
📅 Day 7: Evaluation Metrics & Confusion Matrix

🎯 Objective

To understand and apply key metrics used to **evaluate classification models**:

- Accuracy
- Precision
- Recall
- F1-score
- Confusion matrix

📘 Context & Dataset

We use the **Breast Cancer Wisconsin dataset** (binary classification):

- Class 0: **Malignant**
- Class 1: **Benign**

Trained using **Logistic Regression** (from Day 8)
Now we **analyze how well it performs**.

🧠 Why Evaluation Matters

A good model should not just “predict something” — it should predict **the right thing in the right context**.

> Imagine a cancer detection model that always says “Benign” — it might have 95% accuracy (because most are benign), but will miss dangerous malignant tumors (low recall).

🔢 1. Confusion Matrix

The **confusion matrix** shows:

- True Positives (TP): Correctly predicted positives
- True Negatives (TN): Correctly predicted negatives
- False Positives (FP): Incorrectly predicted as positive
- False Negatives (FN): Incorrectly predicted as negative

```
          Predicted
        |  0  |  1
-------|-----|-----
 Actual  0   | TP | FP
         1   | FN | TN
```

In Python:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(confusion_matrix=cm).plot()
```

📐 2. Evaluation Metrics

✅ Accuracy

textAccuracy=fracTP+TNTP+TN+FP+FN\\text{Accuracy} = \\frac{TP + TN}{TP + TN + FP + FN}

Measures overall correctness

🎯 Precision

textPrecision=fracTPTP+FP\\text{Precision} = \\frac{TP}{TP + FP}

Out of all predicted positives, how many were correct?

Use-case: **Spam detection**
You don't want to mark a real email as spam (high precision needed).

🔍 Recall (Sensitivity)

textRecall=fracTPTP+FN\\text{Recall} = \\frac{TP}{TP + FN}

> Out of all actual positives, how many did you catch?

Use-case: **Disease detection**
You don't want to miss actual patients (high recall important).

⚖️ F1-Score

textF1=2∗fracPrecision∗RecallPrecision+Recall\\text{F1} = 2 * \\frac{Precision * Recall}{Precision + Recall}

> Harmonic mean of precision and recall
> Good when you need **balance** (e.g., fraud detection)

📊 Classification Report (all in one)

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))

Metric	Class 0 (Malignant)	Class 1 (Benign)
Precision	% of correct malignant predictions	% of correct benign predictions
Recall	% of actual malignant detected	% of actual benign detected
F1-score	Balance of both	Balance of both

📋 Summary Table

Metric	Formula	Use Case Example
Accuracy	(TP + TN) / All	Balanced datasets
Precision	TP / (TP + FP)	Avoid false alarms
Recall	TP / (TP + FN)	Don't miss positives
F1-Score	2 × (P × R) / (P + R)	Imbalanced datasets

✅ Intern Exercise Ideas

- Try this on another dataset (e.g., `digits`, `titanic`)
- Change thresholds using `predict_proba() > 0.3` and observe confusion matrix
- Plot Precision-Recall curves and ROC (Day 10 bonus)

Great! Here's the full **detailed explanation** for **Day 10: Decision Trees** — perfect for interns learning machine learning.

📅 Decision Trees

🎯 Objective

Learn how Decision Trees work and how to implement one for a classification task using Scikit-learn.

🌳 What is a Decision Tree?

A **Decision Tree** is a supervised learning algorithm used for:

- **Classification**
- **Regression**

It mimics human decision-making by splitting data into branches based on decision rules.

🧠 Key Concepts

🔹 Nodes

- **Root Node**: First decision point

- **Internal Node**: A condition-based branch (e.g., `age < 30`)
- **Leaf Node**: A final class label (e.g., “Benign” or “Malignant”)

🔹 Splitting Criteria

Used to decide which feature to split on:

- **Gini Impurity** (default in Scikit-learn)
- **Entropy / Information Gain**

🔹 Overfitting

- Deep trees tend to memorize training data.
- Solution: **limit depth**, **prune tree**, or set **min_samples_split**.

🔢 Step-by-Step Implementation

✅ Dataset: Breast Cancer Wisconsin

We continue using this dataset for binary classification.

📦 Step 1: Load and Prepare Data

```
from sklearn.datasets import load_breast_cancer
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load data
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Train-test split
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the data
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

🌲 Step 2: Train a Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

model = DecisionTreeClassifier(max_depth=4, random_state=42)
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

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

- `max_depth=4`: Limits depth to reduce overfitting
- You can experiment with different values

🖼️ Step 3: Visualize the Tree

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 10))
plot_tree(model, filled=True, feature_names=data.feature_names,
class_names=data.target_names)
plt.show()
```

This gives you a **tree-like diagram** showing:

- What feature was used at each split
- Threshold values
- Class distribution at leaves

🔁 Step 4: Try Modifying Parameters

Try experimenting with:

model = DecisionTreeClassifier(max_depth=2, criterion='entropy')

Or use:

- `min_samples_split`
- `min_samples_leaf`
- `max_leaf_nodes`

✅ Summary

Concept	Explanation
Decision Tree	Predicts class by learning decision rules
Gini / Entropy	Measures used to choose best splits
Overfitting	Controlled using max_depth, pruning
Visualization	Helps explain model decisions

🧠 Intern Challenge

Try building two trees:

1. One with `max_depth=2`
2. One with no limit

Compare:

- Accuracy
- Confusion Matrix
- Overfitting behavior