Compute performance metrics for the given Y and Y_score without sklearn

Importing the libraries

In [1]:

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

Reading the file

In [2]:

```
df_a=pd.read_csv("5_a.csv")
df_b=pd.read_csv("5_b.csv")
df_c=pd.read_csv("5_c.csv")
df_d=pd.read_csv("5_d.csv")
```

Converting probality value to output class label

In [3]:

```
df_a['y_pred']=df_a['proba'].apply(lambda x: 1 if x>=.5 else 0)
df_b['y_pred']=df_b['proba'].apply(lambda x: 1 if x>=.5 else 0)
```

confusion matrix

In [4]:

```
def confusion_matrix(data):
    count_tn=len(data[(data['y']==0) & (data['y_pred']==0)])
    count_tp=len(data[(data['y']==1) & (data['y_pred']==1)]) #calculati
ng tn,tp,fn,fp
```

```
    count_fn=len(data[(data['y']==1) & (data['y_pred']==0)])
    count_fp=len(data[(data['y']==0) & (data['y_pred']==1)])
    return count_fn,count_fp,count_tn,count_tp
```

f1 score

In [5]:
```
def f1_score(data):
    fn,fp,tn,tp=confusion_matrix(data)
    precision=tp/(tp+fp)                      # calculating precision and
 recall
    recall=tp/(tp+fn)
    f1=2*((precision*recall)/(precision+recall))
    return f1
```

accuracy value

In [6]:
```
def accuracy(data):
    fn,fp,tn,tp=confusion_matrix(data)
    acc=((tp+tn)/(tp+fp+fn+tn))
    return acc
```

auc value

In [18]:
```
def auc_score(data):
    tpr_array=[]
    fpr_array=[]
    sort= data.sort_values("proba",ascending=False) # sort sart based o
n probability scores
    for i in range(0,len(sort)):
        sort['y_pred']=np.where(sort['proba']>=sort.iloc[i]['proba'],1,
0) # predicting the y based on each threshold
        FN,FP,TN,TP=confusion_matrix(sort)    # for each threshold calc
ulating confusion matrix
        fpr_rate=FP/(TN+FP)
        tpr_rate=TP/(TP+FN)
        tpr_array.append(tpr_rate)
        fpr_array.append(fpr_rate)
    c=np.trapz(tpr_array, fpr_array)
    return c
```

A.

Compute performance metrics for the given data 5_a.csv

1.confusion matrix of data 5_a.csv

In [8]:

```
FN,FP,TN,TP=confusion_matrix(df_a)
print("FALSE NEGATIVE :",FN)
print("FALSE POSITIVE :",FP)
print("TRUE NEGATIVE :",TN)
print("TRUE POSITIVE :",TP )
```

```
FALSE NEGATIVE : 0
FALSE POSITIVE : 100
TRUE NEGATIVE : 0
TRUE POSITIVE : 10000
```

2. f1 score of data 5_a.csv

In [9]:

```
f1=f1_score(df_a)
print("F1 SCORE :",f1)
```

```
F1 SCORE : 0.9950248756218906
```

3. accuracy value of data 5_a.csv

In [10]:

```
acc=accuracy(df_a)
print('ACCURACY VALUE :',acc)
```

```
ACCURACY VALUE : 0.9900990099009901
```

4. auc value of data 5_a.csv

In [21]:

```
auc=auc_score(df_a)
print('AUC VALUE :',auc)
```

```
AUC VALUE : 0.48829900000000004
```

B.

Compute performance metrics for the given data 5_b.csv

1.confusion matrix of data 5_b.csv

In [22]:

```
FN,FP,TN,TP=confusion_matrix(df_b)
print("FALSE NEGATIVE :",FN)
print("FALSE POSITIVE :",FP)
print("TRUE NEGATIVE :",TN)
print("TRUE POSITIVE :",TP )
```

```
FALSE NEGATIVE : 45
FALSE POSITIVE : 239
TRUE NEGATIVE : 9761
TRUE POSITIVE : 55
```

2. f1 score of data 5_b.csv

In [23]:

```
f1=f1_score(df_b)
print("F1 SCORE :",f1)
```

```
F1 SCORE : 0.2791878172588833
```

3. accuracy value of data 5_b.csv

In [24]:
```
acc=accuracy(df_b)
print('ACCURACY VALUE :',acc)
```

```
ACCURACY VALUE : 0.9718811881188119
```

4. auc value of data 5_b.csv

In [30]:
```
auc=auc_score(df_b)
print("AUC VALUE :",auc)
```

```
AUC VALUE : 0.9377570000000001
```

C.

1. Compute the best threshold of probability which gives lowest values of metric A for the given data 5_c.csv

In [26]:
```
def best_threshold(data):
    check=0
    thresh=[]
    A=[]
    sorted= data.sort_values("prob",ascending=False) # sorting data based on probability
    for i in range(0,len(sorted)):
        if check==(sorted.iloc[i]['prob']): # checking unique probability
            continue
        check=sorted.iloc[i]['prob']
        thresh.append(check)
        sorted['y_pred']=np.where(sorted['prob']>=sorted.iloc[i]['prob'],1,0)
        FN,FP,TN,TP=confusion_matrix(sorted) # calculating confusion matrix for each threshold
        value=500*FN+100*FP
        A.append(value)  # calculating the metric A
    index=A.index(min(A)) # finding the index of A with minimium value
    return thresh[index]
```

In [27]:

```
best=best_threshold(df_c)
print('BEST THRESHOLD VALUE :',best)
```

```
BEST THRESHOLD VALUE : 0.2300390278970873
```

D.

Compute performance metrics(for regression) for the given data 5_d.csv

In [111]:

```
def regression_metrics(data):
    n=len(data)
    data['ei']= data.apply(lambda x: abs(x['y'] - x['pred']), axis=1) #
 calculating absolute differnce between Y and y^
    data['mse']= data['ei'].apply(lambda x: x*x) # calculating the squa
res of ei
    total=data['mse'].sum()
    mse=total/n
    mape=(data['ei'].sum())/(data['y'].sum())
    mean=(data['y'].sum())/n # calculating simple mean of yi's
    ssres=data['mse'].sum()
    data['sstotal']= data.apply(lambda x: (x['y'] - mean), axis=1)
    data['sstotal']= data['sstotal'].apply(lambda x: x*x)
    sstotal=data['sstotal'].sum()
    rsquared=1-(ssres/sstotal)
    return mse,mape,rsquared
```

In [115]:

```
mse,mape,rsquared=regression_metrics(df_d)
print('MEAN SQUARED ERROR :',mse)
print('MEAN ABSOLUTE PERCENTAGE ERROR :',mape*100)
print('R SQUARED :',rsquared)
```

```
MEAN SQUARED ERROR : 177.16569974554707
MEAN ABSOLUTE PERCENTAGE ERROR : 12.91202994009687
R SQUARED : 0.9563582786990937
```