ensemble learning methods**.

---

1. Linear Classifiers

Definition

Linear classifiers create a **linear decision boundary** between classes. They are simple, computationally efficient, and work well when the data is linearly separable.

Examples of Linear Classifiers

1. **Logistic Regression**
   - **Description**: Predicts the probability of a binary class outcome using a logistic function.
   - **Use Case**: Spam detection.
2. **Support Vector Machines (SVM) with Kernel = 'linear'**
   - **Description**: Maximizes the margin between classes with a linear boundary.
   - **Use Case**: Text classification.
3. **Single-Layer Perceptron**
   - **Description**: A basic neural network layer that classifies linearly separable data.
   - **Use Case**: Predicting customer churn.
4. **Stochastic Gradient Descent (SGD) Classifier**
   - **Description**: Optimizes the model by minimizing the loss function using stochastic gradient descent.
   - **Use Case**: Large-scale classification tasks.

---

2. Non-linear Classifiers

Definition

Non-linear classifiers create a **non-linear decision boundary**, allowing them to capture complex relationships between features and target variables.

Examples of Non-linear Classifiers

1. **K-Nearest Neighbours (KNN)**
   - **Description**: Classifies data points based on the majority vote of their k-nearest neighbors.
   - **Use Case**: Handwritten digit recognition.
2. **Kernel SVM**
   - **Description**: Extends SVM with non-linear kernels like RBF or polynomial.
   - **Use Case**: Image classification.
3. **Naive Bayes**
   - **Description**: A probabilistic classifier based on Bayes' theorem.
   - **Use Case**: Sentiment analysis.
4. **Decision Tree Classification**
   - **Description**: Splits the data into subsets based on feature values, creating a tree-like structure.
   - **Use Case**: Predicting customer segmentation.

---

3. Ensemble Learning Classifiers

Definition

Ensemble learning combines predictions from multiple models to improve overall accuracy and robustness.

Examples of Ensemble Classifiers

1. **Random Forests**
   - **Description**: An ensemble of decision trees built using bootstrapped datasets.
   - **Use Case**: Loan approval prediction.
2. **AdaBoost**
   - **Description**: Combines weak classifiers iteratively to create a strong classifier.
   - **Use Case**: Fraud detection.
3. **Bagging Classifier**
   - **Description**: Aggregates predictions from multiple base models trained on random subsets of the data.
   - **Use Case**: Credit scoring.
4. **Voting Classifier**
   - **Description**: Combines predictions from multiple classifiers using majority voting.
   - **Use Case**: Diagnosing diseases.
5. **ExtraTrees Classifier**
   - **Description**: Similar to Random Forest but uses random thresholds for splitting.
   - **Use Case**: Stock market predictions.

---

4. Multi-layer Artificial Neural Networks (ANN)

Description

ANNs are a class of deep learning models consisting of multiple layers of interconnected nodes (neurons). They can handle highly complex and non-linear relationships.

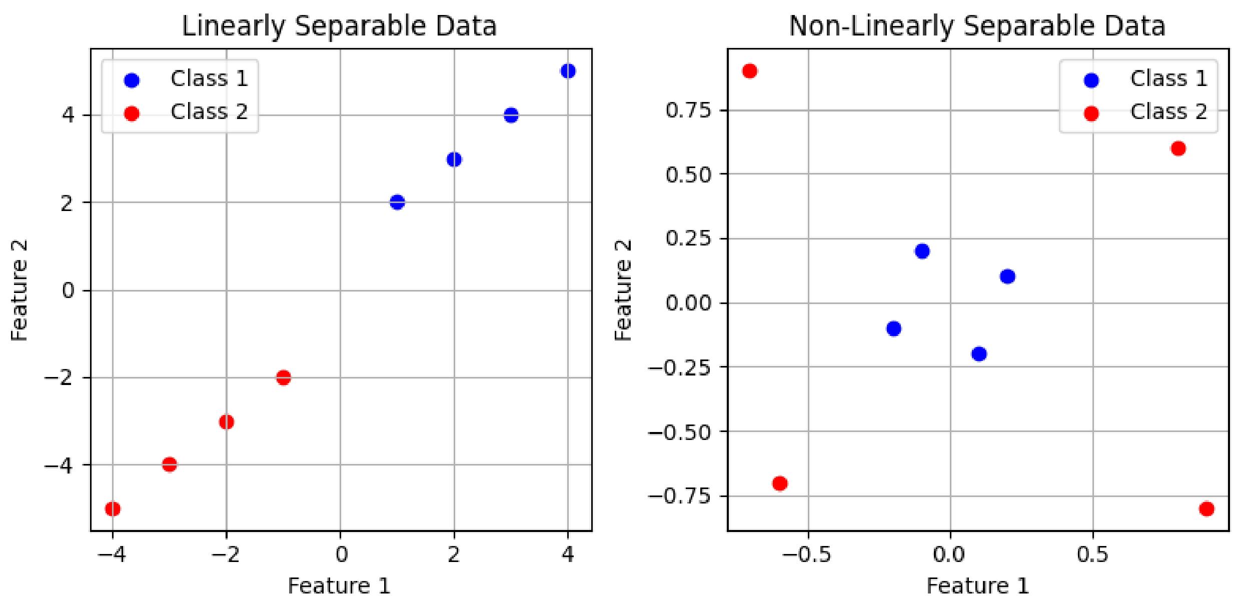
- **Use Case**: Image recognition, speech recognition, and natural language processing.

Choosing the Right Classifier

Scenario	Recommended Algorithm
Linear relationships in data	Logistic Regression, Linear SVM
Complex non-linear relationships	Kernel SVM, Decision Trees
Robust performance across large datasets	Random Forest, AdaBoost
Handling imbalanced datasets	Ensemble methods (e.g., Bagging)
High-dimensional data	Naive Bayes, Neural Networks

By understanding the strengths of each classification algorithm, you can select the most appropriate one for your problem domain and dataset characteristics. ▯

```
# Import necessary libraries
import matplotlib.pyplot as plt

# Predefined linearly separable data
linear_x1 = [[1, 2], [2, 3], [3, 4], [4, 5]]  # Class 1 points
linear_x2 = [[-1, -2], [-2, -3], [-3, -4], [-4, -5]]  # Class 2 points

# Predefined non-linearly separable data
nonlinear_x1 = [[0.2, 0.1], [0.1, -0.2], [-0.2, -0.1], [-0.1, 0.2]]  #
Class 1 points (inside a circle)
nonlinear_x2 = [[0.8, 0.6], [-0.7, 0.9], [0.9, -0.8], [-0.6, -0.7]]  #
Class 2 points (outside the circle)

# Convert to separate x and y coordinates for easier plotting
linear_x1 = list(zip(*linear_x1))
linear_x2 = list(zip(*linear_x2))
nonlinear_x1 = list(zip(*nonlinear_x1))
nonlinear_x2 = list(zip(*nonlinear_x2))

# Plot linearly separable data
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.scatter(linear_x1[0], linear_x1[1], color='blue', label='Class 1')
plt.scatter(linear_x2[0], linear_x2[1], color='red', label='Class 2')
plt.title("Linearly Separable Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)

# Plot non-linearly separable data
plt.subplot(1, 2, 2)
plt.scatter(nonlinear_x1[0], nonlinear_x1[1], color='blue',
label='Class 1')
plt.scatter(nonlinear_x2[0], nonlinear_x2[1], color='red',
label='Class 2')
plt.title("Non-Linearly Separable Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



Evaluating a Classification Model

Evaluating a classification model is a critical step in machine learning to assess its **performance** and **generalization ability** on unseen data. Several metrics can be used for this purpose, depending on the specific problem and requirements.

---

1. Popular Evaluation Metrics

1.1 Accuracy

Definition:
Accuracy is the ratio of the number of correct predictions to the total number of predictions.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

- **Strengths**: Simple and intuitive.
- **Limitations**: Can be misleading in imbalanced datasets.

Example:

Suppose we have a spam classification task with 1,000 emails:

- 950 are **not spam** (Negative class).
- 50 are **spam** (Positive class).

If the model predicts all emails as **not spam**, the accuracy will be:

$$\text{Accuracy} = \frac{950}{1000} = 95\%$$

Issue: Despite high accuracy, the model fails to identify any spam emails, making it unsuitable for this task.

---

2. Confusion Matrix

The **Confusion Matrix** groups classification results into four categories:

Definitions:

- **True Positive (TP)**: Correctly predicted as positive.
- **True Negative (TN)**: Correctly predicted as negative.
- **False Positive (FP)**: Predicted positive but is actually negative.
- **False Negative (FN)**: Predicted negative but is actually positive.

Type 1 and Type 2 Errors in Classification

- **Type 1 Error (False Positive)**:
  Occurs when the model incorrectly predicts a negative instance as positive.
  - **Example**: A healthy patient is diagnosed with a disease.
  - **Impact**: Can lead to unnecessary actions or treatments.
- **Type 2 Error (False Negative)**:
  Occurs when the model incorrectly predicts a positive instance as negative.
  - **Example**: A patient with a disease is diagnosed as healthy.
  - **Impact**: Critical in cases like medical diagnosis, as it may delay necessary treatment.

---

3. Precision

Definition:
Precision measures the proportion of true positives out of all positive predictions:

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

- **Focus**: Highlights how many predicted positives are actually correct.
- **Use Case**: Important when the cost of **false positives** is high.

Example:
In spam detection, Precision measures how many emails flagged as spam are actually spam.

---

4. Recall (Sensitivity)

Definition:
Recall (also known as Sensitivity) measures the proportion of true positives out of all actual positives:

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

- **Focus**: Highlights how many actual positives are correctly identified.
- **Use Case**: Important when the cost of **false negatives** is high.

Example:
In spam detection, Recall measures how many actual spam emails are correctly detected.

---

5. Precision vs Recall

Tradeoff:

- Improving **Precision** often reduces **Recall**, and vice versa.
- The choice depends on the specific task:
  - For spam detection, prioritize **Precision** to avoid important emails being flagged as spam.
  - For disease detection, prioritize **Recall** to minimize missed diagnoses.

---

6. F1 Score

Definition:
The **F1 Score** provides a balance between Precision and Recall:

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

- **Use Case**: Useful when both Precision and Recall are equally important.

Example:
For a model with Precision = 0.8 and Recall = 0.6:

$$F1 = 2 \times \frac{0.8 \times 0.6}{0.8 + 0.6} = 2 \times \frac{0.48}{1.4} = 0.686$$

---

7. Choosing the Right Metric

- **Accuracy**: Suitable for balanced datasets.
- **Precision**: Prioritize when false positives have higher costs.
- **Recall**: Prioritize when false negatives have higher costs.
- **F1 Score**: Use when you need a balance between Precision and Recall.

Summary

- **Accuracy** is simple but insufficient for imbalanced datasets.
- The **Confusion Matrix** helps understand the distribution of predictions.
- Metrics like **Precision**, **Recall**, and **F1 Score** provide deeper insights into model performance.
- Choosing the right metric depends on the specific task and its requirements.

```python
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score

# Binary vectors
y_true = np.array([1, 0, 1, 1, 0, 0, 1, 1])  # Actual labels
y_pred = np.array([1, 0, 0, 1, 1, 0, 1, 0])  # Predicted labels

# Confusion Matrix
cm = confusion_matrix(y_true, y_pred)
```

```
tn, fp, fn, tp = cm.ravel()
print("Confusion Matrix:")
print(cm)
print(f"True Negative (TN): {tn}, False Positive (FP): {fp}, False
Negative (FN): {fn}, True Positive (TP): {tp}")

# Accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"\nAccuracy: {accuracy:.2f}")

# Precision
precision = precision_score(y_true, y_pred)
print(f"Precision: {precision:.2f}")

# Recall
recall = recall_score(y_true, y_pred)
print(f"Recall: {recall:.2f}")

# F1 Score
f1 = f1_score(y_true, y_pred)
print(f"F1 Score: {f1:.2f}")
```

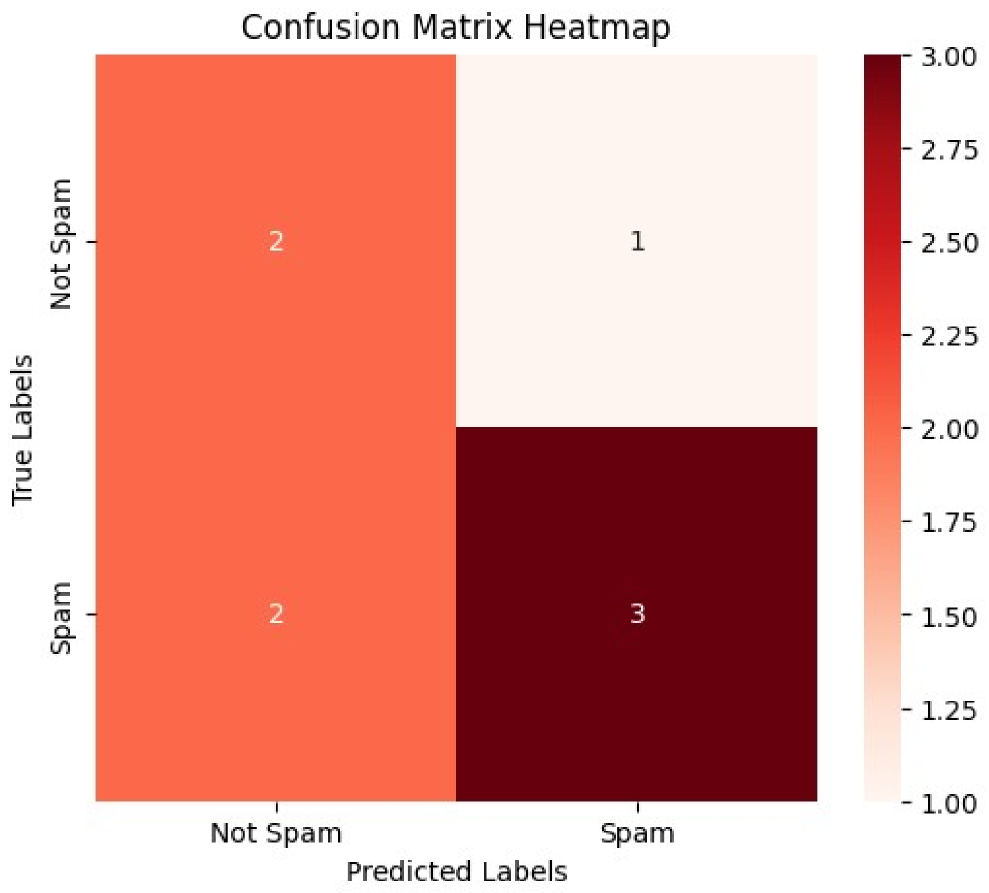
```
Confusion Matrix:
[[2 1]
 [2 3]]
True Negative (TN): 2, False Positive (FP): 1, False Negative (FN): 2,
True Positive (TP): 3

Accuracy: 0.62
Precision: 0.75
Recall: 0.60
F1 Score: 0.67
```

```
from sklearn.metrics import classification_report

# Generate Classification Report
report = classification_report(y_true, y_pred, target_names=['Class
0', 'Class 1'])
print("Classification Report:\n")
print(report)
```

```
Classification Report:

              precision    recall  f1-score   support

     Class 0       0.50      0.67      0.57         3
     Class 1       0.75      0.60      0.67         5

    accuracy                           0.62         8
   macro avg       0.62      0.63      0.62         8
```

```
weighted avg       0.66      0.62      0.63         8
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Example data
y_true = np.array([1, 0, 1, 1, 0, 0, 1, 1])  # Actual labels
y_pred = np.array([1, 0, 0, 1, 1, 0, 1, 0])  # Predicted labels

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Plot confusion matrix heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=['Not
Spam', 'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```



Multi-Class Classification

When we have more than two classes (e.g., Class 0, Class 1, Class 2), classification tasks become **multi-class classification** problems. The metrics for binary classification like **Accuracy**, **Precision**, **Recall**, and **F1 Score** can still be applied, but with some modifications.

---

1. Dataset Example

Let's consider a **customer support email classification** problem with three classes:

- **Class 0**: Shipping
- **Class 1**: Returns
- **Class 2**: Tracking

Below is the example dataset showing **Actual vs Predicted** values:

---

2. Confusion Matrix

For multi-class problems, the confusion matrix becomes a **3x3 matrix**. Each row represents the **actual class**, and each column represents the **predicted class**.

---

3. Accuracy

Accuracy is calculated as:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

- Correct Predictions = (3 + 4 + 3 = 10)
- Total Predictions = (15)

Accuracy:

$$\text{Accuracy} = \frac{10}{15} = 67\,\%$$

---

4. Precision, Recall, and F1 Score (Per Class)

Class 0 - Shipping

- **Precision**:

$$\text{Precision} = \frac{\text{TP}}{\text{TP + FP}} = \frac{3}{3+1} = 75\,\%$$

- **Recall**:

$$\text{Recall} = \frac{\text{TP}}{\text{TP + FN}} = \frac{3}{3+2} = 60\,\%$$

- **F1 Score**:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 67\,\%$$

Class 1 - Returns

- **Precision**:

$$\text{Precision} = \frac{4}{4+2} = 67\,\%$$

- **Recall**:

$$\text{Recall} = \frac{4}{4+1} = 80\,\%$$

- **F1 Score**:

$$F\,1 = 2 \times \frac{67 \times 80}{67+80} = 79\,\%$$

Class 2 - Tracking

- **Precision**:

$$\text{Precision} = \frac{3}{3+2} = 60\,\%$$

- **Recall**:

$$\text{Recall} = \frac{3}{3+2} = 60\,\%$$

- **F1 Score**:

$$F\,1 = 2 \times \frac{60 \times 60}{60+60} = 53\,\%$$

---

5. Macro-Average Precision, Recall, and F1 Score

When dealing with multi-class problems, we use **Macro-Average** to summarize the metrics across all classes.

Macro-Average Precision:

$$\text{Macro Precision} = \frac{\text{Precision}_0 + \text{Precision}_1 + \text{Precision}_2}{3}$$

$$\text{Macro Precision} = \frac{75+67+60}{3} = 67.3\,\%$$

Macro-Average Recall:

$$\text{Macro Recall} = \frac{\text{Recall}_0 + \text{Recall}_1 + \text{Recall}_2}{3}$$

$$\text{Macro Recall} = \frac{60+80+60}{3} = 66.7\,\%$$

Macro-Average F1 Score:

$$\text{Macro F1} = \frac{F1_0 + F1_1 + F1_2}{3}$$

$$\text{Macro F1} = \frac{67 + 79 + 53}{3} = 66.3\%$$

---

6. Summary Table

Class	Precision	Recall	F1 Score
0 - Shipping	75%	60%	67%
1 - Returns	67%	80%	79%
2 - Tracking	60%	60%	53%
Macro-Average	**67.3%**	**66.7%**	**66.3%**

Conclusion

1. **Confusion Matrix** helps in understanding misclassifications for each class.

2. **Precision**, **Recall**, and **F1 Score** can be computed per class.

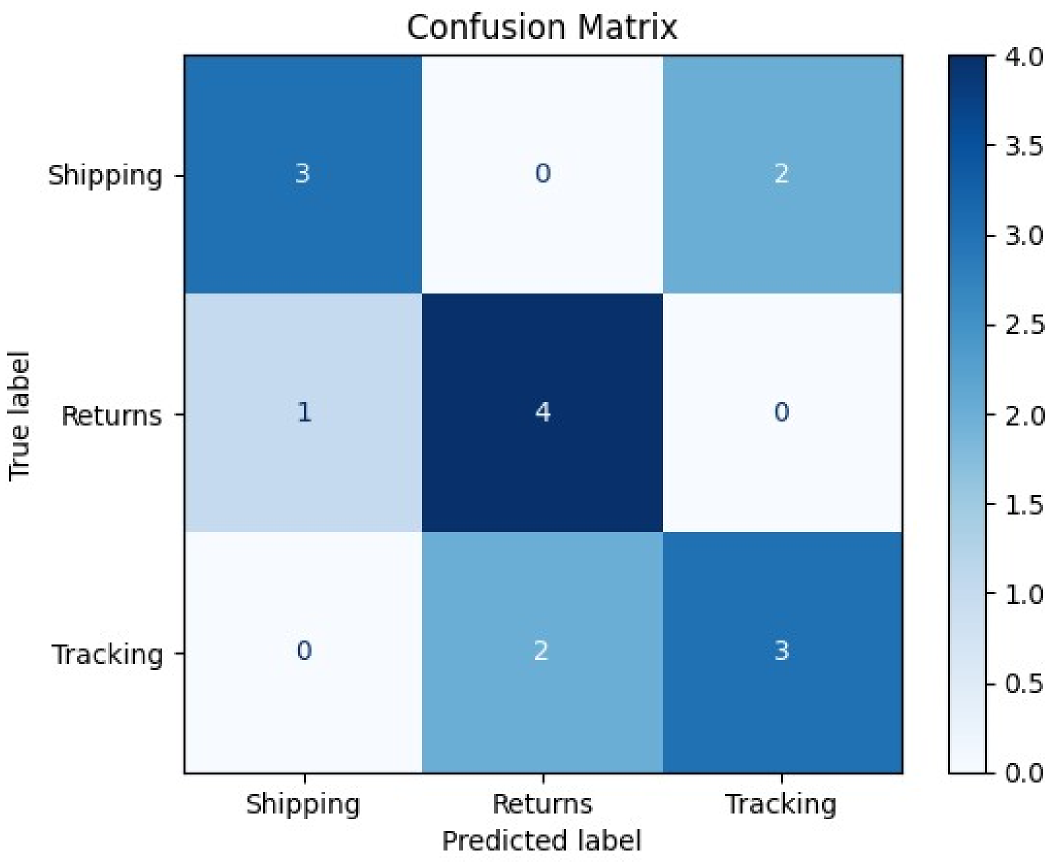
3. **Macro-Average** provides a single value for each metric, summarizing overall performance.

These metrics are critical in evaluating multi-class classification models, ensuring a balanced understanding of their strengths and weaknesses.

```python
# Import libraries
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Define actual and predicted labels
y_true = ["Shipping", "Shipping", "Shipping", "Shipping", "Shipping",
          "Returns", "Returns", "Returns", "Returns", "Returns",
          "Tracking", "Tracking", "Tracking", "Tracking", "Tracking"]

y_pred = ["Shipping", "Shipping", "Tracking", "Shipping", "Tracking",
          "Returns", "Returns", "Returns", "Shipping", "Returns",
          "Returns", "Returns", "Tracking", "Tracking", "Tracking"]

# Unique class labels
class_labels = ["Shipping", "Returns", "Tracking"]
```

```
# Compute confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred, labels=class_labels)
print("Confusion Matrix:\n", conf_matrix)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
display_labels=class_labels)
disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()

# Calculate Accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Calculate Precision, Recall, and F1 Score (Per Class and Macro-
Average)
precision = precision_score(y_true, y_pred, average=None,
labels=class_labels)
recall = recall_score(y_true, y_pred, average=None,
labels=class_labels)
f1 = f1_score(y_true, y_pred, average=None, labels=class_labels)

macro_precision = precision_score(y_true, y_pred, average='macro')
macro_recall = recall_score(y_true, y_pred, average='macro')
macro_f1 = f1_score(y_true, y_pred, average='macro')

# Display Metrics
print("\nClass-wise Metrics:")
for i, label in enumerate(class_labels):
    print(f"Class '{label}' -> Precision: {precision[i]*100:.2f}%,
Recall: {recall[i]*100:.2f}%, F1: {f1[i]*100:.2f}%")

print("\nMacro-Averaged Metrics:")
print(f"Precision: {macro_precision*100:.2f}%")
print(f"Recall: {macro_recall*100:.2f}%")
print(f"F1 Score: {macro_f1*100:.2f}%")
```

```
Confusion Matrix:
 [[3 0 2]
 [1 4 0]
 [0 2 3]]
```



```
Accuracy: 0.67

Class-wise Metrics:
Class 'Shipping' -> Precision: 75.00%, Recall: 60.00%, F1: 66.67%
Class 'Returns' -> Precision: 66.67%, Recall: 80.00%, F1: 72.73%
Class 'Tracking' -> Precision: 60.00%, Recall: 60.00%, F1: 60.00%

Macro-Averaged Metrics:
Precision: 67.22%
Recall: 66.67%
F1 Score: 66.46%

# Import libraries
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Define actual and predicted labels
y_true = ["Shipping", "Shipping", "Shipping", "Shipping", "Shipping",
          "Returns", "Returns", "Returns", "Returns", "Returns",
          "Tracking", "Tracking", "Tracking", "Tracking", "Tracking"]

y_pred = ["Shipping", "Shipping", "Tracking", "Shipping", "Tracking",
          "Returns", "Returns", "Returns", "Shipping", "Returns",
          "Returns", "Returns", "Tracking", "Tracking", "Tracking"]

# Unique class labels
class_labels = ["Shipping", "Returns", "Tracking"]

# Generate Classification Report
print("Classification Report:")
print(classification_report(y_true, y_pred,
target_names=class_labels))

Classification Report:
              precision    recall  f1-score   support

    Shipping       0.67      0.80      0.73         5
     Returns       0.75      0.60      0.67         5
    Tracking       0.60      0.60      0.60         5

    accuracy                           0.67        15
   macro avg       0.67      0.67      0.66        15
weighted avg       0.67      0.67      0.66        15
```

Additional Metrics for Evaluating Classification Models

In addition to Accuracy, Precision, Recall, and F1-Score, there are other important metrics for evaluating classification models. These include **Specificity**, **ROC-AUC Curve**, and **Log Loss**. Each metric provides unique insights into model performance.

---

1. Specificity

Definition

Specificity measures the proportion of **actual negatives** that are correctly identified as negative. It is also known as the **True Negative Rate**.

$$\text{Specificity} = \frac{\text{True Negatives (TN)}}{\text{True Negatives (TN)} + \text{False Positives (FP)}}$$

Key Points

- Specificity complements Recall (True Positive Rate) to give a holistic understanding of model performance.

- A high specificity means the model has fewer **False Positives**.

When to Use

Specificity is particularly useful when correctly identifying negatives is critical.

- **Example**: In fraud detection, it is important to minimize False Positives to avoid flagging legitimate transactions.

---

2. ROC-AUC Curve

Definition

- The **ROC (Receiver Operating Characteristic)** curve is a graphical plot that shows the tradeoff between the **True Positive Rate (Recall)** and the **False Positive Rate** as the decision threshold changes.

- The **AUC (Area Under the Curve)** represents the probability that the model will rank a randomly chosen positive instance higher than a randomly chosen negative one.

$$\text{False Positive Rate} = 1 - \text{Specificity}$$

Interpretation

- **AUC = 1**: Perfect classifier (ideal).

- **AUC = 0.5**: Random guessing.

- **Higher AUC**: Indicates better model performance.

When to Use

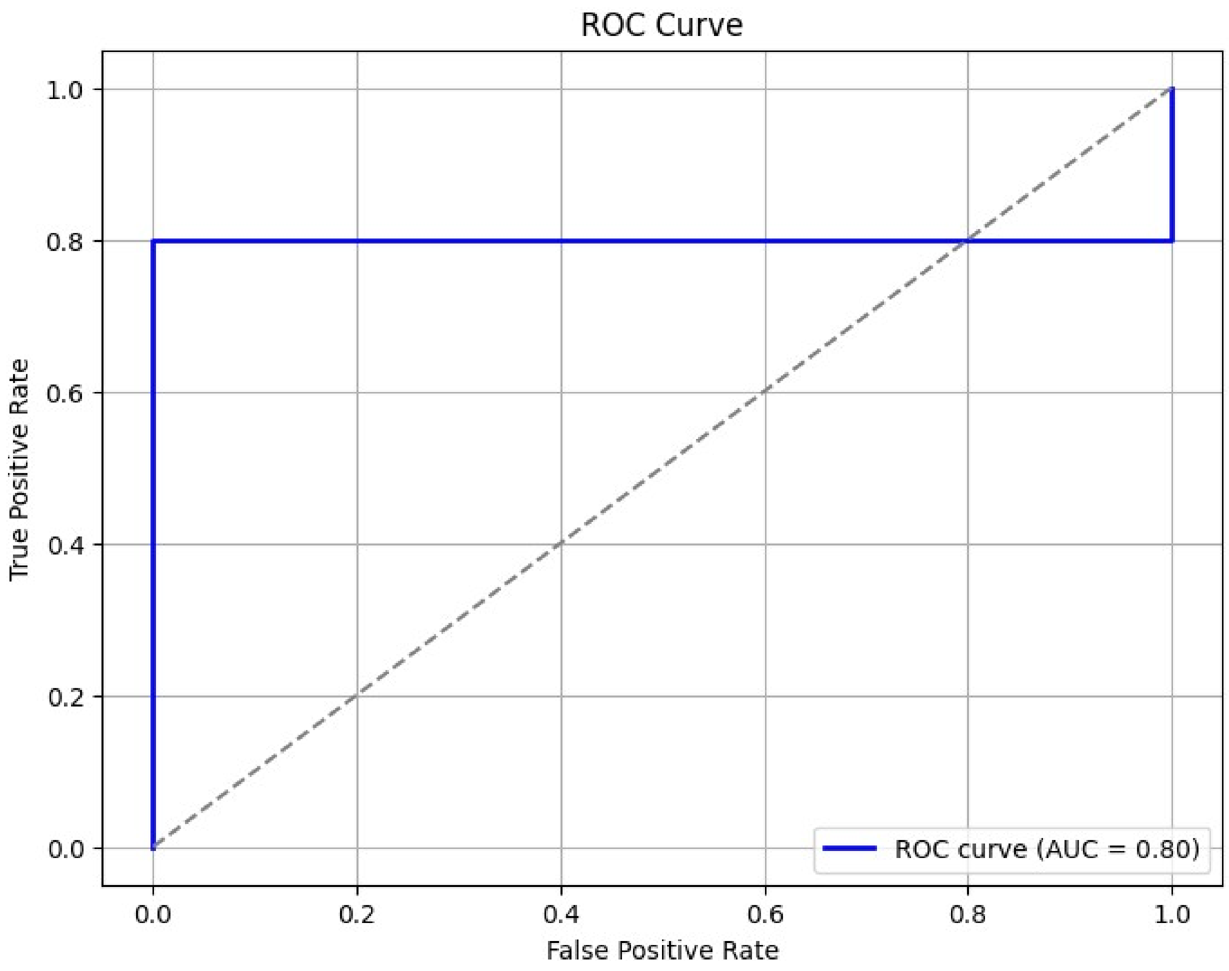
- Use ROC-AUC when you need to evaluate how well the model separates positive and negative classes across different thresholds.

- **Example**: Medical diagnosis models where balancing False Positives and False Negatives is critical.

```python
# Import libraries
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve
import numpy as np
import matplotlib.pyplot as plt

# Actual labels and predicted probabilities
y_true = [1, 0, 1, 0, 0, 1, 0, 0, 1, 1]
y_pred_prob = [0.9, 0.2, 0.8, 0.7, 0.3, 0.85, 0.1, 0.4, 0.95, 0.05]

# Convert probabilities to binary predictions (threshold = 0.5)
y_pred = [1 if prob >= 0.5 else 0 for prob in y_pred_prob]
```

```python
# Compute Confusion Matrix
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

# Calculate Specificity
specificity = tn / (tn + fp)
print(f"Specificity: {specificity:.2f}")

# Calculate ROC-AUC Score
roc_auc = roc_auc_score(y_true, y_pred_prob)
print(f"ROC-AUC Score: {roc_auc:.2f}")

# Plot ROC Curve
fpr, tpr, thresholds = roc_curve(y_true, y_pred_prob)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

```
Specificity: 0.80
ROC-AUC Score: 0.80
Log Loss: 0.59
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.80)