CSC:522 : Homework 2

Brijesh K Bhayana(bkbhayan), Dhruv Purohit (dpurohi)
Akhil Chawla(achawla), Prateek N Kamath(pnkamath)

Problem 1.

(a)

Income Level	Credit Score	Employment Status	Loan Amount	Approved	Predicted
High	Bad	Full-time	High	Yes	No
Medium	Bad	Full-time	Medium	No	No
Low	Bad	Part-time	Low	No	No
Low	Bad	Part-time	Low	No	No
Low	Bad	Self-employed	High	No	No
High	Bad	Self-employed	Medium	Yes	No
Medium	Good	Full-time	High	No	Yes
High	Good	Full-time	High	Yes	Yes
High	Good	Full-time	Low	Yes	Yes
Low	Good	Full-time	Low	Yes	Yes
Medium	Good	Full-time	Medium	Yes	Yes
Low	Good	Part-time	High	No	No
Medium	Good	Part-time	Medium	Yes	No
Medium	Good	Self-employed	High	Yes	No
Medium	Good	Self-employed	Low	Yes	Yes

Confusion Matrix:

TP=5	FP=1
FN=4	TN=5

Accuracy:

$$\frac{TP + TN}{TOTAL\ PREDICTIONS} = \frac{5+5}{15} = \frac{2}{3} = 0.66666$$

Error Rate:

$$\frac{FP + FN}{TOTAL\ \ PEDICTIONS} = \frac{1+4}{15} = \frac{1}{3} = 0.33333$$

Precision:

$$\frac{TP}{TP + FP} = \frac{5}{1+5} = \frac{5}{6} = 0.83333$$

Recall:

$$\frac{TP}{TP + FN} = \frac{5}{9} = 0.55555$$

F1 Score:

$$\frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * 0.8333 * 0.5555}{0.8333 + 0.5555} = \frac{0.9258}{1.3888} = 0.6666$$

(b)

Optimistic Error before splitting:

$$\frac{5}{13} = 0.3846$$

Optimistic Error after splitting:

$$(\frac{0}{4} * \frac{4}{13}) + (\frac{2}{5} * \frac{5}{13}) + (\frac{1}{4} * \frac{4}{13}) = \frac{2}{13} + \frac{1}{13} = \frac{3}{13} = 0.23076$$

Should we prune to reduce optimistic error?

No we should not prune the children. This would lead to an increase in the optimistic error as we can see from the calculations above. Moreover, we should not consider optimistic error as a measure to prune the tree, it would go against the training process in Hunt's algorithm, as it grows the tree to reduce the optimistic error only. Rather, pessimistic error should be considered to make a decision on pruning.

(c)

Pessimistic Error before splitting:

$$\frac{num_{wrong\ predictions} + \alpha * (num_{children})}{total\ predictions} = \frac{5 + 0.8 * 1}{13} = \frac{5.8}{13} = 0.4461$$

Pessimistic Error after splitting:

$$\frac{num_{wrong\ predictions} + \alpha * (num_{children})}{total\ predictions} = \frac{3 + 0.8 * 3}{13} = \frac{5.4}{13} = 0.4153$$

Should we prune to reduce pessimistic error?

No we should not prune the children. This would lead to an increase in the pessimistic error as we can see from the calculations above. Pessimistic error gives us a better estimate of the generalization error and while training our model, we should give the highest importance to reducing it. By pruning, we would increase it causing a poorer performing model.

(d)

Income Level	Credit Score	Employment Status	Loan Amount	Approved	Predicted
High	Bad	Full-time	High	Yes	No
Medium	Bad	Full-time	Medium	No	No
Low	Bad	Part-time	Low	No	No
Low	Bad	Part-time	Low	No	No
Low	Bad	Self-employed	High	No	No
High	Bad	Self-employed	Medium	Yes	No
Medium	Good	Full-time	High	No	Yes
High	Good	Full-time	High	Yes	Yes
High	Good	Full-time	Low	Yes	Yes
Low	Good	Full-time	Low	Yes	Yes
Medium	Good	Full-time	Medium	Yes	Yes
Low	Good	Part-time	High	No	No
Medium	Good	Part-time	Medium	Yes	No
Medium	Good	Self-employed	High	Yes	No
Medium	Good	Self-employed	Low	Yes	Yes

Error Rate:

$$\frac{FP + FN}{TOTAL\ \ PEDICTIONS} = \frac{1 + 4}{15} = \frac{1}{3} = 0.33333$$

Was the original tree (with the Income Level node) over-fitting?

No, the original tree was not overfitting, as we did not observe any decrease in the test error rate after pruning it. The error rate has remained constant. However, this conclusion is only based on the data provided in hw2q1_f24.csv. We would have concluded that the model was overfitting, had the testing error gone down. We observed similar conclusion when we were looking at the pessimistic error for the Income Level node in part c. However, shorter trees are preferred as they lead to quicker inferences

Hence, based on data given in hw2q1_f24.csv, we can conclude that the model was not overfitting.

Problem 2:

Started with importing necessary libraries and importing the data given.

```python
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
```

[44] ✓ 0.0s Python

```python
X = np.asarray([
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    [-3.44, -6.48, 0.93, 0.2, -6.69, -5.85, 3.0, -0.36, 1.68, -0.45],
    [1, 5, -2, 2, 13, 4, 0, 0, -3, -3]], dtype=np.float64).T
X = pd.DataFrame(X, columns=['ID', 'x1', 'x2'])
```

[45] ✓ 0.0s Python

\+ Code \+ Markdown

```python
y = np.asarray([
    np.asarray([1,2,3,4,5,6,7,8,9,10], dtype=np.float64),
    ["star", "spade", "star", "spade", "star", "star", "spade", "spade", "spade", "star"]]).T
y = pd.DataFrame(y, columns=["ID",'y'])
```

[46] ✓ 0.0s Python

```python
print(X.shape)
print(y.shape)
```

[47] ✓ 0.0s Python

```
(10, 3)
(10, 2)
```

(a) To solve this part, I saved the indexes for which we need to find the neighbors. Since, we follow 0 indexing in python, therefore saved them as 1 and 7. (Question asked for 2 and 8).

Then I instantiated a KNN classifier and set algorithm to be brute force and number of neighbors to be 3. Then I made sure that equal weights are given to all points here and set p=2 so that the classifier uses euclidean distances or the L2 norm here. I then removed the given indexes from the training data and called model.fit so that the classifier saves the given data in memory. Since, this is a lazy learning model, it doesn't learn any parameters and only memorizes the training data set. Now, we just send the given data objects to the model and get the 3 nearest neighbors and their distances and print it. Below is the code for the above approach

Code and output:

```python
indices = [1, 7]
for index in indices:
    x_train = np.delete(X, index, axis=0)
    y_train = np.delete(y, index, axis = 0)
    model = KNeighborsClassifier(n_neighbors=3, weights='uniform', algorithm='brute', p=2, metric='minkowski')
    model.fit(x_train[:,1:], y_train[:, 1:])
    dist, nbrs = model.kneighbors([X.iloc[index, 1:]])
    print(X.iloc[index, :1])
    for i in range(len(nbrs[0])):
        print("Distance: "+str(dist[0][i]), end=" | Point: ")
        print(x_train[nbrs[0][i]])
```

[52] ✓ 0.0s Python

```
ID    2.0
Name: 1, dtype: float64
Distance: 1.1819052415485778 | Point: [ 6.   -5.85  4.  ]
Distance: 5.024101909794426 | Point: [ 1.   -3.44  1.  ]
Distance: 7.322731730713614 | Point: [4.  0.2 2. ]
ID    8.0
Name: 7, dtype: float64
Distance: 2.076920797719547 | Point: [4.  0.2 2. ]
Distance: 2.379936973955403 | Point: [ 3.    0.93 -2.  ]
Distance: 3.001349696386611 | Point: [10.   -0.45 -3.  ]
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
```

As we can see. For the 1st data point [-6.48, 5, ‘spade’] the 3 nearest neighbors are: [-5.85, 4, ‘star’], [-3.44, 1, ‘star’], [0.2, 2, ‘spade’]. The corresponding distances are [1.181905, 5.024101, 7.322731] respectively.

Similarly for the 2nd data point [-0.36, 0, ‘spade’] the 3 nearest neighbors are: [0.2, 2, ‘spade’], [0.93, -2, ‘star’], [-0.45, -3, ‘star’]. The corresponding distances are [2.076920, 2.37993,3.0013] respectively.

(b) To calculate LOOCV error for a 1NN on this dataset. We modify the code used for part a. We do this by iterating over all examples as for LOOCV, k = n where n is the size of the dataset. Now, we delete each particular example one by one from the training dataset and fit our model on the remaining data. Then we use the removed data point from the set as a test example and calculate the error rate. We would do this n times as we have n-folds. Now, in the end we would get a mean of all the classification errors to get a LOOCV error for the given model. Along with that we would also modify the n_neighbors parameter to 1 in the KNeighbors model instantiation.

Code and Output:

```python
errors = []
for i in range(len(X)):
    x_train = np.delete(X, i, axis=0)
    y_train = np.delete(y, index, axis=0)
    model = KNeighborsClassifier(n_neighbors=1, weights='uniform', algorithm='brute', p=2, metric='minkowski')
    model.fit(x_train[:, 1:], y_train[:, 1:])
    result = model.predict([X.iloc[i, 1:]])
    if(result[0] == y.iloc[i]['y']):
        errors.append(0)
    else:
        errors.append(1)

print(np.mean(errors))
```

[53] ✓ 0.0s Python

```
0.7
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
```

(c)

For this part, we split the dataset into train and test by creating a mask using the relation that is given. After that, for each fold, we fit the model and test it on the fold we left out of the training data and get the accuracy score. We subtract it from 1 to get the error and append it to an array. Once all folds have been done, we compute the mean to get the 3-fold CV Error.

Output and code:

```python
error_rates = []
for i in range(1, 4):
    x_test = X[X['ID'] % 3 == i-1]
    x_train = X[X['ID'] % 3 != i-1]
    y_test = y[X['ID'] % 3 == i-1]
    y_train = y[X['ID'] % 3 != i-1]

    model = KNeighborsClassifier(n_neighbors=3, weights='uniform', algorithm='brute', p=2, metric='minkowski')
    model.fit(x_train.iloc[:, 1:], y_train.iloc[:, 1:])

    y_pred = model.predict(x_test.iloc[:, 1:])
    error_rate = 1 - accuracy_score(y_test.iloc[:, 1:], y_pred)
    error_rates.append(error_rate)

print(error_rates)
print(np.mean(error_rates))
```

[15] ✓ 0.0s Python

```
[0.6666666666666667, 0.5, 0.6666666666666667]
0.6111111111111112
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
/Users/korupt/CodeWorkspace/CSC-522-ALDA-Fall24/.venv/lib/python3.9/site-packages/sklearn/neighbors/_classification.py:238: DataConversionWarni
  return self._fit(X, y)
```

We get CV error of 0.611111 in this case.

(d)

Based on the results of part b) and c) we cannot determine which is a better classifier as we have run different CV for both the models. For part B we ran LOOCV which considers each data point as a test dataset individually whereas in 3fold CV, the test dataset consists of multiple different data points. We could however have been able to make a conclusion about the same had we tested the both using the same CV technique.

Problem 3.
(a)
(M1) Threshold (0.06)

Probability	Actual	Predicted
0.07	-	+
0.14	-	+
0.32	-	+
0.44	-	+
0.47	+	+
0.48	+	+
0.55	-	+
0.61	+	+
0.62	+	+
0.78	+	+

TPR = $\dfrac{TP}{TP+FN} = \dfrac{5}{5} = 1$

FPR = $\dfrac{FP}{FP+TN} = \dfrac{5}{5} = 1$

(M1) Threshold (0.13)

Probability	Actual	Predicted
0.07	-	-
0.14	-	+
0.32	-	+
0.44	-	+
0.47	+	+
0.48	+	+
0.55	-	+
0.61	+	+
0.62	+	+
0.78	+	+

TPR = $\frac{5}{5} = 1$

FPR = $\frac{4}{4+1} = \frac{4}{5} = 0.8$

(M1) Threshold (0.31)

Probability	Actual	Predicted
0.07	-	-
0.14	-	-
0.32	-	+
0.44	-	+
0.47	+	+
0.48	+	+
0.55	-	+
0.61	+	+
0.62	+	+
0.78	+	+

TPR = $\frac{5}{5} = 1$

FPR = $\frac{3}{3+2} = \frac{3}{5} = 0.6$

(M1) Threshold (0.43)

Probability	Actual	Predicted
0.07	-	-
0.14	-	-
0.32	-	-
0.44	-	+
0.47	+	+
0.48	+	+
0.55	-	+
0.61	+	+
0.62	+	+
0.78	+	+

TPR = $\frac{5}{5} = 1$

FPR = $\frac{2}{2+3} = \frac{2}{5} = 0.4$

M1) Threshold (0.46)

Probability	Actual	Predicted
0.07	-	-
0.14	-	-
0.32	-	-
0.44	-	-
0.47	+	+
0.48	+	+
0.55	-	+
0.61	+	+
0.62	+	+
0.78	+	+

TPR = $\frac{5}{5} = 1$

FPR = $\frac{1}{1+4} = \frac{1}{5} = 0.2$

M1) Threshold (0.47)

Probability	Actual	Predicted
0.07	-	-
0.14	-	-
0.32	-	-
0.44	-	-
0.47	+	-
0.48	+	+
0.55	-	+
0.61	+	+
0.62	+	+
0.78	+	+

TPR = $\dfrac{4}{4+1} = \dfrac{4}{5} = 0.8$

FPR = $\dfrac{1}{1+4} = \dfrac{1}{5} = 0.2$

M1) Threshold (0.54)

Probability	Actual	Predicted
0.07	-	-
0.14	-	-
0.32	-	-
0.44	-	-
0.47	+	-
0.48	+	-
0.55	-	+
0.61	+	+
0.62	+	+
0.78	+	+

TPR = $\dfrac{3}{5} = 0.6$

FPR = $\dfrac{1}{5} = 0.2$

M1) Threshold (0.60)

Probability	Actual	Predicted
0.07	-	-
0.14	-	-
0.32	-	-
0.44	-	-
0.47	+	-
0.48	+	-
0.55	-	-
0.61	+	+
0.62	+	+
0.78	+	+

TPR = $\frac{3}{5} = 0.6$

FPR = $\frac{0}{0+5} = 0$

M1) Threshold (0.61)

Probability	Actual	Predicted
0.07	-	-
0.14	-	-
0.32	-	-
0.44	-	-
0.47	+	-
0.48	+	-
0.55	-	-
0.61	+	-
0.62	+	+
0.78	+	+

TPR = $\frac{2}{5} = 0.4$

FPR = $\frac{0}{5} = 0$

M1) Threshold (0.77)

Probability	Actual	Predicted
0.07	-	-
0.14	-	-
0.32	-	-
0.44	-	-
0.47	+	-
0.48	+	-
0.55	-	-
0.61	+	-
0.62	+	-
0.78	+	+

TPR = $\frac{1}{5} = 0.2$

FPR = $\frac{0}{5} = 0$

M1) Threshold (0.78)

Probability	Actual	Predicted
0.07	-	-
0.14	-	-
0.32	-	-
0.44	-	-
0.47	+	-
0.48	+	-
0.55	-	-
0.61	+	-
0.62	+	-
0.78	+	-

TPR = $\frac{0}{5} = 0$

FPR = $\frac{0}{5} = 0$

M2) Threshold (0.005)

Probability	Actual	Predicted
0.01	-	+
0.06	+	+
0.08	+	+
0.09	+	+
0.09	-	+
0.38	-	+
0.39	-	+
0.48	+	+
0.61	+	+
0.62	-	+

TPR = $\frac{5}{5} = 1$

FPR = $\frac{5}{5} = 1$

M2) Threshold (0.05)

Probability	Actual	Predicted
0.01	-	-
0.06	+	+
0.08	+	+
0.09	+	+
0.09	-	+
0.38	-	+
0.39	-	+
0.48	+	+
0.61	+	+
0.62	-	+

TPR = $\frac{5}{5} = 1$

FPR = $\frac{4}{5} = 0.8$

M2) Threshold (0.07)

Probability	Actual	Predicted
0.01	-	-
0.06	+	-
0.08	+	+
0.09	+	+
0.09	-	+
0.38	-	+
0.39	-	+
0.48	+	+
0.61	+	+
0.62	-	+

TPR = $\frac{4}{5} = 0.8$

FPR = $\frac{4}{5} = 0.8$

M2) Threshold (0.08)

Probability	Actual	Predicted
0.01	-	-
0.06	+	-
0.08	+	-
0.09	+	+
0.09	-	+
0.38	-	+
0.39	-	+
0.48	+	+
0.61	+	+
0.62	-	+

TPR = $\frac{3}{5} = 0.6$

FPR = $\frac{4}{5} = 0.8$

M2) Threshold (0.37)

Probability	Actual	Predicted
0.01	-	-
0.06	+	-
0.08	+	-
0.09	+	-
0.09	-	-
0.38	-	+
0.39	-	+
0.48	+	+
0.61	+	+
0.62	-	+

TPR = $\frac{2}{5} = 0.4$

FPR = $\frac{3}{5} = 0.6$

M2) Threshold (0.38)

Probability	Actual	Predicted
0.01	-	-
0.06	+	-
0.08	+	-
0.09	+	-
0.09	-	-
0.38	-	-
0.39	-	+
0.48	+	+
0.61	+	+
0.62	-	+

TPR = $\frac{2}{5} = 0.4$

FPR = $\frac{2}{5} = 0.4$

M2) Threshold (0.47)

Probability	Actual	Predicted
0.01	-	-
0.06	+	-
0.08	+	-
0.09	+	-
0.09	-	-
0.38	-	-
0.39	-	-
0.48	+	+
0.61	+	+
0.62	-	+

TPR = $\frac{2}{5} = 0.4$

FPR = $\frac{1}{5} = 0.2$

M2) Threshold (0.60)

Probability	Actual	Predicted
0.01	-	-
0.06	+	-
0.08	+	-
0.09	+	-
0.09	-	-
0.38	-	-
0.39	-	-
0.48	+	-
0.61	+	+
0.62	-	+

TPR = $\frac{1}{5} = 0.2$

FPR = $\frac{1}{5} = 0.2$

M2) Threshold (0.61)

Probability	Actual	Predicted
0.01	-	-
0.06	+	-
0.08	+	-
0.09	+	-
0.09	-	-
0.38	-	-
0.39	-	-
0.48	+	-
0.61	+	-
0.62	-	+

TPR = $\frac{0}{5} = 0$

FPR = $\frac{1}{5} = 0.2$

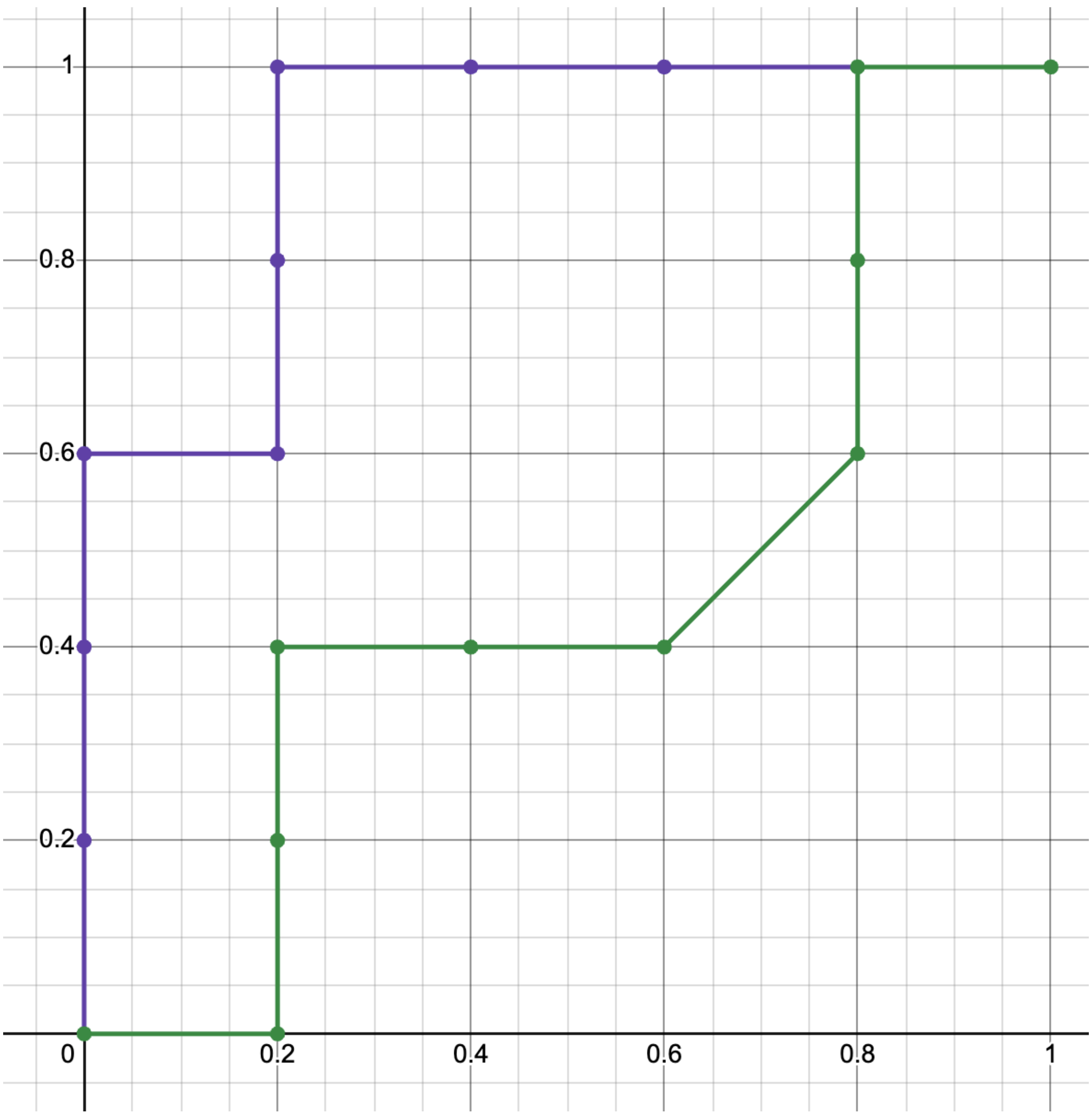
M2) Threshold (0.62)

Probability	Actual	Predicted
0.01	-	-
0.06	+	-
0.08	+	-
0.09	+	-
0.09	-	-
0.38	-	-
0.39	-	-
0.48	+	-
0.61	+	-
0.62	-	-

TPR = $\frac{0}{5} = 0$

FPR = $\frac{0}{5} = 0$

ROC Curves: (Purple Curve: M1, Green Curve: M2)



True Positive Rate: (y), False Positive Rate (x)

According to the ROC Curve obtained, M1's ROC curve has more Area Under the Curve (AUC) hence, leading to a conclusion that it is a better model than M2. Whenever, we are comparing models using ROC curves, we pick the one with the higher AUC. Hence, we pick M1 as the better model here.

(b)
M1) Threshold (0.5)

Probability	Actual	Predicted
0.07	-	-
0.14	-	-
0.32	-	-
0.44	-	-
0.47	+	-
0.48	+	-
0.55	-	+
0.61	+	+
0.62	+	+
0.78	+	+

Confusion Matrix:

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

Precision:

$$\frac{TP}{TP + FP} = \frac{3}{3 + 1} = \frac{3}{4} = 0.75$$

Recall:

$$\frac{TP}{TP + FN} = \frac{3}{3 + 2} = \frac{3}{5} = 0.6$$

F1 Score:

$$\frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * 0.75 * 0.6}{0.75 + 0.6} = \frac{0.9}{1.35} = 0.66666$$

c)
M2) Threshold (0.5)

Probability	Actual	Predicted
0.01	-	-
0.06	+	-
0.08	+	-
0.09	+	-
0.09	-	-
0.38	-	-
0.39	-	-
0.48	+	-
0.61	+	+
0.62	-	+

Confusion Matrix:

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

Precision:

$$\frac{TP}{TP + FP} = \frac{1}{1 + 1} = \frac{1}{2} = 0.5$$

Recall:

$$\frac{TP}{TP + FN} = \frac{1}{1 + 4} = \frac{1}{5} = 0.2$$

F1 Score:

$$\frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * 0.5 * 0.2}{0.5 + 0.2} = \frac{0.2}{0.7} = 0.2857$$

Here, we observe that the F1 Score for M1 is greater than M2. Hence, this helps us in making the conclusion that M1 is a better model than M2. The results are also consistent with the results we obtained from ROC curve.