Compute performance metrics for the given Y and Y_score without sklearn

In [4]:
```
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

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

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/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 [5]:
```
# write your code here
import pandas as pd
data= pd.read_csv("5_a.csv")
data['predicted_val'] =(data.proba>=0.5).astype('float')
data.head()
#k = data.proba.tolist()
#print(k)
#print(k.sort(reverse=True))
```

Out[5]:

	y	proba	predicted_val
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

confusion matrix

In [6]:
```
#confusion matrix
def tp(y,predicted_val):
    return sum((y == 1)&(predicted_val == 1))
def fn(y,predicted_val):
    return sum((y == 1)&(predicted_val == 0))
def fp(y,predicted_val):
    return sum((y == 0)&(predicted_val == 1))
def tn(y,predicted_val):
    return sum((y == 0)&(predicted_val == 0))
```

In [7]:
```
print('TP:',tp(data.y.values,data.predicted_val.values))
print('FN:',fn(data.y.values,data.predicted_val.values))
print('FP:',fp(data.y.values,data.predicted_val.values))
print('TN:',tn(data.y.values,data.predicted_val.values))
```

```
TP: 10000
FN: 0
FP: 100
TN: 0
```

In [8]:
```
def confusion_matrix(y,predicted_val):
    TP = tp(y,predicted_val)
    FN = fn(y,predicted_val)
    FP = fp(y,predicted_val)
    TN= tn(y,predicted_val)
    return TP,FN,FP,TN
def The_confusion_matrix(y,predicted_val):
    TP,FN,FP,TN =  confusion_matrix(y,predicted_val)
    return np.array([[TN,FP],[FN,TP]])
The_confusion_matrix(data.y.values,data.predicted_val.values)
```

Out[8]:
```
array([[    0,   100],
       [    0, 10000]])
```

In [10]:
```
#comparing values by using sklearn
from sklearn.metrics import confusion_matrix
confusion_matrix(data.y.values,data.predicted_val.values)
```

Out[10]:
```
array([[    0,   100],
       [    0, 10000]], dtype=int64)
```

f1 score and accuracy score

In [11]:
```
TP = tp(data.y.values,data.predicted_val.values)
FN = fn(data.y.values,data.predicted_val.values)
FP = fp(data.y.values,data.predicted_val.values)
```

```
TN = tn(data.y.values,data.predicted_val.values)
acr = ((TP+TN)/(TP+TN+FP+FN))
pr =(TP/(TP+FP))
#pr =(10000/(10000+100))
re = TP/(FN+TP)
print("precision:",+pr)
print("recall:",+re)
f1 = ((2*pr*re)/(pr+re))
print("F1score",+f1)
print("accuracy",+acr)
```

```
precision: 0.9900990099009901
recall: 1.0
F1score 0.9950248756218906
accuracy 0.9900990099009901
```

In [12]:

```
#comparing values with sklearn
from sklearn.metrics import accuracy_score,f1_score
print("accuracy score",accuracy_score(data.y.values,data.predicted_val.
values))
print("F1 score",f1_score(data.y.values,data.predicted_val.values))
```

```
accuracy score 0.9900990099009901
F1 score 0.9950248756218906
```

AUC score

In [2]:

```
# write your code here
import pandas as pd
data= pd.read_csv("5_b.csv")
data['predicted_val'] =(data.proba>=0.5).astype('float')
data.head()
```

Out[2]:

	y	proba	predicted_val
0	0.0	0.281035	0.0

	y	proba	predicted_val
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 [12]:
```
from tqdm import tqdm
data=pd.read_csv('5_a.csv')
negative_points= data['y'].value_counts()[0]
positive_points = data['y'].value_counts()[1]
tpr_array = []
fpr_array = []
sorted_data = data.sort_values('proba',ascending = False)
for threshold in tqdm(sorted_data['proba']):
    y_pred = []
    for k in sorted_data['proba']:

        if (k<threshold):
            y_pred.append(0.0)
        else:
            y_pred.append(1.0)
    sorted_data['y_pred'] = y_pred
    for k in data:
        TP = ((sorted_data['y']==1.0) & (sorted_data['y_pred'] == 1.0))
.sum()
        FP = ((sorted_data['y']==0.0) & (sorted_data['y_pred'] == 1.0))
.sum()
    tpr_array.append( TP/positive_points)
    fpr_array.append(FP/negative_points)
auc = np.trapz(tpr_array,fpr_array)
print("The auc score without using sklearn:",auc)
```

```
100%|██████████| 10100/10100 [02:54<00:00, 57.96it/s]
```

```
The auc score without using sklearn: 0.48829900000000004
```

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 ypred=[0 if y_score < 0.5 else 1]ypred=[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 [14]:
```
# write your code here
import pandas as pd
data= pd.read_csv("5_b.csv")
data['predicted_val'] =(data.proba>=0.5).astype('float')
data.head()
#k = data.proba.tolist()
#print(k)
#print(k.sort(reverse=True))
```

Out[14]:

	y	proba	predicted_val
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 [15]:
```
#confusion matrix
def tp(y,predicted_val):
    return sum((y == 1)&(predicted_val == 1))
def fn(y,predicted_val):
    return sum((y == 1)&(predicted_val == 0))
def fp(y,predicted_val):
    return sum((y == 0)&(predicted_val == 1))
def tn(y,predicted_val):
    return sum((y == 0)&(predicted_val == 0))
```

In [16]:
```
print('TP:',tp(data.y.values,data.predicted_val.values))
print('FN:',fn(data.y.values,data.predicted_val.values))
print('FP:',fp(data.y.values,data.predicted_val.values))
print('TN:',tn(data.y.values,data.predicted_val.values))
```

```
TP: 55
FN: 45
FP: 239
TN: 9761
```

In [17]:
```
def confusion_matrix(y,predicted_val):
    TP = tp(y,predicted_val)
    FN = fn(y,predicted_val)
    FP = fp(y,predicted_val)
    TN= tn(y,predicted_val)
    return TP,FN,FP,TN
def The_confusion_matrix(y,predicted_val):
    TP,FN,FP,TN =  confusion_matrix(y,predicted_val)
    return np.array([[TN,FP],[FN,TP]])
The_confusion_matrix(data.y.values,data.predicted_val.values)
```

Out[17]:
```
array([[9761,  239],
       [  45,   55]])
```

In [18]:
```
#comparing values by using sklearn
from sklearn.metrics import confusion_matrix
confusion_matrix(data.y.values,data.predicted_val.values)
```

Out[18]:
```
array([[9761,  239],
       [  45,   55]], dtype=int64)
```

f1 score and accuracy

In [19]:
```
FN = fn(data.y.values,data.predicted_val.values)
FP = fp(data.y.values,data.predicted_val.values)
TN = tn(data.y.values,data.predicted_val.values)
TP = tp(data.y.values,data.predicted_val.values)
acr = ((TP+TN)/(TP+TN+FP+FN))
pr =(TP/(TP+FP))
#pr =(10000/(10000+100))
re = TP/(FN+TP)
print("precision:",+pr)
print("recall:",+re)
f1 = ((2*pr*re)/(pr+re))
print("F1score:",+f1)
print("accuracy:",+acr)
```

```
precision: 0.1870748299319728
recall: 0.55
F1score: 0.2791878172588833
accuracy: 0.9718811881188119
```

In [20]:
```
#comparing values with sklearn
from sklearn.metrics import accuracy_score,f1_score
print("accuracy score",accuracy_score(data.y.values,data.predicted_val.
values))
print("F1 score",f1_score(data.y.values,data.predicted_val.values))
```

```
accuracy score 0.9718811881188119
F1 score 0.2791878172588833
```

AUC score

In [21]:
```
from tqdm import tqdm
data=pd.read_csv('5_b.csv')
negative_points= data['y'].value_counts()[0]
positive_points = data['y'].value_counts()[1]
tpr_array = []
fpr_array = []
sorted_data = data.sort_values('proba',ascending = False)
for threshold in tqdm(sorted_data['proba']):
    y_pred = []
    for k in sorted_data['proba']:

        if (k<threshold):
            y_pred.append(0.0)
        else:
            y_pred.append(1.0)
    sorted_data['y_pred'] = y_pred
    for k in data:
        TP = ((sorted_data['y']==1.0) & (sorted_data['y_pred'] == 1.0))
.sum()
        FP = ((sorted_data['y']==0.0) & (sorted_data['y_pred'] == 1.0))
.sum()
    tpr_array.append( TP/positive_points)
    fpr_array.append(FP/negative_points)
auc = np.trapz(tpr_array,fpr_array)
print("The auc score without using sklearn:",auc)
```

```
100%|██████████| 10100/10100 [02:40<00:00, 63.05it/s]
```

```
The auc score without using sklearn: 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 [2]:
```
# write your code here
import pandas as pd
data= pd.read_csv("5_c.csv")
#len(data)
data.head()
```

Out[2]:

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

In [4]:
```
import numpy as np
unique_prob = list(sorted(set(data['prob'])))
A=100000000
for i in unique_prob:
    data['y_thrshld'] = np.where(data['prob']>=i,1,0)
    FP=len(data.loc[(data['y']==0) & (data['y_thrshld']==1)])
    FN=len(data.loc[(data['y']==1) & (data['y_thrshld']==0)])
    matric_value = (500*FN)+(100*FP)
    if(matric_value<A):
        A=matric_value
        threshold_value = i
    del data['y_thrshld']
```

```
print("Matric value: ",A)
print("Threshold value: ",threshold_value)
```

```
Matric value:  141000
Threshold value:  0.2300390278970873
```

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 [26]:
```
import pandas as pd
data= pd.read_csv("5_d.csv")
data
len(data)
```

Out[26]: 157200

Mean square error

In [27]:
```
import numpy as np
y = data.y.values
```

```
pred = data.pred.values
MSE = np.square(np.subtract(y,pred)).mean()
print("mean square error",MSE)
```

```
mean square error 177.16569974554707
```

In [28]:
```
#comparing with sklearn
from sklearn.metrics import mean_squared_error
from math import sqrt
y = data.y.values
pred = data.pred.values
p = mean_squared_error(y,pred)
p
```

Out[28]:
```
177.16569974554707
```

R squared error

In [29]:
```
def r2_val(y,pred):
    a = sum((y-pred)**2)
    b = (len(y)-1)*np.var(y)
    r2_score = 1-(a/b)
    return r2_score
print("r2_score",r2_val(y,pred))
```

```
r2_score 0.9563580010782354
```

In [30]:
```
#comparing with sklearn
from sklearn.metrics import r2_score
y = data.y.values
pred = data.pred.values
r2_score(y,pred)
```

Out[30]:
```
0.9563582786990937
```

MAPE

In [31]:
```
def MAPE(y,pred):
    y,pred = np.array(y),np.array(pred)
    l = np.mean(y)
    return np.mean(np.abs((y-pred)/l))*100
k = MAPE(y,pred)
k
```

Out[31]: 12.912029940096867