Compute performance metrics for the given Y and Y_score without sklearn

In [1]:
```
import os
os.getcwd()
```

Out[1]:
```
'C:\\Users\\sai\\applied AI Assignments\\5thassignment'
```

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

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

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 [112…

```
# Read the data:

data = pd.read_csv("5_a.csv")
print(data.head()) # display the head of the data
print("\n")

# Derive class labels from given probability scores:
# Let y_pred be the derived class labels
data["y_pred"] = pd.Series([o if data.iloc[i,1]<0.5 else 1 for i in range(len(data))])

print("Display the data after including a new column :")
print(data.head())

print("\n")

print("The shape of the data is :")
print(data.shape)
```

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


Display the data after including a new column :
     y     proba  y_pred
0  1.0  0.637387       1
1  1.0  0.635165       1
2  1.0  0.766586       1
3  1.0  0.724564       1
4  1.0  0.889199       1


The shape of the data is :
(10100, 3)
```

In [113…

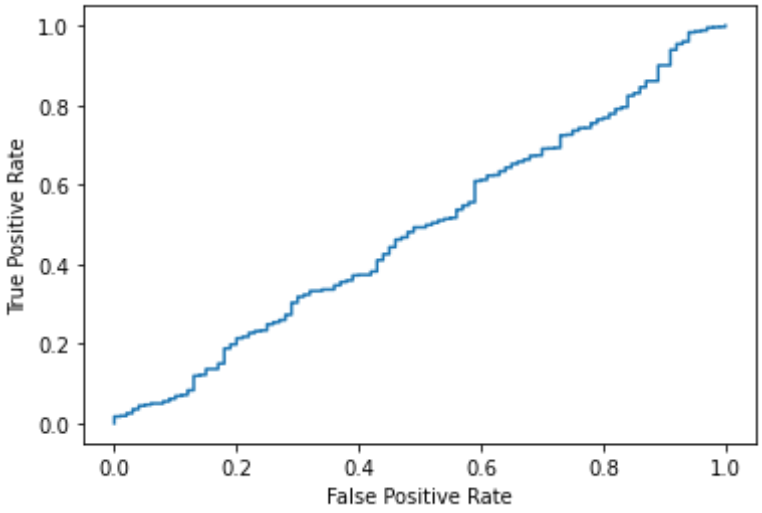
```
# write your code here:

# 1) CONFUSION MATRIX:

actual = data["y"].tolist()           # convert series to list
```

```
predicted = data["y_pred"].tolist()   # convert series to list
TN,FN,FP,TP = 0,0,0,0            # initialize confusion matrix scores

# Calculate the scores using for loop
for i in tqdm(range(len(data))):
    if (actual[i] == 0) and (predicted[i] == 0):
        TN +=1
    if (actual[i] == 1) and (predicted[i] == 0):
        FN +=1
    if (actual[i] == 0) and (predicted[i] == 1):
        FP +=1
    if (actual[i] == 1) and (predicted[i] == 1):
        TP +=1
print("1) The values of confusion matrix :")
print("True negative is",TN)
print("False negative is",FN)
print("False positive is",FP)
print("True positive is",TP,"\n")

# 2) F1 SCORE:

# Use the formula for precision and recall
precision = float(TP/(TP + FP))
recall = float(TP/(TP + FN))

# Use the Harmonic mean and compute F1Score
F1_score = float((2*precision*recall)/(precision+recall))
print("2) The F1 score is :",F1_score,"\n")

# 3) AUC SCORE:

# pick unique threshold probability values
n_unique_thresholds = data["proba"].unique()
print("3) For AUC Score :")
print("The number of unique threshold values are: ",len(n_unique_thresholds))

# Sort thresholds in ascending order and create a list of yscores/probability scores.
sorted_thresholds = sorted(n_unique_thresholds)
yscores = data["proba"].tolist()
TPR = []   # create list to store TPR values
FPR = []   # create list to store FPR values
```

```
for j in tqdm(sorted_thresholds): # Iterate through each threshold values

    temp = []                     # A temporary list to store class labels
    count_tp = 0                  # A list to store true positive counts
    count_fp = 0                  # A list to store false positive counts

    # Compare the class labels accordingly and store the counts:
    for value in yscores:
        if value >= j:
            temp.append(1)
        else:
            temp.append(0)
    for i in range(len(temp)):    # Compute True positives and false positives
        if (actual[i] == 1) and (temp[i] == 1):
            count_tp += 1
        if (actual[i] == 0) and (temp[i] == 1):
            count_fp += 1

    tpr = float((count_tp)/actual.count(1))
    fpr = float((count_fp)/actual.count(0))

    # Append each value into the TPR AND FPR list
    TPR.append(tpr)
    FPR.append(fpr)

#Ensure dimension of TPR and FPR lists are same
print("The length of TPR list :",len(TPR))
print("The length of FPR list :",len(FPR))

# Plot ROC curve using plot()
import matplotlib.pyplot as plt
plt.plot(FPR,TPR)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()

# Calculate Area under Curve:
auc = np.trapz(np.array(TPR[::-1]),np.array(FPR[::-1]))
print("The area under the curve is :",float(auc),"\n")


# 4) ACCURACY SCORE:
```

```
accuracy = float((TP+TN)/(TP+FP+TN+FN))*100 # This is the formula for accuracy
print("4) The accuracy score is :",accuracy)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 10100/10100 [00:00<00:00, 506346.54it/
s]
  0%|                                                                                | 12/10100 [00:00<01:32, 109.38it/
s]
1) The values of confusion matrix :
True negative is 0
False negative is 0
False positive is 100
True positive is 10000

2) The F1 score is : 0.9950248756218906

3) For AUC Score :
The number of unique threshold values are:  10100
100%|██████████████████████████████████████████████████████████████████████████| 10100/10100 [01:23<00:00, 120.60it/
s]
The length of TPR list : 10100
The length of FPR list : 10100
```




```
The area under the curve is : 0.48829900000000004

4) The accuracy score is : 99.00990099009901
```

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</pre>

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

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
4. Compute Accuracy Score

In [114…

```
# Read the data:
data_b = pd.read_csv("5_b.csv")
print(data_b.head()) # display the head of the data
print("\n")

# Derive class labels from given probability scores:
# Let y_pred be the derived class labels
data_b["y_pred"] = pd.Series([0 if data_b.iloc[i,1]< 0.5 else 1 for i in range(len(data_b))])

print("Display the data after including a new column :")
print(data_b.head())

print("\n")

print("The shape of the data is :")
print(data_b.shape)
```

```
     y     proba
0  0.0  0.281035
1  0.0  0.465152
2  0.0  0.352793
3  0.0  0.157818
```

```
4  0.0  0.276648


Display the data after including a new column :
     y     proba  y_pred
0  0.0  0.281035       0
1  0.0  0.465152       0
2  0.0  0.352793       0
3  0.0  0.157818       0
4  0.0  0.276648       0


The shape of the data is :
(10100, 3)
```

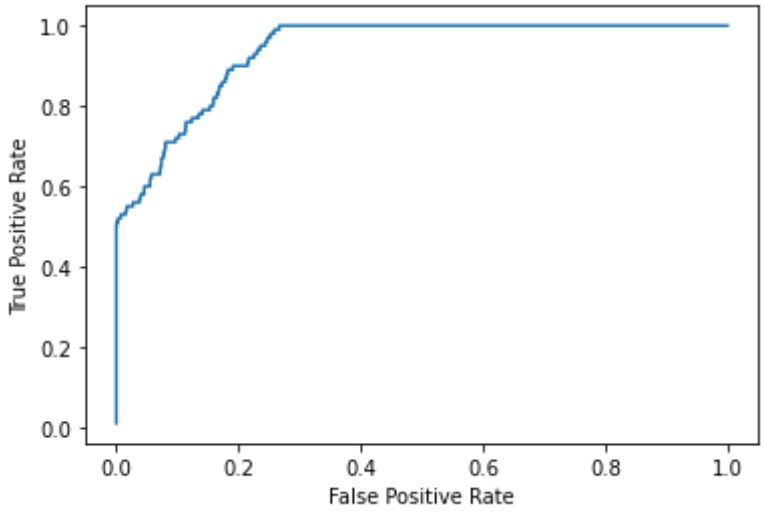
In [115…

```python
# write your code here:

# 1) CONFUSION MATRIX:

actual = data_b["y"].tolist()           # convert series to list
predicted = data_b["y_pred"].tolist()   # convert series to list
TN,FN,FP,TP = 0,0,0,0                   # initialize confusion matrix scores

# Calculate the scores using for loop
for i in tqdm(range(len(data_b))):
    if (actual[i] == 0) and (predicted[i] == 0):
        TN +=1
    if (actual[i] == 1) and (predicted[i] == 0):
        FN +=1
    if (actual[i] == 0) and (predicted[i] == 1):
        FP +=1
    if (actual[i] == 1) and (predicted[i] == 1):
        TP +=1
print("1) The values of confusion matrix :")
print("True negative is",TN)
print("False negative is",FN)
print("False positive is",FP)
print("True positive is",TP,"\n")

# 2) F1 SCORE:

# Use the formula for precision and recall
precision = float(TP/(TP + FP))
```

```
recall = float(TP/(TP + FN))

# Use the Harmonic mean and compute F1Score
F1_score = float((2*precision*recall)/(precision+recall))
print("2) The F1 score is :",F1_score,"\n")

# 3) AUC SCORE:

# pick unique threshold probability values
n_unique_thresholds = data_b["proba"].unique()
print("3) For AUC Score :")
print("The number of unique threshold values are: ",len(n_unique_thresholds))

# Sort thresholds in ascending order and create a list of yscores/probability scores.
sorted_thresholds = sorted(n_unique_thresholds)
yscores = data_b["proba"].tolist()
TPR = []                          # create list to store TPR values
FPR = []                          # create list to store FPR values

for j in tqdm(sorted_thresholds): # Iterate through each threshold values

    temp = []                     # A temporary list to store class labels
    count_tp = 0                  # A list to store true positive counts
    count_fp = 0                  # A list to store false positive counts

    # Compare the class labels accordingly and store the counts:
    for value in yscores:
        if value >= j:
            temp.append(1)
        else:
            temp.append(0)
    for i in range(len(temp)):    # Compute True positives and false positives
        if (actual[i] == 1) and (temp[i] == 1):
            count_tp += 1
        if (actual[i] == 0) and (temp[i] == 1):
            count_fp += 1

    tpr = float((count_tp)/actual.count(1))
    fpr = float((count_fp)/actual.count(0))

    # Append each value into the TPR AND FPR list
    TPR.append(tpr)
```

```
    FPR.append(fpr)

#Ensure dimension of TPR and FPR lists are same
print("The length of TPR list :",len(TPR))
print("The length of FPR list :",len(FPR))

# Plot ROC curve using plot()
import matplotlib.pyplot as plt
plt.plot(FPR,TPR)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()

# Calculate Area under Curve:
auc = np.trapz(np.array(TPR[::-1]),np.array(FPR[::-1]))
print("The area under the curve is :",float(auc),"\n")


# 4) ACCURACY SCORE:
accuracy = float((TP+TN)/(TP+FP+TN+FN))*100 # This is the formula for accuracy
print("4) The accuracy score is :",accuracy)
```

```
100%|██████████████████████████████████████████████████████████████████████| 10100/10100 [00:00<00:00, 482241.11it/
s]
  0%|                                                                                | 11/10100 [00:00<01:32, 109.18it/
s]
1) The values of confusion matrix :
True negative is 9761
False negative is 45
False positive is 239
True positive is 55

2) The F1 score is : 0.2791878172588833

3) For AUC Score :
The number of unique threshold values are:  10100
100%|██████████████████████████████████████████████████████████████████████| 10100/10100 [01:30<00:00, 111.77it/
s]
The length of TPR list : 10100
The length of FPR list : 10100
```




```
The area under the curve is : 0.9377570000000001

4) The accuracy score is : 97.18811881188118
```

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

$A = 500 \times \text{number of false negative} + 100 \times \text{numebr of false positive}$

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 [129…

```
# Read the data:
data_c = pd.read_csv("5_c.csv")
print("(i) Display top 5 rows of the data :")
print(data_c.head(),"\n") # display the head of the data
print("(ii) The shape of the data is :")
print(data_c.shape,"\n")

# create a list of actual class labels:
```

```
actual = data_c["y"].tolist()

# pick unique threshold probability values:
n_unique_thresholds = data_c["prob"].unique()
print("(iii) The number of unique threshold values are: ",len(n_unique_thresholds),"\n")

# Sort thresholds in ascending order and create a list of yscores/probability scores.
sorted_thresholds = sorted(n_unique_thresholds)
yscores = data_c["prob"].tolist()
FN = []                          # create list to store FN values
FP = []                          # create list to store FP values

for j in sorted_thresholds: # Iterate through each threshold values

    temp = []                    # A temporary list to store class labels
    count_fn = 0                 # A list to store false negative counts
    count_fp = 0                 # A list to store false positive counts

    # Compare the class labels accordingly and store the counts:
    for value in yscores:
        if value >= j:
            temp.append(1)
        else:
            temp.append(0)
    for i in range(len(temp)):    # Compute false positives and false negatives
        if (actual[i] == 1) and (temp[i] == 0):
            count_fn += 1
        if (actual[i] == 0) and (temp[i] == 1):
            count_fp += 1

    fn = count_fn
    fp = count_fp

    # Append each value into the FN AND FP list
    FN.append(fn)
    FP.append(fp)

#Ensure dimension of FN and FP lists are same-->HERE: FP means false +ve FN means false -ve
print("(iv) The length of FN list :",len(FN),"\n")
print("(v) The length of FP list :",len(FP),"\n")

# compute metric A for pairs of FN & FP values:
```

```
A_list = []                          # create an empty list to store values of metric A
for i in range(len(FN)):
    A_list.append((500*FN[i])+(100*FP[i]))

# Pick the lowest value and its corresponding threshold:
lowest_val = sorted(A_list)[0]
print("(vi) The lowest value of metric A :",lowest_val,"\n")
print("(vii) The best threshold value :",sorted_thresholds[A_list.index(lowest_val)])
```

```
(i) Display top 5 rows of the data :
   y      prob
0  0  0.458521
1  0  0.505037
2  0  0.418652
3  0  0.412057
4  0  0.375579

(ii) The shape of the data is :
(2852, 2)

(iii) The number of unique threshold values are:  2791

(iv) The length of FN list : 2791

(v) The length of FP list : 2791

(vi) The lowest value of metric A : 141000

(vii) The best 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
    </pre>

In [10]:

```
# Read the data:

data_d = pd.read_csv("5_d.csv")
print("Display Top 5 rows of the data :")
print(data_d.head(),"\n")
print("The shape the data :",data_d.shape,"\n")

# 1) Compute mean square error:

y = data_d["y"].tolist()                          # Store actual values in a list
y_mean = float(sum(y)/len(y))                     # compute mean of actual values
y_hat = data_d["pred"].tolist()                   # y_hat --> list of predicted values

squared_errors = []                               # store squared errors into this list
for i in range(len(data_d)):
    squared_errors.append((y[i]-y_hat[i])**2)

# use the formula to find MSE:
MSE = float((sum(squared_errors))/len(squared_errors))
print("1) The value of mean squared error :",MSE,"\n")


# 2) Compute mean absolute percentage error:

errors = []                                       # Store absolute value of error into errors
for j in range(len(data_d)):
    errors.append(abs((y[j]-y_hat[j])))

# use the formula to find MAPE:
MAPE = float(sum(errors)/sum(y))
print("2) The Mean absolute percentage error :",MAPE,"\n")



# 3) Compute R^2:
total_errors = []                                 #To store squared errors of simple mean model
for k in range(len(y)):
    total_errors.append((y[k]-y_mean)**2)

SS_residuals = sum(squared_errors)                # compute sum of squared residuals
SS_total = sum(total_errors)                      # compute sum of squares total
```

```python
# use the formula to find R^2:
R_squared = float(1-(SS_residuals/SS_total))
print("3) The Coefficient of determination :",R_squared)
```

```
Display Top 5 rows of the data :
       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

The shape the data : (157200, 2)

1) The value of mean squared error : 177.16569974554707

2) The Mean absolute percentage error : 0.1291202994009687

3) The Coefficient of determination : 0.9563582786990964
```