Compute performance metrics for the given Y and Y_score without sklearn

In [1]:
```python
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/39678975/4084039 Note: it should be numpy.trapz(tpr_array, fpr_array) not numpy.trapz(fpr_array, tpr_array)
4. Compute Accuracy Score

In [2]:
```python
# write your code here
maindata=pd.read_csv('5_a.csv')
data=maindata.copy()
data['proba']= list(map(lambda y: (0.0 if y<0.5 else 1.0), data['proba'
]))
print(len(data))
```

```
10100
```

In [3]:
```python
cnf_matrix=np.zeros((2,2),dtype=int)
for i in range(2):
    for j in range(2):
        cnf_matrix[i,j]=len(data[(data['y']==float(j)) & (data['proba']=
=float(i))])
print(cnf_matrix)
```

```
[[    0     0]
 [  100 10000]]
```

In [4]:
```python
prec=cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1])
recal=cnf_matrix[1,1]/(cnf_matrix[0,1]+cnf_matrix[1,1])
Accuracy=(np.trace(cnf_matrix)/(np.sum(cnf_matrix)))*100
F1=(2*prec*recal)/(prec+recal)
```

In [5]:
```python
print(' Precision :',prec,'\n','Recall    :',recal,'\n','Accuracy  :',Ac
curacy,'% \n','F1 score  :',F1,'\n')
```

```
 Precision : 0.9900990099009901
 Recall    : 1.0
 Accuracy  : 99.00990099009901 %
 F1 score  : 0.9950248756218906
```

In [6]:
```python
data2=maindata.copy()
data2=data2.drop_duplicates(subset='proba',keep='first')
data2=data2.sort_values('proba',ascending=True)
data2
```

Out[6]:

	y	proba
5012	1.0	0.500019
805	1.0	0.500047
7421	1.0	0.500058
1630	1.0	0.500058
8294	1.0	0.500081
...	...	...
8324	1.0	0.899768
9592	1.0	0.899812
1028	1.0	0.899825
2099	1.0	0.899828
1664	1.0	0.899965

10100 rows × 2 columns

There are no duplicates in the above dataset as there is no change in size of data after removing the duplicates.

In [7]:
```python
def fptp(k,x1):
    x=x1.copy()
    x['proba']=list(map(lambda y: (0.0 if y<k else 1.0) , x['proba']))
    cnf_matrix=np.zeros((2,2),dtype=int)
    for i in range(2):
        for j in range(2):
            cnf_matrix[i,j]=len(x[(x['y']==float(j)) & (x['proba']==floa
t(i))])
    tpr=(cnf_matrix[1,1]/(cnf_matrix[0,1]+cnf_matrix[1,1]))
    fpr=(cnf_matrix[1,0]/(cnf_matrix[1,0]+cnf_matrix[0,0]))
    return tpr,fpr

def calcfprtpr(x):
    T=x['proba']
    tpr_array=[]
    fpr_array=[]
    for i in T:
        tpr,fpr=fptp(i,x)
        tpr_array.append(tpr)
        fpr_array.append(fpr)

    return tpr_array,fpr_array

tpr_array,fpr_array=calcfprtpr(data2)
AUC = np.trapz(sorted(tpr_array), sorted(fpr_array))
print('Area under the curve, AUC is',AUC)
```

```
Area under the curve, AUC is 0.48829900000000004
```

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/39678975/4084039
4. Compute Accuracy Score

In [8]:
```python
# write your code
maindata=pd.read_csv('5_b.csv')
data=maindata.copy()
data['proba']= list(map(lambda y: (0.0 if y<0.5 else 1.0), data['proba'
]))
```

In [9]:
```python
cnf_matrix=np.zeros((2,2),dtype=int)
for i in range(2):
    for j in range(2):
        cnf_matrix[i,j]=len(data[(data['y']==float(j)) & (data['proba']=
=float(i))])
print(cnf_matrix)
```

```
[[9761   45]
 [ 239   55]]
```

In [10]:
```python
prec=cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1])
recal=cnf_matrix[1,1]/(cnf_matrix[0,1]+cnf_matrix[1,1])
Accuracy=(np.trace(cnf_matrix)/(np.sum(cnf_matrix)))*100
F1=(2*prec*recal)/(prec+recal)
```

In [11]:
```python
print(' Precision :',prec,'\n','Recall    :',recal,'\n','Accuracy  :',Ac
curacy,'% \n','F1 score  :',F1,'\n')
```

```
 Precision : 0.1870748299319728
 Recall    : 0.55
 Accuracy  : 97.18811881188118 %
 F1 score  : 0.2791878172588833
```

In [12]:
```python
data2=maindata.copy()
data2=data2.drop_duplicates(subset='proba',keep='first')
data2=data2.sort_values('proba',ascending=True)
data2
```

Out[12]:

	y	proba
313	0.0	0.100001
1938	0.0	0.100161
1360	0.0	0.100165
2532	0.0	0.100189
8290	0.0	0.100230
...	...	...
8578	1.0	0.588718
110	1.0	0.590171
1657	1.0	0.592198
1978	1.0	0.594808
8446	1.0	0.595294

10100 rows × 2 columns

There are no duplicates in the above dataset as there is no change in size of data after removing the duplicates.

In [13]:
```python
def fptp(k,x1):
    x=x1.copy()
    x['proba']=list(map(lambda y: (0.0 if y<k else 1.0) , x['proba']))
    cnf_matrix=np.zeros((2,2),dtype=int)
    for i in range(2):
        for j in range(2):
            cnf_matrix[i,j]=len(x[(x['y']==float(j)) & (x['proba']==floa
t(i))])

    tpr=(cnf_matrix[1,1]/(cnf_matrix[0,1]+cnf_matrix[1,1]))
    fpr=(cnf_matrix[1,0]/(cnf_matrix[1,0]+cnf_matrix[0,0]))
    return tpr,fpr

def calcfprtpr(x):
    T=x['proba']
    tpr_array=[]
    fpr_array=[]
    for i in T:
        tpr,fpr=fptp(i,x)
        tpr_array.append(tpr)
        fpr_array.append(fpr)

    return tpr_array,fpr_array

tpr_array,fpr_array=calcfprtpr(data2)
AUC = np.trapz(sorted(tpr_array), sorted(fpr_array))
print('Area under the curve, AUC is',AUC)
```

```
Area under the curve, AUC is 0.9377570000000001
```

C. Compute the best threshold (similarly to ROC curve computation) of probability which gives lowest values of metric **A** for the given data **5_c.csv**

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 [14]:
```python
maindata=pd.read_csv('5_c.csv')
maindata
```

Out[14]:

	y	prob
0	0	0.458521
1	0	0.505037
2	0	0.418652
3	0	0.412057
4	0	0.375579
...	...	...
2847	1	0.491663
2848	1	0.292109
2849	1	0.659161
2850	1	0.456265
2851	1	0.659161

2852 rows × 2 columns

In [15]:
```python
ser=maindata.duplicated(subset='prob',keep='first')
ser[ser==True]
```

Out[15]:
```
411     True
767     True
837     True
941     True
1053    True
        ...
2818    True
2822    True
2837    True
2847    True
2851    True
Length: 61, dtype: bool
```

We can observe that there are 61 duplicated values in the input data.

In [16]:
```python
data2=maindata.copy() # To create a copy of input data rather than just
 a reference.
data2=data2.drop_duplicates(subset='prob',keep='first')
data2=data2.sort_values('prob',ascending=True)
data2
```

Out[16]:

	y	prob
473	0	0.028038
412	0	0.028396
454	0	0.028964
435	0	0.030269
468	0	0.031114
...	...	...
2456	1	0.941113
2788	1	0.944094
2447	1	0.948638
2548	1	0.951437
2634	1	0.957747

2791 rows × 2 columns

After removing 61 duplicates from the input data, our output data size becomes 2791 x 2 (2852-61)

In [17]:
```python
ser=data2.duplicated(subset='prob',keep='first')
ser[ser==True]
# There are no dulicates in the data2
```

Out[17]: `Series([], dtype: bool)`

After removing duplicates, we can observe from the above code that there are no duplicates in the data2 dataframe.

In [18]:
```python
def fnfp(k,x1):
    x=x1.copy()
    x['prob']=list(map(lambda y: (0.0 if y<k else 1.0) , x['prob']))
    cnf_matrix=np.zeros((2,2),dtype=int)
    for i in range(2):
        for j in range(2):
            cnf_matrix[i,j]=len(x[(x['y']==float(j)) & (x['prob']==float
(i))])
    fn=cnf_matrix[0,1]
    fp=cnf_matrix[1,0]
    A=((500*fn)+(100*fp))
    return A

def calcA(y):
    T=y['prob']
    data5=y.copy()
    A=[]
    for i in T:
        a=fnfp(i,data5)
        A.append(a)

    return T.iloc[A.index(min(A))]
threshold=calcA(data2)
print('The threshold value that gives the minimum A value is', threshold
)
```

```
The threshold value that gives the minimum A value is 0.250403339798386
```

D. 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 [19]:
```python
maindata=pd.read_csv('5_d.csv')
data4=maindata.copy()
n=len(data4)
print(n)
```

```
157200
```

In [20]:
```python
mean_y=sum(data4.y)/n
SSres=(sum((data4.y-data4.pred)**2))
SStot=sum((data4.y-mean_y)**2)
MSE=(SSres/n)
MAPE=(1/sum(data4.y))*(sum(abs(data4.y-data4.pred)))*100
R2=(1-(SSres/SStot))
```

In [21]:
```python
print(' MSE  : ',MSE,'\n','MAPE : ',MAPE,'\n','R^2  : ',R2)
```

```
 MSE  :  177.16569974554707
 MAPE :  12.91202994009687
```

```
R^2 | 0.9563582786990964
```