Compute performance metrics for the given Y and Y_score without sklearn

In [2]:

```
import numpy as np
import pandas as pd
from tqdm import tqdm
# 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

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

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) Note: it should be numpy.trapz(tpr_array, fpr_array) not numpy.trapz(fpr_array, tpr_array)

4. Compute Accuracy Score

In [2]:

```
#Function to obtain derived class y
def yclass(row):
    if row['proba'] < 0.5:
        return 0.0
    else:
        return 1.0

#Function to obtain derived class y for Auc score
def auc_yclass(row,i):
    if row['proba'] < i:
        return 0.0
    else:
        return 1.0

#Function to return tpr_array,fpr_array for AUC calculation
def auc_score(auc_threshold):
    tpr_array=[]
    fpr_array=[]
    for i in tqdm(auc_threshold):
        y_d1_class=[]
        data['y_d1_class'] = data.apply(lambda row: auc_yclass(row,i), axis = 1)
        y_a_class_lst=data['y'].tolist()
        y_d1_class_lst=data['y_d1_class'].tolist()
        y_class1_merged=()
        y_class1_merged=tuple(zip( y_d1_class_lst,y_a_class_lst))
        TN,FN,FP,TP=tpr_fpr(y_class1_merged)
        #TPR,FPR,TNR,FNR
        TPR=TP/(TP+FN)
        TNR=TN/(TN+FP)
        FPR=FP/(TN+FP)
        FNR=FN/(FN+TP)
        tpr_array.append(TPR)
        fpr_array.append(FPR)
    return tpr_array,fpr_array

#function to return TN,FN,FP,TP values
def tpr_fpr(y_class_merged):
    TN,FP,FN,TP=0,0,0,0
    for i in range(len(y_class_merged)):
        if  y_class_merged[i][0]==0.0 and y_class_merged[i][1]==0.0:
            TN=TN+1
        elif y_class_merged[i][0]==0.0 and y_class_merged[i][1]==1.0:
            FN=FN+1
        elif y_class_merged[i][0]==1.0 and y_class_merged[i][1]==0.0:
            FP=FP+1
        else:
            TP=TP+1
    return TN,FN,FP,TP


#load 5_a.csv dataset
data=pd.read_csv('5_a.csv')

#derive class labels from probability score
data['y_d_class'] = data.apply(lambda row: yclass(row), axis = 1)

y_a_class_lst=[]
y_d_class_lst=[]
y_a_class_lst=data['y'].tolist()
```

```
y_d_class_lst=data['y_d_class'].tolist()
y_class_merged=tuple(zip( y_d_class_lst,y_a_class_lst))

#function call to calculate TN,FN,TP,FP
TN,FN,FP,TP=tpr_fpr(y_class_merged)

#confusion matrix
confusion_matrix_lst=[]
confusion_matrix_lst.append(TN)
confusion_matrix_lst.append(FN)
confusion_matrix_lst.append(FP)
confusion_matrix_lst.append(TP)
cm_array=np.asarray(confusion_matrix_lst)
confusion_matrix = cm_array.reshape(2, 2)
print("confusion_matrix :\n {}\n\n " .format(confusion_matrix))

#TPR,FPR,TNR,FNR
TPR=TP/(TP+FN)
TNR=TN/(TN+FP)
FPR=FP/(TN+FP)
FNR=FN/(FN+TP)

#F1_score calculation
precision=TP/(TP+FP)
recall=TPR
F1_score=2*((precision*recall)/(precision+recall))
print('Precision :\n {} \n Recall :\n {} \n F1_Score :\n{}\n'.format(precision,recall,F
1_score))

#AUC score calculation
auc_threshold=[]
auc_threshold=data['proba'].tolist()
auc_threshold.sort(reverse=True)
#AUC_score function call
tpr_array,fpr_array=auc_score(auc_threshold)

Auc_score=np.trapz(tpr_array, fpr_array)
print('AUC Score : \n {} \n\n' .format(Auc_score))

#Accuracy score
Accuracy_score=(int(confusion_matrix[0][0])+int(confusion_matrix[1][1])) / int(len(data
))
print('Accuracy_score :\n{}\n\n'.format(Accuracy_score))
```

```
confusion_matrix :
 [[    0     0]
 [  100 10000]]


Precision :
 0.9900990099009901
 Recall :
 1.0
 F1_Score :
0.9950248756218906


100%|██████████████████████████████████| 10100/10100 [1:00:36<00:00,  2.73
it/s]

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

4. Compute Accuracy Score

In [3]:

```
#Function to obtain derived class y
def yclass(row):
    if row['proba'] < 0.5:
        return 0.0
    else:
        return 1.0

#Function to obtain derived class y for Auc score
def auc_yclass(row,i):
    if row['proba'] < i:
        return 0.0
    else:
        return 1.0

#Function to return tpr_array,fpr_array for AUC calculation
def auc_score(auc_threshold):
    tpr_array=[]
    fpr_array=[]
    for i in tqdm(auc_threshold):
        y_d1_class=[]
        data['y_d1_class'] = data.apply(lambda row: auc_yclass(row,i), axis = 1)
        y_a_class_lst=data['y'].tolist()
        y_d1_class_lst=data['y_d1_class'].tolist()
        y_class1_merged=()
        y_class1_merged=tuple(zip( y_d1_class_lst,y_a_class_lst))
        TN,FN,FP,TP=tpr_fpr(y_class1_merged)
        #TPR,FPR,TNR,FNR
        TPR=TP/(TP+FN)
        TNR=TN/(TN+FP)
        FPR=FP/(TN+FP)
        FNR=FN/(FN+TP)
        tpr_array.append(TPR)
        fpr_array.append(FPR)
    return tpr_array,fpr_array

#function to return TN,FN,FP,TP values
def tpr_fpr(y_class_merged):
    TN,FP,FN,TP=0,0,0,0
    for i in range(len(y_class_merged)):
        if  y_class_merged[i][0]==0.0 and y_class_merged[i][1]==0.0:
            TN=TN+1
        elif y_class_merged[i][0]==0.0 and y_class_merged[i][1]==1.0:
            FN=FN+1
        elif y_class_merged[i][0]==1.0 and y_class_merged[i][1]==0.0:
            FP=FP+1
        else:
            TP=TP+1
    return TN,FN,FP,TP


#load 5_a.csv dataset
data=pd.read_csv('5_b.csv')

#derive class labels from probability score
data['y_d_class'] = data.apply(lambda row: yclass(row), axis = 1)

y_a_class_lst=[]
y_d_class_lst=[]
y_a_class_lst=data['y'].tolist()
```

```
y_d_class_lst=data['y_d_class'].tolist()
y_class_merged=tuple(zip( y_d_class_lst,y_a_class_lst))

#function call to calculate TN,FN,TP,FP
TN,FN,FP,TP=tpr_fpr(y_class_merged)

#confusion matrix
confusion_matrix_lst=[]
confusion_matrix_lst.append(TN)
confusion_matrix_lst.append(FN)
confusion_matrix_lst.append(FP)
confusion_matrix_lst.append(TP)
cm_array=np.asarray(confusion_matrix_lst)
confusion_matrix = cm_array.reshape(2, 2)
print("confusion_matrix :\n {}\n\n " .format(confusion_matrix))

#TPR,FPR,TNR,FNR
TPR=TP/(TP+FN)
TNR=TN/(TN+FP)
FPR=FP/(TN+FP)
FNR=FN/(FN+TP)

#F1_score calculation
precision=TP/(TP+FP)
recall=TPR
F1_score=2*((precision*recall)/(precision+recall))
print('Precision :\n {} \n Recall :\n {} \n F1_Score :\n{}\n'.format(precision,recall,F
1_score))

#AUC score calculation
auc_threshold=[]
auc_threshold=data['proba'].tolist()
auc_threshold.sort(reverse=True)
#AUC_score function call
tpr_array,fpr_array=auc_score(auc_threshold)

Auc_score=np.trapz(tpr_array, fpr_array)
print('AUC Score : \n {} \n\n' .format(Auc_score))

#Accuracy score
Accuracy_score=(int(confusion_matrix[0][0])+int(confusion_matrix[1][1])) / int(len(data
))
print('Accuracy_score :\n{}\n\n'.format(Accuracy_score))
```

```
confusion_matrix :
 [[9761   45]
 [ 239   55]]


Precision :
 0.1870748299319728
 Recall :
 0.55
 F1_Score :
0.2791878172588833


100%|██████████████████████████████████| 10100/10100 [1:00:54<00:00,  2.71
it/s]

AUC Score :
 0.9377570000000001


Accuracy_score :
0.9718811881188119
```

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 positi
ve points
   Note 2: use pandas or numpy to read the data from 5_c.csv
```

In [4]:

```
#Function to obtain derived class y for Auc score
def auc_yclass(row,i):
    if row['prob'] < i:
        return 0.0
    else:
        return 1.0

#Function to return tpr_array,fpr_array for AUC calculation
def auc_score(auc_threshold):
    tpr_array=[]
    fpr_array=[]
    metric_a_lst=[]
    for i in tqdm(auc_threshold):
        y_d1_class=[]
        data['y_d1_class'] = data.apply(lambda row: auc_yclass(row,i), axis = 1)
        y_a_class_lst=data['y'].tolist()
        y_d1_class_lst=data['y_d1_class'].tolist()
        y_class1_merged=()
        y_class1_merged=tuple(zip( y_d1_class_lst,y_a_class_lst))
        TN,FN,FP,TP=tpr_fpr(y_class1_merged)
        metric_a_lst.append(500*FN+100*FP)

        #TPR,FPR,TNR,FNR
        TPR=TP/(TP+FN)
        TNR=TN/(TN+FP)
        FPR=FP/(TN+FP)
        FNR=FN/(FN+TP)
        tpr_array.append(TPR)
        fpr_array.append(FPR)
    return tpr_array,fpr_array,metric_a_lst

#function to return TN,FN,FP,TP values
def tpr_fpr(y_class_merged):
    TN,FP,FN,TP=0,0,0,0
    for i in range(len(y_class_merged)):
        if  y_class_merged[i][0]==0.0 and y_class_merged[i][1]==0.0:
            TN=TN+1
        elif y_class_merged[i][0]==0.0 and y_class_merged[i][1]==1.0:
            FN=FN+1
        elif y_class_merged[i][0]==1.0 and y_class_merged[i][1]==0.0:
            FP=FP+1
        else:
            TP=TP+1
    return TN,FN,FP,TP


#load 5_a.csv dataset
data=pd.read_csv('5_c.csv')

#AUC score calculation
auc_threshold=[]
auc_threshold=data['prob'].tolist()
auc_threshold.sort(reverse=True)
#AUC_score function call
tpr_array,fpr_array,metric_a_lst=auc_score(auc_threshold)

#Best threshold prob value
metric_a_dict = dict(zip(auc_threshold, metric_a_lst))
```

```
metric_a_lst_asc=sorted(metric_a_dict,key=metric_a_dict.get)[:1]
for i in metric_a_lst_asc:
    print('Best threshold probabilty : {0}\t Metric_A value : {1}'.format(i,metric_a_di
ct[i]))
```

```
100%|██████████████████████████████████████| 2852/2852 [05:13<00:00,  7.55
it/s]

Best threshold probabilty : 0.2300390278970873   Metric_A value : 141000
```

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 [23]:

```
#load 5_a.csv dataset
data=pd.read_csv('5_d.csv')

#Mean square error,average of square of y-pred
y_lst=[]
y_pred_lst=[]
y_lst=data['y'].tolist()
y_pred_lst=data['pred'].tolist()

error_sum=0
y_sum=0
for i in range(len(y_lst)):
    error_sum= error_sum + ((y_lst[i]-y_pred_lst[i])**2)

mean_sq_err=error_sum/len(y_lst)
print('Mean Squared Error : \n {} \n '.format(mean_sq_err))

#MAPE,Mean absolute percentage error,sum of abs value of error(y-pred) divided by sum o
f values of y
error_sum=0
y_sum=0
for i in range(len(y_lst)):
    error_sum= error_sum +abs(y_pred_lst[i]-y_lst[i])
    y_sum=y_sum+y_lst[i]
MAPE=(error_sum/y_sum)*100
print('MAPE : \n {} \n '.format(MAPE))

#R squared,1-(ss_res/ss_total)
ss_total=0
ss_res=0
for i in range(len(y_lst)):
    ss_total=ss_total+((y_lst[i]-y_avg)**2)
    ss_res=ss_res+((y_lst[i]-y_pred_lst[i])**2)

R_2=1-(ss_res/ss_total)
print('R^2 : \n {} '.format(R_2))
```

```
Mean Squared Error : 
 177.16569974554707 
 
MAPE : 
 12.91202994009687 
 
R^2 : 
 0.9563582786990964 
```

In [ ]:

```
```