Compute performance metrics for the given Y and Y_score without sklearn

In [13]:

```
import numpy as np
import pandas as pd
# other than these two you should not import any other packages
from tqdm import tqdm
import pdb
```

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 negative s 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 [14]:

```
#Note 2: use pandas or numpy to read the data from 5_a.csv
data = pd.read_csv('5_a.csv')
data.head(10)
```

Out[14]:

	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
5	1.0	0.601600
6	1.0	0.666323
7	1.0	0.567012
8	1.0	0.650230

	y	proba
9	1.0	0.829346

In [15]:

```
#Note 3: you need to derive the class labels from given score
data.proba = list(map(lambda datapoint: 1 if (datapoint >= 0.5) else 0, data.proba))
data.head(10)
```

Out[15]:

	y	proba
0	1.0	1
1	1.0	1
2	1.0	1
3	1.0	1
4	1.0	1
5	1.0	1
6	1.0	1
7	1.0	1
8	1.0	1
9	1.0	1

In [16]:

```
# 1. Compute Confusion Matrix
def compute_confusion_matrix(y_score, y_predicted):
    confusionMX = []
    tp = tn = fp = fn = 0
    for i in range(len(y_score)):
        if(y_predicted[i] == 0 and y_score[i] == 0):
            tn += 1
        if(y_predicted[i] == 0 and y_score[i] == 1):
            fn += 1
        if(y_predicted[i] == 1 and y_score[i] == 0):
            fp += 1
        if(y_predicted[i] == 1 and y_score[i] == 1):
            tp += 1
    confusionMX.append([tn, fn])
    confusionMX.append([fp, tp])

    return confusionMX
```

In [17]:

```
confusion_matrix = compute_confusion_matrix(data.y, data.proba)
print('Confusion Matrix: ', confusion_matrix)
```

```
Confusion Matrix:  [[0, 0], [100, 10000]]
```

In [18]:

```
# 2. Compute F1 Score
def compute_f1_score(confusion_matrix):
    n = confusion_matrix[0][0] + confusion_matrix[1][0]
    p = confusion_matrix[0][1] + confusion_matrix[1][1]

    tp = confusion_matrix[1][1]
    fp = confusion_matrix[1][0]

    precision = tp/(tp + fp)
    recall = tp/p

    f1_score = 2 * ((precision * recall)/(precision + recall))
```

```python
    return f1_score

f1_score = compute_f1_score(confusion_matrix)
print('F1 Score is ', f1_score)
```

```
F1 Score is  0.9950248756218906
```

In [19]:

```python
#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)

data = pd.read_csv('5_a.csv')
def compute_tpr_fpr_array(data, y_score, y_predicted):
    column = 0
    tpr_array = []
    fpr_array = []

    for i in tqdm(range(len(data.y))):
        data[column] = list(map(lambda dp: 1 if (dp >= y_predicted[column]) else 0, y_predicted))
        confusion_matrix = compute_confusion_matrix(y_score, data[column])

        p = confusion_matrix[0][1] + confusion_matrix[1][1]
        n = confusion_matrix[0][0] + confusion_matrix[1][0]

        tpr = confusion_matrix[1][1]/p
        fpr = confusion_matrix[0][1]/n

        tpr_array.append(tpr)
        fpr_array.append(fpr)

        column += 1

    return tpr_array, fpr_array
```

In [20]:

```python
tpr_array, fpr_array = compute_tpr_fpr_array(data, data.y, data.proba)
```

```
100%|██████████████████████████████| 10100/10100 [1:33:16<00:00,  1.80it/s]
```

In [21]:

```python
#Computing AUC score
auc_score = np.trapz(tpr_array, fpr_array)
print('AUC Score: ', auc_score)
```

```
AUC Score:  6.545454499999918
```

In [22]:

```python
#Compute Accuracy Score
def compute_accuracy_score(confusion_matrix):
    tn = confusion_matrix[0][0]
    tp = confusion_matrix[1][1]
    fp = confusion_matrix[1][0]
    fn = confusion_matrix[0][1]

    accuracy_score = (tn + tp)/(tp + tn + fp + fn)

    return accuracy_score
```

In [23]:

```python
accuracy_score = compute_accuracy_score(confusion_matrix)
```

```
print('Accuracy Score: ', accuracy_score)
```

```
Accuracy Score:  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 [52]:

```
#Note 2: use pandas or numpy to read the data from 5_b.csv
data = pd.read_csv('5_b.csv')
data.head(10)
```

Out[52]:

	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
5	0.0	0.190260
6	0.0	0.320328
7	0.0	0.435013
8	0.0	0.284849
9	0.0	0.427919

In [53]:

```
#Note 3: you need to derive the class labels from given score
data['y_predicted'] = list(map(lambda datapoint: 1 if (datapoint >= 0.5) else 0, data.pr
oba))
data.head(10)
```

Out[53]:

	y	proba	y_predicted
0	0.0	0.281035	0

	y	proba	y_predicted
1	0.0	0.465152	0
2	0.0	0.352793	0
3	0.0	0.157818	0
4	0.0	0.276648	0
5	0.0	0.190260	0
6	0.0	0.320328	0
7	0.0	0.435013	0
8	0.0	0.284849	0
9	0.0	0.427919	0

In [54]:

```
#Compute Confusion Matrix
confusion_matrix = compute_confusion_matrix(data.y, data.y_predicted)
print('Confusion Matrix: ', confusion_matrix)
```

```
Confusion Matrix:  [[9761, 45], [239, 55]]
```

In [55]:

```
#Compute F1 Score
f1_score = compute_f1_score(confusion_matrix)
print('F1 Score: ',f1_score)
```

```
F1 Score:  0.2791878172588833
```

In [56]:

```
#Compute tpr_array and fpr_array
tpr_array, fpr_array = compute_tpr_fpr_array(data, data.y, data.proba)
```

```
100%|██████████████████████████████████████████████████| 10100/10100 [1:37:51<00:00,  1.72it/s]
```

In [57]:

```
#Computing AUC score
auc_score = np.trapz(tpr_array, fpr_array)
print('AUC Score: ', auc_score)
```

```
AUC Score:  -3.2309224740068O3e-17
```

In [58]:

```
def compute_accuracy_score(confusion_matrix):
    tn = confusion_matrix[0][0]
    tp = confusion_matrix[1][1]
    fp = confusion_matrix[1][0]
    fn = confusion_matrix[0][1]

    accuracy_score = (tn + tp)/(tp + tn + fp + fn)

    return accuracy_score

accuracy_score = compute_accuracy_score(confusion_matrix)
print('Accuracy Score: ', accuracy_score)
```

```
Accuracy Score:  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 [31]:

```python
# Note 2: use pandas or numpy to read the data from 5_c.csv
data = pd.read_csv('5_c.csv')
data.head(10)
```

Out[31]:

	y	prob
0	0	0.458521
1	0	0.505037
2	0	0.418652
3	0	0.412057
4	0	0.375579
5	0	0.595387
6	0	0.370288
7	0	0.299273
8	0	0.297000
9	0	0.266479

In [32]:

```python
# Computing best Threshold
def compute_best_threshold(data, y_score, y_predicted):
    col = 0
    metric_A = {}

    for i in tqdm(range(len(y_score))):
        data[col] = list(map(lambda dp: 1 if (dp > y_predicted[col]) else 0, y_predicted
))
        confusion_matrix = compute_confusion_matrix(y_score, data[col])
        A = 500*confusion_matrix[0][1] + 100*confusion_matrix[1][0]
        metric_A[A] = y_predicted[col]
        col += 1

    threshold = sorted(metric_A.items())
    best_threshold = threshold[0][1]

    return best_threshold
```

In [33]:

```python
best_threshold = compute_best_threshold(data)
print('Best Threshold: ', best_threshold)
```

```
100%|██████████████████████████████████████████████████| 2852
/2852 [08:05<00:00,  5.87it/s]
```

```
Best Threshold:  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 [44]:

```python
#Note 2: use pandas or numpy to read the data from 5_d.csv
data = pd.read_csv('5_d.csv')
data.head(10)
```

Out[44]:

	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
5	133.0	153.0
6	148.0	139.0
7	172.0	145.0
8	153.0	162.0
9	162.0	154.0

In [45]:

```python
# Function to Compute Mean Square Error
def compute_mean_square_error(y_score, y_pred):
    squared_error = []
    n = len(y_score)
    mean_square_error = 0
    for i in tqdm(range(n)):
        error = y_score[i] - y_pred[i]
        error *= error
        squared_error.append(error)

    for i in squared_error:
        mean_square_error += i

    mean_square_error /= n

    return mean_square_error
```

In [47]:

```python
# Compute Mean Square Error
mse = compute_mean_square_error(data.y, data.pred)
print('Mean Square Error: ', mse)
```

```
100%|██████████████████████████████████████████████████| 157200/157200 [00:02<00:00, 66105.83it/s]
```

```
Mean Square Error:  177.16569974554707
```

In [48]:

```python
# Function to compute Mean Absolute Percentage Error (MAPE)
def compute_mape(y_score, y_predicted):
    n = len(y_score)
    e = 0
    a_bar = 0
    for i in range(n):
        e += abs(y_score[i])
        a_bar += y_predicted[i]

    mape = (e/a_bar)

    return mape
```

In [49]:

```python
# Compute MAPE
mape = compute_mape(data.y, data.pred)
print('mape: ', mape)
```

```
mape:  1.0011788074907857
```

In [50]:

```python
#Function to Compute R^2 error
def compute_R_squared_error(y_score, y_predicted):
    n = len(y_score)
    y_bar = 0
    for i in tqdm(range(n)):
        y_bar += y_score[i]
    y_bar = (y_bar/n)

    for i in range(n):
        ss_total = ((y_score[i])*(y_bar))*((y_score[i])*(y_bar))
        ss_res = (y_score[i] - y_predicted[i])*(y_score[i] - y_predicted[i])


    R_squared = 1 - (ss_res/ss_total)

    return R_squared
```

In [51]:

```python
#Compute R^2 error
R_squared_error = compute_R_squared_error(data.y, data.pred)
print('R_squared_error: ', R_squared_error)
```

```
100%|██████████████████████████████| 157200/15720
0 [00:01<00:00, 118367.96it/s]
```

```
R_squared_error:  0.9999818016597894
```