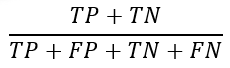
**Performance Measures**

Evaluating a classifier is often significantly trickier than evaluating a regressor. Here, we will discuss the various ways to check the performance of our machine learning or deep learning model and why to use one in place of the other. We will discuss terms like:

1. Accuracy
2. Confusion matrix
3. Precision
4. Recall
5. Specificity
6. F1 score
7. Precision-Recall or PR curve
8. **ROC** (**R**eceiver **O**perating **C**haracteristics) curve
9. PR vs ROC curve.

**Accuracy**

The most commonly used metric to judge a model and is actually not a clear indicator of the performance. The worse happens when classes are imbalanced.



**Measuring Accuracy Using Cross-Validation**

A good way to evaluate a model is to use cross-validation:

from sklearn.model\_selection import cross\_val\_score

cross\_val\_score(model, X\_train, y\_train, cv=3, scoring="accuracy")

Take for example a cancer detection model. The chances of actually having cancer are very low. Let’s say out of 100, 90 of the patients don’t have cancer and the remaining 10 actually have it. We don’t want to miss on a patient who is having cancer but goes undetected (false negative). Detecting everyone as not having cancer gives an accuracy of 90% straight. The model did nothing here but just gave cancer free for all the 100 predictions. This demonstrates why accuracy is generally not the preferred performance measure for classifiers, especially when you are dealing with skewed datasets (i.e., when some classes are much more frequent than others).

We surely need better alternatives.

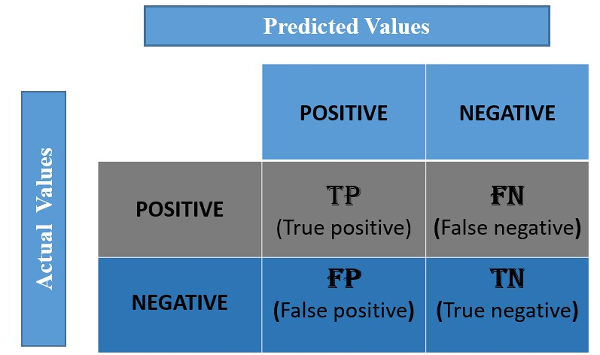
**Confusion matrix**

## **What Is a Confusion Matrix?**

* The confusion Matrix gives a comparison between actual and predicted values.
* It is used for the optimization of machine learning models.
* The confusion matrix is a N x N matrix, where N is the number of classes or outputs.
* For 2 classes, we get a 2 x 2 confusion matrix.
* For 3 classes, we get a 3 X 3 confusion matrix.

## **How to Create a 2X2 Confusion Matrix?**

Below is the representation of the confusion matrix:



There are 4 terms you must understand in order to correctly interpret or read a Confusion Matrix:

* True Positive(TP): TP represents the number of True Positives. This refers to the total number of observations that belong to the positive class and have been predicted correctly.
* False Positive(FP): FP is the number of False Positives. It is also known as a Type 1 Error. This is the total number of observations that have been predicted to belong to the positive class, but instead, actually, belong to the negative class.
* True Negative(TN): TN represents the number of True Negatives. This is the total number of observations that belong to the negative class and have been predicted correctly.
* False Negative(FN):FN is the number of False Negatives. It may be  
  referred to as a Type 2 Error. This is the total number of observations that  
  have been predicted to be a part of the negative class but instead belong to  
  the positive class.

## **Confusion Matrix for Binary Classification**

Let us see how to construct a confusion matrix and understand its terminologies. Consider we have to model a classifier that classifies 2 kinds of fruits. We have 2 types of fruits – apples and grapes – and we want our machine-learning model to identify or classify the given fruit as an apple or grape.

So we take 15 samples of 2 fruits, out of which 8 belong to Apples, and 7 belong to the Grapes class. Class is nothing but the output, in this example, we have 2 output classes – Apples and Grapes. We will represent Apple as 1 and Grape as 0 class.

The actual class for 8 apples and 7 grapes can be represented as:

Actual = [1,1,1,1,1,1,1,1,0,0,0,0,0,0,0]

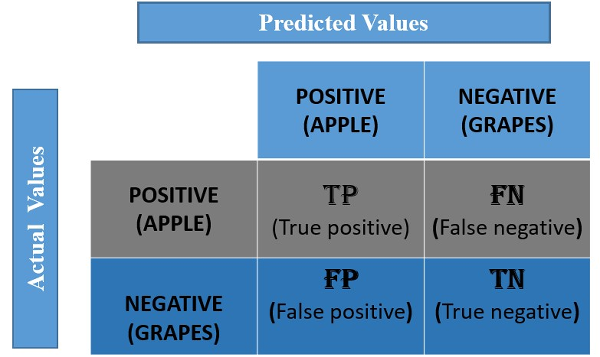
The classifier model predicts 1 for Apple and 0 for grape.

Assume that the classifier takes all 15 inputs and makes the following predictions:

* Out of 8 apples, it will classify 5 correctly as apples and wrongly predict 3 as grapes.
* Out of 7 grapes, it will classify 5 correctly as grapes and wrongly predicts 2 as apples

The prediction of the classifier may be as follows:

Prediction = [1,0,0,0,1,1,1,1,0,0,0,0,0,1,1]



For our example, the positive value is Apple, and the negative value is Grapes.

* **True Positive:**It means the actual value and also the predicted values are the same. In our case, the actual value is also an apple, and the model prediction is also an apple. If you observe the TP cell, the positive value is the same for Actual and predicted.
* **False Negative:**This means the actual value is positive. In our case, it is apple, but the model has predicted it as negative, i.e., grapes. So the model has given the wrong prediction. It was supposed to give a positive (apple), but it has given a negative (grape). So whatever the negative output we got is false; hence the name False Negative.
* **False Positive:**This means the actual value is negative. In our case, it is grapes, but the model has predicted it as positive, i.e., apple. So the model has given the wrong prediction. It was supposed to give a negative (grape), but it has given a positive (apple), so whatever the positive output we got is false, hence the name False Positive.
* **True Negative:**It means the actual value and also the predicted values are the same. In our case, the actual values are grapes, and the prediction is also Grapes. The values for the above example are: TP=5, FN=3, FP=2, TN=5.

## **Confusion Matrix for Multi-Class Classification**

The above example is a binary classification model with only 2 outputs, so we got a 2 X 2 matrix.

Now, what if the outputs are greater than 2 classes, i.e., Multi-class classification?

How to calculate TP, FN, FP, and TN?

## **Confusion Matrix for a 3-Class Classification**

Let’s try to understand the confusion matrix for 3 classes and the confusion matrix for multiple classes with a popular dataset – the IRIS DATASET.

The dataset has 3 flowers as outputs or classes, Versicolor, Virginia, and Setosa.



With the help of petal length, petal width, sepal length, and sepal width, the model has to classify the given instance as Versicolor or Virginia, or Setosa flower.

Let’s apply a classifier model here. We can use logistic regression, but a decision tree classifier is applied to the above dataset. The dataset has 3 classes; hence we get a 3 X 3 confusion matrix.

But how to know the TP, TN, FP, and FN values?

In the multi-class classification problem, we won’t get TP, TN, FP, and FN values directly as in the binary classification problem. For validation, we need to calculate for each class.

## **Confusion Matrix Using Scikit-learn in Python**

You know the theory – now let’s put it into practice. Let’s code a confusion matrix with the [Scikit-learn (sklearn) library](https://www.analyticsvidhya.com/blog/2020/02/everything-you-should-know-scikit-learn/?utm_source=blog&utm_medium=confusion-matrix-machine-learning" \t "_blank) in Python.

Sklearn has two great functions: **confusion\_matrix()** and **classification\_report()**.

* Sklearn **[confusion\_matrix()](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html" \t "_blank)** returns the values of the Confusion matrix. The output is, however, slightly different from what we have studied so far. It takes the rows as Actual values and the columns as Predicted values. The rest of the concept remains the same.
* Sklearn **[classification\_report()](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html" \t "_blank)** outputs precision, recall, and f1-score for each target class. In addition to this, it also has some extra values: **micro avg**, **macro avg**, and **weighted avg. We will see them later.**

#importing packages

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

#Importing of dataset to dataframe.

df = pd.read\_csv("../input/iris-flower-dataset/IRIS.csv")

#Speceis is the output class,to know the count of each class we use value\_counts()

df['Species'].value\_counts()

#Separating independant variable and dependent variable("Species")

X = df.drop(['Species'], axis=1)

y = df['Species']

# Splitting the dataset to Train and test

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

#We use Support Vector classifier as a classifier

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix

#training the classifier using X\_Train and y\_train

clf = SVC(kernel = 'linear').fit(X\_train,y\_train)

clf.predict(X\_train)

#Testing the model using X\_test and storing the output in y\_pred

y\_pred = clf.predict(X\_test)

# Creating a confusion matrix,which compares the y\_test and y\_pred

cm = confusion\_matrix(y\_test, y\_pred)

# Creating a dataframe for a array-formatted Confusion matrix,so it will be easy for plotting.

cm\_df = pd.DataFrame(cm,index = ['SETOSA','VERSICOLR','VIRGINICA'], columns = ['SETOSA','VERSICOLR','VIRGINICA'])

#Plotting the confusion matrix

plt.figure(figsize=(5,4))

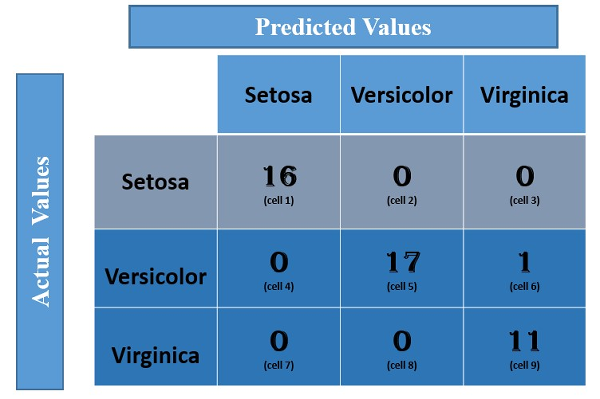
sns.heatmap(cm\_df, annot=True)

plt.title('Confusion Matrix')

plt.ylabel('Actual Values')

plt.xlabel('Predicted Values')

plt.show()



## **How to Calculate FN, FP, TN, and TP Values?**

As discussed earlier, FN: The False-negative value for a class will be the sum of values of corresponding rows except for the TP value. FP: The False-positive value for a class will be the sum of values of the corresponding column except for the TP value. TN: The True-negative value for a class will be the sum of the values of all columns and rows except the values of that class that we are calculating the values for. And TP: the True-positive value is where the actual value and predicted value are the same.

Let us calculate the TP, TN, FP, and FN values for the class **Setosa**using the Above tricks:

**TP**: The actual value and predicted value should be the same. So concerning Setosa class, the value of cell 1 is the TP value.

**FN**: The sum of values of corresponding rows except for the TP value

FN = (cell 2 + cell3)

= (0 + 0)

= 0

**FP:**The sum of values of the corresponding column except for the TP value.

FP = (cell 4 + cell 7)

= (0 + 0)

= 0

**TN:**The sum of values of all columns and rows except the values of that class that we are calculating the values for.

TN = (cell 5 + cell 6 + cell 8 + cell 9)

= 17 + 1 +0 + 11

= 29

Similarly, for the **Versicolor** class, the values/metrics are calculated as below:

TP: 17 (cell 5)

FN : 0 + 1 = 1 (cell 4 +cell 6)

FP : 0 + 0 = 0 (cell 2 + cell 8)

TN : 16 +0 +0 + 11 =27 (cell 1 + cell 3 + cell 7 + cell 9).

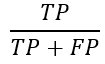
You can try for the Virginia class.

## **Why Use Confusion Matrix?**

The confusion Matrix allows us to measure Recall and Precision, which, along with Accuracy and the AUC-ROC curve, are the metrics used to measure the performance of ML models.

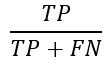
**Precision**

Percentage of positive instances out of the ***total predicted positive***instances. Here denominator is the model prediction done as positive from the whole given dataset. Take it as to find out ‘*how much the model is right when it says it is right’.*



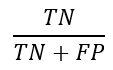
**Recall/Sensitivity/True Positive Rate**

Percentage of positive instances out of the***total actual positive*** instances. Therefore denominator (*TP + FN)*here is the *actual*number of positive instances present in the dataset. Take it as to find out ‘*how much extra right ones, the model missed when it showed the right ones’.*



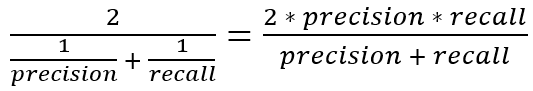
**Specificity**

Percentage of negative instances out of the***total actual negative*** instances. Therefore denominator (*TN + FP)*here is the *actual*number of negative instances present in the dataset. It is similar to recall but the shift is on the negative instances.*Like finding out how many healthy patients were not having cancer and were told they don’t have cancer*. Kind of a measure to see how separate the classes are.



**F1 score**

Itis the harmonic mean of precision and recall. This takes the contribution of both, so higher the F1 score, the better. See that due to the product in the numerator if one goes low, the final F1 score goes down significantly. So a model does well in F1 score if the positive predicted are actually positives (precision) and doesn't miss out on positives and predicts them negative (recall).



One drawback is that both precision and recall are given equal importance due to which according to our application we may need one higher than the other and F1 score may not be the exact metric for it. Therefore either weighted-F1 score or seeing the PR or ROC curve can help.

**Have you asked why F1-Score is a Harmonic Mean(HM) of Precision and Recall?**

Let us say have a model:

**Model with a P of 0.9 and R of 0.4.**

So the AM = (0.9+0.4)/2=0.65

GM = sqrt(0.9 \* 0.4)=0.6

HM = 2\*(0.9\*0.4)/(0.9+0.4)= 0.55

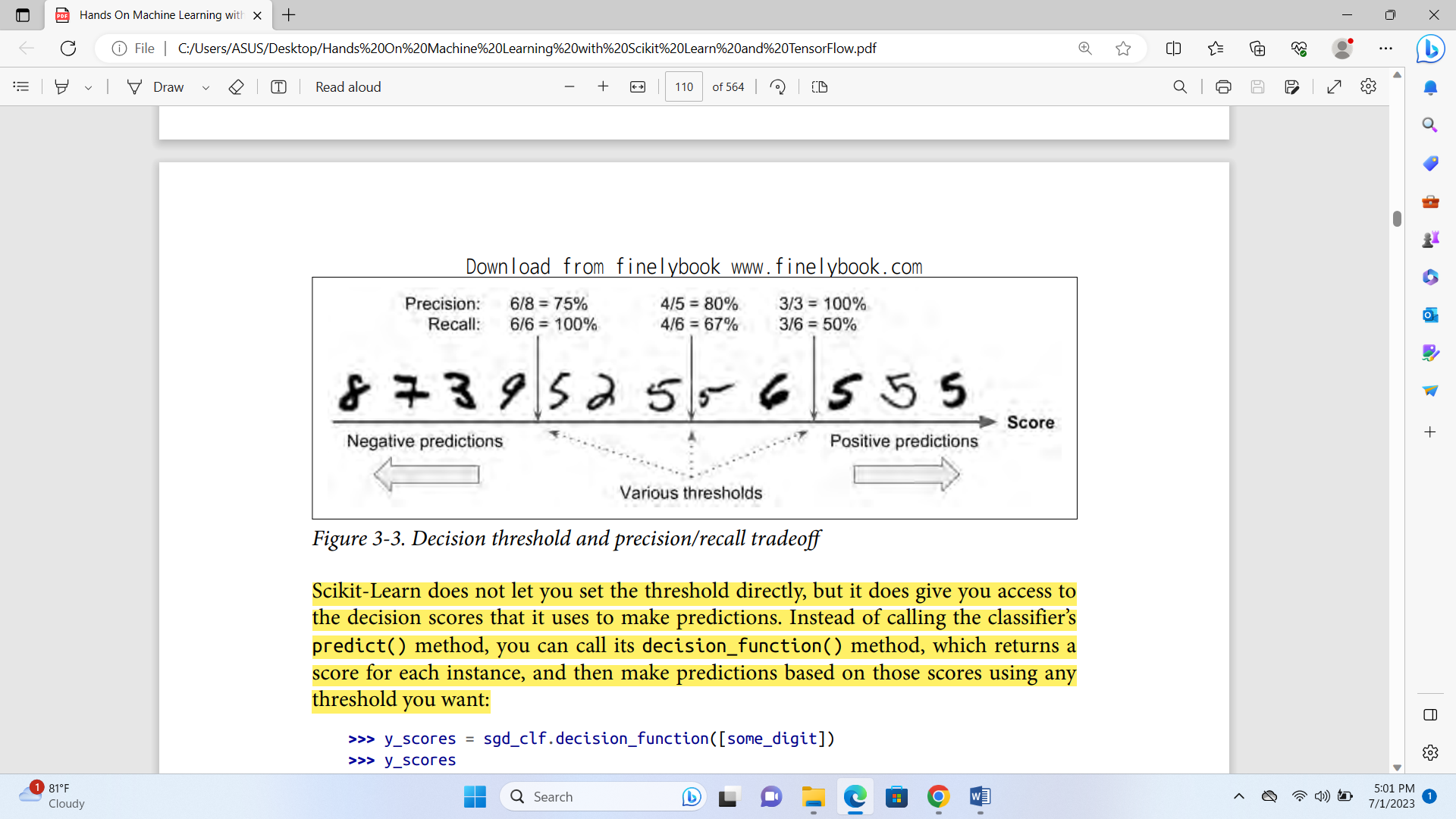
**It is clear that compared to AM and GM, HM penalizes model the most when even one of Precision and Recall is low.**

Whereas the regular mean treats all values equally, the harmonic mean gives much more weight to low values.

**Precision/Recall Tradeoff and Thresholds**

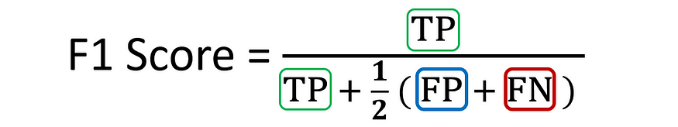
The F1 score favors classifiers that have similar precision and recall. This is not always what you want: in some contexts you mostly care about precision, and in other con‐ texts you really care about recall. For example, if you trained a classifier to detect vid‐ eos that are safe for kids, you would probably prefer a classifier that rejects many good videos (low recall) but keeps only safe ones (high precision), rather than a clas‐ sifier that has a much higher recall but lets a few really bad videos show up in your product (in such cases, you may even want to add a human pipeline to check the clas‐ sifier’s video selection). On the other hand, suppose you train a classifier to detect shoplifters on surveillance images: it is probably fine if your classifier has only 30% precision as long as it has 99% recall (sure, the security guards will get a few false alerts, but almost all shoplifters will get caught). Unfortunately, you can’t have it both ways: increasing precision reduces recall, and vice versa. This is called the precision/recall tradeoff.

Below figure shows a few digits positioned from the lowest score on the left to the highest score on the right. Suppose the deci‐ sion threshold is positioned at the central arrow (between the two 5s): you will find 4 true positives (actual 5s) on the right of that threshold, and one false positive (actually a 6). Therefore, with that threshold, the precision is 80% (4 out of 5). But out of 6 actual 5s, the classifier only detects 4, so the recall is 67% (4 out of 6). Now if you raise the threshold (move it to the arrow on the right), the false positive (the 6) becomes a true negative, thereby increasing precision (up to 100% in this case), but one true positive becomes a false negative, decreasing recall down to 50%. Conversely, lowering the threshold increases recall and reduces precision.



# Micro, Macro & Weighted Averages of F1 Score

If we express it in terms of True Positive (TP), False Positive (FP), and False Negative (FN), we get this equation:

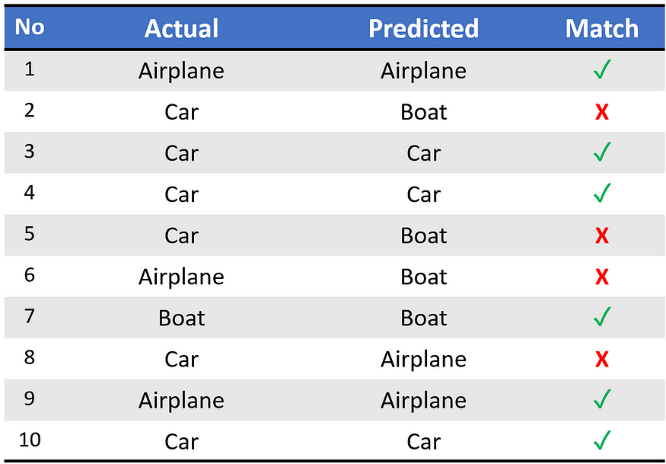


To illustrate the concepts of averaging F1 scores, we will use the following example in the context of this tutorial.

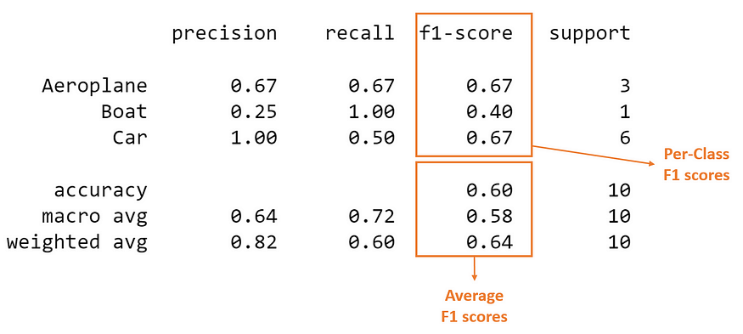
Imagine we have trained an **image classification model** on a **multi-class** dataset containing images of **three** classes: **A**irplane, **B**oat, and **C**ar.



We use this model to **predict**the classes of **ten**test set images. Here are the **raw predictions**:



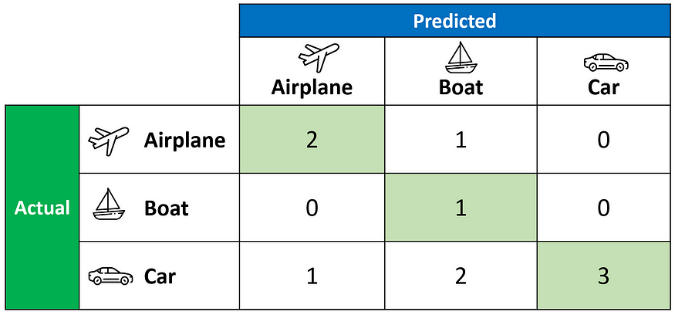
Upon running sklearn.metrics.classification\_report, we get the following classification report:



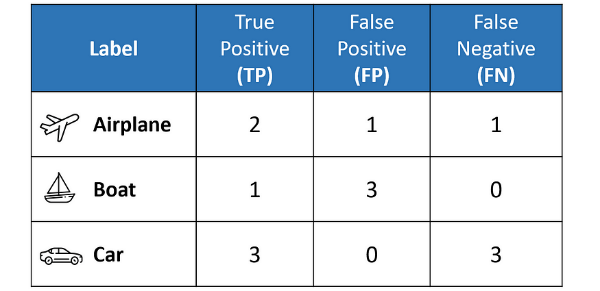
The columns (in orange) with the **per-class** scores (i.e., score for each class) and **average**scores are the focus of our discussion.

We can see from the above that the dataset is **imbalanced**(only one out of ten test set instances is ‘Boat’). Thus the **proportion of correct matches** (aka accuracy) would be ineffective in assessing model performance.

Instead, let us look at the **confusion matrix** for a holistic understanding of the model predictions.



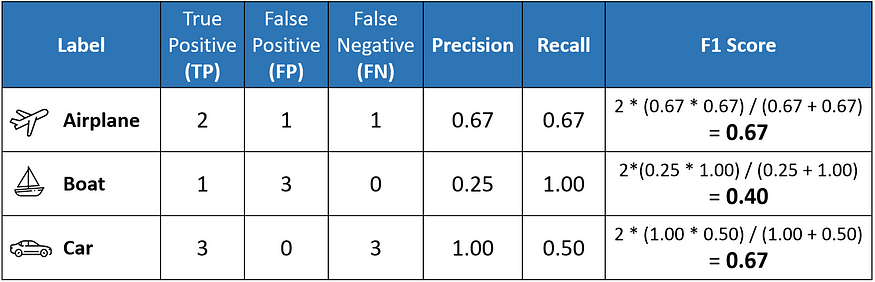
The confusion matrix above allows us to compute the critical values of True Positive (**TP**), False Positive (**FP**), and False Negative (**FN**), as shown below.



The above table sets us up nicely to compute the **per-class** values of **precision**, **recall**, and F1 score for each of the three classes.

It is important to remember that in **multi-class classification, we calculate the F1 score for each class in a One-vs-Rest (OvR)**approach instead of a single overall F1 score, as seen in binary classification.

In this **OvR** approach, we determine the metrics for each class separately, as if there is a different classifier for each class. Here are the per-class metrics (with the F1 score calculation displayed):



However, instead of having multiple per-class F1 scores, it would be better to **average**them to obtain a **single number** to describe overall performance.

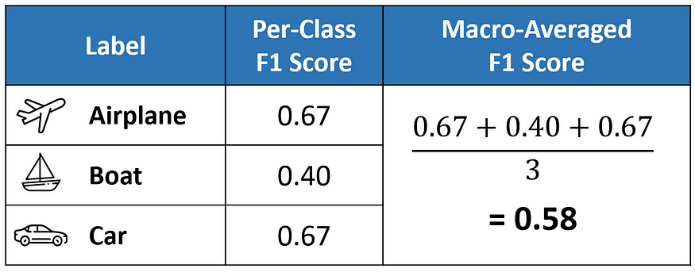
Now, let’s discuss the **averaging**methods that led to the classification report’s **three different average F1 scores**.

# Macro Average

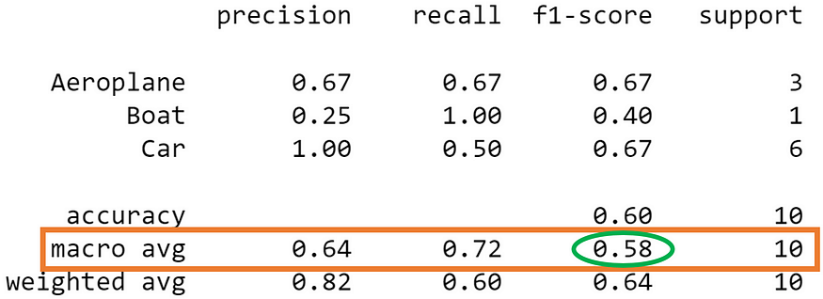
**Macro averaging**is perhaps the most straightforward among the numerous averaging methods.

The macro-averaged F1 score (or macro F1 score) is computed using the arithmetic mean (aka **unweighted**mean) of all the per-class F1 scores.

This method treats all classes equally regardless of their **support**values.



The value of **0.58** we calculated above matches the macro-averaged F1 score in our classification report.

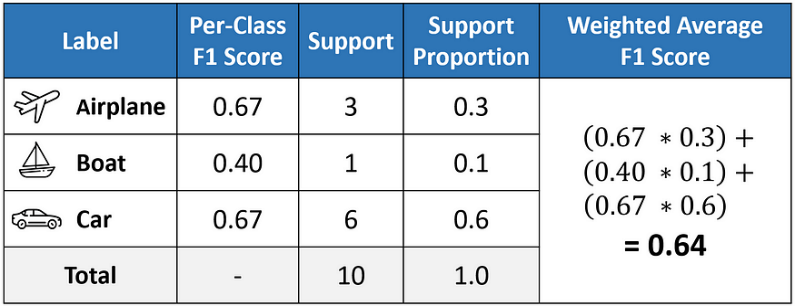


# Weighted Average

The **weighted-averaged**F1 score is calculated by taking the mean of all per-class F1 scores **while considering each class’s support**.

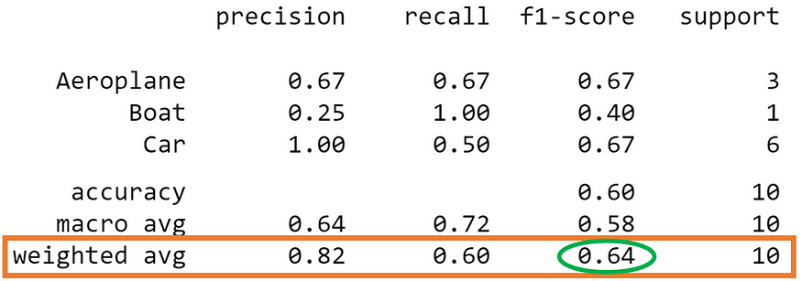
*S****upport****refers to the number of actual occurrences of the class in the dataset. For example, the support value of 1 in****Boat****means that there is only one observation with an actual label of Boat.*

The ‘weight’ essentially refers to the proportion of each class’s support relative to the sum of all support values.



With weighted averaging, the output average would have accounted for the contribution of each class as weighted by the number of examples of that given class.

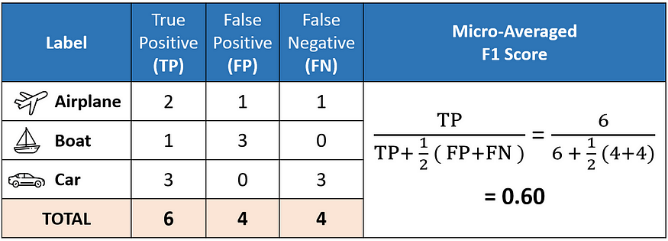
The calculated value of **0.64** tallies with the weighted-averaged F1 score in our classification report.



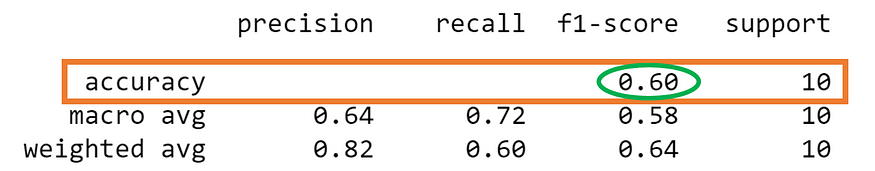
# Micro Average

Micro averaging computes a **global average**F1 score by counting the **sums** of the True Positives (**TP**), False Negatives (**FN**), and False Positives (**FP**).

We first sum the respective TP, FP, and FN values across all classes and then plug them into the F1 equation to get our micro F1 score.

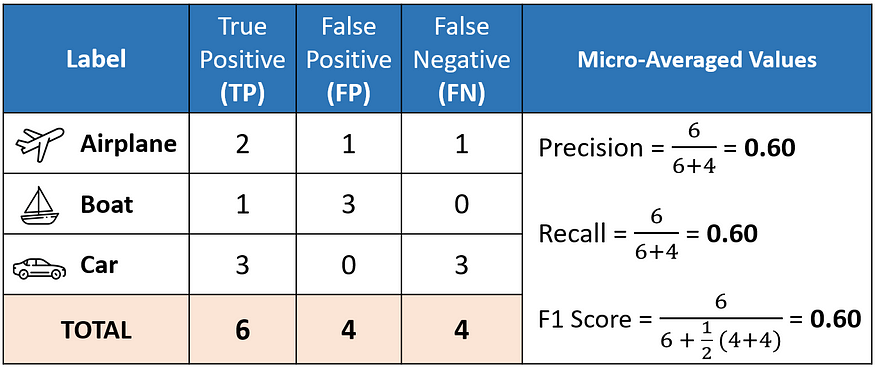


In the classification report, you might be wondering why our micro F1 score of **0.60** is displayed as ‘accuracy’ and why there is **NO row stating**‘**micro avg’**.



This is because micro-averaging essentially computes the **proportion**of **correctly classified** observations out of all observations. If we think about this, this definition is what we use to calculate overall **accuracy**.

Furthermore, if we were to do micro-averaging for precision and recall, we would get the same value of **0.60**.



These results mean that in multi-class classification cases where each observation has a **single label**, the **micro-F1**, **micro-precision**, **micro-recall,** and **accuracy**share the **same**value (i.e.,**0.60** in this example).

And this explains why the classification report **only needs to display a single accuracy value** since micro-F1, micro-precision, and micro-recall also have the same value.

**NOTE: micro-F1 = accuracy = micro-precision = micro-recall**

# Which average should I choose?

In general, if you are working with an imbalanced dataset where all classes are equally important, using the **macro**average would be a good choice as it treats all classes equally.

It means that for our example involving the classification of airplanes, boats, and cars, we would use the macro-F1 score.

If you have an imbalanced dataset but want to assign greater contribution to classes with more examples in the dataset, then the **weighted**average is preferred.

This is because, in weighted averaging, the contribution of each class to the F1 average is weighted by its size.

Suppose you have a balanced dataset and want an easily understandable metric for overall performance regardless of the class. In that case, you can go with accuracy, which is essentially our **micro** F1 score.

# Beyond the F-1 score: A look at the F-beta score

While many machine learning practitioners frequently use the F-1 score, fewer are familiar with its generalized form, the F-beta score. The F-beta score calculation follows the same form as the F-1 score, however it also allows you to decide how to weight the balance between precision and recall using the **beta** parameter. The F-beta score can be calculated as follows:

https://miro.medium.com/v2/resize:fit:550/1*2aggmDVBW0EAqr_tEy3uqg.png

When beta=1, the F-beta score is equivalent to the F-1 Score. When beta=0.5, this score is the F-0.5 score, and so on.

So what is actually happening when we adjust the beta parameter, and why would we want to do it? Let’s explore with an example.

## **Imbalanced Classification and F-beta Scores**

I’ll be demonstrating this concept using the [Haberman’s Survival Dataset](https://www.kaggle.com/gilsousa/habermans-survival-data-set" \t "_blank) available on Kaggle to illustrate these points. This dataset contains information for patients who had undergone surgery for breast cancer between 1958 and 1970, and contains 308 observations with four attributes: (1) age of patient at time of operation, (2) year of the operation, (3) the number of positive axillary nodes detected, and (4) whether the patient died within five years. The Haberman’s dataset is an example of class imbalance — of the 306 patients, 226 (74%) survived at least five years, while only 81 (26%) died within 5 years.

First, I load in the data set, and partition the data using a 75–25% train-test split.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Load heart data

df = pd.read\_csv('haberman.csv')

df.head()

# https://miro.medium.com/v2/resize:fit:240/1*FnFxReilItVUP59xNMFiXA.png

# Perform train-test split

from sklearn.model\_selection import 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=0.25, random\_state=24)

# Next, I calculate the F-0.5, F-1, and F-2 scores while varying the ****threshold probability****that a Logistic Regression classifier uses to predict whether a patient died within five years (target=2) or didn’t (target=1). Remember, while Logistic Regression is used to assign a class label, what it’s actually doing is determining the probability that an observation belongs to a specific class. In a typical binary classification problem, an observation must have a probability of > 0.5 to be assigned to the positive class. However, in this case, I will vary that threshold probability value incrementally from 0 to 1. This will result in the range of varying F-beta scores at different thresholds.

# 

# 

# Now that I’ve run the code above and obtained the F-beta scores, I can plot the results against all probability thresholds used.

# Plot f\_beta, precision, recall curves

fig, ax = plt.subplots(figsize=(8,6))

ax.plot(probability\_thresholds, f0\_5\_scores , label='f0.5')

ax.plot(probability\_thresholds, f1\_scores, label='f1')

ax.plot(probability\_thresholds, f2\_scores, label='f2')

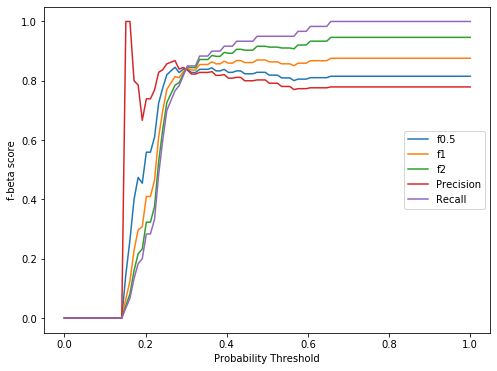
ax.plot(probability\_thresholds, precision\_scores, label='Precision')

ax.plot(probability\_thresholds, recall\_scores, label='Recall')

ax.set\_xlabel('Probability Threshold')

ax.set\_ylabel('f-beta score')

ax.legend(loc='center right');



The F-0.5, F-1, and F-2 curves all have a similar shape across the Logistic Regression probability thresholds. However, the three measures have very different maximum values — F-0.5 at ~0.25 probability, F-1 at ~0.5, and F-2 at ~0.65.

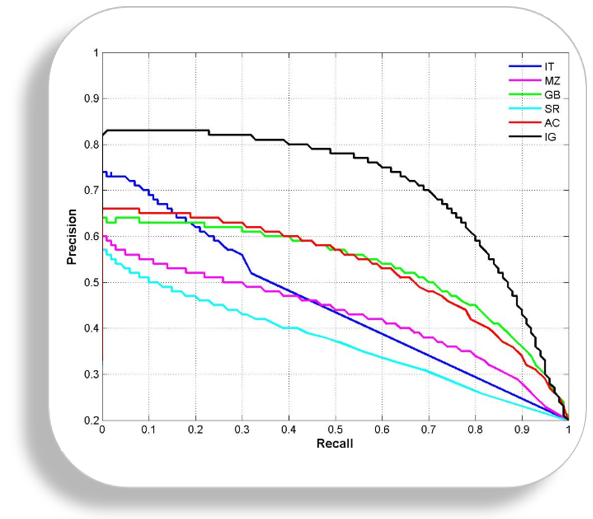
We know that the F-1 score equally balances precision and recall, but when would we want to use a different value for beta?

When we care more about **minimizing false positives** than minimizing false negatives, we would want to select a **beta value of < 1** for the F-beta score. In other words, precision would be given more weight than recall in this scenario. In the example above, using the F-0.5 score, a logistic regression probability threshold of ~0.25 would yield the best balance of precision and recall.

On the other hand, when the priority is to minimize false negatives, we would want to select a **beta value of >1** for the F-beta score. Recall would be considered more important than precision in this scenario. We can imagine that in a real-world problem like the Haberman’s data set, it would probably be desirable to choose a higher beta value to ensure that as many positive cases (people that died within five years) could be identified, at the expense of having more false positives. Using the F-2 score, a logistic regression probability threshold of ~0.65 might be better for this scenario.

# ****PR curve****

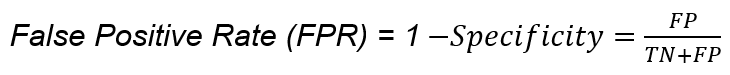
It is the curve between precision and recall for various threshold values. In the figure below we have 6 predictors showing their respective precision-recall curve for various threshold values. The top right part of the graph is the ideal space where we get high precision and recall. Based on our application we can choose the predictor and the threshold value. PR AUC is just the area under the curve. The higher its numerical value the better.

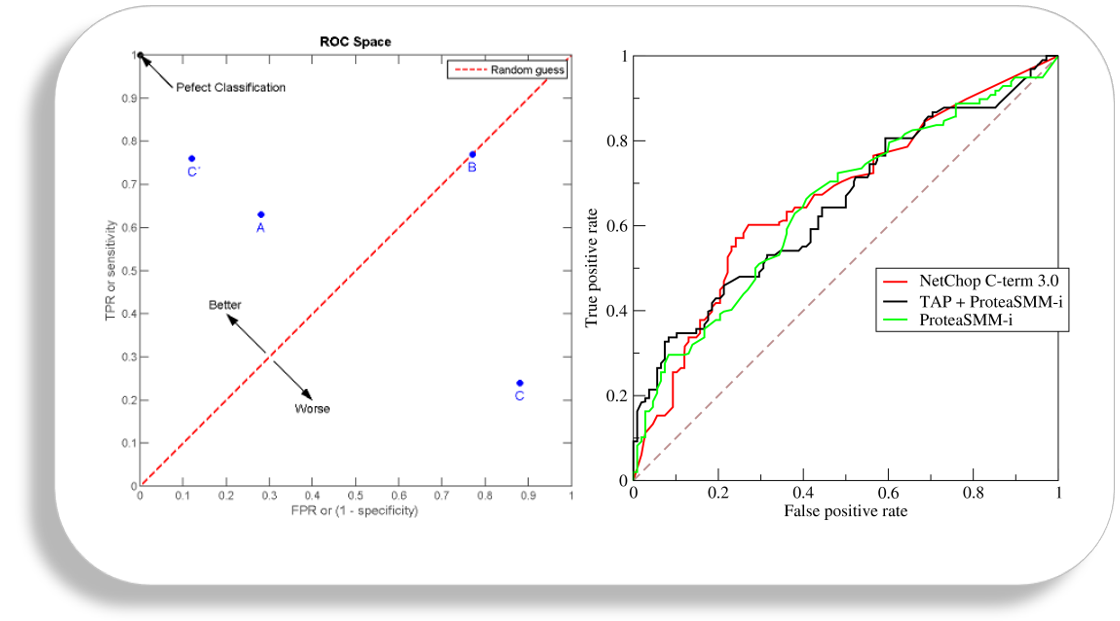


# ****ROC curve****

ROC stands for receiver operating characteristic and the graph is plotted against TPR and FPR for various threshold values. As TPR increases FPR also increases. As you can see in the first figure, we have four categories and we want the threshold value that leads us closer to the top left corner. Comparing different predictors (here 3) on a given dataset also becomes easy as you can see in figure 2, one can choose the threshold according to the application at hand. ROC AUC is just the area under the curve, the higher its numerical value the better.



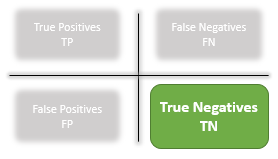




# PR vs ROC curve

Both the metrics are widely used to judge a models performance.

**Which one to use PR or ROC?**



The answer lies in TRUE NEGATIVES.

**Due to the absence of TN in the precision-recall equation, they are useful in imbalanced classes**. In the case of class imbalance when there is a majority of the negative class. The metric doesn’t take much into consideration the high number of TRUE NEGATIVES of the negative class which is in majority, giving better resistance to the imbalance. This is important when the detection of the positive class is very important.

Like to detect cancer patients, which has a high class imbalance because very few have it out of all the diagnosed. We certainly don’t want to miss on a person having cancer and going undetected (recall) and be sure the detected one is having it (precision).

**Due to the consideration of TN or the negative class in the ROC equation, it is useful when both the classes are important to us.**Like the detection of cats and dog. The importance of true negatives makes sure that both the classes are given importance, like the output of a CNN model in determining the image is of a cat or a dog.