Activity_5

October 28, 2023

1 Team Number: 7

2 Team Captain: Jacob Silva

3 Team: Juliana Steele, Joel Hurtado

3.1 1. Read the CSV file "Microsoft_Results.CSV".

[4]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

data = pd.read_csv('Microsoft_Results.csv')
data
```

[4]:

	MACHINEID	TRUE_STATUS	PRED_Probability
0	1	0	0.626452
1	2	0	0.239536
2	3	0	0.504254
3	4	1	0.623444
4	5	1	0.546973
…	…	…	…
999995	999996	0	0.302164
999996	999997	0	0.419462
999997	999998	1	0.336044
999998	999999	1	0.490187
999999	1000000	1	0.405500

[1000000 rows x 3 columns]

4 2. Write a program to create a new file "Microsoft_Ranked" which contains an additional column "Rank" based on PRED_Probability. The case with smaller PRED_probability has smaller rank.

[5]:
```
data['Rank'] = data['PRED_Probability'].rank()

new_csv_file_path = 'Microsoft_Ranked.csv'

data.to_csv(new_csv_file_path, index=False)
```

[6]:
```
# Run this to get the directory it saves to in google collab. Use the file tab
 ↪to the left.
!pwd
```

```
/content
```

5 3. Write a program (or function) to calculate the following statistics: "True Positive", "False Positive", "True Negative", "False Negative", "Sensitivity", "Specificity", "Accuracy", and "Precision" for any given cut-off probability (i.e., "PRED_Probability") using the data set "Microsoft_Results.CSV". The output data set has nine variables including "Cut off Probability", "True Positive", "False Positive", "True Negative", "False Negative", "Sensitivity", "Specificity", "Accuracy", and "Precision". The input of your program (or function) has one input data file name and the cut-off probability. The output of your program is a output file with all desired statistics.

[7]:
```
df = pd.DataFrame()

output_file = open('output.txt', 'w')

def evaluate_model(threshold, true_labels, predicted_probabilities,
 ↪column_name):

    df[column_name] = (predicted_probabilities >= threshold).astype(int)

    conf_matrix = confusion_matrix(true_labels, df[column_name], labels=[1, 0])

    # Print the confusion matrix
    print("Confusion Matrix:")
```

```
    print(conf_matrix)
    print()

    # true positive, false positive, true negative, false negative
    TP = conf_matrix[0, 0]
    FP = conf_matrix[0, 1]
    TN = conf_matrix[1, 1]
    FN = conf_matrix[1, 0]

    # true positive rate, false positive rate
    TPR = TP / (TP + FN)
    TNR = TN / (TN + FP)

    print("True-Positive: ", TP, "\nFalse-Positive: ", FP, "\nTrue-Negative: ",␣
 ↪TN, "\nFalse-Negative: ", FN)

    print("\nSensitivity: ", round(TPR, 3), "\nSpecificity: ", round(TNR, 3))

    # Correctly predicted cases to the total number of cases
    accuracy = (TP + TN) / (TP + TN + FP + FN)

    print("Accuracy: ", round(accuracy, 3))

    # Measure of model's ability to make positive predictions correctly
    precision = TP / (TP + FP)

    print("Precision: ", round(precision, 3), "\n")

    # output file
    output_file.write("True-Positive: " + str(TP) +  "\nFalse-Positive: " +␣
 ↪str(FP) + "\nTrue-Negative: " + str(TN) + "\nFalse-Negative: " + str(FN))
    output_file.write("\nSensitivity: " + str(round(TPR, 3)) + "\n")
    output_file.write("Specificity: " + str(round(TNR, 3)) + "\n")
    output_file.write("Accuracy: " + str(round(accuracy, 3)) + "\n")
    output_file.write("Precision: " + str(round(precision, 3)) + "\n\n")

threshold = 0.5
column_name = 'prob1'
evaluate_model(threshold, data['TRUE_STATUS'], data['PRED_Probability'],␣
 ↪column_name)
```

```
Confusion Matrix:
[[306004 192997]
 [170809 330190]]

True-Positive:  306004
False-Positive:  192997
```

```
True-Negative:  330190
False-Negative:  170809

Sensitivity:  0.642
Specificity:  0.631
Accuracy:  0.636
Precision:  0.613
```

6 4. (a) Use cut-off probability 0.3 and the function in Problem 2 to produce desired output. Present your output.

[8]:
```
threshold = 0.3
column_name = 'prob2'
evaluate_model(threshold, data['TRUE_STATUS'], data['PRED_Probability'],␣
 ↪column_name)
```

```
Confusion Matrix:
[[478925  20076]
 [432025  68974]]

True-Positive:  478925
False-Positive:  20076
True-Negative:  68974
False-Negative:  432025

Sensitivity:  0.526
Specificity:  0.775
Accuracy:  0.548
Precision:  0.96
```

7 4. (b) Use cut-off probability 0.6 and the function in Problem 2 to produce desired output. Present your output.

[9]:
```
threshold = 0.6
column_name = 'prob3'
evaluate_model(threshold, data['TRUE_STATUS'], data['PRED_Probability'],␣
 ↪column_name)
```

```
Confusion Matrix:
[[156745 342256]
 [ 51740 449259]]

True-Positive:  156745
False-Positive:  342256
```

```
True-Negative:  449259
False-Negative:  51740

Sensitivity:  0.752
Specificity:  0.568
Accuracy:  0.606
Precision:  0.314
```

[10]:
```
# Run this when you want to close and download file.
output_file.close()
```

8 5. Write a program (or function) to calculate the following statistics: "c-Statistics" (AUC) and Gini Index using the data set "Microsoft_Ranked". Report both c-statistics and Gini Index.

[11]:
```
data = pd.read_csv('Microsoft_Ranked.csv')
data.head(10)
```

[11]:

	MACHINEID	TRUE_STATUS	PRED_Probability	Rank
0	1	0	0.626452	832532.0
1	2	0	0.239536	25421.0
2	3	0	0.504254	535791.0
3	4	1	0.623444	828371.0
4	5	1	0.546973	665918.0
5	6	0	0.298289	86546.0
6	7	1	0.554385	687215.0
7	8	1	0.394333	270880.0
8	9	1	0.602833	796556.0
9	10	1	0.890603	991325.0

[12]:
```
# function to calculate AUC score
def calc_auc(positive_ranks, total_positive_samples, total_negative_samples):
    auc = (sum(positive_ranks) - (total_positive_samples *␣
 ↪(total_positive_samples + 1) / 2)) / (total_positive_samples *␣
 ↪total_negative_samples)
    return auc

# function to calculate GINI INDEX from AUC
def calc_gini(auc_score):
    gini = (auc_score * 2) - 1
    return gini
```

[13]:
```
positive_ranks = [data['Rank'][i] for i in range(len(data['TRUE_STATUS'])) if␣
 ↪data['TRUE_STATUS'][i] == 1]
```

```
total_positive_samples = data['TRUE_STATUS'].value_counts()[1]
total_negative_samples = data['TRUE_STATUS'].value_counts()[0]

auc_score = calc_auc(positive_ranks, total_positive_samples,␣
 ↪total_negative_samples)
print(auc_score)
gini_score = calc_gini(auc_score)
print(gini_score)
```

```
0.6937894745983577
0.3875789491967154
```