Accuracy and Confusion Matrix | Type 1 and Type 2 Errors

Accuracy

- **Accuracy** is one of the most straightforward performance metrics in classification.
- It tells you the percentage of correctly predicted labels out of the total number of predictions.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where:

- **TP** = True Positives: Correct positive predictions.
- **TN** = True Negatives: Correct negative predictions.
- **FP** = False Positives: Incorrect positive predictions (negative cases incorrectly predicted as positive).
- **FN** = False Negatives: Incorrect negative predictions (positive cases incorrectly predicted as negative).

Use Case:

Accuracy is suitable when:

- Classes are balanced.
- You are dealing with a simple problem where predicting a large number of correct labels is the main objective.

However, accuracy might not be very informative when there is an

imbalanced dataset (where one class is much larger than the other). In that case, other metrics like precision, recall, or F1-score might be more informative.

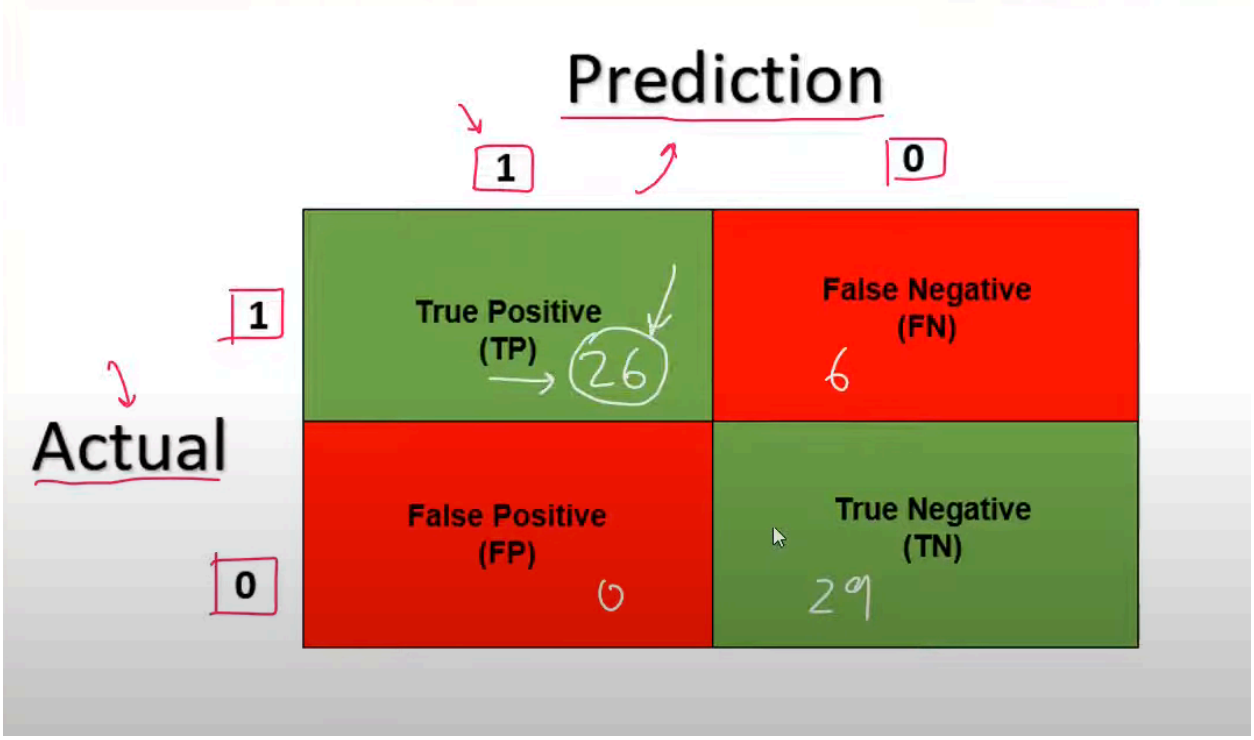
Cons:

- It doesn't give you the type of inaccuracy.
    - Type 1 vs Type 2 error
- Confusion Matrix solves this problem

Confusion Matrix

- The **confusion matrix** is a **2×2** matrix (in binary classification) that shows the performance of a classification model in terms of its true positive, true negative, false positive, and false negative predictions.

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



$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} = \frac{TP + TN}{TP + TN + FP + FN}$$

type I error

- **False Positive (FP)**: The model incorrectly predicted the positive class ((type I error).

type II error

- **False Negative (FN)**: The model incorrectly predicted the negative class (type II error).

Python Code

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_
report
import seaborn as sns
```

```
import matplotlib.pyplot as plt

# Example data
y_true = [0, 1, 0, 1, 1, 0, 0, 1, 0, 1]  # Actual labels
y_pred = [0, 1, 0, 0, 1, 0, 1, 1, 0, 1]  # Predicted labels

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

# Generate confusion matrix
cm = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:")
print(cm)
```

Output:

Accuracy: 0.80
Confusion Matrix:
[[4 1]
[1 4]]

Multi-Class Confusion Matrix

- When you have more than 2 output columns

```
from sklearn.metrics import multilabel_confusion_matrix

# Example multi-class data
y_true_multi = [0, 1, 2, 0, 1, 2]
y_pred_multi = [0, 2, 1, 0, 0, 1]

# Multi-class confusion matrix
mcm = multilabel_confusion_matrix(y_true_multi, y_pred_multi)
```

```
print("Multi-Class Confusion Matrix:")
print(mcm)
```

Output:

Multi-Class Confusion Matrix:
[[[3 1]
[0 2]]

[[2 2]
[2 0]]

[[3 1]
[2 0]]]

Another Multi-Class Confusion Matrix:

markdown

	Predicted		
	0	1	2
Actual			
0	15	0	0
1	0	14	3
2	0	3	13

- Class 0: 15 correctly predicted, 0 incorrectly predicted as 1 or 2.
- Class 1: 14 correctly predicted, 3 predicted as Class 2.
- Class 2: 13 correctly predicted, 3 predicted as Class 1.