Compute performance metrics for the given Y and Y_score without sklearn

In [2]:
```
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/q/53603376/4084039), https://stackoverflow.com/a/39678975/4084039 (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 [3]:
```
# write your code here
#Load 5_a.csv into a pandas dataFrame.
data1= pd.read_csv("C:/Users/aaa/Desktop/saraswati aaic/5_a.csv")
data1.head(10)
```

Out[3]:

	y	proba
0	1.0	0.637387
1	1.0	0.635165
2	1.0	0.766586
3	1.0	0.724564
4	1.0	0.889199
5	1.0	0.601600
6	1.0	0.666323
7	1.0	0.567012
8	1.0	0.650230
9	1.0	0.829346

In [4]:
```
# derive the class labels from given score
# ypred=[0 if y_score < 0.5 else 1]
# data1['ypred'] = [0 if ''< 0.5 else 1 for x in data1['proba']]
f = lambda x: 0 if x < 0.5 else 1
data1['ypred'] = data1['proba'].map(f)
data1.head(10)
# data1.shape
```

Out[4]:

	y	proba	ypred
0	1.0	0.637387	1
1	1.0	0.635165	1
2	1.0	0.766586	1
3	1.0	0.724564	1
4	1.0	0.889199	1
5	1.0	0.601600	1
6	1.0	0.666323	1
7	1.0	0.567012	1
8	1.0	0.650230	1
9	1.0	0.829346	1

Compute Confusion Matrix

In [5]:
```
positive=0
negative=0
for i in range(0,len(data1)):
    if data1['y'].loc[i]==1:
            positive+=1
    elif data1['y'].loc[i]==0:
            negative+=1
print("positive numbers are",positive)
print("negative numbers are",negative)
```

```
positive numbers are 10000
negative numbers are 100
```

In [ ]:
```

```

In [6]:
```
# computing confusion matrix
confusion_matrix=[]
TP=int(data1[(data1.y == 1) & (data1.ypred == 1)].count()[0])
TN=int(data1[(data1.y == 0) & (data1.ypred == 0)].count()[0])
FP=int(data1[(data1.y == 0) & (data1.ypred == 1)].count()[0])
FN=int(data1[(data1.y == 1) & (data1.ypred == 0)].count()[0])

print(TP)
print(TN)
print(FP)
print(FN)
confusion_matrix.append(TN)
confusion_matrix.append(FN)
confusion_matrix.append(FP)
confusion_matrix.append(TP)
# print(confusion_matrix)
x=np.reshape(confusion_matrix,(2,2))
print(x)
```

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

compute precision , recall and F1 Score

In [7]:

```
precision=((TP)/(TP+FP))
print('precision ',precision)

recall=((TP)/(TP+FN))
print('recall ',recall)

F1_score=2*(precision*recall)/(precision+recall)
print('F1-score ',F1_score)
```

```
precision  0.9900990099009901
recall  1.0
F1-score  0.9950248756218906
```

Compute Accuracy Score

In [8]:

```
accuracy_score=(TP+TN)/(TP+FP+FN+TN)
print('accuracy score',accuracy_score)
```

```
accuracy score 0.9900990099009901
```

Compute AUC_CURVE

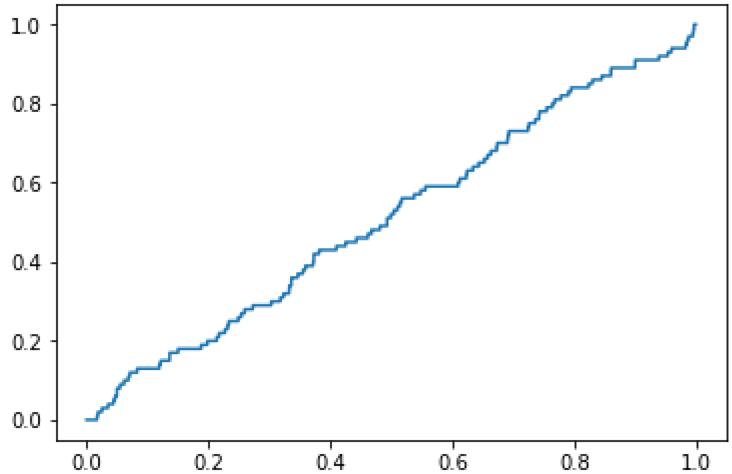
In [32]:

```
from tqdm import tqdm
unique=list(set(data1.proba))
unique.sort()
# print(unique)

TPR=[]
FPR=[]
for i in tqdm(unique):
    data1.loc[data1['proba'] < i, 'ypred'] = 0
    data1.loc[data1['proba'] > i, 'ypred'] = 1

    TP=int(data1[(data1.y == 1) & (data1.ypred == 1)].count()[0])
    TN=int(data1[(data1.y == 0) & (data1.ypred == 0)].count()[0])
    FP=int(data1[(data1.y == 0) & (data1.ypred == 1)].count()[0])
    FN=int(data1[(data1.y == 1) & (data1.ypred == 0)].count()[0])

    tpr=(TP/(TP+FN))
    TPR.append(tpr)
    fpr=(FP/(FP+TN))
    FPR.append(fpr)

```

```
100%|██████████████████████████████████| 10100/10100 [04:18<00:00, 39.09it/s]
```

In [33]:
```
x=sorted(TPR)
y=sorted(FPR)
from matplotlib import pyplot as plt
a=plt.plot(x,y)
print(a)
```

```
[<matplotlib.lines.Line2D object at 0x0000000009FAD2E8>]
```



In [34]:
```
AUC_CURVE= np.trapz(x,y)
AUC_CURVE
```

Out[34]: 0.48829900000000004

/p

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

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

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/q/53603376/4084039), https://stackoverflow.com/a/39678975/4084039 (https://stackoverflow.com/a/39678975/4084039)

4. Compute Accuracy Score

In [12]:

```
# write your code
#Load 5_b.csv into a pandas dataFrame.
data2= pd.read_csv("C:/Users/aaa/Desktop/saraswati aaic/5_b.csv")
data2.head(10)
```

Out[12]:

	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	0.276648
5	0.0	0.190260
6	0.0	0.320328
7	0.0	0.435013
8	0.0	0.284849
9	0.0	0.427919

In [13]:
```
f = lambda x: 0 if x < 0.5 else 1
data2['ypred'] = data2['proba'].map(f)
data2.head(10)
```

Out[13]:

	y	proba	ypred
0	0.0	0.281035	0
1	0.0	0.465152	0
2	0.0	0.352793	0
3	0.0	0.157818	0
4	0.0	0.276648	0
5	0.0	0.190260	0
6	0.0	0.320328	0
7	0.0	0.435013	0
8	0.0	0.284849	0
9	0.0	0.427919	0

In [14]:
```
positive=0
negative=0
for i in range(0,len(data2)):
    if data2['y'].loc[i]==1:
            positive+=1
    elif data2['y'].loc[i]==0:
            negative+=1
print("positive numbers are",positive)
print("negative numbers are",negative)
```

```
positive numbers are 100
negative numbers are 10000
```

compute confusion matrix

In [15]:

```
# computing confusion matrix
confusion_matrix=[]
TP=int(data2[(data2.y == 1) & (data2.ypred == 1)].count()[0])
TN=int(data2[(data2.y == 0) & (data2.ypred == 0)].count()[0])
FP=int(data2[(data2.y == 0) & (data2.ypred == 1)].count()[0])
FN=int(data2[(data2.y == 1) & (data2.ypred == 0)].count()[0])

print(TP)
print(TN)
print(FP)
print(FN)
confusion_matrix.append(TN)
confusion_matrix.append(FN)
confusion_matrix.append(FP)
confusion_matrix.append(TP)
# print(confusion_matrix)
x=np.reshape(confusion_matrix,(2,2))
print(x)
```

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

compute precision ,recall and f1_score

In [16]:

```
precision=((TP)/(TP+FP))
print('precision ',precision)

recall=((TP)/(TP+FN))
print('recall ',recall)

F1_score=2*(precision*recall)/(precision+recall)
print('F1-score ',F1_score)
```

```
precision  0.1870748299319728
recall  0.55
F1-score  0.2791878172588833
```

compute accuracy score

In [17]:

```
accuracy_score=(TP+TN)/(TP+FP+FN+TN)
print('accuracy score',accuracy_score)
```

```
accuracy score 0.9718811881188119
```

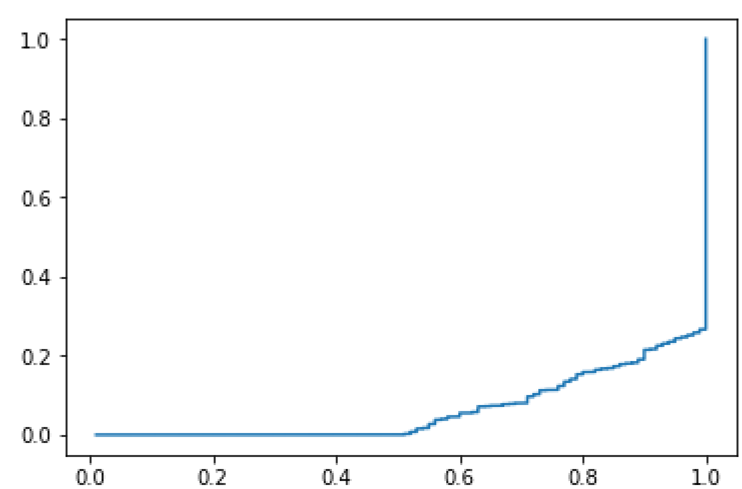
compute AUC_CURVE

In [18]:
```python
from tqdm import tqdm
unique=list(set(data2.proba))
unique.sort()
# print(unique)

TPR=[]
FPR=[]
for i in tqdm(unique):
    data2.loc[data2['proba'] < i, 'ypred'] = 0
    data2.loc[data2['proba'] > i, 'ypred'] = 1

    TP=int(data2[(data2.y == 1) & (data2.ypred == 1)].count()[0])
    TN=int(data2[(data2.y == 0) & (data2.ypred == 0)].count()[0])
    FP=int(data2[(data2.y == 0) & (data2.ypred == 1)].count()[0])
    FN=int(data2[(data2.y == 1) & (data2.ypred == 0)].count()[0])

    tpr=(TP/(TP+FN))
    TPR.append(tpr)
    fpr=(FP/(FP+TN))
    FPR.append(fpr)

```

```
100%|██████████████████████████████| 10100/10100 [03:37<00:00, 46.49it/s]
```

In [19]:
```python
x=sorted(TPR)
y=sorted(FPR)
from matplotlib import pyplot as plt
a=plt.plot(x,y)
print(a)
```

```
[<matplotlib.lines.Line2D object at 0x0000000009E8D358>]
```



In [20]:
```python
AUC_CURVE= np.trapz(x,y)
AUC_CURVE
```

Out[20]:
```
0.9376570000000001
```

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 [60]:
```
# write your code
#Load 5_c.csv into a pandas dataFrame.
data3=pd.read_csv("C:/Users/aaa/Desktop/saraswati aaic/5_c.csv")
data3.head(10)
```

Out[60]:

	y	prob
0	0	0.458521
1	0	0.505037
2	0	0.418652
3	0	0.412057
4	0	0.375579
5	0	0.595387
6	0	0.370288
7	0	0.299273
8	0	0.297000
9	0	0.266479

In [22]:
```
f = lambda x: 0 if x < 0.5 else 1
data3['ypred'] = data3['prob'].map(f)
data3.head(10)
```

Out[22]:

	y	prob	ypred
0	0	0.458521	0
1	0	0.505037	1
2	0	0.418652	0
3	0	0.412057	0
4	0	0.375579	0
5	0	0.595387	1
6	0	0.370288	0
7	0	0.299273	0
8	0	0.297000	0
9	0	0.266479	0

Compute the best threshold

In [61]:

```
from tqdm import tqdm
unique=list(data3.prob)
unique.sort()
s={}
for i in tqdm(unique):
    data3.loc[data3['prob'] < i, 'ypred'] = 0
    data3.loc[data3['prob'] > i, 'ypred'] = 1
#     TP=int(data3[(data3.y == 1) & (data3.ypred == 1)].count()[0])
#     TN=int(data3[(data3.y == 0) & (data3.ypred == 0)].count()[0])
    FP=int(data3[(data3.y == 0) & (data3.ypred == 1)].count()[0])
    FN=int(data3[(data3.y == 1) & (data3.ypred == 0)].count()[0])

    A=(500*FN)+(100*FP)
    s[i]=A


```

```
100%|██████████████████████████████| 2852/2852 [00:51<00:00, 55.80it/s]
```

In [97]:

```
val = min(s.values())
for threshold in s:
    if s[threshold]==val:
        print("the minimum value {} and the threshold {}".format(val,threshol
```

```
the minimum value 141000 and the threshold 0.2300390278970873
```

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 [110]:
```
#Load 5_c.csv into a pandas dataFrame.
data4=pd.read_csv("C:/Users/aaa/Desktop/saraswati aaic/5_d.csv")
data4.head(10)
```

Out[110]:

	y	pred
0	101.0	100.0
1	120.0	100.0
2	131.0	113.0
3	164.0	125.0
4	154.0	152.0
5	133.0	153.0
6	148.0	139.0
7	172.0	145.0
8	153.0	162.0
9	162.0	154.0

Compute Mean Square Error

In [111]:
```
lst1=list(data4.y)
lst2=list(data4.pred)
MSE=np.square(np.subtract(lst1,lst2)).mean()
MSE
```

Out[111]: 177.16569974554707

Compute mean absolute percentage error

In [133]:
```
for i in range(0,len(data4)):
    data4['pred'].loc[i]=abs((data4.y[i])-(data4.pred[i]))
errors=data4['pred'].sum()
actual_y=data4['y'].sum()
MAPE=(errors/actual_y)*100
print(MAPE)
```

12.663242720450786

Compute R^2 error

In [125]:
```
matrix = np.corrcoef(lst1, lst2)
xyz = matrix[0,1]
r_squared = xyz**2
print(r_squared)
```

0.9563600409880488