# Precision, Recall and F1 Score

from sklearn.metrics import recall\_score,precision\_score,f1\_score

These three metrics are used to evaluate the performance of classification models, especially in scenarios where the class distribution is imbalanced.

## **Precision**

- Definition: The proportion of predicted positives that are actually positive.
- Precision answers the question, "Of all the instances that were predicted as positive, how many were actually positive?"
- Formula: Precision=True Positives (TP)+False Positives (FP)True Positives (TP)

$$ext{Precision} = rac{ ext{True Positives (TP)}}{ ext{True Positives (TP)} + ext{False Positives (FP)}}$$

When to Use: Use precision when the cost of false positives is high.

## **Example:**

- In email spam detection, high precision means that most emails marked as spam really are spam.
  - You don't want to label legitimate emails as spam (false positive).

# **Recall (Sensitivity or True Positive Rate)**

• The proportion of actual positives that are correctly identified.

 Recall answers the question, "Of all the actual positive cases, how many did the model correctly identify?"

$$Recall = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Negatives\ (FN)}$$

When to Use: Use recall when the cost of false negatives is high.

## **Example:**

- In disease screening, high recall means most patients with the disease are correctly identified.
- Recall is more important when the cost of missing a positive case (false negative) is high.
  - For example, in medical diagnosis (e.g., cancer detection), you don't want to miss any potential positive cases (false negatives).

## F1 Score

- The harmonic mean of precision and recall, balancing the two metrics.
- It's harmonic mean of precision & recall.
  - Arithmetic mean is in the centre
  - Harmonic mean is in the lower side
  - It penalises the lower value.
    - eg. If precision is low, it will penalise it.

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

#### When to Use:

- Use F1 Score when you need a balance between precision and recall, especially in cases of class imbalance.
- It's useful when you have an imbalanced dataset, where one class is much larger than the other.

# When to Use Each Metric:

- **Precision**: Useful when false positives are more costly (e.g., spam detection, fraud detection).
- **Recall**: Important when false negatives are more critical (e.g., cancer detection, disease outbreak).
- **F1 Score**: Ideal when you need to balance both precision and recall, especially in situations where classes are imbalanced.

## Python code:

```
from sklearn.metrics import precision_score, recall_score, f1_score

# Example true labels and predicted labels for a binary classification task
y_true = [1, 0, 1, 1, 0, 1, 1, 1, 0]
y_pred = [1, 0, 0, 1, 0, 1, 1, 0, 1, 0]

precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
```

```
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

Output:
Precision: 1.0
Recall: 0.7142857142857143
F1 Score: 0.83333333333333333333
```

# We trained 2 models (Logistic regression & Decision Tree) on the heart disease dataset.

from sklearn.metrics import recall\_score,precision\_score,f1\_score

```
print("For Logistic regression Model")
print("-"*50)
cdf = pd.DataFrame(confusion_matrix(y_test,y_pred1),columns=list(range(0, 2)))
print(cdf)
print("-"*50)
print("Precision - ",precision_score(y_test,y_pred1))
print("Recall - ",recall_score(y_test,y_pred1))
print("F1 score - ",f1_score(y_test,y_pred1))
```

```
Output:

For Logistic regression Model

-----

0 1

0 82 23
1 10 90
```

```
Precision - 0.7964601769911505

Recall - 0.9

F1 score - 0.8450704225352113
```

```
print("For DT Model")
print("-"*50)
cdf = pd.DataFrame(confusion_matrix(y_test,y_pred2),columns=list(range(0, 2)))
print(cdf)
print("-"*50)
print("Precision - ",precision_score(y_test,y_pred2))
print("Recall - ",recall_score(y_test,y_pred2))
print("F1 score - ",f1_score(y_test,y_pred2))
Output:
For DT Model
```

```
Output:

For DT Model

0 1
0 101 4
1 0 100

Precision - 0.9615384615384616

Recall - 1.0
F1 score - 0.9803921568627451
```

# **Multi-Class Precision & Recall**

#### We'll use Iris dataset

df = pd.read\_csv(r'https://raw.githubusercontent.com/G1Codes/Datasets/refs/
heads/main/Iris.csv')

from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df['Species'] = encoder.fit\_transform(df['Species'])

#### We have converted Cat → Num

df

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0
145	146	6.7	3.0	5.2	2.3	2
146	147	6.3	2.5	5.0	1.9	2
147	148	6.5	3.0	5.2	2.0	2
148	149	6.2	3.4	5.4	2.3	2
149	150	5.9	3.0	5.1	1.8	2

#### TTS:

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

X\_train,X\_test,y\_train,y\_test = train\_test\_split(df.iloc[:,0:-1],df.iloc[:,-1],test\_size
=0.2,random\_state=1)

```
from sklearn.tree import DecisionTreeClassifier

clf1 = LogisticRegression()
clf2 = DecisionTreeClassifier()

clf1.fit(X_train,y_train)
clf2.fit(X_train,y_train)

y_pred1 = clf1.predict(X_test)

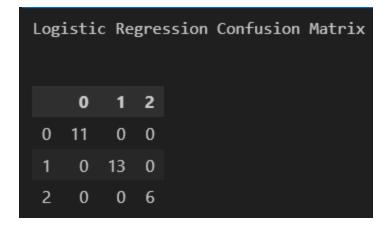
y_pred2 = clf2.predict(X_test)

from sklearn.metrics import accuracy_score,confusion_matrix
print("Accuracy of Logistic Regression",accuracy_score(y_test,y_pred1))
print("Accuracy of Decision Trees",accuracy_score(y_test,y_pred2))
```

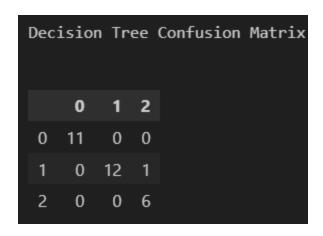
### **Output:**

Accuracy of Logistic Regression 1.0
Accuracy of
Decision Trees 0.9666666666666667

```
print("Logistic Regression Confusion Matrix\n")
pd.DataFrame(confusion_matrix(y_test,y_pred1),columns=list(range(0,3)))
```



print("Decision Tree Confusion Matrix\n")
pd.DataFrame(confusion\_matrix(y\_test,y\_pred2),columns=list(range(0,3)))



```
result = pd.DataFrame()
result['Actual Label'] = y_test
result['Logistic Regression Prediction'] = y_pred1
result['Decision Tree Prediction'] = y_pred2
result.sample(10)
```

	Actual Label	Logistic Regression Prediction	Decision Tree Prediction
31	0	0	0
99	1	1	2
131	2	2	2
125	2	2	2
84	1	1	1
73	1	1	1
40	0	0	0
92	1	1	1
66	1	1	1
90	1	1	1

# Now, calculate precision & recall score:

from sklearn.metrics import precision\_score,recall\_score precision\_score(y\_test,y\_pred1,average=None)

## **Output:**

array([1., 1., 1.])

from sklearn.metrics import precision\_score,recall\_score precision\_score(y\_test,y\_pred1,average=None)

## **Output:**

array([1., 1., 1.])



If you want average, use 'macro'

# You need to specify an appropriate average parameter for multiclass classification.

The average parameter can take one of the following values:

- None: Returns the recall score for each class.
- 'micro': Calculates metrics globally by counting the total true positives, false negatives, and false positives.
- 'macro': Calculates metrics for each label and finds their unweighted mean.
  This does not take label imbalance into account.
  - Average of all scores
  - Use macro when the classes are Equal
- 'weighted': Calculates metrics for each label and finds their average, weighted by support (the number of true instances for each label). This accounts for label imbalance.
  - You multiply the value with weight of the class
  - Use weighted when classes are imbalanced

## To print everything → Use classification report

from sklearn.metrics import classification\_report

from sklearn.metrics import classification\_report print(classification\_report(y\_test,y\_pred1))

	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	6
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

from sklearn.metrics import classification\_report print(classification\_report(y\_test,y\_pred2))

	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1 2	1.00 0.86	0.92 1.00	0.96 0.92	13 6
accuracy			0.97	30
macro avg	0.95	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

• support → How many times 0, 1, 2 occur?