Compute performance metrics for the given Y and Y_score without sklearn

udaylunawat@gmail.com

```
import numpy as np
import pandas as pd
from tqdm import tqdm_notebook
```

```
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/My Drive/Appliedai colab/Assignment 5 - Compute performance metrics without sklearn
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.a

Enter your authorization code:
··········
Mounted at /content/drive
/content/drive/My Drive/Appliedai colab/Assignment 5 - Compute performance metrics without sklearn
```

```
# Old and terribly slow code
# def perf_measure(y, y_pred):
#     TP = 0
#     FP = 0
#     TN = 0
#     FN = 0
#     for i in range(len(y_pred)):
#         if y[i]==y_pred[i]==1.0:
#            TP += 1
#         elif y_pred[i]==1.0 and y[i]!=y_pred[i]:
#            FP += 1
#         elif y[i]==y_pred[i]==0:
```

```
#            TN += 1
#         elif y_pred[i]==0 and y[i]!=y_pred[i]:
#            FN += 1
#     return(TP, FP, TN, FN)

#Calculating TP, FP, TN, FN for confusion matrix
#Using np.sum and np.logical as it improves performance drastically
def perf_measure(y, y_pred):
    TP = np.sum(np.logical_and(y_pred == 1, y == 1))
    FP = np.sum(np.logical_and(y_pred == 1, y == 0))
    TN = np.sum(np.logical_and(y_pred == 0, y == 0))
    FN = np.sum(np.logical_and(y_pred == 0, y == 1))
    return TP,FP,TN,FN

#Inspired by -https://webcache.googleusercontent.com/search?q=cache:i-sKM7SjAKAJ:https://kawahara.ca/how-to-compute-truefalse-po

neg_score = {} # For computing and storing A in Ques. c
def AUC(data):
    #Storing list of FPR and TPR used in a and b
    list_FPR = []
    list_TPR = []
    for i in tqdm_notebook(data['proba']): #Used tqdm to measure performance
        t = [0 if x<i else 1 for x in data['proba']] #Creating Taoo columns using thresholds one by one
        TP, FP, TN, FN = perf_measure(data['y'],np.array(t)) #Confusion matri values
        P = TP+FN
        N = TN+FP
        TPR = TP/P
        FPR = FP/N
        A = (500*FN) + (100*FP) #For Ques c
        neg_score.update({i:A}) #Dict with Taoo thresholds and their custom scores A as value
        list_FPR.append(FPR)
        list_TPR.append(TPR)
    TPR_ARR = np.array(list_TPR)
    FPR_ARR = np.array(list_FPR)
    AUC_sc = np.trapz(TPR_ARR,FPR_ARR) #Calculating AUC
    return AUC_sc, list_FPR, list_TPR
```

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
4. Compute Accuracy Score

```
a = pd.read_csv('5_a.csv')
print("No. of positive points: ",a['y'].value_counts()[1])
print("No. of negative points: ",a['y'].value_counts()[0])
a.head()
```

```
No. of positive points:  10000
No. of negative points:  100
```

	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

Highly unbalanced dataset, hence default AUC score can be invalidated

Confusion matrix and accuracy score

```
#Creating a y_pred column which uses ypred=[0 if y_score < 0.5 else 1]
a['y_pred'] = [0 if x<0.5 else 1  for x in a['proba']]

#Computing and printing confusion matrix

TP, FP, TN, FN = perf_measure(np.array(a['y']),np.array(a['y_pred']))
x = np.array([[TN, FP,],
               [FN, TP]])
row_labels = ['Actual NO','Actual YES']
column_labels = ['Predicted NO','Predicted YES']
df = pd.DataFrame(x, columns=column_labels, index=row_labels)
print('Confusion matrix\n\n',df)
# print(df)
print('\nAccuracy =',(TP+TN)/len(a))
```

```
Confusion matrix

             Predicted NO  Predicted YES
Actual NO               0            100
Actual YES              0          10000

Accuracy = 0.9900990099009901
```

```
#calculating F1 score
f1_score = 2*TP/(2*TP + FP + FN)
print('F1 score',f1_score)
```

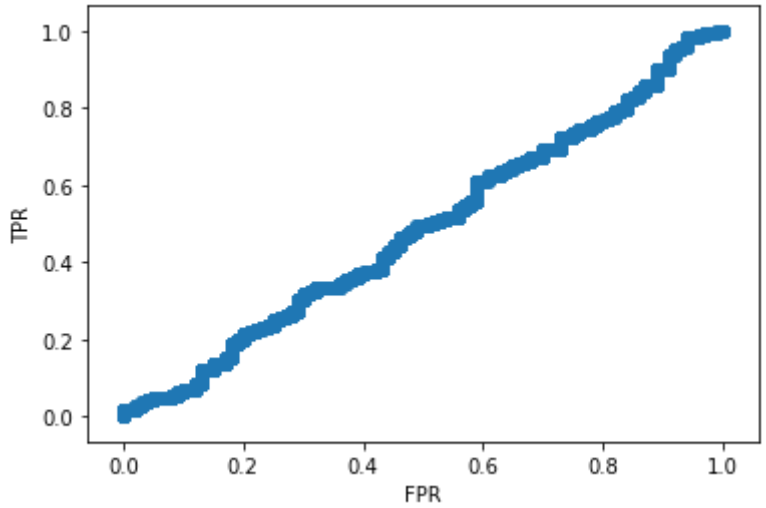
```
F1 score 0.9950248756218906
```

```
#Sorting dataframe a in descending order of proba
a.sort_values(by = 'proba',ascending=False, inplace = True)
```

```
AUC_score, FPR_list, TPR_list = AUC(a)
```

```
100% 10100/10100 [00:48<00:00, 207.33it/s]
```

```
import matplotlib.pyplot as plt
plt.scatter(FPR_list,TPR_list);
plt.xlabel("FPR");
plt.ylabel("TPR");
```




```
print("AUC_score of a : ",AUC_score)
```

```
AUC_score of a :  0.48829900000000004
```

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

4.  Compute Accuracy Score
```

```
b = pd.read_csv("5_b.csv")
print("No. of positive points: ",b['y'].value_counts()[1])
print("No. of negative points: ",b['y'].value_counts()[0])
#Creating a y_pred column which uses ypred=[0 if y_score < 0.5 else 1]
b['y_pred'] = [0 if x<0.5 else 1 for x in b['proba']]
b.head()
```

```
No. of positive points:  100
No. of negative points:  10000
```

	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

Highly unbalanced dataset, hence default AUC score can be invalidated

```
#Computing and printing confusion matrix
TP, FP, TN, FN = perf_measure(b['y'],b['y_pred'])
x = np.array([[TN, FP,],
              [FN, TP]])
row_labels = ['Actual NO','Actual YES']
```

```
column_labels = ['Predicted NO','Predicted YES']
df = pd.DataFrame(x, columns=column_labels, index=row_labels)
print('\n\nConfusion matrix\n', df)
print('\nAccuracy =',(TP+TN)/len(a))
```

```
Confusion matrix
             Predicted NO  Predicted YES
Actual NO            9761            239
Actual YES             45             55

Accuracy = 0.9718811881188119
```

```
#Calculating F1 score
f1_score = 2*TP/(2*TP + FP + FN)
print('F1 score',f1_score)
```

```
F1 score 0.27918781725888325
```

Pretty low AUC score, hence an insensible model.

```
#Sorting dataframe a in descending order of proba
b.sort_values(by = 'proba',ascending=False, inplace = True)
```

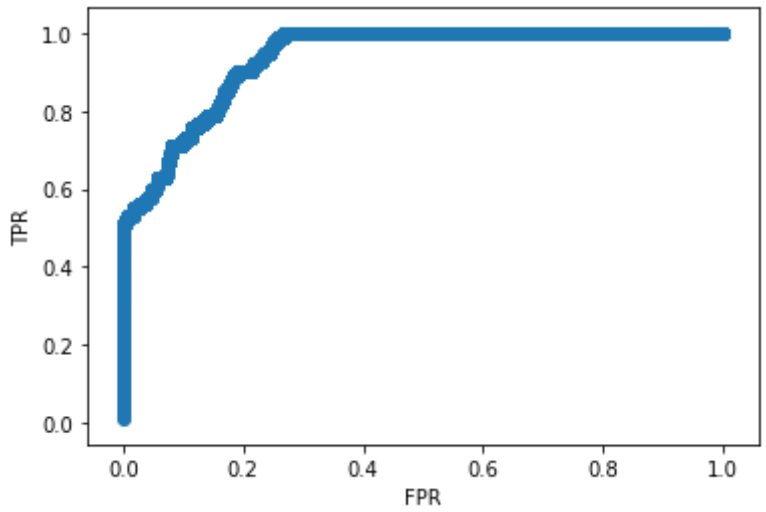
```
AUC_score, FPR_list, TPR_list = AUC(b)
```

100% 10100/10100 [00:46<00:00, 215.57it/s]

```
b['y'].value_counts()
```

```
0.0    10000
1.0      100
Name: y, dtype: int64
```

```
import matplotlib.pyplot as plt
plt.scatter(FPR_list,TPR_list);
plt.xlabel("FPR");
plt.ylabel("TPR");
```




```
print("AUC_score of b : ",AUC_score)
```

```
AUC_score of b :  0.9377570000000001
```

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**

```
c = pd.read_csv("5_c.csv")
```

```
#Renaming column prob to proba to match the already written function
c.rename({'prob':'proba'},inplace=True,axis = 1)
c.head()
```

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

```
print("No. of positive points: ",c['y'].value_counts()[0])
print("No. of negative points: ",c['y'].value_counts()[1])
```

```
No. of positive points:  1805
No. of negative points:  1047
```

```
#Computing and printing confusion matrix
c['y_pred'] = [0 if x<0.5 else 1 for x in c['proba']]
TP, FP, TN, FN = perf_measure(c['y'],c['y_pred'])
x = np.array([[TN, FP,],
              [FN, TP]])
row_labels = ['Actual NO','Actual YES']
column_labels = ['Predicted NO','Predicted YES']
df = pd.DataFrame(x, columns=column_labels, index=row_labels)
print('\n\nConfusion matrix\n', df)
print('\nAccuracy =',(TP+TN)/len(c))
```

```
Confusion matrix
             Predicted NO  Predicted YES
Actual NO            1637            168
Actual YES            462            585

Accuracy = 0.7791023842917251
```

Quite unbalanced dataset and we can see that the ratio of FP:FN is 1:5. Hence we're balancing it by calculating

A = 500 * FP + 100 * FN

So, we get out best threshold that minimizes A

```
#Sorting df by column proba
c.sort_values(by = 'proba',ascending=False, inplace = True)
```

```
#Generating FPR and TPR list, and calculating AUC score
#these are not important for this question but our function also generates a dictionary with the needed metric score
AUC_score, FPR_list, TPR_list = AUC(c)
```

100% 2852/2852 [00:08<00:00, 325.56it/s]

```
s_c = sorted(neg_score.items(), key = lambda kv:(kv[1]),reverse = False)
print("Best threshold: {:.2f}".format(s_c[0][0]))
```

```
Best threshold: 0.23
```

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

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

	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 average squared difference between the estimated values and what is estimated
MSE = sum([(i-j)**2 for i,j in zip(d['y'],d['pred'])])/len(d)
print("Mean Sqaured Error :",MSE)
```

```
Mean Sqaured Error : 177.16569974554707
```

```
e = sum([abs(j-i) for i,j in zip(d['y'],d['pred'])])
mean_y = sum(d['y']) / len(d)
MAPE = (e/mean_y)/len(d)
```

```
print("MAPE :",MAPE)
```

```
MAPE : 0.1291202994009687
```

```
res_ss = sum([(i-j)**2 for i,j in zip(d['y'],d['pred'])])
tot_ss = sum([(i-mean_y)**2 for i in d['y']])
r2 = 1 - (res_ss / tot_ss)
print("R^2 error : ",r2)
```

```
R^2 error :  0.9563582786990964
```

```

```