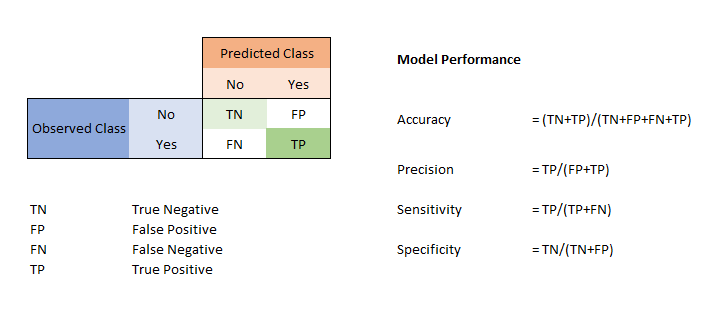
**Classification Model Performances**

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.

**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.

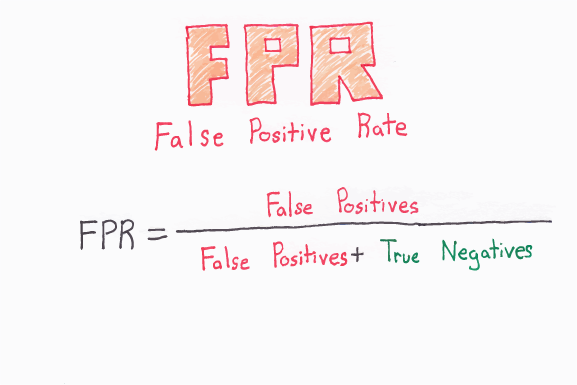
* **False Positive & False Negative:**

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.

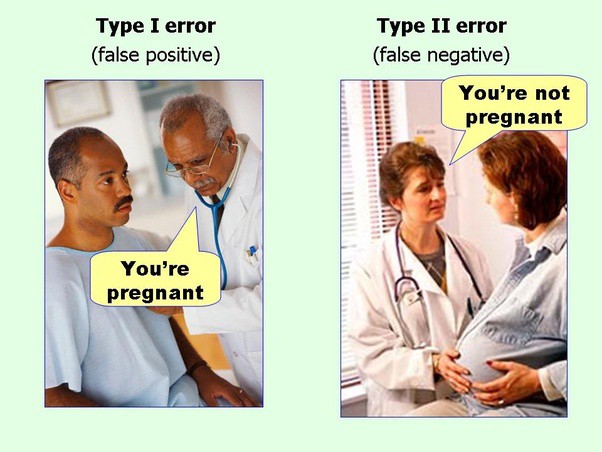
* **False positive**(type I error)— when you reject a true null hypothesis



**False Negative:**

An example in which the model mistakenly predicted the [**negative class**](https://developers.google.com/machine-learning/glossary/#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.

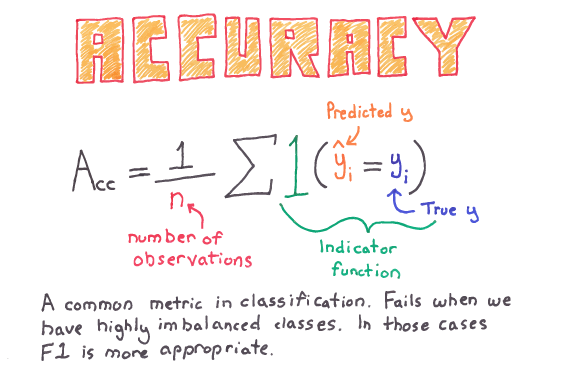


This picture easily illustrates the above metrics . The man’s test results say “You ’re pregnant” is False Positive as a man cannot be pregnant, and a pregnant woman’s test results say “Pregnant” is False Negative as from the image it easily identified that the woman is pregnant.

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

#### **Accuracy:**

Accuracy is the fraction of predictions our model got right.



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]**

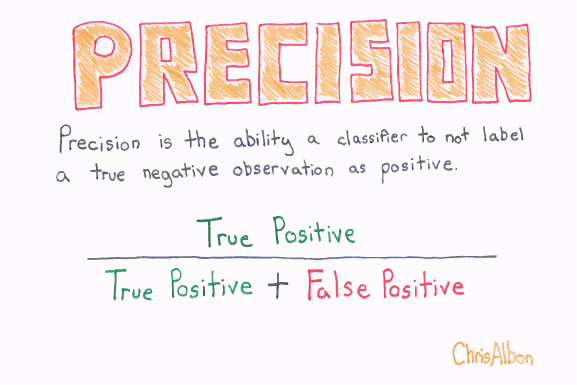
**y\_true = [0, 1, 2, 3]**

**accuracy\_score(y\_true, y\_pred) – 0.5**

**accuracy\_score(y\_true, y\_pred, normalize=False) – 2**

#### **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...**

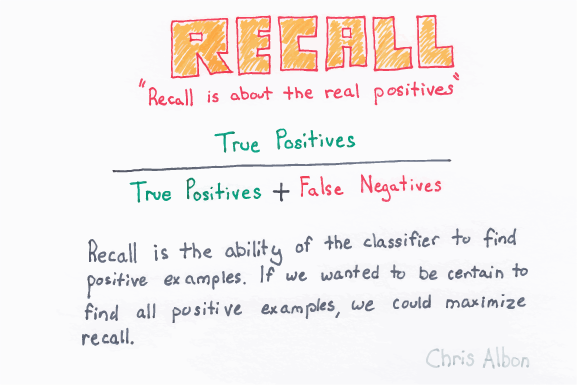
#### **precision\_score(y\_true, y\_pred, average='weighted') - 0.22...**

#### **precision\_score(y\_true, y\_pred, average=None)**

#### **array([0.66..., 0. , 0. ])**

#### **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...

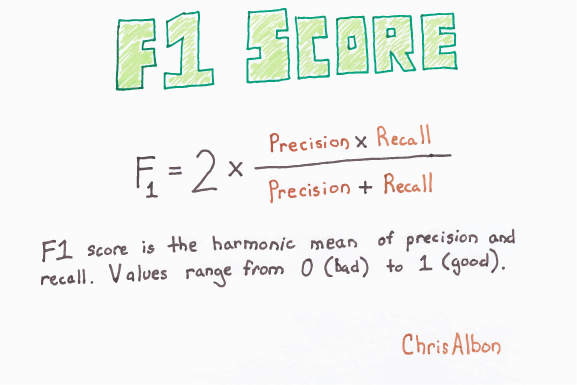
recall\_score(y\_true, y\_pred, average='weighted') - 0.33...

recall\_score(y\_true, y\_pred, average=None)

array([1., 0., 0.]

#### **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.



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...**

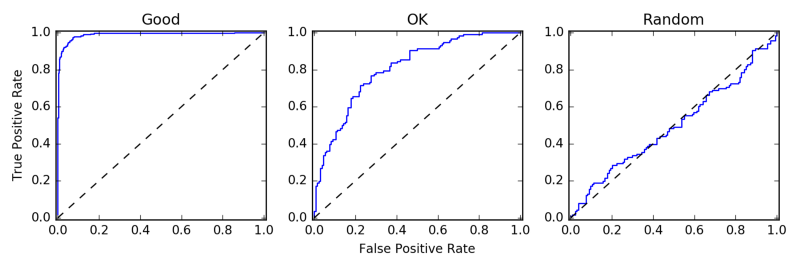
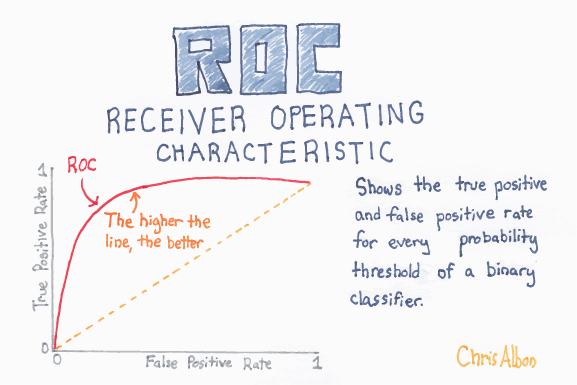
**f1\_score(y\_true, y\_pred, average='weighted') - 0.26...**

**f1\_score(y\_true, y\_pred, average=None)**

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

1. **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**