Compute performance metrics for the given Y and Y_score without sklearn

In [1]:

```
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 [2]:

```
# write your code here
data = pd.read_csv("5_a.csv")  # Read csv file
data.shape
```

Out[2]:

```
(10100, 2)
```

In [3]:

```
print(data.columns)  # checking the columns name
```

```
Index(['y', 'proba'], dtype='object')
```

In [4]:

```
data["y"].value_counts()  # counting the number of ones and zeros
```

Out[4]:

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

In [5]:

```
# this function will the convert probability score(proba in data) into 1 & 0 according to t

def predict(df,y,threshold):
    y_pred = []
    for label in df[y]:   # iterating through the proba
        if label<threshold:   # checking the threshhold is greater than the probability sco
            y_pred.append(0)   # if yes then append 0 in y_pred
        else:
            y_pred.append(1)    # else append 1 in y_pred
    return y_pred
```

In [6]:

```
data['y_pred'] = predict(data,'proba',0.5)
data['y_pred'].head(5)
```

Out[6]:

```
0    1
1    1
2    1
3    1
4    1
Name: y_pred, dtype: int64
```

In [7]:

```
# this fucntion will return the confusion matrix

def con_matrix(df,y,y_pred):
    tp=0
    tn=0
    fn=0
    fp=0
    for val1,val2 in enumerate(df[y]):
        if(df.y_pred[val1]==1) and df.y[val1]==1:
            tp=tp+1
        if(df.y_pred[val1]==0) and df.y[val1]==0:
            tn=tn+1
        if(df.y_pred[val1]==0) and df.y[val1]==1:
            fn=fn+1
        if(df.y_pred[val1]==1) and df.y[val1]==0:
            fp=fp+1
    return {'tn':tn,'tp':tp,'fn':fn,'fp':fp}
```

In [8]:

```
con_matrix(data,'y','y_pred')
```

Out[8]:

```
{'tn': 0, 'tp': 10000, 'fn': 0, 'fp': 100}
```

In [9]:

```
# this function will return the F1 score

def f1_score(df,confusion_matrix):
    x=df.y.value_counts()
    P=x[1]

    precision=confusion_matrix['tp']/(confusion_matrix['tp']+confusion_matrix['fp'])
    recall=confusion_matrix['tp']/P

    F1=2*precision*recall/(precision+recall)
    print('the F1 score is: ',F1)
```

In [10]:

```
Z = con_matrix(data,'y','y_pred')
f1_score(data,Z)
```

```
the F1 score is:  0.9950248756218906
```

In [11]:

```
# Accuracy
Acc=(Z['tp']+Z['tn'])/data.shape[0]
print('the accuracy is: ',Acc)
```

```
the accuracy is:  0.9900990099009901
```

In [12]:

```
from tqdm import tqdm
def auc(df):
    s = df['y'].value_counts()
    P = s[1]
    N = s[0]
    tpr = []
    fpr = []
    for elem in tqdm(df['proba']):
        df['y_pred']=predict(df,'proba',elem)
        confusion_matrix=con_matrix(df,'y','y_pred')
        tpr.append(confusion_matrix['tp']/P)
        fpr.append(confusion_matrix['fp']/N)
        df.drop(columns=['y_pred'])
    return np.trapz(tpr,fpr)
```

In [12]:

```
data=data.sort_values(by='proba',ascending=False)
data.drop(columns=['y_pred'])
AUC_score=auc(data)
print ('the AUC Score is :',AUC_score)
```

```
100%|████████████████████████████████████████████████████████████████
███| 10100/10100 [2:17:38<00:00,  1.24it/s]

the AUC Score is : 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)

4. Compute Accuracy Score

In [13]:

```
# write your code
data_B = pd.read_csv('5_b.csv')
data_B.shape
```

Out[13]:

```
(10100, 2)
```

In [14]:

```
print(data_B.columns)  # checking the columns name
```

```
Index(['y', 'proba'], dtype='object')
```

In [15]:

```
data_B["y"].value_counts()  # counting the number of ones and zeros
```

Out[15]:

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

In [16]:

```
data_B['y_pred'] = predict(data_B,'proba',0.5)
data_B['y_pred'].head(5)
```

Out[16]:

```
0    0
1    0
2    0
3    0
4    0
Name: y_pred, dtype: int64
```

In [17]:

```
con_matrix(data_B,'y','y_pred')
```

Out[17]:

```
{'tn': 9761, 'tp': 55, 'fn': 45, 'fp': 239}
```

In [18]:

```
Z_B = con_matrix(data_B,'y','y_pred')
f1_score(data_B,Z_B)
```

```
the F1 score is:  0.2791878172588833
```

In [19]:

```
Acc_B=(Z_B['tp']+Z_B['tn'])/data_B.shape[0]
print('the Accuracy is :',Acc_B)
```

```
the Accuracy is : 0.9718811881188119
```

In [20]:

```
data_B=data_B.sort_values(by='proba',ascending=False)
data_B.drop(columns=['y_pred'])
AUC_score_B=auc(data_B)
print('the AUC Score is: ',AUC_score_B)
```

```
100%|██████████████████████████████████████████████████████████████████████
███| 10100/10100 [2:11:05<00:00,  1.29it/s]

the AUC Score is:  0.9377570000000001
```

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 [20]:

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

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

In [21]:

```
def low_metric(data):
    metric={}    # declaring the empty dictionary for storing value
    for elem in tqdm(data['prob']):
        data['y_pred']= predict(data,'prob',elem)  # convert probability score(proba in dat
        con_mat = con_matrix(data,'y','y_pred')   # computing the confusion marix
        metric_val=(500*con_mat['fn'])+(100*con_mat['fp'])  # implimenting the given formul
        metric[elem]=metric_val
        data.drop(columns=['y_pred'])
    return(metric)
```

In [23]:

```
result=low_metric(data)
```

```
100%|██████████████████████████████████████████████████
█████| 2852/2852 [14:40<00:00,  4.31it/s]
```

In [25]:

```
temp = min(result.values())
res = [key for key in result if result[key] == temp]
print('the key:value pair for min value of the specified metric is-',res,temp)
```

```
the key:value pair for min value of the specified metric is- [0.230039027897
0873] 141000
```

In [ ]:

```

```

> **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 [27]:

```
data_d = pd.read_csv("5_d.csv")
data_d.head()
```

Out[27]:

	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 [29]:

```
def ss_res(df,col):
    val=0
    for index,value in enumerate(df[col]):
        val=val+(value*value)
    return val

def ss_tot(df,col):
    val=0
    mean_val=data_d['y'].mean()
    for index,value in enumerate(df[col]):
        val=val+ (value-mean_val)*(value-mean_val)
    return val
```

In [30]:

```
def error(df,col1,col2):
    val=[]
    for index, (value1, value2) in enumerate(zip(df[col1], df[col2])):
        val.append(value1-value2)
    return val

def absolute_error(df,col):
    val=[]
    for index,value in enumerate(df[col]):
        val.append(abs(value))
    return val


data_d['error']=error(data_d,'y','pred')
data_d['abs_error']=absolute_error(data_d,'error')
```

In [33]:

```
# computing MAPE value
val=sum(data_d['abs_error'])/sum(data_d['y'])
print(val)
```

```
0.1291202994009687
```

In [34]:

```
# computing r^2 value
SS_RES=ss_res(data_d,'error')
SS_TOT=ss_tot(data_d,'y')
R_square= 1- (SS_RES/SS_TOT)
print('the Co-efficient of determination value is: ',R_square)
```

```
the Co-efficient of determination value is:  0.9563582786990964
```

In [37]:

```
# computing mean square error
MSE = ss_res(data_d,'error')/len(data_d['error'])
print(MSE)
```

```
177.16569974554707
```

In [ ]: