Performance Metrics

February 19, 2021

1 Compute performance metrics for the given Y and Y_score without sklearn

[1]:
```
# Importing packages
import numpy as np
import pandas as pd
```

[2]:
```
def get_conf_mat(df):
    """
    This function takes the dataset as the input and returns a 2x2.
    True Negative  - [0,0]
    False Negative - [0,1]
    False Positive - [1,0]
    True Positive  - [1,1]
    """
    # Initializing an empty array
    conf_mat = [[0,0],[0,0]]
    # Iterative over the values of dataset and filling in the values based on␣
 ↪the conditions
    for i in range(len(df)):
        if(df[i][0] == 1 and df[i][2] == 1):
            conf_mat[1][1] += 1
        elif(df[i][0] == 1 and df[i][2] == 0):
            conf_mat[0][1] += 1
        elif(df[i][0] == 0 and df[i][2] == 1):
            conf_mat[1][0] += 1
        elif(df[i][0] == 0 and df[i][2] == 0):
            conf_mat[0][0] += 1
    return conf_mat
```

[3]:
```
def get_auc(df, tot_neg, tot_pos, thresholds):
    """
    This function takes the dataset, total number of negative points, total␣
 ↪number of positive points and the unique
    threshold values as an input and returns the Area Under Curve Score of the␣
 ↪dataset.
    """
```

```
    # Creating two empty arrays to store the Total Positive Rate(TPR) and the␣
↪False Positive Rate(FPR)
    tpr, fpr = [],[]
    # Iterating over the unique threshold values
    for threshold in thresholds:
        # Counting the number of false positives and true positives.
        # Initializing them to 0 each iteration
        fp_temp, tp_temp = 0,0
        # Iterating over the whole dataset
        for i in range(len(df)):
            # If the probability is >= threshold i.e., Model predicts 1
            if(df[i][1] >= threshold):
                # If the actual value is 1, ie., It's a True Positive.␣
↪Incrementing it's value
                if(df[i][0] == 1):
                    tp_temp += 1
                # Else if the actual value is 0, i.e., It's a False Positive.␣
↪Incrementing it's value
                elif(df[i][0] == 0):
                    fp_temp += 1
            # We don't need the False Negatives and True Negatives.
            # If the probability is < threshold., we can just break and exit␣
↪the for loop.
            # Before breaking, calculating the FPR and TPR values and appending␣
↪them to the list.
            elif(df[i][1] < threshold):
                fpr.append(fp_temp/tot_neg)
                tpr.append(tp_temp/tot_pos)
                break
    # Calculating the AUC Score using the TPR and FPR lists.
    # Before calculating, reversing the lists as the default order returns␣
↪negative AUC Score
    fpr = fpr[::-1]
    tpr = tpr[::-1]
    return np.trapz(tpr, fpr)
```

[4]:
```
def get_best_threshold(df, thresholds):
    """
    This function takes the dataset and returns the threshold that gives the␣
↪lowest metric.
    """
    # lowest_a will store the lowest_a value calculated so far. Initially it's␣
↪going to be -1
    lowest_a = -1
    # best_threshold will store the threshold value that produced the lowest_a
    best_threshold = 0
```

```
    # Iterative over all the thresholds
    for threshold in thresholds:
        # Storing the values of the confusion matrix at each iteration
        # a_temp stores the current A value
        fp_temp, tp_temp, fn_temp, tn_temp, a_temp = 0,0,0,0,0
        # Iterating over the dataset
        for i in range(len(df)):
            # If the probability is >= threshold ie., Model predicts 1
            if(df[i][1] >= threshold):
                # If the actual value is also 1, incrementing the value of True␣
↪Positive
                if(df[i][0] == 1):
                    tp_temp += 1
                # If the actual value is 0, incrementing the value of False␣
↪Positive
                elif(df[i][0] == 0):
                    fp_temp += 1
            # If the probability is < threshold ie., Model predicts 0
            elif(df[i][1] < threshold):
                # If the actual value is 1, incrementing the value of False␣
↪Negative
                if(df[i][0] == 1):
                    fn_temp += 1
                # If the actual value is alse 0, incrementing the value of True␣
↪Negative
                elif(df[i][0] == 0):
                    tn_temp += 1
        # Calculating the value of a
        a_temp = (500*fn_temp)+(100*fp_temp)
        # If the value of a is -1 ie., this is the first iteration
        if(lowest_a == -1):
            # Storing the value of a in lowest_a
            lowest_a = a_temp
        # Else if the value of a is less than lowest_a
        elif(a_temp <= lowest_a):
            # Storing the value of a in lowest_a
            lowest_a = a_temp
            # Storing the threshold in best_threshold
            best_threshold = threshold
    # Returning the best_threshold
    return best_threshold
```

[22]:
```
# Importing the dataset using numpy
df_5a = np.genfromtxt('./Datasets/5_a.csv', delimiter=',')
# Removing the first row of the dataset (names of columns)
df_5a = np.delete(df_5a, 0, 0)
```

```
# Making forecasts based on the probability scores and storing them in a preds␣
↪array
preds_5a = np.array([1 if df_5a[i][1] >= 0.5 else 0 for i in range(len(df_5a))])
# Adding the predictions array to the dataset as a new column
df_5a = np.c_[df_5a, preds_5a]
```

[6]:
```
# Using the get_conf_mat to get the Confusion Matrix
conf_mat_5a = get_conf_mat(df_5a)
# Calculating the Precision by dividing the True Positive by sum of True and␣
↪False positives
precision_5a = conf_mat_5a[1][1]/(conf_mat_5a[1][0]+conf_mat_5a[1][1])
# Calculating the Recall by dividing the True Positive by sum of False Negative␣
↪and True Positive
recall_5a = conf_mat_5a[1][1]/(conf_mat_5a[0][1] + conf_mat_5a[1][1])
# Calculating the F1 score by taking the Harmonic Mean of Precision and Recall
f1_5a = 2*(precision_5a*recall_5a)/(precision_5a+recall_5a)
# Printing the F1 scores and the confusion Matrix
print(f"The Confusion Matrix is {conf_mat_5a}")
print(f"The F1 Score is {f1_5a}")
```

```
The Confusion Matrix is [[0, 0], [100, 10000]]
The F1 Score is 0.9950248756218906
```

[7]:
```
# Sorting the dataset by the probabilites
df_5a_sorted = df_5a[df_5a[:,1].argsort()][::-1]
# Getting the unique probabilites and storing them in a thresholds list
thresholds_5a = np.unique(df_5a_sorted[:,1])
# Getting the total number of Positive and Negative points by summing up the␣
↪columns of the Confusion Matrix
tot_neg_5a, tot_pos_5a  = conf_mat_5a[0][0]+conf_mat_5a[1][0],␣
↪conf_mat_5a[0][1]+conf_mat_5a[1][1]
```

[8]:
```
# Getting the AUC Score using the get_auc function
auc_5a = get_auc(df_5a_sorted, tot_neg_5a, tot_pos_5a, thresholds_5a)
print(f"The AUC Score is {auc_5a}")
```

```
The AUC Score is 0.48829900000000004
```

[9]:
```
# Calculating the Accuracy
accuracy_5a = (conf_mat_5a[0][0]+conf_mat_5a[1][1])/
↪(conf_mat_5a[0][0]+conf_mat_5a[0][1]+conf_mat_5a[1][0]+conf_mat_5a[1][1])
print(f"The accuracy of the dataset is {accuracy_5a}")
```

```
The accuracy of the dataset is 0.9900990099009901
```

[10]:
```
# Importing the dataset using numpy
df_5b = np.genfromtxt('./Datasets/5_b.csv', delimiter=',')
# Removing the first row of the dataset (names of columns)
```

```
df_5b = np.delete(df_5b, 0, 0)
# Making forecasts based on the probability scores and storing them in a preds␣
 ↪array
preds_5b = np.array([1 if df_5b[i][1] >= 0.5 else 0 for i in range(len(df_5b))])
# Adding the predictions array to the dataset as a new column
df_5b = np.c_[df_5b, preds_5b]
```

[11]:
```
# Using the get_conf_mat to get the Confusion Matrix
conf_mat_5b = get_conf_mat(df_5b)
# Calculating the Precision by dividing the True Positive by sum of True and␣
 ↪False positives
precision_5b = conf_mat_5b[1][1]/(conf_mat_5b[1][0]+conf_mat_5b[1][1])
# Calculating the Recall by dividing the True Positive by sum of False Negative␣
 ↪and True Positive
recall_5b = conf_mat_5b[1][1]/(conf_mat_5b[0][1] + conf_mat_5b[1][1])
# Calculating the F1 score by taking the Harmonic Mean of Precision and Recall
f1_5b = 2*(precision_5b*recall_5b)/(precision_5b+recall_5b)
# Printing the F1 scores and the confusion Matrix
print(f"The Confusion Matrix is {conf_mat_5b}")
print(f"The F1 Score is {f1_5b}")
```

```
The Confusion Matrix is [[9761, 45], [239, 55]]
The F1 Score is 0.2791878172588833
```

[12]:
```
# Sorting the dataset by the probabilites
df_5b_sorted = df_5b[df_5b[:,1].argsort()][::-1]
# Getting the unique probabilites and storing them in a thresholds list
thresholds_5b = np.unique(df_5b_sorted[:,1])
# Getting the total number of Positive and Negative points by summing up the␣
 ↪columns of the Confusion Matrix
tot_neg_5b, tot_pos_5b  = conf_mat_5b[0][0]+conf_mat_5b[1][0],␣
 ↪conf_mat_5b[0][1]+conf_mat_5b[1][1]
```

[13]:
```
# Getting the AUC Score using the get_auc function
auc_5b = get_auc(df_5b_sorted, tot_neg_5b, tot_pos_5b, thresholds_5b)
print(f"The AUC Score is {auc_5b}")
```

```
The AUC Score is 0.9376570000000001
```

[14]:
```
# Calculating the Accuracy
accuracy_5b = (conf_mat_5b[0][0]+conf_mat_5b[1][1])/
 ↪(conf_mat_5b[0][0]+conf_mat_5b[0][1]+conf_mat_5b[1][0]+conf_mat_5b[1][1])
print(f"The accuracy of the dataset is {accuracy_5b}")
```

```
The accuracy of the dataset is 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

[15]:
```
# Importing the dataset using numpy
df_5c = np.genfromtxt('./Datasets/5_c.csv', delimiter=',')
# Removing the first row of the dataset (names of columns)
df_5c = np.delete(df_5c, 0, 0)
# Making forecasts based on the probability scores and storing them in a preds␣
 ↪array
preds_5c = np.array([1 if df_5c[i][1] >= 0.5 else 0 for i in range(len(df_5c))])
# Adding the predictions array to the dataset as a new column
df_5c = np.c_[df_5c, preds_5c]
```

[16]:
```
# Using the get_conf_mat to get the Confusion Matrix
conf_mat_5c = get_conf_mat(df_5c)
# Calculating the Precision by dividing the True Positive by sum of True and␣
 ↪False positives
df_5c_sorted = df_5c[df_5c[:,1].argsort()][::-1]
# Calculating the Recall by dividing the True Positive by sum of False Negative␣
 ↪and True Positive
thresholds_5c = np.unique(df_5c_sorted[:,1])
# Calculating the F1 score by taking the Harmonic Mean of Precision and Recall
tot_neg_5c, tot_pos_5c  = conf_mat_5c[0][0]+conf_mat_5c[1][0],␣
 ↪conf_mat_5c[0][1]+conf_mat_5c[1][1]
```

[17]:
```
# Using the get_best_threshold to get the best threshold value resposnsible for␣
 ↪lowest A
best_t_5c = get_best_threshold(df_5c_sorted, thresholds_5c)
```

[18]:
```
best_t_5c
```

[18]:
```
0.2300390278970873
```

[19]:
```
# Importing the dataset using numpy
df_5d = np.genfromtxt('./Datasets/5_d.csv', delimiter=',')
# Removing the first row of the dataset (names of columns)
df_5d = np.delete(df_5d, 0, 0)
```

[20]:
```
# Storing the mean of actual values in mean_y
mean_y = np.mean(df_5d[:,0])
# Initializing the values of ss_total, ss_residual and MAPE to 0
ss_total, ss_residual, mape, mse = 0, 0, 0, 0
# Iterating over the dataset
for i in range(len(df_5d)):
    # Calculating the ss_total, ss_residual and mape numerator
    ss_total += (df_5d[i][0] - mean_y)**2
    ss_residual += (df_5d[i][0] - df_5d[i][1])**2
    mape += np.abs(df_5d[i][0] - df_5d[i][1])
# Calculating the MSE, MAPE and R^2 values
mse = ss_residual/len(df_5d)
```

```
mape /= np.sum(df_5d[:,0])
r_squared = 1-(ss_residual/ss_total)
```

[21]:
```
# Printing obtained values
print(f"1. MSE : {mse}\n2. MAPE : {mape}\n3. R^2 : {r_squared}")
```

```
1. MSE : 177.16569974554707
2. MAPE : 0.1291202994009687
3. R^2 : 0.9563582786990964
```

[ ]:
```

```