Homework 02

CSC 522 (002) Fall 2024 Automated Learning and Data Analysis

Team Member #1: **Kahaan Patel, kpatel48**
Team Member #2: **Priyanshu Malaviya, pmalavi**
Team Member #3: **Nirmit Patel, npatel37**

Q1

1a) The table of Actual vs Predicted values for each sample can be constructed as:

Sr. No.	Actual Values	Predicted Values
1	Yes	No
2	No	No
3	No	No
4	No	No
5	No	No
6	Yes	No
7	No	Yes
8	Yes	Yes
9	Yes	Yes
10	Yes	Yes
11	Yes	Yes
12	No	No
13	Yes	No
14	Yes	No
15	Yes	Yes

Here, “Yes” represents the cases where the loan was approved whereas “No” represents the cases where the loan was denied.

- Total number of cases where the actual true values were predicted to be true (TP) = 5
- Total number of cases where the actual false values were predicted to be false (TN) = 5
- Total number of cases where the actual true values were predicted to be false (FN) = 4
- Total number of cases where the actual false values were predicted to be true (FP) = 1

Hence, the **Confusion Matrix** can be constructed as:

		Actual Value	
		+	-
Predicted Value	+	5	1
	-	4	5

- **Accuracy:**
  Accuracy = (TP + TN)/(TP + TN + FP + FN) = (5 + 5)/(5 + 5 + 1 + 4) = 10/15 = 0.667 = 66.67%

- **Error Rate:**
  Error Rate = (FP + FN)/(TP + TN + FP + FN) = (1 + 4)/(5 + 5 + 1 + 4) = 5/15 = 0.33 = 33.33%

- **Precision:**
  Precision = (TP)/(TP + FP) = (5)/(5 + 1) = 5/6 = 0.83 = 83.33%

- **Recall:**
  Recall = (TP)/(TP + FN) = (5)/(5 + 4) = 5/9 = 0.55 = 55.56%

- **F1 Score:**
  F1 Score = (2 * Recall * Precision)/(Recall + Precision) = (2 * 0.55 * 0.83)/(0.55 + 0.83) = 0.913/1.38 = 0.66159 = 66.16%

1b)

- **Optimistic training classification error before splitting**
  Let's calculate the majority class for each leaf nodes under the subtree "Income Level":
  1. Low Income: Y: 0, N: 4 (Majority: N)
  2. Medium Income: Y: 2, N: 3 (Majority: N)
  3. High Income: Y: 3, N: 1 (Majority: Y)

  The majority class in the entire subtree would be "N" because the majority label is "N" in two out of three categories. Let's sum up the errors for each branch:
  1. Low Income: All are classified as N, so no error here.
  2. Medium Income: Majority class is N, so there are 2 errors (2 instances of Y misclassified).
  3. High Income: Majority class is N, so there are 3 errors (3 instances of Y misclassified).

  Total errors before splitting: **0 + 2 + 3 = 5.**
  Error Rate: 5/13 = 0.3846 = 38.46%

- **Optimistic training classification error after splitting**
  Now, let's calculate the error after splitting for each branch:
  1. Low Income: Y: 0, N: 4 (All correctly classified as N, so no error).
  2. Medium Income: Y: 2, N: 3 (Majority class is N, so 2 errors).
  3. High Income: Y: 3, N: 1 (Majority class is Y, so 1 error).

  So, after splitting, the total number of errors remains **0 + 2 + 1 = 3.**
  Error Rate: 3/13 = 0.2308 = 23.08%

The optimistic training classification error rate before splitting on the "Income Level" node is 38.46%, while after splitting, it decreases to 23.08%. This reduction in error suggests that splitting on the "Income Level" attribute reduces the classification error. If the node's children were pruned, the model would revert to using the majority class for the entire subtree, resulting in the higher pre-split error of 38.46%. In contrast, keeping the split lowers the error to 23.08%, indicating that the split on "Income Level" reduces the number of misclassifications. Therefore, to minimize the optimistic training classification error, the children of the "Income Level" node should not be pruned. Retaining the split results in a lower classification error and prevents the error rate from increasing by reverting to the higher pre-split value.

1c)
Pessimistic Error Calculation:

- **Before Splitting:**
  The optimistic error before splitting was previously calculated as 5/13, with 5 classification errors out of 13 instances in the "Income Level" subtree. Since there is only one leaf node (i.e., no further split), we apply a penalty of 0.8 for this leaf. Therefore, the pessimistic error before splitting is:

  Pessimistic Error (before splitting) = (5 + 0.8)/13 = 5.8/13 = 0.446 = 44.6%

- **After Splitting:**
  After splitting, we have three leaf nodes (corresponding to the Low, Medium, and High Income categories). The optimistic error after splitting was calculated as 3 misclassifications. For each leaf node, a penalty of 0.8 is applied, so the total penalty is 3 * 0.8 = 2.4. Therefore, the pessimistic error after splitting is:

  Pessimistic Error (after splitting) = (3 + 3 * 0.8)/13 = 5.4/13 = 0.415 = 41.5%

The pessimistic error before splitting is approximately 0.446, while after splitting, it reduces to 0.415. Since the pessimistic error decreases after the split, it indicates that the split on the "Income Level" node results in fewer misclassifications after accounting for the penalty. To minimize the pessimistic error rate, the node’s children should not be pruned, as pruning would increase the error from 0.415 (after splitting) to 0.446 (before splitting). Retaining the split on "Income Level" results in a lower overall pessimistic error.

1d)
The table of Actual vs Predicted values for each sample after pruning the “Income Level” node can be constructed as:

Sr. No.	Actual Values	Predicted Values
1	Yes	No
2	No	No
3	No	No
4	No	No
5	No	No
6	Yes	No
7	No	Yes
8	Yes	Yes
9	Yes	Yes
10	Yes	Yes
11	Yes	Yes
12	No	No
13	Yes	No
14	Yes	No
15	Yes	Yes

Here, “Yes” represents the cases where the loan was approved whereas “No” represents the cases where the loan was denied.

- Total number of cases where the actual true values were predicted to be true (TP) = 5
- Total number of cases where the actual false values were predicted to be false (TN) = 5
- Total number of cases where the actual true values were predicted to be false (FN) = 4
- Total number of cases where the actual false values were predicted to be true (FP) = 1

Hence, the **Confusion Matrix** can be constructed as:

		Actual Value	
		+	-
Predicted Value	+	5	1
	-	4	5

- **Accuracy:**
  Accuracy = (TP + TN)/(TP + TN + FP + FN) = (5 + 5)/(5 + 5 + 1 + 4) = 10/15 = 0.667 = 66.67%

- **Error Rate:**
  Error Rate = (FP + FN)/(TP + TN + FP + FN) = (1 + 4)/(5 + 5 + 1 + 4) = 5/15 = 0.33 = 33.33%

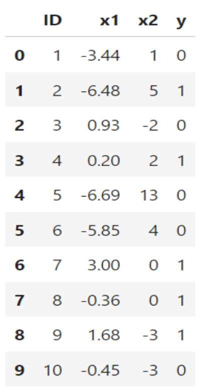
No, the original tree with the "Income Level" node was not overfitting. After pruning the node, the accuracy and error rate remained the same, indicating that the Income Level node did not introduce unnecessary complexity or noise. This suggests that the original tree was not overly complex and was generalizing well to the test data, rather than fitting too closely to the training data.

2.

Implementation:

a) Importing the required Libraries:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter('ignore')
```

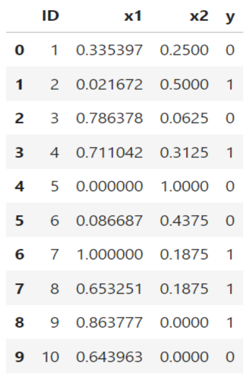
Creating a DataFrame of the given dataset:
Code:
```
df=pd.DataFrame({'ID':range(1,11),'x1':[-3.44,-6.48,0.93,0.2,-6.69,-5.85,3.0,-
0.36,1.68,-0.45],'x2':[1,5,-2,2,13,4,0,0,-3,-3],'y':[0,1,0,1,0,0,1,1,1,0]})
df
```

Output:


	ID	x1	x2	y
0	1	-3.44	1	0
1	2	-6.48	5	1
2	3	0.93	-2	0
3	4	0.20	2	1
4	5	-6.69	13	0
5	6	-5.85	4	0
6	7	3.00	0	1
7	8	-0.36	0	1
8	9	1.68	-3	1
9	10	-0.45	-3	0

Creating a deepcopy of the original data and normalize the attributes using MinMaxScaler:
Code:
```
data=df.copy(deep=True)
from sklearn.preprocessing import MinMaxScaler
m=MinMaxScaler()
data[['x1','x2']]=m.fit_transform(data[['x1','x2']])
data
```

Output:



	ID	x1	x2	y
0	1	0.335397	0.2500	0
1	2	0.021672	0.5000	1
2	3	0.786378	0.0625	0
3	4	0.711042	0.3125	1
4	5	0.000000	1.0000	0
5	6	0.086687	0.4375	0
6	7	1.000000	0.1875	1
7	8	0.653251	0.1875	1
8	9	0.863777	0.0000	1
9	10	0.643963	0.0000	0

Dividing data into independent and dependent variables i.e. X and y. And then finding nearest neighbors of point 2 and point 8 using KNeighborsClassifier:

Code:

```
X=data.drop(['ID','y'],axis=1)
y=data['y']
from sklearn.neighbors import KNeighborsClassifier
nbr=KNeighborsClassifier(n_neighbors=4, metric='euclidean')
nbr.fit(X,y)
distance_2,indices_2=nbr.kneighbors([X.loc[1]],n_neighbors=4)
print('Nearest neighbors of point 2 are points:',indices_2[0][1:]+1)
print('Their Respective Euclidean distance from point 2 are:',distance_2[0][1:])
```

Output:

```
Nearest neighbors of point 2 are points: [6 1 5]
Their Respective Euclidean distance from point 2 are: [0.0901846  0.40115294 0.50046945]
```

Code:

```
distance_8,indices_8=nbr.kneighbors([X.loc[7]],n_neighbors=4)
print('Nearest neighbors of point 8 are points:',indices_8[0][1:]+1)
print('Their Respective Euclidean distance from point 8 are:',distance_8[0][1:])
```

Output:

```
Nearest neighbors of point 8 are points: [ 4  3 10]
Their Respective Euclidean distance from point 8 are: [0.13771297 0.18261375 0.1877299 ]
```

Here we use n_neighbors=4, because sklearn also considers the point itself as its neighbor, so on choosing n_neighbors=4 would give other 3 nearest neighbors apart from itself.

Nearest neighbors for data points 2 are point: 6, 1 and 5 having Euclidean distance from point 2 as: 0.0901846, 0.40115294, 0.50046945 respectively

Nearest neighbors for data points 8 are point: 4, 3 and 10 having Euclidean distance from point 8 as: 0.13771297, 0.18261375, 0.1877299 respectively

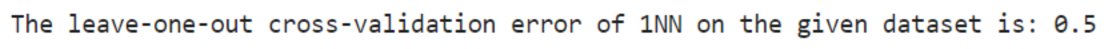
b) Initializing 1-NN model and using cross_val_score, LeaveOneOut function of sklearn to compute loocv error on 1-NN on the given dataset:

Code:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import LeaveOneOut,cross_val_score
```

```
knn_1=KNeighborsClassifier(n_neighbors=1)
looc=LeaveOneOut()
scores=cross_val_score(knn_1, X, y,cv=looc, scoring='accuracy')
loocv_error = 1-scores.mean()
print('The leave-one-out cross-validation error of 1NN on the given dataset 
is:',loocv_error)
```

Output:



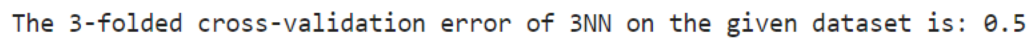
```
The leave-one-out cross-validation error of 1NN on the given dataset is: 0.5
```

Therefore, Leave one out cross validation error using 1-NN on this dataset is 0.5.

c) Calculating 3-folded cross-validation error of 3-NN on the given dataset, given condition: For the ith fold, the testing dataset is composed of all the data points whose (ID mod 3 = i – 1),
Code:

```
from sklearn.metrics import accuracy_score
knn_3=KNeighborsClassifier(n_neighbors=3)
indices=[]
for i in range(3):
    train_index = [x-1 for x in np.arange(1,len(X)+1) if x%3==i]
    test_index = [x for x in np.arange(len(X)) if x not in train_index]
    indices.append([test_index,train_index])
scores=cross_val_score(knn_3, X, y,cv=indices, scoring='accuracy')
knn3_error = 1-scores.mean()
print('The 3-folded cross-validation error of 3NN on the given dataset is:',knn3_error)
```

Output:



```
The 3-folded cross-validation error of 3NN on the given dataset is: 0.5
```

Therefore, 3-folded cross-validation error of 3NN on given dataset is: 0.5.

d) From (b) and (c), it can be seen that both of them have an error of 0.5. Based on the result, it cannot be determined whether one is better than the other. However, some observations can be drawn. Firstly, the dataset only contains 10 instances, which makes it very small to judge a method’s performance properly. Moreover, the cross-validation techniques used to compare the classifiers are both different. LOOCV with 1NN uses 9 instances for training and 1 instance for testing, repeating this process 10 times, whereas 3-Fold CV with 3NN uses 2/3rd of the data for training and the rest for testing in each fold. This along with the small dataset size makes it difficult to judge the performance of any technique or the models themselves. A better assessment can be made if either LOOCV or 3-Fold-CV is used for both the models.

3

Given Data:

Instances	True Class	P(+|A, ..., Z,M1)	P(+|A, ..., Z,M2)
1	+	0.78	0.61
2	+	0.62	0.08
3	-	0.44	0.62
4	-	0.55	0.39
5	+	0.61	0.48
6	+	0.47	0.09
7	-	0.07	0.38
8	-	0.14	0.09
9	+	0.48	0.06
10	-	0.32	0.01

a) We start by first sorting the posterior probabilities of models M1 and M2, and taking them as thresholds, calculate the True Positives (TP), False Positives (FP), False Negatives (FN), True Negatives (TN). Using the below given formula we calculate True Positive Rate (TPR) and False Positive Rate (FPR) at each threshold,
TPR = TP / (TP + FN)
FPR = FP / (FP + TN)
Using the above steps we can easily plot the ROC curve for model M1 and M2.

Below 2 table shows the TP, FP, FN, TN, TPR, and FPR of models M1 and M2 at different thresholds:

For Model 1 (M1):

	-	-	-	-	+	+	-	+	+	+	
	0.07	**0.14**	**0.32**	**0.44**	**0.47**	**0.48**	**0.55**	**0.61**	**0.62**	**0.78**	**1.00**
TP	5	5	5	5	5	4	3	3	2	1	0
FP	5	4	3	2	1	1	1	0	0	0	0
TN	0	1	2	3	4	4	4	4	5	5	5
FN	0	0	0	0	0	1	2	2	3	4	5
TPR	1.0	1.0	1.0	1.0	1.0	0.8	0.6	0.6	0.4	0.2	0.0
FPR	1.0	0.8	0.6	0.4	0.2	0.2	0.2	0.0	0.0	0.0	0.0

For Model 2 (M2):

	-	-	-	-	+	+	-	+	+	
	0.01	**0.06**	**0.08**	**0.09**	**0.38**	**0.39**	**0.48**	**0.61**	**0.62**	**1.00**
TP	5	5	4	3	2	2	2	1	0	0
FP	5	4	4	4	3	2	1	1	1	0
TN	0	1	1	1	2	3	4	4	4	5
FN	0	0	1	2	3	3	3	4	5	5
TPR	1.0	1.0	0.8	0.6	0.4	0.4	0.4	0.2	0.0	0.0
FPR	1.0	0.8	0.8	0.8	0.6	0.4	0.2	0.2	0.2	0.0

Plotting the ROC curve for M1 and M2:
Code:

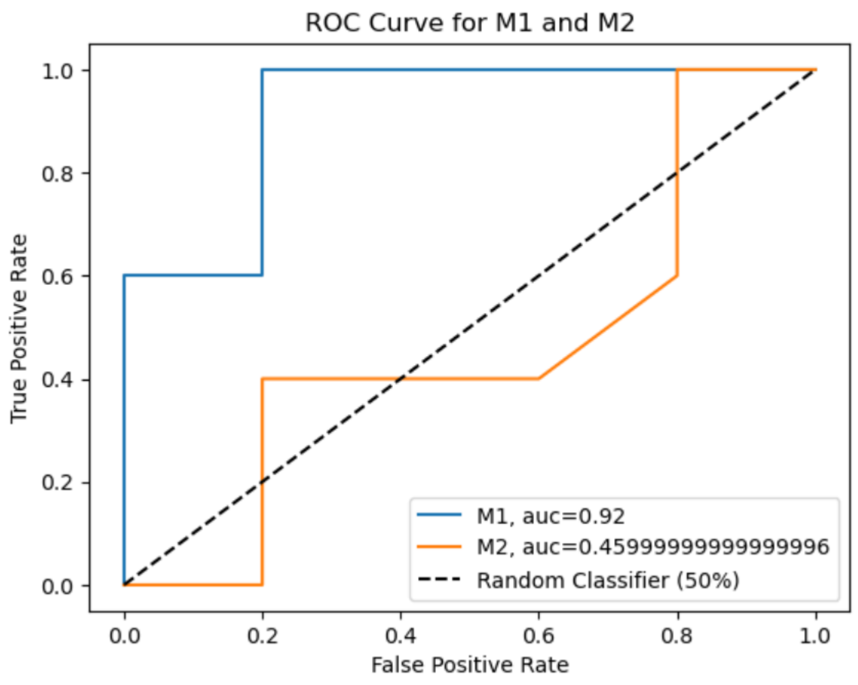
```
from sklearn import metrics
prob_M1 = [0.78,0.62,0.44,0.55,0.61,0.47,0.07,0.14,0.48,0.32]
prob_M2 = [0.61,0.08,0.62,0.39,0.48,0.09,0.38,0.09,0.06,0.01]
y = [1,1,0,0,1,1,0,0,1,0]
M1_fpr, M1_tpr, M1_thresh = metrics.roc_curve(y,prob_M1)
M2_fpr, M2_tpr, M2_thresh = metrics.roc_curve(y,prob_M2)
M1_auc_score = metrics.roc_auc_score(y,prob_M1)
M2_auc_score = metrics.roc_auc_score(y,prob_M2)

plt.figure(0).clf()
plt.plot(M1_fpr,M1_tpr,label="M1, auc="+str(M1_auc_score))
plt.plot(M2_fpr,M2_tpr,label="M2, auc="+str(M2_auc_score))
plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier (50%)')

plt.legend(loc=0)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for M1 and M2')

# Show the plot
plt.show()
```

Output:



Explanation:

M1's ROC curve is closer to the ideal top-left corner, suggesting that it has a strong ability to discriminate between classes. Also, indicated by its higher AUC score of 0.92. This means that M1 can distinguish between the classes effectively, achieving a high true positive rate while keeping the false positive rate low.

M2's ROC curve is more toward the diagonal (closer to random guessing), showing it struggles with classification. M2, with an AUC of 0.46, performs worse than random guessing, indicating poor class separation and unreliable predictions.

b) Given threshold t=0.5

Computing predictions of Model M1 on given instances in the dataset,

Instances	**True Class**	**P(+|A, ..., Z,M1)**	**Predicted Class**	**True Positive**	**False Positive**	**True Negative**	**False Negative**
1	+	0.78	+	1	0	0	0
2	+	0.62	+	1	0	0	0
3	-	0.44	-	0	0	1	0
4	-	0.55	+	0	1	0	0
5	+	0.61	+	1	0	0	0
6	+	0.47	-	0	0	0	1
7	-	0.07	-	0	0	1	0
8	-	0.14	-	0	0	1	0
9	+	0.48	-	0	0	0	1
10	-	0.32	-	0	0	1	0

So, any test instances whose posterior probability is greater than t will be classified as a positive example. Using the given data we compute the following True positives (TP), False Positives (FP), False Negatives (FN), and True Negatives (TN) for model M1. Above table shows instances which are either TP, FP, TN or FN.

So, calculating total number of TP, FP, TN and FN using the table:

TP = 3, FP = 1, FN = 2, TN = 4

Precision is the proportion of true positives among all predicted positives, indicating how many of the predicted positives were actually correct.

Precision = TP / (TP + FP)

= 3 / (3 + 1) = 3/4

= 0.75

Recall is the measures how many true positives can the model successfully recall out of all positives, indicating how well the model identifies positive instances.

Recall = TP / (TP + FN)

= 3 / (3 +2)

= 3 / 5

= 0.60

F-measure denotes the harmonic mean of precision and recall, providing a balance between the two, especially when there is an uneven class distribution.

F-measure = 2 * Recall * Precision / (Recall + Precision)

= 2 * 0.75 * 0.6 / (0.75 + 0.6)

= 0.6666 = 0.67

Hence, the values of precision, recall and F-measure, when threshold=0.5 is 0.75, 0.60, and 0.67 respectively.

c) Given threshold t=0.5

Computing predictions of Model M2 on given instances in the dataset,

Instances	**True Class**	**P(+|A, ..., Z,M2)**	**Predicted Class**	**True Positive**	**False Positive**	**True Negative**	**False Negative**
1	+	0.61	+	1	0	0	0
2	+	0.08	-	0	0	0	1
3	-	0.62	+	0	1	0	0
4	-	0.39	-	0	0	1	0
5	+	0.48	-	0	0	0	1
6	+	0.09	-	0	0	0	1
7	-	0.38	-	0	0	1	0
8	-	0.09	-	0	0	1	0
9	+	0.06	-	0	0	0	1
10	-	0.01	-	0	0	1	0

Similarly, for model M2,
Threshold = 0.5
Using the given data we compute the following True positives (TP), False Positives (FP), False Negatives (FN), and True Negatives (TN) for model M2:

TP = 1, FP = 1, FN = 4, TN = 4

Precision = TP / (TP + FP)
= 1 / (1 + 1)
= 1 / 2
= 0.50

Recall = TP / (TP + FN)
= 1 / (1 + 4)
= 1 / 5
= 0.20

F-measure = 2 * Recall * Precision / (Recall + Precision)
= 2 * 0.5 * 0.2 / (0.5 + 0.2)
= 0.2857
= 0.286

Thus, the value of precision, recall and F-measure for model M2, when cutoff threshold=0.5 are 0.50, 0.20, and 0.286 respectively.

Comparing the models on the basis of F-measure:
The F-measure of model M1 is significantly higher compared to that of model M2. This significant difference indicates that M1 is more balanced in correctly identifying positive instances, while keeping false positive at a minimum. Hence, based on F-measure metric, M1 shows better performance compared to M2.

The results are indeed consistent with what we expect from the ROC curve:
Model M1 outperforms Model M2 based on its higher precision, recall, and F-measure values for the positive class. These metrics confirm that M1 is more effective at identifying positive instances and minimizing false

positives. This conclusion aligns with the results from the ROC curve analysis, where M1 showed a much higher AUC, further demonstrating its superior performance compared to M2.