Performance_Metrics_Assignment

April 9, 2020

1 A. Performance Metrics for Data 5_a.csv

[1]:
```
import numpy as np
import pandas as pd
```

[2]:
```
data=pd.read_csv("5_a.csv") #loading Dataset
data.head()
```

[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
```

[3]:
```
data["y"].value_counts()
```

[3]:
```
1.0    10000
0.0      100
Name: y, dtype: int64
```

[4]:
```
data["y_predict"]=(data["proba"]>=0.5)*1 #predicting values for Label "y"
data.head()
```

[4]:
```
     y     proba  y_predict
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
```

[5]:
```
actual_values=np.array(data["y"]) #actual values of label "y"
predicted_values=np.array(data["y_predict"]) #predicted values of label "y"
```

[6]:
```
#function to calculate confusion matrix using actual and predicted values of "y"

def cal_confusion_matrix(actual_values,predicted_values):
```

```
    confusion_matrix=np.zeros((2,2),dtype="int")
    n=len(actual_values)
    for i in range(n):
        if actual_values[i]==0 and predicted_values[i]==0: #finding count of␣
 ↪true negative
            confusion_matrix[0,0]+=1
        elif actual_values[i]==1 and predicted_values[i]==0: #count of false␣
 ↪negative
            confusion_matrix[0,1]+=1
        elif actual_values[i]==0 and predicted_values[i]==1: #count of false␣
 ↪postive
            confusion_matrix[1,0]+=1
        else:                              #count of true postive
            confusion_matrix[1,1]+=1
    return confusion_matrix
```

[7]:
```
confusion_matrix=cal_confusion_matrix(actual_values,predicted_values)
print("confusion matrix is")
print(confusion_matrix)
```

```
confusion matrix is
[[    0     0]
 [  100 10000]]
```

[8]:
```
TN=confusion_matrix[0,0] # values of TN,FN,FP,TP
FN=confusion_matrix[0,1] # from confusion matrix
FP=confusion_matrix[1,0]
TP=confusion_matrix[1,1]

P=TP+FN #total count of postives
N=FP+TN #total count of negatives


precision=TP/(TP+FP) #precision formula
#print(precision)

recall=TP/(TP+FN) #recall formula
#print(recall)

F1_Score=2*precision*recall/(precision+recall) #calculating F1_SCORE using␣
 ↪precision and recall
print("F1 Score is {}".format(F1_Score))
```

```
F1 Score is 0.9950248756218906
```

[9]:
```
proba=np.array(data["proba"].sort_values(ascending=False)) #sorting probability␣
 ↪values in descending order
```

```
#print(proba)

length=len(proba)
tpr_values=[]
fpr_values=[]
y_values=np.array(data["y"])
for i in range(length):
    predict_values=np.array((data["proba"]>=proba[i])*1)        #predicted␣
↪values of "y" using
    conf_matrix=cal_confusion_matrix(y_values,predict_values)   # different␣
↪thresolds
    TN_V=conf_matrix[0,0]
    FN_V=conf_matrix[0,1]
    FP_V=conf_matrix[1,0]
    TP_V=conf_matrix[1,1]
    P=TP_V+FN_V
    N=FP_V+TN_V
    tpr=TP_V/P
    fpr=FP_V/N
    tpr_values.append(tpr)
    fpr_values.append(fpr)
tpr_array=np.array(tpr_values) #TPR values for different thresolds
fpr_array=np.array(fpr_values) #FPR values for different thresolds

AUC_Score=np.trapz(tpr_array, fpr_array) #calculating AUC Using TPR AND FPR␣
↪Values
print("AUC Score is {} ".format(AUC_Score))
```

```
AUC Score is 0.48829900000000004
```

[10]:
```
accuracy_score=(TP+TN)/(TP+FP+TN+FN) #calculating accuracy score
print("Accuracy Score is {}".format(accuracy_score))
```

```
Accuracy Score is 0.9900990099009901
```

2 B. Performance Metrics for Data 5_b.csv

[11]:
```
import numpy as np
import pandas as pd
```

[12]:
```
data=pd.read_csv("5_b.csv") #loading Dataset
data.head()
```

[12]:

	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
```

[13]:
```
data["y"].value_counts()
```

[13]:
```
0.0    10000
1.0      100
Name: y, dtype: int64
```

[14]:
```
data["y_predict"]=(data["proba"]>=0.5)*1 #predicting values for Label "y"
data.head()
```

[14]:
```
     y     proba  y_predict
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
```

[15]:
```
actual_values=np.array(data["y"]) #actual values of label "y"
#print(actual_values)
predicted_values=np.array(data["y_predict"]) #predicted values of label "y"
#print(predicted_values)
```

[16]:
```
#function to calculate confusion matrix using actual and predicted values of "y"

def cal_confusion_matrix(actual_values,predicted_values):
    confusion_matrix=np.zeros((2,2),dtype="int")
    n=len(actual_values)
    for i in range(n):
        if actual_values[i]==0 and predicted_values[i]==0:   #count of true␣
 ↪negative
            confusion_matrix[0,0]+=1
        elif actual_values[i]==1 and predicted_values[i]==0:   #count of false␣
 ↪negative
            confusion_matrix[0,1]+=1
        elif actual_values[i]==0 and predicted_values[i]==1:   #countpf false␣
 ↪postive
            confusion_matrix[1,0]+=1
        else:                                  #count of true postive
            confusion_matrix[1,1]+=1
    return confusion_matrix
```

[17]:
```
confusion_matrix=cal_confusion_matrix(actual_values,predicted_values)
print("confusion matrix is")
print(confusion_matrix)
```

```
confusion matrix is
[[9761   45]
 [ 239   55]]
```

[18]:
```
TN=confusion_matrix[0,0] # values of TN,FN,FP,TP
FN=confusion_matrix[0,1] # from confusion matrix
FP=confusion_matrix[1,0]
TP=confusion_matrix[1,1]

P=TP+FN #total postives
N=FP+TN #total negatives


precision=TP/(TP+FP) #precision formula
#print(precision)

recall=TP/(TP+FN) #recall formula
#print(recall)

F1_Score=2*precision*recall/(precision+recall) #calculating F1_SCORE using␣
↪precision and recall
print("F1 Score is {}".format(F1_Score))
```

```
F1 Score is 0.2791878172588833
```

[19]:
```
proba=np.array(data["proba"].sort_values(ascending=False))  #sorting␣
↪probability values in descending order
#print(proba)

length=len(proba)
tpr_values=[]
fpr_values=[]
y_values=np.array(data["y"])
for i in range(length):
    predict_values=np.array((data["proba"]>=proba[i])*1)         #predicted␣
↪values of "y" using
    conf_matrix=cal_confusion_matrix(y_values,predict_values)    # different␣
↪thresolds
    TN_V=conf_matrix[0,0]
    FN_V=conf_matrix[0,1]
    FP_V=conf_matrix[1,0]
    TP_V=conf_matrix[1,1]
    P=TP_V+FN_V
    N=FP_V+TN_V
    tpr=TP_V/P
    fpr=FP_V/N
    tpr_values.append(tpr)
```

```
    fpr_values.append(fpr)
tpr_array=np.array(tpr_values)   #TPR values for different thresolds
fpr_array=np.array(fpr_values)    #FPR values for different thresolds

AUC_Score=np.trapz(tpr_array, fpr_array)  #calculating AUC Using TPR AND FPR␣
 ↪Values
print("AUC Score is {} ".format(AUC_Score))
```

```
AUC Score is 0.9377570000000001
```

[20]:
```
accuracy_score=(TP+TN)/(TP+FP+TN+FN) #calculating accuracy score
print("Accuracy Score is {}".format(accuracy_score))
```

```
Accuracy Score is 0.9718811881188119
```

3 C. Best Thresold of Metric A for Data 5_c.csv

[21]:
```
import numpy as np
import pandas as pd
```

[22]:
```
data=pd.read_csv("5_c.csv")   #loading dataset
data.head()
```

[22]:

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

[23]:
```
data["y"].value_counts()
```

[23]:
```
0    1805
1    1047
Name: y, dtype: int64
```

[24]:
```
#function to calculate confusion matrix using actual and predicted values of "y"

def cal_confusion_matrix(actual_values,predicted_values):
    confusion_matrix=np.zeros((2,2),dtype="int")
    n=len(actual_values)
    for i in range(n):
        if actual_values[i]==0 and predicted_values[i]==0:    #count of true␣
 ↪negative
            confusion_matrix[0,0]+=1
```

```
        elif actual_values[i]==1 and predicted_values[i]==0:   #fcount of alse␣
↪negative
            confusion_matrix[0,1]+=1
        elif actual_values[i]==0 and predicted_values[i]==1:  # count of false␣
↪postive
            confusion_matrix[1,0]+=1
        else:                                  # count of true postive
            confusion_matrix[1,1]+=1
    return confusion_matrix
```

[25]:
```
proba=np.array(data["prob"].sort_values(ascending=False))  #sorting probability␣
↪values in descending order
#print(proba)

length=len(proba)
A_metric=[]
y_values=np.array(data["y"]) #actual values of label "y"
for i in range(length):
    predict_values=np.array((data["prob"]>=proba[i])*1)         #predicted␣
↪values of "y" using
    conf_matrix=cal_confusion_matrix(y_values,predict_values)    # different␣
↪thresolds
    FN=conf_matrix[0,1]
    FP=conf_matrix[1,0]
    A=500*FN+100*FP         # calculating A metrices
    A_metric.append(A)
A_array=np.array(A_metric)
```

[26]:
```
A_min=A_array.min()
A_min_index=A_array.argmin()  #getting index of minimum value of A
#print(A_min)
#print(A_min_index)

print("Best threshold with lowest value of metric A is")
print(proba[A_min_index]) ##getting best thresold value from index A_min_index
```

```
Best threshold with lowest value of metric A is
0.2300390278970873
```

4 D. Performance Metrics(for regression) for Data 5_d.csv

[27]:
```
import numpy as np
import pandas as pd
```

[28]:
```
data=pd.read_csv("5_d.csv")   # loading dataset
data.head()
```

[28]:

	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

[29]:
```
y=np.array(data["y"])          # given actual values of "y"
y_cap=np.array(data["pred"])  #given predicted values of "y"

Mean_Square_Error=np.mean(np.square(y-y_cap))  #calculating Mean Square Error
print("Mean Square Error is {}".format(Mean_Square_Error))
```

```
Mean Square Error is 177.16569974554707
```

[30]:
```
sum_absolute_errors=np.sum((np.absolute(y-y_cap)))  #sum of absolute errors
sum_actual_values=np.sum(y)                          #sum of actual values of␣
 ↪"y"

MAPE=(sum_absolute_errors/sum_actual_values)*100     #mean absolute pecentage␣
 ↪error formula
print("Mean Absolute Percentage Error is {}".format(MAPE))
```

```
Mean Absolute Percentage Error is 12.91202994009687
```

[31]:
```
total_sum_of_squares=np.sum(np.square(y-np.mean(y)))   #total sum of squares␣
 ↪SS_tot
sum_of_squares_of_residuals=np.sum(np.square(y-y_cap))  #sum of squares of␣
 ↪residuals SS_res

R2_Error=1-(sum_of_squares_of_residuals/total_sum_of_squares) ␣
 ↪#Rsquare_Error=1-(SS_res/SS_total)
print("R^2 Error is")
print(R2_Error)
```

```
R^2 Error is
0.9563582786990937
```

[ ]:
```

```