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/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]:

```
# read CSV
df_5a = pd.read_csv("5_a.csv")
df_5a.head()
```

Out[2]:

	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

In [3]:

```
#converting probability scores to labels
#df_5a['y_pred'] = [1.0 if i > 0.5 else 0.0 for i in df_5a['proba']]
#df_5a['y_pred'].head()
#len(df_5a)
```

In [4]:

```
def pred(df, t):
  y_pred = []
  for i in df['proba']:
    if i > t:
      y_pred.append(1.0)
    else:
      y_pred.append(0.0)
  return y_pred
```

In [5]:

```
df_5a['y_pred'] = pred(df_5a, 0.5)

# df_5a = df_5a.drop(columns = ['pred', 'y_pred1'])
print(df_5a)
```

```
         y     proba  y_pred
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
...    ...       ...     ...
10095  1.0  0.665371     1.0
10096  1.0  0.607961     1.0
10097  1.0  0.777724     1.0
10098  1.0  0.846036     1.0
10099  1.0  0.679507     1.0
```

```
[10100 rows x 3 columns]
```

In [6]:

```
def confusion_mat(df):
  TP, TN, FP, FN = 0, 0, 0, 0
  for i in range(len(df)):
    if df['y'][i] == 1.0 and df['y_pred'][i] == 1.0:
      TP += 1
    if df['y'][i] == 0.0 and df['y_pred'][i] == 0.0:
      TN += 1
    if df['y'][i] == 1.0 and df['y_pred'][i] == 0.0:
      FN += 1
    if df['y'][i] == 0.0 and df['y_pred'][i] == 1.0:
      FP += 1
  return TP, TN, FN, FP
```

In [7]:

```
#1. Confusion Matrix
TP, TN, FN, FP = confusion_mat(df_5a)
print(TP, TN, FN, FP)
```

```
10000 0 0 100
```

In [8]:

```
#2. precision and recall, F1 Score
def F1_score(TP, TN, FP, FN):
  pre = TP / (TP + FP)
  rec = TP / (TP + FN)
  print("Precision and Recall ", pre, rec)
  F1_scr = 2*(pre * rec) / (pre + rec)
  print("F1 Score", F1_scr)
F1_score(TP, TN, FP, FN)
```

```
Precision and Recall  0.9900990099009901 1.0
F1 Score 0.9950248756218906
```

In [9]:

```
df_5a = df_5a.sort_values(by = 'proba', ascending = False)
```

In [10]:

```
def AUC(df):
  tpr_arr = []
  fpr_arr = []
  s = df['y'].value_counts()
  P = s[1]
  N = s[0]
  for thrshld in df['proba']:
    df['y_pred'] = pred(df, thrshld)
    #print(df['y_pred'])
    TP_, TN_, FN_, FP_ = confusion_mat(df)
    # print(TP_,TN_,FN_,FP_)
    tpr_arr.append(TP_/P)
    fpr_arr.append(FP_/N)
    # print(tpr_arr)
    # print(fpr_arr)
  return np.trapz(tpr_arr, fpr_arr)
```

In [14]:

```
#AUC_scr = AUC(df_5a)
```

In [12]:

```
#3. AUC Score
print('AUC SCORE: ', AUC_scr)
#here i got a negative value bz sorted in ascending. corrected for 5_b dataset
```

```
AUC SCORE:  0.48829900000000004
```

In [12]:

```
#4. Accuracy Score
def accuracy(TP, TN, FP, FN):
  Acc = (TP + TN) / (TP + TN + FN + FP)
  print("Accuracy Score", Acc)
accuracy(TP, TN, FP, FN)
```

```
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

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

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/a/39678975/4084039

4. Compute Accuracy Score

In [13]:

```
# read CSV
df_5b = pd.read_csv("5_b.csv")
df_5b.head()
```

Out[13]:

	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

In [14]:

```
#converting probability scores to labels
df_5b['y_pred'] = pred(df_5b, 0.5)
```

In [15]:

```
#1. Confusion Matrix
TP_5b, TN_5b, FN_5b, FP_5b = confusion_mat(df_5b)
print(TP_5b, TN_5b, FN_5b, FP_5b)
```

```
55 9761 45 239
```

In [16]:

```
#2. precision and recall, F1 Score
F1_score(TP_5b, TN_5b, FN_5b, FP_5b)
```

```
Precision and Recall  0.55 0.1870748299319728
F1 Score 0.2791878172588833
```

In [47]:

```
#3. AUC score
df_5b = df_5b.sort_values(by = 'proba', ascending = False)
AUC_scr_5b = AUC(df_5b)
print(AUC_scr_5b)
```

```
0.9376570000000001
```

In [18]:

```
#4. Accuracy Score
accuracy(TP_5b, TN_5b, FN_5b, FP_5b)
```

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

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

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 [34]:

```
 # Best thershold
 # https://github.com/mustaffa-hussain/Performance-Metric/blob/master/Performance%20metric.ipynb
def opt_thrsh(df):
  tpr_arr = []
  fpr_arr = []
  s = df['y'].value_counts()
  P = s[1]
  N = s[0]
  bst_t = {}
  for thrshld in df['prob']:
    df['y_pred'] = pred1(df, thrshld)
    TP_, TN_, FN_, FP_ = confusion_mat(df)
    # tpr_arr.append(TP_/P)
    # fpr_arr.append(FP_/N)
    A = (500 * FN_) + (100 * FP_)
    bst_t[thrshld] = A
  return bst_t
  #return np.trapz(tpr_arr, fpr_arr)
```

In [33]:

```
def pred1(df, t):
    y_pred1 = []
    [y_pred1.append(1.0) if i > t else y_pred1.append(0.0) for i in df['prob']]
    return y_pred1
```

In [31]:

```
#pred1(df_5c)
```

In [35]:

```
df_5c = pd.read_csv("5_c.csv")
print(df_5c.head(5))
df_5c = df_5c.sort_values(by = 'prob', ascending = False)
result = opt_thrsh(df_5c)
```

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

In [45]:

```
#print(min(result.values()))
```

```
141000
```

In [46]:

```
temp = min(result.values())
thr = [k for k in result if result[k] == temp]
print("best thershold:", thr)
```

```
best thershold: [0.22987164436159915]
```

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 [2]:

```
df_5d = pd.read_csv('5_d.csv')
df_5d.head()
```

Out[2]:

	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 [3]:

```
def loss(data):
    loss1 = []
    [loss1.append(i - j) for index, (i, j) in enumerate(zip(data['y'], data['pred']))]
    return loss1
```

In [4]:

```
def MSE(data):
    mse = SS_res(df_5d) / len(data)
    return mse
```

In [7]:

```
#1. MSE
print("Mean Squared error:", MSE(df_5d))
```

```
Mean Squared error: 177.16569974554707
```

In [9]:

```
def SS_res(data):
    x = 0
    y = loss(data)
    for i in y:
        x += (i * i)
    return x
```

In [10]:

```
def SS_total(data):
    x = 0
    mean = df_5d['y'].mean()
    for i in df_5d['y']:
        x += (i - mean) * (i - mean)
    return x
```

In [8]:

```
# SS_total(df_5d)
```

In [11]:

```
def R_sqr(data):
    return 1 - (SS_res(df_5d) / SS_total(df_5d))
```

In [12]:

```
# 3. R_sqr error
print("coeffient of determination: ", R_sqr(df_5d))
```

```
coeffient of determination:  0.9563582786990964
```

In [22]:

```
def abs_val_err(data):
    x = 0
    l = loss(data)
    for i in l:
        x += abs(i)
    return x
```

In [25]:

```
def MAPE(data):
    #to avoid the divide by zero just take mean of actual values. sum(|error|) / sum(actual value). n is cancelle
    MAP = abs_val_err(data) / sum(data['y'])
    return MAP
```

In [26]:

```
#2. MAPE
print('Mean Absolute Percentage Error: ', MAPE(df_5d))
```

```
Mean Absolute Percentage Error:  0.1291202994009687
```

In [ ]:

Processing math: 100%