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

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

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
   Note: it should be numpy.trapz(tpr_array, fpr_array) not numpy.trapz(fpr_array, tpr_array)
   Note- Make sure that you arrange your probability scores in descending order while calculating AUC
4. Compute Accuracy Score

In [2]:
```
df_a = pd.read_csv('5_a.csv')
df_a.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]:
```
# https://numpy.org/doc/stable/reference/generated/numpy.where.html#numpy-where

df_a['y_pred'] = np.where(df_a['proba'] > .5, 1, 0)
df_a['y_pred'].value_counts()
```

Out[3]:
```
1    10100
Name: y_pred, dtype: int64
```

In [4]:
```
# https://scikit-learn.org/stable/_images/sphx_glr_plot_confusion_matrix_001.png

def confusion_matrix_scores(df, actual, predicted):
    true_neg = df[(df[actual] == 0) & (df[predicted] == 0)].shape[0]
    false_pos = df[(df[actual] == 0) & (df[predicted] == 1)].shape[0]

    true_pos = df[(df[actual] == 1) & (df[predicted] == 1)].shape[0]
    false_neg = df[(df[actual] == 1) & (df[predicted] == 0)].shape[0]

    df_confusion_mat = pd.DataFrame({'Predicted No': [true_neg, false_neg],
                      'Predicted Yes' :[false_pos, true_pos]},
                     index = ['Actual No', 'Actual Yes'])

    return true_neg, false_pos, false_neg, true_pos, df_confusion_mat

# true_negative, false_positive, false_negative, true_positive, df_confusion_matrix
```

In [5]:
```
# 1. Compute Confusion Matrix

true_negative, false_positive, false_negative, true_positive, df_confusion_matrix = \
                                            confusion_matrix_scores(df_a, 'y', 'y_pred')
```

```
# true_negative = df_a[(df_a.y == 0) & (df_a.y_pred == 0)].shape[0]
# false_positive = df_a[(df_a.y == 0) & (df_a.y_pred == 1)].shape[0]

# true_positive = df_a[(df_a.y == 1) & (df_a.y_pred == 1)].shape[0]
# false_negative = df_a[(df_a.y == 1) & (df_a.y_pred == 0)].shape[0]

confusion_matrix = [[true_negative, false_positive], [false_negative, true_positive]]
print('Confusion Matrix :' ,confusion_matrix)

df_confusion_matrix
```

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

Out[5]:

	Predicted No	Predicted Yes
Actual No	0	100
Actual Yes	0	10000

In [6]:

```
# 2. Compute F1 Score

def f1_score_calc(true_pos, false_pos, false_neg):
    '''
    Computes F1-Score

    Input : true_positive, false_positive, false_negative

    Output : F1-Score
    '''
    precission = true_pos / (true_pos + false_pos)
    recall = true_pos / (true_pos + false_neg)
    f1_score = 2 * ((precission * recall) / (precission + recall))

    return round(f1_score, 5)

f1_score_ = f1_score_calc(true_positive, false_positive, false_negative)

print('F1-Score :', round(f1_score_, 5))
```

```
F1-Score : 0.99502
```

In [7]:

```
# 4. Compute Accuracy Score

def accuracy_score(true_pos, true_neg, df):
    '''
    Computes Accuracy Score

    Input : true_positive, true_negative, dataframe

    Output : Accuracy Score
    '''
    accuracy_sco = (true_pos + true_neg) / df.shape[0]

    return round(accuracy_sco, 5)

accuracy_ = accuracy_score(true_positive, true_negative, df_a)

print('Accuracy :', accuracy_)
```

```
Accuracy : 0.9901
```

In [8]:

```
# 3. Compute AUC Score

# print(df_a.proba.unique())
# print(sorted(df_a.proba.unique(), reverse = False)[:5])

def compute_auc_score(df):
    '''
    Computes AUC Score

    Input : dataframe

    Output : AUC Score
    '''
    thresholds_list = sorted(df.proba.unique(), reverse = True)

    #lists for storing true_positive & false_positive rates
    true_positive_rate_list = []
    false_positive_rate_list = []

#     print(df.head())
    for threshold in thresholds_list:

        # Mapping predictions based on threshold value
        df['threshold'] = np.where(df['proba'] <= threshold, 0, 1)
```

```
        true_negative, false_positive, false_negative, true_positive, df_confusion_matrix = \
                                                confusion_matrix_scores(df ,'y', 'threshold')

        true_positive_rate = true_positive / (true_positive + false_negative)
        false_positive_rate = false_positive / (true_negative + false_positive)

        true_positive_rate_list.append(true_positive_rate)
        false_positive_rate_list.append(false_positive_rate)

    '''
    df['TP_rate'] = true_positive_rate_list
    df['FP_rate'] = false_positive_rate_list
    '''
    auc_score = np.trapz(true_positive_rate_list,false_positive_rate_list)

    return round(auc_score, 5)

auc_score_ = compute_auc_score(df_a)

print('AUC Score :', auc_score_)
```

```
AUC Score : 0.4883
```

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

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

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 Note- Make sure that you arrange your probability scores in descending order while calculating AUC
4. Compute Accuracy Score

In [9]:
```
df_b=pd.read_csv('5_b.csv')
df_b.head()
```

Out[9]:

	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 [10]:
```
# write your code here for task B

df_b['y_pred'] = np.where(df_b['proba'] > .5, 1, 0)
df_b['y_pred'].value_counts()
```

Out[10]:
```
0    9806
1     294
Name: y_pred, dtype: int64
```

In [11]:
```
# 1. Compute Confusion Matrix

# https://scikit-learn.org/stable/_images/sphx_glr_plot_confusion_matrix_001.png

true_negative, false_positive, false_negative, true_positive, df_confusion_matrix = \
                                                confusion_matrix_scores(df_b, 'y', 'y_pred')

confusion_matrix = [[true_negative, false_positive], [false_negative, true_positive]]
print('Confusion Matrix :' ,confusion_matrix)
```

```
df_confusion_matrix
```

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

Out[11]:

	Predicted No	Predicted Yes
Actual No	9761	239
Actual Yes	45	55

In [12]:

```
# 2. Compute F1 Score

f1_score_ = f1_score_calc(true_positive, false_positive, false_negative)

print('F1-Score :', f1_score_)
```

```
F1-Score : 0.27919
```

In [13]:

```
# 4. Compute Accuracy Score

accuracy_ = accuracy_score(true_positive, true_negative, df_b)

print('Accuracy :', accuracy_)
```

```
Accuracy : 0.97188
```

In [14]:

```
# 3. Compute AUC Score

auc_score_ = compute_auc_score(df_b)

print('AUC Score :', auc_score_)
```

```
AUC Score : 0.93766
```

C. Compute the best threshold (similarly to ROC curve computation) of probability which gives lowest values of metric **A** for the given data

you will be predicting label of a data points like this: $y^{pred}= \text{[0 if y_score < threshold else 1]}$

$ 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 [15]:

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

Out[15]:

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

In [16]:

```
# write your code for task C

thresholds_list = sorted(df_c.prob.unique(), reverse = True)

#lists for storing valuation_metric scores
valuation_metric_list = []

for threshold in thresholds_list:

#     Mapping predictions based on threshold value
    df_c['threshold'] = np.where(df_c['prob'] <= threshold, 0, 1)

#     confusion_matrix_scores calculation
    true_negative, false_positive, false_negative, true_positive, df_confusion_matrix = \
                                            confusion_matrix_scores(df_c ,'y', 'threshold')

    valuation_metric = ((500 * false_negative) + (100 * false_positive))

#     storing valuation_metric scores
    valuation_metric_list.append(valuation_metric)

# https://numpy.org/doc/stable/reference/generated/numpy.argmin.html
```

```
minimun_value_position = np.argmin(valuation_metric_list)

print(f"Minimum 'A' at location {valuation_metric_list[minimun_value_position]} and threshold\
 value at that point is {round(thresholds_list[minimun_value_position],5)}")
```

```
Minimum 'A' at location 141000 and threshold value at that point is 0.22987
```

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 [17]:

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

Out[17]:

	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 [18]:

```
# write your code for task 5d

# 1. Compute Mean Square Error

# Splitting data into 2 lists
act_y = df_d.y
pred_y = df_d.pred
# print('Length of act_y\t\t\t: ', len(act_y))
# print('Length of pred_y\t\t: ', len(pred_y))

diff_act_y_pred_y = act_y - pred_y
# print('Length of diff_act_y_pred_y\t: ',len(diff_act_y_pred_y))

# https://numpy.org/doc/stable/reference/generated/numpy.power.html

mean_square_error = np.mean(np.power(diff_act_y_pred_y, 2))
print('Mean Square Error : ', round(mean_square_error,5))
```

```
Mean Square Error :  177.1657
```

In [19]:

```
# 2. Compute MAPE

# https://numpy.org/doc/stable/reference/generated/numpy.absolute.html

abs_diff_act_y_pred_y = np.absolute(diff_act_y_pred_y)
mean_act_y = np.mean(act_y)

mape_score = np.mean(abs_diff_act_y_pred_y / mean_act_y)

# since it's a percentage error, multiplying by 100
print(f'MAPE error is {round(mape_score * 100, 5)} %')
```

```
MAPE error is 12.91203 %
```

In [20]:

```
# 3. Compute R^2 error: https://en.wikipedia.org/wiki/Coefficient_of_determination#Definitions

residual_sum_square_error = np.power(diff_act_y_pred_y, 2).sum()
total_sum_of_squares = np.power((act_y - mean_act_y),2).sum()

r2_score = 1 - (residual_sum_square_error/total_sum_of_squares)

print('Coefficient of determination OR R^2 value is :', round(r2_score, 5))
```

```
Coefficient of determination OR R^2 value is : 0.95636
```