Compute performance metrics for the given Y and Y_score without sklearn

In [92]:

```
import numpy as np
import pandas as pd
```

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)
   Note- Make sure that you arrange your probability scores in descending order while calculating AUC

4. Compute Accuracy Score

In [97]:

```
df_a=pd.read_csv('5_a.csv')
# class labels are stored in y_hat based on the condition ypred=[0 if y_score
df_a.loc[(df_a['proba']>=0.5),'y_hat']=1
df_a.loc[(df_a['proba']<0.5),'y_hat']=0
df_a.head()
```

Out[97]:

	y	proba	y_hat
0	1.0	0.637387	1.0
1	1.0	0.635165	1.0
2	1.0	0.766586	1.0
3	1.0	0.724564	1.0
4	1.0	0.889199	1.0

In [113]:

```
##function for confusion matrix
def confusion_matrix(df,pred):
    tp=tn=fp=fn=0
    for row in df.index:
        if df['y'][row] == df[pred][row]:
            #if the predicted and actual values matches and if the predicted
            if df[pred][row] == 1:
                tp+=1
            # else we increment true negative
            else: tn+=1
        else:
            # if the predicted and actual are different, then its a false pre
            if df[pred][row] == 1:
                fp+=1
            else: fn+=1
    return tp, tn, fp, fn

tp, tn, fp, fn =confusion_matrix(df_a,'y_hat')
print(f'True Positives:{tp}\n False Positives:{fp}\n True Negatives:{tn}\n Fa
```

```
True Positives:10000
 False Positives:100
 True Negatives:0
 False Negatives:0
```

In [114]:

```
#function to calculate precision
def precision_calc(tp, fp):
    return tp/(tp+fp)

#function to calculate recall
def recall_calc(tp, fn):
    return tp/(tp+fn)

#function to calculate fpr
def fpr_calc(fp,tn):
    return fp/(fp+tn)

#function to calculate f1_score
def f1_score_calc(precision, recall):
    f1_score = (2*precision*recall)/(precision+recall)
    return f1_score
#function to calculate accuracy
def accuracy(tp,fp,tn,fn):
    return (tp+tn)/(tp+fp+tn+fn)

print(f'precision:{precision_calc(tp,fp)}')
print(f'recall:{recall_calc(tp,fn)}')
print(f'f1_score:{f1_score_calc(precision, recall)}')
print(f'accuracy:{accuracy(tp,fp,tn,fn)}')
```

```
precision:0.9900990099009901
recall:1.0
f1_score:0.9950248756218906
accuracy:0.9900990099009901
```

In [129]:

```
from tqdm import tqdm
# function to implement roc_auc curve

def roc_auc(df):
    #first we sort the threshold values(probabilities)
    threshold = df.proba.sort_values(ascending= False)
    TPR = []
    FPR = []
    #for each value in threshold
    for t in tqdm(threshold):
        df.loc[(df['proba']>=t),'threshold']=1
        df.loc[(df['proba']<t),'threshold']=0
        tp, tn, fp, fn =confusion_matrix(df,'threshold')
        tpr = recall_calc(tp, fn)
        fpr = fpr_calc(fp,tn)
        TPR.append(tpr)
        FPR.append(fpr)
    AUC_score = np.trapz(np.array(TPR), np.array(FPR))
    return AUC_score
print(f'AUC score:{roc_auc(df_a)}')
```

```
100%|██████████████████████████████████████████
█████| 10100/10100 [22:47<00:00,  7.38it/s]

AUC score:0.48829900000000004
```

In [ ]:

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/q/53603376/4084039), https://stackoverflow.com/a/39678975/4084039 (https://stackoverflow.com/a/39678975/4084039)
   Note- Make sure that you arrange your probability scores in descending order while calculating AUC

4. Compute Accuracy Score

In [130]:

```
df_b=pd.read_csv('5_b.csv')
df_b.loc[(df_b['proba']>=0.5),'y_hat']=1
df_b.loc[(df_b['proba']<0.5),'y_hat']=0
df_b.head()
```

Out[130]:

	y	proba	y_hat
0	0.0	0.281035	0.0
1	0.0	0.465152	0.0
2	0.0	0.352793	0.0
3	0.0	0.157818	0.0
4	0.0	0.276648	0.0

In [131]:

```
tp, tn, fp, fn =confusion_matrix(df_b,'y_hat')
print(f'True Positives:{tp}\n False Positives:{fp}\n True Negatives:{tn}\n Fa
print(f'precision:{precision_calc(tp,fp)}')
print(f'recall:{recall_calc(tp,fn)}')
print(f'f1_score:{f1_score_calc(precision_calc(tp,fp), recall_calc(tp,fn))}')
print(f'accuracy:{accuracy(tp,fp,tn,fn)}')
print(f'AUC score:{roc_auc(df_b)}')
```

```
True Positives:55
 False Positives:239
 True Negatives:9761
 False Negatives:45

precision:0.1870748299319728
recall:0.55
f1_score:0.2791878172588833
accuracy:0.9718811881188119

100%|██████████████████████████████████████████████████████████████████████
██████| 10100/10100 [24:09<00:00,  6.97it/s]

AUC score:0.9377570000000001
```

In [ ]:

```
```

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 o
f positive points
   Note 2: use pandas or numpy to read the data from 5_c.csv
```

In [123]:

```
df_c=pd.read_csv('5_c.csv')
df_c.head()
```

Out[123]:

	y	prob
0	0	0.458521
1	0	0.505037
2	0	0.418652
3	0	0.412057
4	0	0.375579

In [148]:

```
def roc_auc(df):
    threshold = df.prob.sort_values(ascending= False)
    A = []
    for t in tqdm(threshold):
        df.loc[(df['prob']>=t),'threshold']=1
        df.loc[(df['prob']<t),'threshold']=0
        tp, tn, fp, fn =confusion_matrix(df,'threshold')
        a = (500 * fn) +(100*fp)
        A.append(a)
    # getting the index value of minimum(A)
    A_min_index = A.index(min(A))
    top_threshold = threshold.get(A_min_index)
    return top_threshold
print(f'best threshold:{roc_auc(df_c)}')
```

```
100%|██████████████████████████████████████████████████
█████████| 2852/2852 [02:26<00:00, 19.41it/s]

best threshold:0.2441047538776655
```

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 [132]:

```
df_d=pd.read_csv('5_d.csv')
df_d.head()
```

Out[132]:

	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 [152]:

```
# function for mean squared error
def mse(actual, pred):
    mean_sq_err = np.square(np.subtract(actual, pred)).mean()
    return mean_sq_err
# function for mean absolute percentage error
def mape(actual, pred):
    mape_val = np.mean(np.abs((actual-pred)/np.mean(actual)))*100
    return mape_val
# function for r square error
def rsquare(actual,pred):
    ss_square = np.square(np.subtract(actual, np.mean(actual))).sum()
    ss_res = np.square(np.subtract(actual,pred)).sum()
    rsquare = 1-(ss_res/ss_square)
    return rsquare
print(f'mean square error: {mse(df_d.y,df_d.pred)}')
print(f'mean absolute percentage error: {mape(df_d.y,df_d.pred)}')
print(f'r square error:{rsquare(df_d.y,df_d.pred)}')
```

```
mean square error: 177.16569974554707
mean absolute percentage error: 12.912029940096314
r square error:0.9563582786990937
```

In [ ]:

```

```

In [ ]:

```

```