Compute performance metrics for the given Y and Y_score without sklearn

In [1]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```python
# from google.colab import files
# uploaded = files.upload()
```

In [3]:
```python
import numpy as np
import pandas as pd
from functools import partial
```

In [6]:
```python
src_url = r'/content/drive/MyDrive/AAIML/5. KPI'
```

In [43]:
```python
df_a = pd.read_csv(fr'{src_url}/5_a.csv')
df_b = pd.read_csv(fr'{src_url}/5_b.csv')
df_c = pd.read_csv(fr'{src_url}/5_c.csv')
df_d = pd.read_csv(fr'{src_url}/5_d.csv')
```

In [10]:
```python
df_a.head()
```

Out[10]:

	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 [11]:
```python
df_a['y'].value_counts()
```

Out[11]:
```
1.0    10000
0.0      100
Name: y, dtype: int64
```

Utils ---

In [8]:
```python
# Actual True (Just for Understanding)
at = pd.Series(
    [0, 0],
    index=['pt', 'pf']
)
# Actual False
af = pd.Series(
    [0, 0],
    index=['pt', 'pf']
)

cm = pd.DataFrame({'at': at, 'af':af})
cm
```

Out[8]:

	at	af
pt	0	0
pf	0	0

In [9]:
```python
# # 1. Confusion Matrix  (DEPRECATED)
# # NOTE: Below Code is Iterator based (hence consume a lot time but indeed is memory efficent)
# def calc_confusion_matrix_o1(d):
#     '''
#     Compute confusion matrix for binary classification
#     :param d: dataframe with 2 series ie actual & predicted
#     '''
#     #d = df[['y', 'y_pred']]
#     tp = fp = fn = tn = 0
#     for _, r in d.iterrows():
#         ya, yp = r
#         if ya == yp:
#             if yp :  # True Positive
#                 tp += 1
#             else:    # True Negative
#                 tn += 1
#         else:
#             if yp :  # False Positive
#                 fp += 1
#             else:    # False Negative
#                 fn += 1

#     ans = np.array([[tn, fn], [fp, tp]])
# #     ans
# #     cm = pd.DataFrame(ans)
#     return ans
```

In [13]:
```python
# TEST
# Count only True Elements in Series
np.count_nonzero(df_a['y']==0.0)
```

Out[13]:
```
100
```

In [25]:
```python
# 1. Confusion Matrix
# NOTE: Below Code is Vectorization Based hence it can utilize concurrent programming of sinlge Core (ie Threads) Hence its much faster in terms of Time
def calc_confusion_matrix(d):
    '''
    Compute confusion matrix for binary classification
    :param d: dataframe with 2 series ie actual & predicted
    :param ya: col name for actual y (target)
    :param yp: col name for predicted y (target)
    '''
    #d = df[['y', 'y_pred']]
    ya, yp = d.columns
    tp = np.count_nonzero((d[yp] == 1) & (d[ya] == 1)) # True Positive
    tn = np.count_nonzero((d[yp] == 0) & (d[ya] == 0)) # True Negative
    fp = np.count_nonzero((d[yp] == 1) & (d[ya] == 0)) # False Positive
    fn = np.count_nonzero((d[yp] == 0) & (d[ya] == 1)) # False Negatove

    ans = np.array([[tn, fn], [fp, tp]])
    return ans
```

In [15]:
```python
# 2. F1 Score
def calc_f1_score(d):
    '''
    :param d: dataframe with 2 series ie actual & predicted
    '''
    cm = calc_confusion_matrix(d)
    tp, fp, fn = cm[1, 1], cm[1, 0], cm[0, 1]

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

    f1_score = 2 / (recall**-1 + precision**-1)
    return f1_score
```

In [ ]:
```python
# def get_tpr_fpr(d):
#     '''
#     :param d: dataframe with 2 series ie actual & predicted
#     '''
#     c1 = d.columns[0] # actual column name
#     tp = fp = 0
#     cnts = d[c1].value_counts()

#     an, ap = cnts.get(0, 0), cnts.get(1, 0)
#     for i, r in d.iterrows():

#         ya, yp = r
#         if ya == yp:
#             if yp :  # True Positive
#                 tp += 1
#         else:
#             if yp :  # False Positive
#                 fp += 1

#     # TODO decide What to do
#     tpr = (tp / ap) if ap else 0
#     fpr = (fp / an) if an else 0

#     return tpr, fpr
```

In [33]:
```python
def get_tpr_fpr(d):
    '''
    :param d: dataframe with 2 series ie actual & predicted
    '''
    ca, cp = d.columns # actual-col, pred-col
    cnts = d[ca].value_counts()

    an, ap = cnts.get(0, 0), cnts.get(1, 0) # actual negative, actual positive
    tp = np.count_nonzero((d[cp] == 1) & (d[ca] == 1)) # True Positive
    fp = np.count_nonzero((d[cp] == 1) & (d[ca] == 0)) # False Positive

    # TODO decide What to do
    tpr = (tp / ap) if ap else 0
    fpr = (fp / an) if an else 0

    return tpr, fpr
```

In [17]:
```python
# 3. AUC Score

def calc_auc_score(d):
    '''
    :param d: dataframe with 2 series ie actual & predicted
    '''
    y_act, y_proba, *_ = d.columns

    # 1) Sort values by pred
    d_s = d.sort_values(by=y_proba, ascending=False)
    unique_proba = d_s[y_proba].unique()
    tpr_array = []
    fpr_array = []

    for thresold in unique_proba:
        # 2) compute new y_hat for given {thresold}
        y_hat = np.where(d_s[y_proba] >= thresold, 1, 0)
        df = pd.DataFrame({'y': d_s[y_act], 'y_hat': y_hat})

        # 3) calculate tpr, fpr
        tpr, fpr = get_tpr_fpr(df)

        # 4) store results to resp arrays
        tpr_array.append(tpr)
        fpr_array.append(fpr)

    # calculate AUC score ie Area Under Curve = Integration => by Trapezoidal method
    auc_score = None
    if tpr_array and fpr_array:
        auc_score = np.trapz(tpr_array, fpr_array)

    return auc_score
```

In [18]:
```python
# 4. Accuracy Score

def calc_accuracy_score(df):
    '''
    :param d: dataframe with 2 series ie actual & predicted
    '''
    cm = calc_confusion_matrix(df)
    total = cm.sum()
    true_pred_total = np.diag(cm).sum()
    accuracy = true_pred_total/total
    return accuracy
```

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)
   Note- Make sure that you arrange your probability scores in descending order while calculating AUC

4. Compute Accuracy Score

In [ ]:
```python
# df_a=pd.read_csv('5_a.csv')
# df_a.head()
```

In [19]:
```python
# Approximate the absolute prediction
df_a['y_pred'] = np.where(df_a['proba']<0.5, 0, 1)
df_a
```

Out[19]:

	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
...	...	...	...
10095	1.0	0.665371	1
10096	1.0	0.607961	1
10097	1.0	0.777724	1
10098	1.0	0.846036	1
10099	1.0	0.679507	1

10100 rows × 3 columns

In [ ]:
```python
# df_a = df_a.sample(frac=0.20)
# df_a.shape
```

Out[ ]:
```
(2020, 3)
```

In [30]:
```python
# Confusion Matrix
calc_confusion_matrix(df_a[['y', 'y_pred']])
```

Out[30]:
```
array([[    0,     0],
       [  100, 10000]])
```

In [31]:
```python
# F1 Score
calc_f1_score(df_a[['y', 'y_pred']])
```

Out[31]:
```
0.9950248756218907
```

In [34]:
```python
# aUC Score
calc_auc_score(df_a[['y', 'proba']])
```

Out[34]:
```
0.48829900000000004
```

In [35]:
```python
# accuracy score
calc_accuracy_score(df_a[['y', 'y_pred']])
```

Out[35]:
```
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
   Note- Make sure that you arrange your probability scores in descending order while calculating AUC

4. Compute Accuracy Score

In [36]:
```python
# df_b=pd.read_csv('5_b.csv')
# df_b.head()

df_b['y_pred'] = np.where(df_b['proba']<0.5, 0, 1)
df_b
```

Out[36]:

	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
...	...	...	...
10095	0.0	0.474401	0
10096	0.0	0.128403	0
10097	0.0	0.499331	0
10098	0.0	0.157616	0
10099	0.0	0.296618	0

10100 rows × 3 columns

In [ ]:
```python
# df_b = df_b.sample(frac=0.20)
# df_b.shape
```

In [37]:
```python
# Confusion Matrix
calc_confusion_matrix(df_b[['y', 'y_pred']])
```

Out[37]:
```
array([[9761,   45],
       [ 239,   55]])
```

In [38]:
```python
# F1 Score
calc_f1_score(df_b[['y', 'y_pred']])
```

Out[38]:
```
0.27918781725888325
```

In [39]:
```python
# aUC Score
calc_auc_score(df_b[['y', 'proba']])
```

Out[39]:
```
0.9377570000000001
```

In [40]:
```python
# accuracy score
calc_accuracy_score(df_b[['y', 'y_pred']])
```

Out[40]:
```
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

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 [ ]:
```python
# df_c=pd.read_csv('5_c.csv')
# df_c.head()
```

In [45]:
```python
def calc_best_thresold(d):
    '''
    :param d: dataframe with 2 series ie actual & predicted
    '''
    # def calc_fn_fp_old(d):
    #     '''
    #         THIS is Iterator Based Approach
    #         calculate the false negative & false positive for given dataframe d
    #         :return : tuple :- (fn, fp)
    #     '''
    #     fp = fn = 0
    #     for _, r in d.iterrows():
    #         ya, yp = r
    #         if ya != yp:
    #             if yp :  # False Positive
    #                 fp += 1
    #             else:    # False Negative
    #                 fn += 1

    #     return fn, fp

    def calc_fn_fp(d):
        '''
            THIS is Vectorization Based Approach
            calculate the false negative & false positive for given dataframe d
            :return : tuple :- (fn, fp)
        '''
        ca, cp = d.columns # actual-col, pred-col

        fp = np.count_nonzero((d[cp] == 1) & (d[ca] == 0)) # False Positive
        fn = np.count_nonzero((d[cp] == 0) & (d[ca] == 1)) # False Negative

        return fn, fp

    def calc_metric_A(d, thresold):
        '''
        :param thrsold: to decide new y_hat values
        :param d: dataframe with actua & predicted series
        '''
        # compute new y_hat for given {thresold}
        y_hat = np.where(d[y_proba] >= thresold, 1, 0)
        df = pd.DataFrame({'y': d[y_act], 'y_hat': y_hat})

        # calculate fp & fn
        fn, fp = calc_fn_fp(df)

        return (500 * fn) + (100 * fp)

    y_act, y_proba, *_ = d.columns

    # Sort values by pred
    d_s = d.sort_values(by=y_proba, ascending=False)
    unique_proba = d_s[y_proba].unique()

    # helper method that fixes the data frame as d_s for all thresold calculation
    # so only thresold differs & dataframe itself remains stagnant
    metric_A_helper = partial(calc_metric_A, d_s)

    return min(unique_proba, key = metric_A_helper)
```

In [46]:
```python
calc_best_thresold(df_c[['y', 'prob']])
```

Out[46]:
```
0.2300390278970873
```

In [47]:
```python
calc_auc_score(df_c[['y', 'prob']])
```

Out[47]:
```
0.8288141557331724
```

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 [49]:
```python
#df_d=pd.read_csv('5_d.csv')
df_d.head()
df_d.shape
```

Out[49]:
```
(157200, 2)
```

In [50]:
```python
# 1. Mean Square Error
def mean_sq_err(d):
    y, y_hat = d.columns
    return pow(d[y] - d[y_hat], 2).sum() / d.shape[0]

mean_sq_err(df_d)
```

Out[50]:
```
177.16569974554707
```

In [51]:
```python
# 2. MAPE (Mean Absolute Percentage Error)
def mape(d):
    y, y_hat = d.columns
    err = (d[y_hat] - d[y]).abs()
    return sum(err) / sum(d[y].abs())

mape(df_d)
```

Out[51]:
```
0.1291202994009687
```

In [52]:
```python
def r_square(d):
    y, y_hat = d.columns
    mean = d[y].mean()

    ss_total = sum((d[y_hat] - mean)**2)
    ss_residue = sum((d[y_hat] - d[y])**2)

    return 1 - ss_residue/ss_total

r_square(df_d)
```

Out[52]:
```
0.9544134826849505
```

In [ ]: