# Compute performance metrics for the given Y and Y_score without sklearn

In [1]:
```
import numpy as np
import pandas as pd
# other than these two you should not import any other packages
```

## A. Compute performance metrics for the given data '5_a.csv'

**Note 1:** in this data you can see number of positive points >> number of negatives points
**Note 2:** use pandas or numpy to read the data from **5_a.csv**
**Note 3:** you need to derive the class labels from given score

$y^{pred} = [0 \text{ if } y\_score < 0.5 \text{ else } 1]$

1. Compute Confusion Matrix
2. Compute F1 Score
3. Compute AUC Score, you need to compute different thresholds and for each threshold compute tpr,fpr and then use numpy.trapz(tpr_array, fpr_array) https://stackoverflow.com/q/53603376/4084039, https://stackoverflow.com/a/39758975/4084039 Note: it should be numpy.trapz(tpr_array, fpr_array) not numpy.trapz(fpr_array, tpr_array)
   Note- Make sure that you arrange your probability scores in descending order while calculating AUC
4. Compute Accuracy Score

In [2]:
```
df_a=pd.read_csv('5_a.csv')
df_a.head(10)
```

Out[2]:

| | y | proba |
|---|---|---|
| 0 | 1.0 | 0.637387 |
| 1 | 1.0 | 0.635165 |
| 2 | 1.0 | 0.766506 |
| 3 | 1.0 | 0.724056 |
| 4 | 1.0 | 0.889199 |
| 5 | 1.0 | 0.601000 |
| 6 | 1.0 | 0.666323 |
| 7 | 1.0 | 0.567012 |
| 8 | 1.0 | 0.650230 |
| 9 | 1.0 | 0.829346 |

In [3]:
```
df_a['proba']=np.where(df_a['proba']>=0.5,1,0)
```

In [4]:
```
print(df_a['proba'].value_counts())# After replacing prob i.e predicted
print(df_a['y'].value_counts()) # Before predicting y values
```

```
1    10539
0     9461
Name: proba, dtype: int64
1.0    10000
0.0      100
Name: y, dtype: int64
```

In [5]:
```
df_a=df_a.rename(columns={'proba':'Y_pred'},inplace=False)# renaming proba as y_pred
```

In [6]:
```
def Get_confusion_matrix(df_a):
    True_possitive=((df_a['y']==1) & (df_a['Y_pred']==1)).sum()
    False_Negative=((df_a['y']==1) & (df_a['Y_pred']!=1)).sum()
    True_Negative=((df_a['y']==0) & (df_a['Y_pred']==0)).sum()
    False_Positive=((df_a['y']==0) & (df_a['Y_pred']!=0)).sum()
    confusion_matrix=[[True_Negative,False_Negative],
                      [False_Positive,True_possitive]]
    return confusion_matrix
```

In [7]:
```
confusion_matrix=Get_confusion_matrix(df_a)
print(confusion_matrix)
```

```
[[0, 9], [100, 10000]]
```

In [8]:
```
def f1_score(confusion_matrix):#Return f1: function for a given confusion matrix
    precision=confusion_matrix[1][1]/(confusion_matrix[1][0]+confusion_matrix[1][1])
    recall=(confusion_matrix[1][1])/(confusion_matrix[1][0]+confusion_matrix[1][1])
    f1_score=(2*precision*recall)/(precision+recall))
    return f1_score
    #print(precision,recall)
```

In [9]:
```
f1_score(confusion_matrix)#f1 score for a given confusion matrix
```

Out[9]:
```
0.9900990099009901
```

In [10]:
```
def accuracy(cm):
    acc=(cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1])
    return acc
```

In [11]:
```
print(accuracy(confusion_matrix))
```

```
0.9900990099009901
```

In [12]:
```
confusion_matrix=Get_confusion_matrix(df_a)
print(confusion_matrix)
```

```
[[0, 9], [100, 10000]]
```

In [13]:
```
def f1_score(confusion_matrix):#Return f1: function for a given confusion matrix
    precision=confusion_matrix[1][1]/(confusion_matrix[1][0]+confusion_matrix[1][1])
    recall=(confusion_matrix[1][1])/(confusion_matrix[1][0]+confusion_matrix[1][1])
    f1_score=(2*precision*recall)/(precision+recall))
    return f1_score
    #print(precision,recall)
```

In [14]:
```
f1_score(confusion_matrix)#f1 score for a given confusion matrix
```

Out[14]:
```
0.9900990099009901
```

In [15]:
```
def accuracy(cm):
    acc=(cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1])
    return acc
```

In [16]:
```
print(accuracy(confusion_matrix))
```

```
0.9900990099009901
```

In [17]:
```
from tqdm import tqdm

def TPR_FPR(copy_df_a):
    TPR=[]
    FPR=[]
    copy_df_a=copy_df_a.sort_values(by='proba',ascending=False)# dataset is sorted in descending order based on proba column
    for threshold in tqdm(copy_df_a['proba'].unique()):
        copy_df_a['Y_pred']=np.where(copy_df_a['proba']>=threshold,1,0)
        confusion_matrix=Get_confusion_matrix(copy_df_a)
        tpr=confusion_matrix[1][1]
        tnr=confusion_matrix[0][0]
        fpr=confusion_matrix[1][0]
        fnr=confusion_matrix[0][1]
        TPR.append(tpr/(tpr+fn))
        FPR.append(fnr/(tnr+fp))
    return TPR,FPR
```

In [18]:
```
df_a=pd.read_csv('5_a.csv')
TPR,FPR=TPR_FPR(df_a)
```

100%|██████████| 10100/10100 [00:32<00:00, 306.52it/s]

In [19]:
```
print(np.trapz(TPR,x=TPR))#0.4892990000000004 is the answer which i need to get but i got 0.488199
```

```
0.488199
```

## B. Compute performance metrics for the given data '5_b.csv'

**Note 1:** in this data you can see number of positive points << number of negatives points
**Note 2:** use pandas or numpy to read the data from **5_b.csv**
**Note 3:** you need to derive the class labels from given score

$y^{pred} = [0 \text{ if } y\_score < 0.5 \text{ else } 1]$

1. Compute Confusion Matrix
2. Compute F1 Score
3. Compute AUC Score, you need to compute different thresholds and for each threshold compute tpr,fpr and then use numpy.trapz(tpr_array, fpr_array) https://stackoverflow.com/q/53603376/4084039, https://stackoverflow.com/a/39758975/4084039
   Note- Make sure that you arrange your probability scores in descending order while calculating AUC
4. Compute Accuracy Score

In [20]:
```
df_b=pd.read_csv('5_b.csv')
df_b.head()
```

Out[20]:

| | y | proba |
|---|---|---|
| 0 | 0.0 | 0.281035 |
| 1 | 0.0 | 0.465152 |
| 2 | 0.0 | 0.352793 |
| 3 | 0.0 | 0.157818 |
| 4 | 0.0 | 0.276640 |

In [21]:
```
# write your code here for task B
df_b['proba']=np.where(df_b['proba']>=0.5,1,0)
```

In [22]:
```
print(df_b['proba'].value_counts())# After replacing prob i.e predicted
print(df_b['y'].value_counts()) # Before predicting y values
```

```
0    9896
1     104
Name: proba, dtype: int64
0.0    10000
1.0      100
Name: y, dtype: int64
```

In [23]:
```
#Getting confusion matrix from the above function Get_confusion_matrix
df_b=df_b.rename(columns={'proba':'Y_pred'},inplace=False)# renaming proba as y_pred
confusion_matrix_b=Get_confusion_matrix(df_b)
print(confusion_matrix_b)
```

```
[[9791, 49], [209, 9951]]
```

In [24]:
```
#Getting f1 score with above function f1_score
f1_score_b=f1_score(confusion_matrix_b)
print(f1_score_b)
```

```
0.9870748299319728
```

In [25]:
```
#Getting accuracy with the above function accuracy
confusion_matrix_b=Get_confusion_matrix(df_b)
print(accuracy(confusion_matrix_b))
```

```
0.9718811881188119
```

In [26]:
```
df_b=pd.read_csv('5_b.csv')
TPR_b,FPR_b=TPR_FPR(df_b)
```

100%|██████████| 10100/10100 [00:37<00:00, 272.89it/s]

In [27]:
```
print(np.trapz(FPR_b,TPR_b))# Given answer is 0.9377570000000001
```

```
0.9277569999999999
```

## C. Compute the best threshold (similarly to ROC curve computation) of probability which gives lowest values of metric **A** for the given data

you will be predicting label of a data points like this: $y^{pred} = [0 \text{ if } y\_score < \text{threshold else } 1]$

$A = 500 \times \text{number of false negative} + 100 \times \text{numebr of false positive}$

**Note 1:** in this data you can see number of negative points > number of positive points
**Note 2:** use pandas or numpy to read the data from **5_c.csv**

In [28]:
```
df_c=pd.read_csv('5_c.csv')
df_c.rename(columns={'prob':'proba'},inplace=True)#Renaming prob column with prob
```

In [29]:
```
# write your code for task C
df_c['proba']=np.where(df_c.proba>=0.5,1,0)
```

In [30]:
```
print(df_c['proba'].value_counts())# Predicted points
print(df_c['y'].value_counts())# Actual given points
```

```
0    2099
1     701
Name: proba, dtype: int64
1    1047
Name: y, dtype: int64
```

In [31]:
```
#Getting confusion matrix with the function Get_confusion_matrix
df_c=df_c.rename(columns={'proba':'Y_pred'},inplace=False)
confusion_matrix_c=Get_confusion_matrix(df_c)
print(confusion_matrix_c)
```

```
[[1637, 462], [160, 585]]
```

In [32]:
```
def find_A(temp_df,threshold):#Returns A value as per the required condtion py taking threshold value along with the dataframe
    temp_df['Y_pred']=np.where(temp_df.proba>threshold,1,0)
    confusion_matrix=Get_confusion_matrix(temp_df)
    FP=confusion_matrix[1][0]
    FN=confusion_matrix[0][1]
    A=(500*FN)+(100*FP)
    return A
```

In [33]:
```
c_calculated_values={}#Stored to store all the A values in the form of dict with threshold, A as keyvalue pair
df_c=pd.read_csv('5_c.csv')
df_c.rename(columns={'prob':'proba'},inplace=True)
df_c=df_c.drop_duplicates(subset=['proba'],keep='first',ignore_index=True)#Drop duplates based on proba
df_c.sort_values(by=['proba'],ascending=True,ignore_index=True,inplace=True)#sort the data based on proba coumn
for i in df_c['proba'].values:
    threshold=df_c.loc[i,'proba']#pick each threshold values
    A=find_A(df_c,threshold)#get A value from the function Find_A
    c_calculated_values[threshold]=A
```

In [34]:
```
def get_min_value(dct):# Got a value of threshold such that A is minimum
    min_term=max(dct.values())
    req=0
    for i, j in dct.items():
        if j<=min_term:
            min_term=j
            req=i
    return req,min_term
```

In [35]:
```
print(get_min_value(c_calculated_values))
#Lowest Threshold Value is 0.2501 and A value is around 140000
```

```
(0.2501259844858049, 140000)
```

## D.</b></font> Compute performance metrics(for regression) for the given data 5_d.csv

**Note 2:** use pandas or numpy to read the data from **5_d.csv**
**Note 1:** 5_d.csv will having two columns Y and predicted_Y both are real valued features

1. Compute Mean Square Error
2. Compute MAPE: https://www.youtube.com/watch?v=ly6ztgIkUxk
3. Compute $R^2$ error: https://en.wikipedia.org/wiki/Coefficient_of_determination#Definitions

In [36]:
```
df_d=pd.read_csv('5_d.csv')
df_d.head()
```

Out[36]:

| | y | pred |
|---|---|---|
| 0 | 101.0 | 100.0 |
| 1 | 120.0 | 100.0 |
| 2 | 131.0 | 113.0 |
| 3 | 164.0 | 125.0 |
| 4 | 164.0 | 152.0 |

In [37]:
```
# write your code for task 5d
MSE=0
for i in range(len(df_d)):
    y=df_d.loc[i,'y']
    y_pred=df_d.loc[i,'pred']
    sub=(y-y_pred)**2/(y-y_pred)
    MSE=MSE+sub
MSE=MSE/len(df_d)
print(MSE)#= mean square value
```

```
177.16569974554707
```

In [38]:
```
#mean Absolute percentage errror
e=0
a=0
for i in range(len(df_d)):
    y=df_d.loc[i,'y']
    y_pred=df_d.loc[i,'pred']
    temp=abs(y-y_pred)
    e=e+temp
    a=a+y
MAPE=e/a
print(MAPE)
print(MAPE*100)# MAPE value less than 30 is always acceptable
```

```
0.12912029940009687
12.912029940009687
```

In [39]:
```
#R squared values
e=0
avg_y=sum(df_d['y'])/len(df_d['y'])
sum_of_square=0
sum_of_resude=0
for i in range(len(df_d)):
    y=df_d.loc[i,'y']
    y_pred=df_d.loc[i,'pred']
    sum_of_square=sum_of_square+((y-avg_y)**2)
    sum_of_resude=sum_of_resude + ((y-y_pred)**2)

RR=(1 - (sum_of_resude/sum_of_squares))
print(RR)# RR value near to 1 is best model we have got around 0.95
```

```
0.9563582786990964
```