

Classification Model Performances

1) Confusion Matrix:

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. It is a table with 4 different combinations of predicted and actual values in the case for a binary classifier. The confusion matrix for a multi-class classification problem can help you determine mistake patterns.

		Predicted Class	
		No	Yes
Observed Class	No	TN	FP
	Yes	FN	TP

TN	True Negative
FP	False Positive
FN	False Negative
TP	True Positive

Model Performance

Accuracy = $(TN+TP)/(TN+FP+FN+TP)$

Precision = $TP/(FP+TP)$

Sensitivity = $TP/(TP+FN)$

Specificity = $TN/(TN+FP)$

Syntax:

```
from sklearn.metrics import confusion_matrix
```

y_true = Ground truth (correct) target values

y_pred = Estimated targets as returned by a classifier.

```
confusion_matrix(y_true, y_pred)
```

Example:

```
y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]
y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]
confusion_matrix(y_true, y_pred, labels=["ant", "bird", "cat"])
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

- **True Positive & True Negative:**

A **true positive** is an outcome where the model *correctly* predicts the *positive* class. Similarly, a **true negative** is an outcome where the model *correctly* predicts the *negative* class.

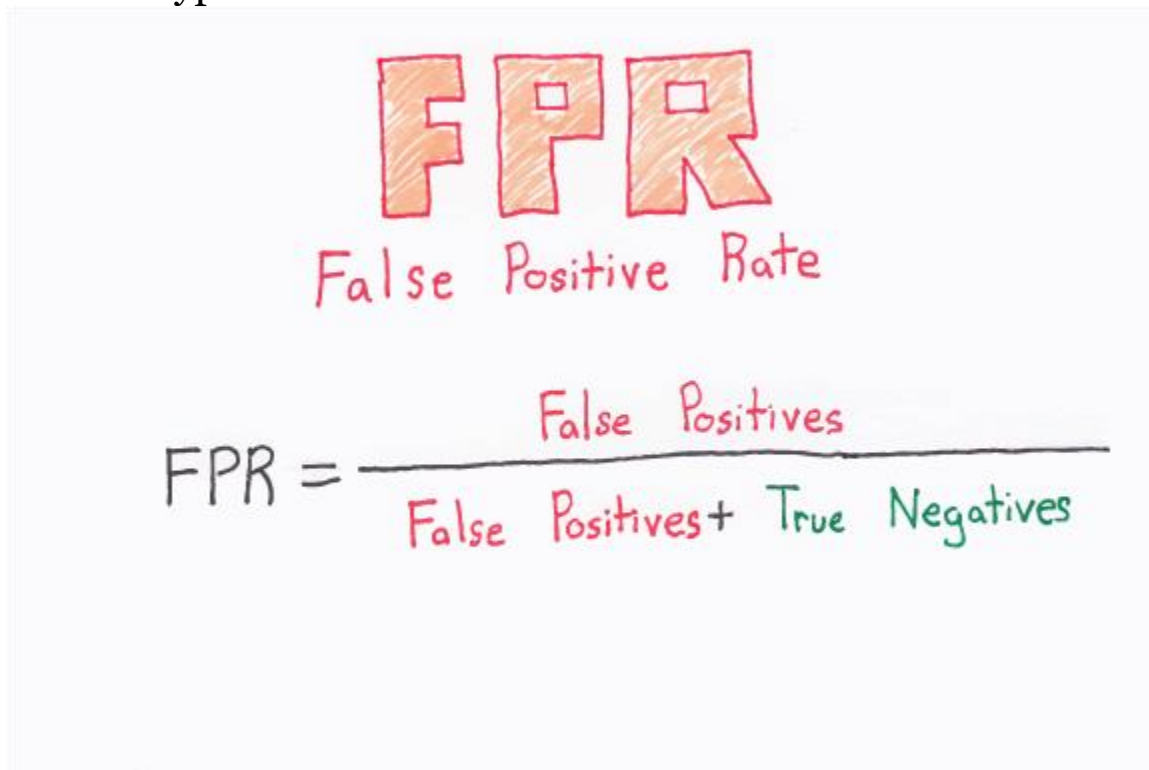
- **False Positive & False Negative:**

The terms False Positive and False Negative are very in determining how well the model is predicted with respect to classification. A false positive is an outcome where the model incorrectly predicts the positive class. And a false negative is an outcome where the model incorrectly predicts the negative class. The more values in main diagonal, better the model whereas the other diagonal gives the worst result for classification.

- **False Positive:**

An example in which the model mistakenly predicted the positive class. For example, the model inferred that a particular email message was spam (the positive class), but that email message was actually not spam. It's like a warning sign that the mistake did should be rectified as it's not much of a serious concern compared to False Negative.

- **False positive** (*type I error*)—when you reject a *true* null hypothesis



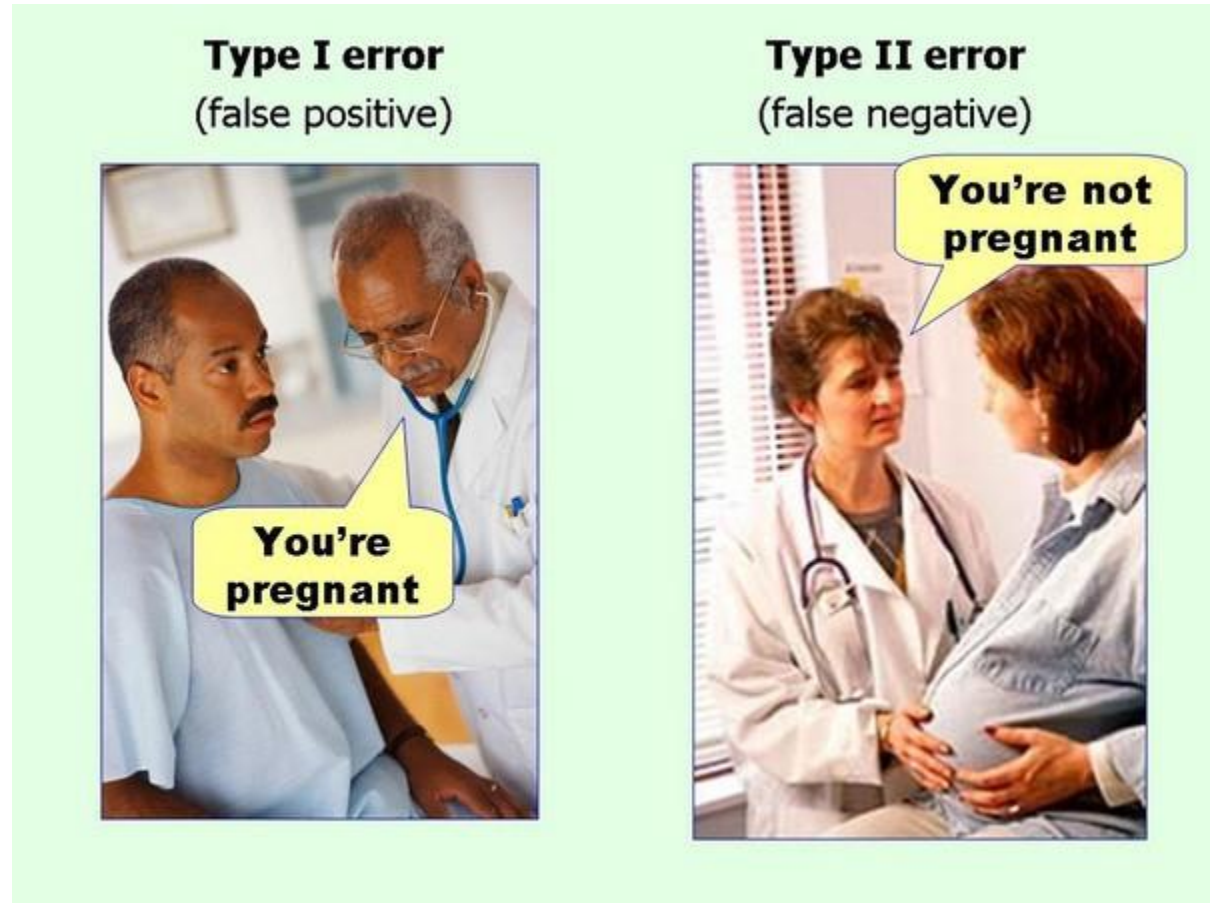
The diagram shows the formula for the False Positive Rate (FPR) in a hand-drawn style. At the top, the letters 'FPR' are written in large, orange, blocky font. Below them, the words 'False Positive Rate' are written in a smaller, red, cursive font. The formula itself is written below a horizontal line. On the left side of the line is 'FPR ='. On the right side, the numerator is 'False Positives' in red, and the denominator is 'False Positives + True Negatives', where 'False Positives' is in red and 'True Negatives' is in green.

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

False Negative:

An example in which the model mistakenly predicted the **negative class**. For example, the model inferred that a particular email message was not spam (the negative class), but that email message actually was spam. It's like a **danger** sign that the mistake did should be rectified at the earliest as it's of a much serious concern compared to False Positive.

False negative (*type II error*) — when you accept a *false* null hypothesis.



This picture easily illustrates the above metrics . The man's test results say "You 're pregnant" is False Positive as a man cannot be pregnant, and a pregnant woman's test results say "Pregnant" is False Negative as from the image it easily identified that the woman is pregnant.

From the Confusion Matrix, we can infer Accuracy, Precision, Recall, F-1 Score.

2) Accuracy:

Accuracy is the fraction of predictions our model got right.

ACCURACY

$$A_{cc} = \frac{1}{n} \sum 1(\hat{y}_i = y_i)$$

Diagram illustrating the formula for Accuracy (A_{cc}):

- n : number of observations (indicated by a red arrow pointing to the denominator).
- \sum : Summation over all observations.
- $1(\hat{y}_i = y_i)$: Indicator function (indicated by a green bracket below the term).
 - \hat{y}_i : Predicted y (indicated by an orange arrow pointing to the predicted value).
 - y_i : True y (indicated by a blue arrow pointing to the true value).

A common metric in classification. Fails when we have highly imbalanced classes. In those cases F1 is more appropriate.

Accuracy alone doesn't tell the full story when you're working with a class-imbalanced data set, where there is a significant disparity between the number of positive and negative labels. Precision and Recall are better metrics for evaluating class-imbalanced problems.

Syntax:

```
from sklearn.metrics import accuracy_score
```

```
y_true = Ground truth (correct) target values
```

```
y_pred = Estimated targets as returned by a classifier.
```

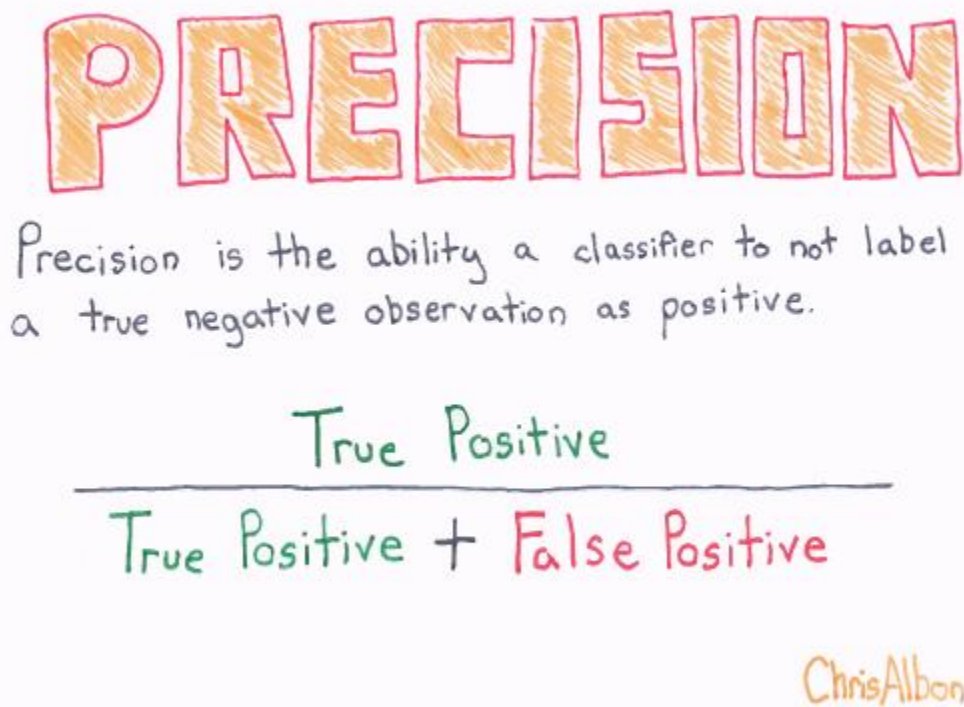
```
accuracy_score(y_true, y_pred)
```

Example:

```
import numpy as np  
from sklearn.metrics import accuracy_score  
y_pred = [0, 2, 1, 3]  
y_true = [0, 1, 2, 3]  
accuracy_score(y_true, y_pred) - 0.5  
accuracy_score(y_true, y_pred, normalize=False) - 2
```

3) Precision:

Out of all the classes, how much we predicted correctly.



Precision should be as **high** as possible.

Syntax:

```
from sklearn.metrics import precision_score
```

`y_true` = Ground truth (correct) target values

`y_pred` = Estimated targets as returned by a classifier.

```
precision_score(y_true, y_pred)
```

Parameters:

average : string, [None, 'binary' (default), 'micro', 'macro', 'samples', 'weighted'] - This parameter is required for multiclass/multilabel targets. If None, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data:

- 'binary': Only report results for the class specified by `pos_label`. This is applicable only if targets (`y_{true,pred}`) are binary.
- 'micro': Calculate metrics globally by counting the total true positives, false negatives and false positives.
- 'macro': Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

- 'weighted': Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance; it can result in an F-score that is not between precision and recall.
- 'samples': Calculate metrics for each instance, and find their average (only meaningful for multilabel classification where this differs from accuracy_score).

Example:

```
from sklearn.metrics import precision_score
```

```
y_true = [0, 1, 2, 0, 1, 2]
```

```
y_pred = [0, 2, 1, 0, 0, 1]
```

```
precision_score(y_true, y_pred, average='macro') - 0.22
```

```
precision_score(y_true, y_pred, average='micro') - 0.33...
```

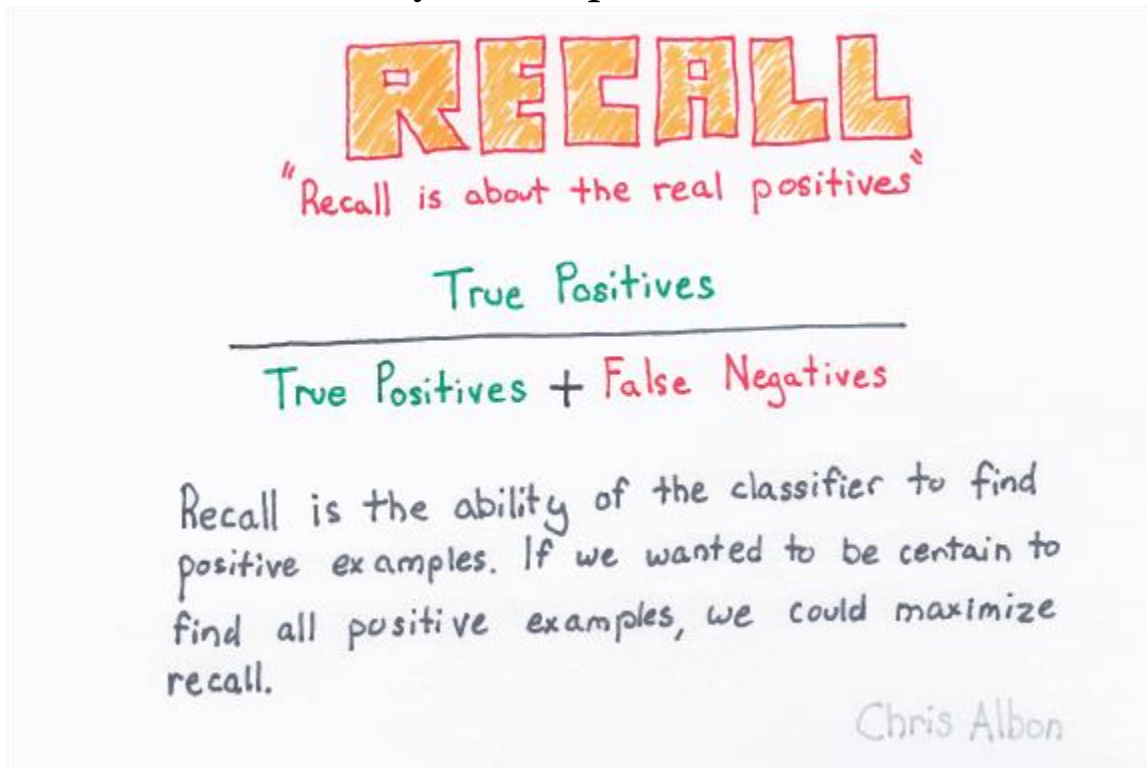
```
precision_score(y_true, y_pred, average='weighted') - 0.22...
```

```
precision_score(y_true, y_pred, average=None)
```

```
array([0.66..., 0.    , 0.    ])
```


4) Recall:

Out of all the positive classes, how much we predicted correctly. It is also called sensitivity or true positive rate (TPR).



Recall should be as **high** as possible.

Syntax:

```
from sklearn.metrics import recall_score
```

`y_true` = Ground truth (correct) target values

`y_pred` = Estimated targets as returned by a classifier.

```
recall_score(y_true, y_pred, labels=None, pos_label=1,  
             average='binary', sample_weight=None)
```

Example:

```
from sklearn.metrics import recall_score  
  
y_true = [0, 1, 2, 0, 1, 2]  
y_pred = [0, 2, 1, 0, 0, 1]  
  
recall_score(y_true, y_pred, average='macro') - 0.33...  
recall_score(y_true, y_pred, average='micro') - 0.33...  
recall_score(y_true, y_pred, average='weighted') - 0.33...  
recall_score(y_true, y_pred, average=None)  
  
array([1., 0., 0.]
```

5) F-1 Score:

It is often convenient to combine precision and recall into a single metric called the F1 score, in particular, if you need a simple way to compare two classifiers. The F1 score is the harmonic mean of precision and recall.

F1 SCORE

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1 score is the harmonic mean of precision and recall. Values range from 0 (bad) to 1 (good).

Chris Albon

The regular mean treats all values equally, the harmonic mean gives much more weight to low values thereby punishing the extreme values more. As a result, the classifier will only get a **high F1 score if both recall and precision are high**.

Syntax:

```
from sklearn.metrics import f1_score
```

y_true = Ground truth (correct) target values

y_pred = Estimated targets as returned by a classifier.

```
f1_score(y_true, y_pred, labels=None, pos_label=1,  
average='binary', sample_weight=None)
```

Example:

```
from sklearn.metrics import f1_score
```

```
y_true = [0, 1, 2, 0, 1, 2]
```

```
y_pred = [0, 2, 1, 0, 0, 1]
```

```
f1_score(y_true, y_pred, average='macro') - 0.26...
```

```
f1_score(y_true, y_pred, average='micro') - 0.33...
```

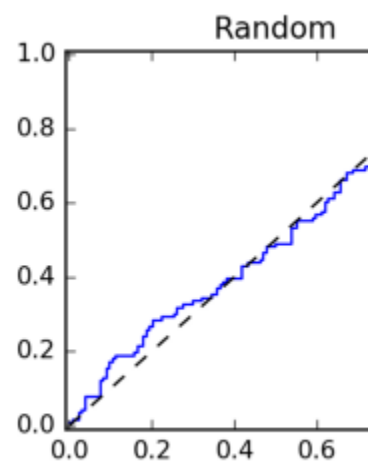
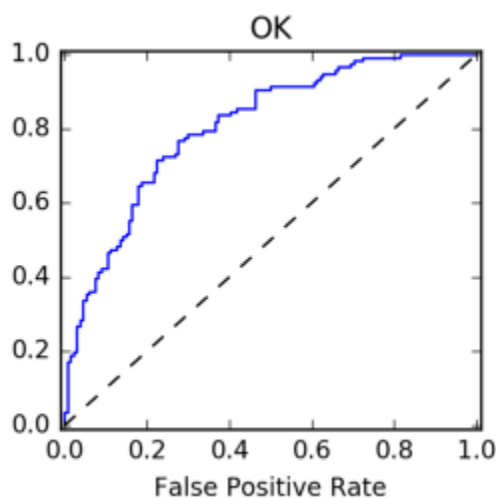
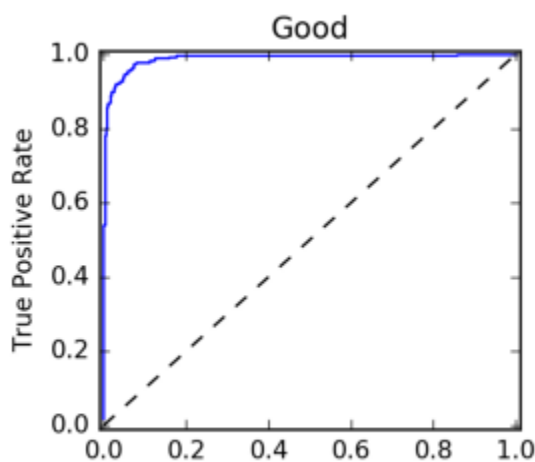
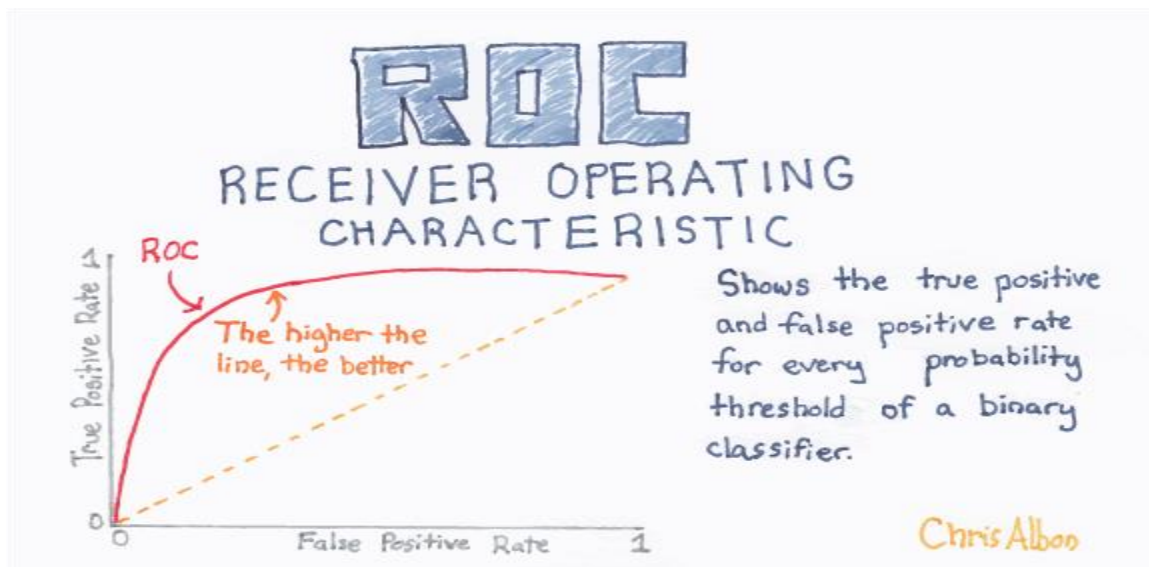
```
f1_score(y_true, y_pred, average='weighted') - 0.26...
```

```
f1_score(y_true, y_pred, average=None)
```

```
array([0.8, 0. , 0. ])
```

6) Receiver Operator Curve(ROC) & Area Under the Curve(AUC):

ROC curve is an important classification evaluation metric. It tells us how good the model is accurately predicted. The ROC curve shows the sensitivity of the classifier by plotting the rate of true positives to the rate of false positives. If the classifier is outstanding, the true positive rate will increase, and the area under the curve will be close to 1. If the classifier is similar to random guessing, the true positive rate will increase linearly with the false positive rate. The better the AUC measure, the better the model.



Example:

```
import numpy as np

from sklearn import metrics

y = np.array([1, 1, 2, 2])

scores = np.array([0.1, 0.4, 0.35, 0.8])

fpr, tpr, thresholds = metrics.roc_curve(y, scores, pos_label=2)

fpr
array([0. , 0. , 0.5, 0.5, 1. ])

tpr
array([0. , 0.5, 0.5, 1. , 1. ])

thresholds
array([1.8 , 0.8 , 0.4 , 0.35, 0.1 ])
```

7) Classification Report:

Scikit-learn does provide a convenience report when working on classification problems to give you a quick idea of the accuracy of a model using a number of measures.

The *classification_report()* function displays the precision, recall, f1-score and support for each class.

The example below demonstrates the report on the binary classification problem.

Example:

```
from sklearn.metrics import classification_report
```

```
y_true = [0, 1, 2, 2, 2]
```

```
y_pred = [0, 0, 2, 2, 1]
```

```
target_names = ['class 0', 'class 1', 'class 2']
```

```
print(classification_report(y_true, y_pred,  
target_names=target_names))
```

	precision	recall	f1-score	support
class 0	0.50	1.00	0.67	1
class 1	0.00	0.00	0.00	1
class 2	1.00	0.67	0.80	3
micro avg	0.60	0.60	0.60	5
macro avg	0.50	0.56	0.49	5
weighted avg	0.70	0.60	0.61	5