Compute performance metrics for the given Y and Y_score without sklearn

In [88]:
```
import numpy as np
import pandas as pd
from tqdm import tqdm
# other than these two you should not import any other packages
```

numpy.trapz(tpr_array, fpr_array)

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

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

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/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 [89]:
```
data_a = pd.read_csv("5_a.csv")
print (data_a.head())
print (data_a['y'].value_counts())
```

```
     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
1.0    10000
0.0      100
Name: y, dtype: int64
```

In [90]:
```
data_a['y_hat'] = list(map(lambda x : 0 if x < 0.5 else 1,data_a.proba))
data_a.sort_values(by = 'proba',ascending =
False,inplace=True,ignore_index=True)
print (data_a.head())
print (data_a.tail())
```

```
     y     proba  y_hat
0  1.0  0.899965      1
1  1.0  0.899828      1
2  1.0  0.899825      1
3  1.0  0.899812      1
4  1.0  0.899768      1
         y     proba  y_hat
10095  1.0  0.500081      1
10096  1.0  0.500058      1
10097  1.0  0.500058      1
10098  1.0  0.500047      1
10099  1.0  0.500019      1
```

In [91]:
```
uniqueProb = data_a['proba'].sort_values(ascending = False).unique()
Total = len(data_a)
P = len(data_a[data_a['y'] == 1])
N = len(data_a[data_a['y'] == 0])
TP = len(data_a[(data_a['y'] == 1) & (data_a['y_hat'] == 1)])
FP = len(data_a[(data_a['y'] == 0) & (data_a['y_hat'] == 1)])
TN = len(data_a[(data_a['y'] == 0) & (data_a['y_hat'] == 0)])
FN = len(data_a[(data_a['y'] == 1) & (data_a['y_hat'] == 0)])
cm_ele = [TN,FN,FP,TP]
CM = np.array(cm_ele).reshape(2,2)
Pr = TP/(TP+FP)
Re = TP/P
f1 = (2*Pr*Re)/(Pr+Re)

print ('Confusion Matrix :\n', CM)
print ('F1 Score :', f1)
```

```
Confusion Matrix :
 [[    0     0]
 [  100 10000]]
F1 Score : 0.9950248756218906
```

In [92]:
```
tpr_array = []
fpr_array = []
for i in tqdm(range(len(uniqueProb))):
    data_a['y_hat'] = list(map(lambda x : 1 if x >= uniqueProb[i]  else
0,data_a.proba))
    TP = len(data_a[(data_a['y'] == 1) & (data_a['y_hat'] == 1)])
```

```
    FP = len(data_a[(data_a['y'] == 0) & (data_a['y_hat'] == 1)])
    TPR = (TP/P)
    FPR = (FP/N)
    tpr_array.append(TPR)
    fpr_array.append(FPR)
```

```
100%|████████████████████████████████████████| 10100/
10100 [01:31<00:00, 109.81it/s]
```

In [93]:

```
# import matplotlib.pyplot as plt


# plt.plot(fpr_array,tpr_array)
# plt.show()
auc_score = np.trapz(tpr_array, fpr_array)
print ("AUC Score :",auc_score )

acc = (TP+TN)/Total
print ("\nAccuracy Score :",acc )
```

```
AUC Score : 0.48829900000000004

Accuracy Score : 0.9900990099009901
```

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/a/39678975/4084039
4. Compute Accuracy Score

In [99]:

```
data_b = pd.read_csv("5_b.csv")
```

```
print (data_b.head())
print (data_b['y'].value_counts())

data_b['y_hat'] = list(map(lambda x : 0 if x < 0.5 else 1,data_b.proba))
data_b.sort_values(by = 'proba',ascending =
False,inplace=True,ignore_index=True)
print (data_b.head())
print (data_b.tail())

uniqueProb = data_b['proba'].sort_values(ascending = False).unique()
Total = len(data_b)
P = len(data_b[data_b['y'] == 1])
N = len(data_b[data_b['y'] == 0])
TP = len(data_b[(data_b['y'] == 1) & (data_b['y_hat'] == 1)])
FP = len(data_b[(data_b['y'] == 0) & (data_b['y_hat'] == 1)])
TN = len(data_b[(data_b['y'] == 0) & (data_b['y_hat'] == 0)])
FN = len(data_b[(data_b['y'] == 1) & (data_b['y_hat'] == 0)])
cm_ele = [TN,FN,FP,TP]
CM = np.array(cm_ele).reshape(2,2)
Pr = TP/(TP+FP)
Re = TP/P
f1 = (2*Pr*Re)/(Pr+Re)

print ('\nConfusion Matrix :\n', CM)
print ('\nF1 Score :', f1)
```

```
     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
0.0    10000
1.0      100
Name: y, dtype: int64
     y     proba  y_hat
0  1.0  0.595294      1
1  1.0  0.594808      1
2  1.0  0.592198      1
3  1.0  0.590171      1
4  1.0  0.588718      1
         y     proba  y_hat
10095  0.0  0.100230      0
10096  0.0  0.100189      0
10097  0.0  0.100165      0
10098  0.0  0.100161      0
10099  0.0  0.100001      0

Confusion Matrix :
 [[9761   45]
```

```
[ 239   55]]

F1 Score : 0.2791878172588833
```

In [95]:

```
tpr_array_b = []
fpr_array_b = []
for i in tqdm(range(len(uniqueProb))):
    data_b['y_hat'] = list(map(lambda x : 1 if x >= uniqueProb[i]  else 
0,data_b.proba))
    TP = len(data_b[(data_b['y'] == 1) & (data_b['y_hat'] == 1)])
    FP = len(data_b[(data_b['y'] == 0) & (data_b['y_hat'] == 1)])
    TPR = (TP/P)
    FPR = (FP/N)
    tpr_array_b.append(TPR)
    fpr_array_b.append(FPR)
```

```
100%|██████████████████████████████████████████████████| 10100/
10100 [01:32<00:00, 109.37it/s]
```

In [96]:

```
auc_score = np.trapz(tpr_array_b, fpr_array_b)
print ("AUC Score :",auc_score )

acc = (TP+TN)/Total
print ("\nAccuracy Score :",acc )
```

```
AUC Score : 0.9377570000000001

Accuracy Score : 0.9763366336633663
```

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: $y^{pred}= \text{[0 if y_score < threshold else 1]}$

$ 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 [69]:

```
data_c = pd.read_csv("5_c.csv")
print (data_c.head())
print (data_c['y'].value_counts())

data_c.sort_values(by = 'prob',ascending =
```

```
False,inplace=True,ignore_index=True)
uniqueProb = data_c['prob'].sort_values(ascending = False).unique()

A_dict = {}
for i in tqdm(range(len(uniqueProb))):
    data_c['y_hat'] = list(map(lambda x : 1 if x >= uniqueProb[i]  else
0,data_c.prob))
    FP = len(data_c[(data_c['y'] == 0) & (data_c['y_hat'] == 1)])
    FN = len(data_c[(data_c['y'] == 1) & (data_c['y_hat'] == 0)])
    A = (500*FN + 100*FP)
    A_dict[uniqueProb[i]] = A



A_max = max(A_dict, key=A_dict.get)
print ('Best Threshold : ',A_max)
```

```
  0%|
| 0/2791 [00:00<?, ?it/s]
   y      prob
0  0  0.458521
1  0  0.505037
2  0  0.418652
3  0  0.412057
4  0  0.375579
0    1805
1    1047
Name: y, dtype: int64
100%|██████████████████████████████████████████████████████████████████████████████| 279
1/2791 [00:46<00:00, 60.34it/s]
Best Threshold :  0.9577467989277196
```

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 [42]:
```
data_d = pd.read_csv("5_d.csv")
```

```
print (data_d.head())
```

```
       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
```

In [43]:
```
data_d['error'] = data_d['y'] - data_d['pred']
data_d['abs_error'] = np.absolute(data_d['y'] - data_d['pred'])
data_d['error_sq'] = np.square(data_d['error'])
y_bar = np.mean(data_d['y'])
data_d['mean_diff_sq'] = np.square(data_d['y'] - y_bar)
print (data_d.head())
```

```
       y   pred  error  abs_error  error_sq  mean_diff_sq
0  101.0  100.0    1.0        1.0       1.0   1185.969885
1  120.0  100.0   20.0       20.0     400.0   2855.610598
2  131.0  113.0   18.0       18.0     324.0   4152.244694
3  164.0  125.0   39.0       39.0    1521.0   9494.146985
4  154.0  152.0    2.0        2.0       4.0   7645.388715
```

In [44]:
```
MSE = np.mean(data_d['error_sq'])
print ('Mean Square Error :', MSE)
```

```
Mean Square Error : 177.16569974554707
```

In [45]:
```
error_sum = np.sum(data_d['abs_error'])
y_actual_sum = np.sum(data_d['y'])
MAPE = (error_sum/y_actual_sum)*100
print ('Modified MAPE :', MAPE)
```

```
Modified MAPE : 12.91202994009687
```

In [46]:
```
SS_res = np.sum(data_d['error_sq'])
SS_total = np.sum(data_d['mean_diff_sq'])
R_squared = 1 - (SS_res/SS_total)
print ('R squared : ', R_squared)
```

```
R squared :  0.9563582786990937
```