Ramkishore Rao

CS5033/HW-6

Exercise 1 – Preprocessing (10 points). You have already done this part in homework 4. However, since you may need to refresh your memory with what you did, this part is worth a few points.

(a) Remove columns 5 and 13 (labeled p1 and stab); p1 is non-predictive and stab is target column that is exactly correlated with the binary target you are trying to predict (if this column is negative, the system is stable).

(b) Change the target variable to a number. If the value is stable, change it to 1, and if the value is unstable, change it to 0.

(c) Remove 20% of the examples and keep them for testing. You may assume that all examples are independent, so it does not matter which 20% you remove. However, the testing data should not be used until after a model has been selected.

(d) Split the remaining examples into training (75%) and validation (25%). Thus, you will train with 60% of the full dataset (75% of 80%) and validate with 20% of the full dataset (25% of 80%).

Code:

```
df = pd.read_csv('C:/Data Science and Analytics/CS 5033/Homeworks/Data_for_UCI_named.csv')

df.drop(columns=['p1'], inplace=True)

df.drop(columns=['stab'], inplace=True)

print(df.head())

# Exercise 1

for i in df.index:

    if df['stabf'][i] == 'stable':

        df['stabf'][i] = 1

    else:

        df['stabf'][i] = 0


print(df.head())

X = df.iloc[:, :-1]

Y = df.iloc[:,-1].astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1, test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))
```

```
print(y_test.head(), len(y_test))

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state = 1,
test_size = 0.25)

print(X_train1.head(), len(X_train1))

print(X_val1.head(), len(X_val1))

print(y_train1.head(), len(y_train1))

print(y_val1.head(), len(y_val1))
```

Output:

```
runfile('C:/Data Science and Analytics/CS 5033/Homeworks/untitled1.py', wdir='C:/Data Science
and Analytics/CS 5033/Homeworks')

       tau1      tau2      tau3  ...        g3        g4     stabf

0  2.959060  3.079885  8.381025  ...  0.887445  0.958034  unstable

1  9.304097  4.902524  3.047541  ...  0.562139  0.781760    stable

2  8.971707  8.848428  3.046479  ...  0.839444  0.109853  unstable

3  0.716415  7.669600  4.486641  ...  0.929381  0.362718  unstable

4  3.134112  7.608772  4.943759  ...  0.656947  0.820923  unstable


     tau1      tau2      tau3      tau4  ...        g2        g3        g4  stabf

0  2.959060  3.079885  8.381025  9.780754  ...  0.859578  0.887445  0.958034      0

1  9.304097  4.902524  3.047541  1.369357  ...  0.862414  0.562139  0.781760      1

2  8.971707  8.848428  3.046479  1.214518  ...  0.766689  0.839444  0.109853      0

3  0.716415  7.669600  4.486641  2.340563  ...  0.976744  0.929381  0.362718      0

4  3.134112  7.608772  4.943759  9.857573  ...  0.455450  0.656947  0.820923      0
```

Split into Train and Test

```
          tau1      tau2      tau3  ...        g2        g3        g4

2694  6.255995  2.542401  7.024714  ...  0.618403  0.685739  0.660088

5140  5.070581  5.490253  8.075688  ...  0.097244  0.916955  0.129254

2568  1.220072  8.804028  3.874283  ...  0.923594  0.238881  0.660156

3671  7.498402  6.697603  8.798626  ...  0.899003  0.964752  0.600598

7427  7.074006  1.337511  6.100756  ...  0.716082  0.836928  0.165162


[5 rows x 11 columns] 8000

          tau1      tau2      tau3  ...        g2        g3        g4

9953  6.877876  4.113820  9.356768  ...  0.845536  0.112440  0.822562
```

```
3850  5.802841  6.271371  4.731540  ...  0.416478  0.912846  0.861306
4962  2.286998  4.385142  2.830232  ...  0.130186  0.703887  0.063811
3886  5.019920  2.209962  6.266080  ...  0.065992  0.427349  0.814648
5437  7.646145  9.187896  5.484219  ...  0.121611  0.787318  0.300314

[5 rows x 11 columns] 2000
2694    0
5140    0
2568    0
3671    0
7427    0
Name: stabf, dtype: int32 8000
9953    0
3850    0
4962    1
3886    1
5437    0
Name: stabf, dtype: int32 2000
```

Split into Train and Validation

```
          tau1      tau2      tau3  ...        g2        g3        g4
4495  5.981366  6.047619  8.405237  ...  0.514232  0.060114  0.401098
6470  8.768688  4.074135  2.062575  ...  0.405332  0.890861  0.438152
2221  8.215859  0.753517  2.269814  ...  0.338388  0.986259  0.471547
7686  6.548323  7.155136  2.075700  ...  0.791773  0.224126  0.322416
9419  3.406631  5.474094  1.571459  ...  0.904382  0.855335  0.287320

[5 rows x 11 columns] 6000
          tau1      tau2      tau3  ...        g2        g3        g4
7857  8.053666  9.477708  8.880026  ...  0.410746  0.070308  0.114784
9924  5.855570  9.554050  2.637931  ...  0.591532  0.348481  0.707757
3087  7.390836  7.652423  5.953608  ...  0.289397  0.854659  0.835010
```

```
3513  2.200650  3.569270  6.543458  ...  0.614464  0.349396  0.783647
2105  7.597574  7.016628  7.031702  ...  0.742712  0.670236  0.099994

[5 rows x 11 columns] 2000
4495    0
6470    1
2221    1
7686    0
9419    0
Name: stabf, dtype: int32 6000
7857    0
9924    1
3087    0
3513    0
2105    0
Name: stabf, dtype: int32 2000
```

Exercise 2 – Artificial Neural Network (20 points). You may use `sklearn.neural_network.MLPClassifier`.

(a) Fit an artificial neural network to the training data using 1 hidden layer of 20 units as well as another neural network that has 2 hidden layers of 10 units each.

(b) For each model made in (a), make a probabilistic prediction for each validation example. Report the cross-entropies between the predictions and the true labels in your writeup.

(c) Which neural network performs the best on the validation data? Report this in your writeup. Train a new neural network using the architecture that performed better among the two using the training and validation data. Make a probabilistic prediction for each testing example using this model and save them for later.

Part a

Code:

Model 1: 1 layer 20 Units; Model 1: 2 layers, 10 Units Each

```
# Model 1 , 1 Hidden Layer of 20 Units

clf = MLPClassifier(random_state=1, hidden_layer_sizes=(20,), max_iter=300).fit(X_train1,
y_train1)
```

 # Model 2, 2 Hidden Layers with 10 Units Each

```
clf1 = MLPClassifier(random_state=1, hidden_layer_sizes=(10, 10), max_iter=300).fit(X_train1,
y_train1)
```

Part b

Code:

 # Model 1 on Validation Examples, Hidden Layer with 20 Units

```
cr_entropy = []

actual1 = []

for i in range(len(y_val1)):

    a = [0 for j in range(2)]

    a[y_val11[i]] = 1

    actual1.append(a)

probability11 = np.zeros((len(y_val1) ,2)) #Initialize Array

for i in range(len(probability1)):

    cross_entropy =0.0

    for j in range(len(probability1[i])):

        if probability1[i][j] ==0:

            probability11[i][j] = probability1[i][j] + 1e-15

        elif probability11[i][j] ==1:

            probability11[i][j] = probability1[i][j] - 1e-15

        else:

            probability11[i][j] = probability1[i][j]

        cross_entropy += - actual1[i][j]*math.log(probability11[i][j])


    cr_entropy.append(cross_entropy)

sum =0.0

for i  in range(len(y_val11)):
```

```
    sum += cr_entropy[i]
mean = sum/len(cr_entropy)
print("Mean Cross Entropy on Validation Dataset, Model 1")
print(mean)
print("_____")
print("Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 1")
print(log_loss(y_val11, probability1))
print("_____ ")
```

 # Model 2 on Validation Examples, 2 Hidden Layers with 10 Units Each

```
# Model 2 on Validation Examples
probability2 = clf1.predict_proba(X_val1)
cr_entropy1 = []
probability22 = np.zeros((len(y_val1) ,2)) #Initialize Array
for i in range(len(probability2)):
    cross_entropy =0.0
    for j in range(len(probability2[i])):
        if probability2[i][j] ==0:
            probability22[i][j] = probability2[i][j] + 1e-15
        elif probability22[i][j] ==1:
            probability22[i][j] = probability2[i][j] - 1e-15
        else:
            probability22[i][j] = probability2[i][j]
        cross_entropy += - actual1[i][j]*math.log(probability22[i][j])

    cr_entropy1.append(cross_entropy)
sum =0.0
for i  in range(len(cr_entropy1)):
    sum += cr_entropy1[i]
mean = sum/len(y_val11)
print("Mean Cross Entropy on Validation Dataset, Model 2")
print(mean)
print("_____")
print("Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 2")
```

```
print(log_loss(y_val11, probability2))

print("_____________________________________________________________________")
```

Output:

The cross entropy from the function I wrote matches that calculated from log_loss function from Sk Learn.

Mean Cross Entropy on Validation Dataset, Model 1

0.14156957472255843

_____

Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 1

0.14156957472255843

_____

Mean Cross Entropy on Validation Dataset, Model 2

0.16614056110642383

_____

Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 2

0.16614056110642392

Part c:

Model 1 has lower cross entropy than Model 2. It performs Model 2

The cross entropy from the function I wrote matches that calculated from log_loss function from Sk Learn.

Code:

```
clf3 = MLPClassifier(random_state=1, hidden_layer_sizes=(20,), max_iter=300).fit(X_train,
y_train)

probability3 = clf3.predict_proba(X_test)

cr_entropy = []

actual2 = []

y_test1 = y_test.to_numpy()

for i in range(len(y_test)):

    a = [0 for j in range(2)]
```

```
    a[y_test1[i]] = 1
    actual2.append(a)
probability33 = np.zeros((len(y_test) ,2)) #Initialize Array
for i in range(len(probability3)):
   cross_entropy =0.0
   for j in range(len(probability3[i])):
       if probability3[i][j] ==0:
           probability33[i][j] = probability3[i][j] + 1e-15
       elif probability33[i][j] ==1:
           probability33[i][j] = probability3[i][j] - 1e-15
       else:
           probability33[i][j] = probability3[i][j]
       cross_entropy += - actual2[i][j]*math.log(probability33[i][j])

   cr_entropy.append(cross_entropy)
sum =0.0
for i  in range(len(cr_entropy)):
    sum += cr_entropy[i]
mean = sum/len(y_test1)
print("Mean Cross Entropy on Test Dataset, Model 1")
print(mean)
print("_____________________________________________________________________________")
print("Log_Loss from Sk Learn In Built Function on Test Dataset, Model 1")
print(log_loss(y_test1, probability3))
print("_____________________________________________________________________________")
```

Output

Mean Cross Entropy on Test Dataset, Model 1

0.1317989706018745

____

Log_Loss from Sk Learn In Built Function on Test Dataset, Model 1

0.13179897060187495

____

Exercise 3 – Decision Trees (20 points). For this problem you can use the scikit-learn method `sklearn.tree.DecisionTreeClassifier`.

(a) Fit a decision tree to the training data using the Gini impurity index and max tree depth of 5.

(b) Using the model created in part (a) make a probabilistic prediction for each validation example. What is the cross-entropy on these predictions and the true labels? Put this value in your writeup.

(c) Fit a decision tree to the training data using information gain and max tree depth of 5.

(d) Using the model created in part (c) make a probabilistic prediction for each validation example. What is the cross-entropy on these predictions and the true labels? Put this value in your writeup.

(e) Which model performed better on the validation data? Report this in your writeup. Train a new decision tree on the training and validation data using whichever measure created the best model in (a)-(d), with a max tree depth of 5. Make a probabilistic prediction for each testing example and save them for later.

Parts a and b:

`# Model 1 on Validation Examples, Gini Index, Max_Tree Depth 5`

The cross entropy from the function I wrote matches that calculated from log_loss function from Sk Learn.

```
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth = 5)

tree_clf.fit(X_train1, y_train1)


probability1 = tree_clf.predict_proba(X_val1)

cr_entropy1 = []

actual = []

probability11 = np.zeros((len(y_val1) ,2))

for i in range(len(y_val1)):

    a = [0 for j in range(2)]

    a[y_val11[i]] = 1

    actual.append(a)

actual = np.array(actual)

for i in range(len(probability1)):

   cross_entropy =0.0

   for j in range(len(probability1[i])):

       if probability1[i][j] ==0:
```

```
            probability11[i][j] = probability1[i][j] + 1e-15
        elif probability1[i][j] ==1:
            probability11[i][j] = probability1[i][j] - 1e-15
        else:
         probability11[i][j] = probability1[i][j]

        cross_entropy += - actual[i][j]*math.log(probability11[i][j])

   cr_entropy1.append(cross_entropy)
sum =0.0
for i  in range(len(cr_entropy1)):
    sum += cr_entropy1[i]
mean = sum/len(y_val11)
print("Mean Cross Entropy on Validation Dataset, Model 1")
print(mean)
print("_____________________________________________________________________")
print("Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 1")
print(log_loss(y_val11, probability1))
print("_____________________________________________________________________")
```

Output:

Mean Cross Entropy on Validation Dataset, Model 1

0.5154716245147285

____

Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 1

0.5154716245147339

____

Parts cand d:

 # Model 2 on Validation Examples, Information Gain, Max_Tree_Depth = 5

```
# Model 2 on Validation Examples
tree_clf1 = DecisionTreeClassifier(criterion='entropy', max_depth = 5)
```

```
tree_clf1.fit(X_train1, y_train1)
probability2 = tree_clf1.predict_proba(X_val1)


cr_entropy2 = []
actual = []
probability22 = np.zeros((len(y_val1) ,2))
for i in range(len(y_val1)):
    a = [0 for j in range(2)]
    a[y_val11[i]] = 1
    actual.append(a)


actual = np.array(actual)
for i in range(len(probability2)):
   cross_entropy =0.0
   for j in range(len(probability2[i])):
      if probability2[i][j] ==0:
          probability22[i][j] = probability2[i][j] + 1e-15
      elif probability2[i][j] ==1:
          probability22[i][j] = probability2[i][j] - 1e-15
      else:
          probability22[i][j] = probability2[i][j]


      cross_entropy += - actual[i][j]*math.log(probability22[i][j])


   cr_entropy2.append(cross_entropy)
sum =0.0
for i  in range(len(cr_entropy2)):
    sum += cr_entropy2[i]
mean = sum/len(y_val11)
print("Mean Cross Entropy on Validation Dataset, Model 2, Validation Dataset")
print(mean)
print("_____________________________________________________________________")
print("Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 2")
```

```
print(log_loss(y_val11, probability2))
print("_____________________________________________________________________")
```

Output:

Mean Cross Entropy on Validation Dataset, Model 2, Validation Dataset

0.5853730429530862

____

Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 2

0.5853730429530868

____

Part e:

Model 1 Performed Better than Model 2

The cross entropy from the function I wrote matches that calculated from log_loss function from Sk Learn.

```
# Model 1 outperforms Model 2 on Cross Entropy Loss
# So Model 1 is to Train on Train+Validation Data
tree_clf.fit(X_train1, y_train1)
probability3 = tree_clf.predict_proba(X_test)
cr_entropy = []
actual = []
probability33 = np.zeros((len(y_test) ,2))
y_test1 = y_test.to_numpy()
for i in range(len(y_test)):
    a = [0 for j in range(2)]
    a[y_test1[i]] = 1
    actual.append(a)
for i in range(len(probability3)):
   cross_entropy =0.0
   for j in range(len(probability3[i])):
       if probability3[i][j] ==0:
           probability33[i][j] = probability3[i][j] + 1e-15
```

```
        elif probability3[i][j] ==1:
            probability33[i][j] = probability3[i][j] - 1e-15
        else:
            probability33[i][j] = probability3[i][j]
        cross_entropy += - actual[i][j]*math.log(probability33[i][j])

    cr_entropy.append(cross_entropy)
sum =0.0
for i  in range(len(cr_entropy)):
    sum += cr_entropy[i]
mean = sum/len(cr_entropy)
print("Mean Cross Entropy on Test Dataset, Model 1")
print("Mean Cross Entropy on Test Dataset")
print(mean)
print("_____________________________________________________________________")
print("Log_Loss from Sk Learn In Built Function on Test Dataset, Model 1")
print(log_loss(y_test1, probability3))
print("_____________________________________________________________________")
```

Output:

Mean Cross Entropy on Test Dataset, Model 1

Mean Cross Entropy on Test Dataset

0.4482956294505133

Log_Loss from Sk Learn In Built Function on Test Dataset, Model 1

0.44829562945051726

Exercise 4 – Boosting (20 points). For this problem you may use `sklearn.ensemble.AdaBoostClassifier`.

(a) Fit boosted decision stumps (max tree depth of 1) to the training data allowing at most 20, 40, and 80 decision stumps (base estimators) in each model.

(b) For each model trained in (a), make a probabilisitc prediction for each validation example. Report the cross-entropies between the predictions and the true labels in your writeup.

(c) Which upper bound on the number of allowed base classifiers generates the best performing model? Report this in your writeup. Train a new AdaBoost classifier using this bound on the number of maximum allowed base classifiers, using the training and validation data. Make a probabilistic prediction for each testing example using this model and save them for later.

Parts a and b:

Code:

```
# Model 1

# Max_depth = 1, n_estimators = 20

ada_clf1 = AdaBoostClassifier(DecisionTreeClassifier(max_depth =1), n_estimators = 20)

ada_clf1.fit(X_train1, y_train1)

y_val11 = y_val1.to_numpy()

# Model 1 on Validation Examples

probability1 = ada_clf1.predict_proba(X_val1)

cr_entropy = []

actual = []

for i in range(len(y_val1)):

    a = [0 for j in range(2)]

    a[y_val11[i]] = 1

    actual.append(a)

actual = np.array(actual)

probability11 = np.zeros((len(y_val1) ,2))

for i in range(len(probability1)):

   cross_entropy =0.0

   for j in range(len(probability1[i])):

       if probability1[i][j] ==0:

           probability11[i][j] = probability1[i][j] + 1e-15
```

```
        elif probability1[i][j] ==1:
            probability11[i][j] = probability1[i][j] - 1e-15
        else:
            probability11[i][j] = probability1[i][j]
        cross_entropy += - actual[i][j]*math.log(probability11[i][j])

    cr_entropy.append(cross_entropy)
sum =0.0
for i  in range(len(cr_entropy)):
    sum += cr_entropy[i]
mean = sum/len(y_val11)
print("Mean Cross Entropy on Validation Dataset, Model 1")
print(mean)
print("_____________________________________________________________________")
print("Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 1")
print(log_loss(y_val11, probability1))
print("_____________________________________________________________________")
```

 # Model 2

 # Max_depth = 1, n_estimators = 40

```
ada_clf2 = AdaBoostClassifier(DecisionTreeClassifier(max_depth =1), n_estimators = 40)
ada_clf2.fit(X_train1, y_train1)
y_val11 = y_val1.to_numpy()
probability2 = ada_clf2.predict_proba(X_val1)
cr_entropy1 = []
actual = []
probability22 = np.zeros((len(y_val1) ,2))
for i in range(len(y_val1)):
    a = [0 for j in range(2)]
    a[y_val11[i]] = 1
    actual.append(a)
actual = np.array(actual)
for i in range(len(probability2)):
```

```
    cross_entropy =0.0
    for j in range(len(probability2[i])):
        if probability2[i][j] ==0:
            probability22[i][j] = probability2[i][j] + 1e-15
        elif probability2[i][j] ==1:
            probability22[i][j] = probability2[i][j] - 1e-15
        else:
            probability22[i][j] = probability2[i][j]
        cross_entropy += - actual[i][j]*math.log(probability22[i][j])

    cr_entropy1.append(cross_entropy)
sum =0.0
for i  in range(len(cr_entropy1)):
    sum += cr_entropy1[i]
mean = sum/len(y_val11)
print("Mean Cross Entropy on Validation Dataset, Model 2")
print(mean)
print("_____________________________________________________________________________")
print("Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 2")
print(log_loss(y_val11, probability2))
print("_____________________________________________________________________________")
```

 # Model 3

 # Max_depth = 1, n_estimators = 80

```
ada_clf3 = AdaBoostClassifier(DecisionTreeClassifier(max_depth =1), n_estimators = 80)
ada_clf3.fit(X_train1, y_train1)
y_val11 = y_val1.to_numpy()
# Model 1 on Validation Examples
probability3 = ada_clf3.predict_proba(X_val1)
cr_entropy2 = []
actual = []
probability33 = np.zeros((len(y_val1) ,2))
for i in range(len(y_val1)):
```

```
    a = [0 for j in range(2)]
    a[y_val11[i]] = 1
    actual.append(a)
actual = np.array(actual)
for i in range(len(probability3)):
   cross_entropy =0.0
   for j in range(len(probability3[i])):
       if probability3[i][j] ==0:
           probability33[i][j] = probability3[i][j] + 1e-15
       elif probability3[i][j] ==1:
           probability33[i][j] = probability3[i][j] - 1e-15
       else:
           probability33[i][j] = probability3[i][j]
       cross_entropy += - actual[i][j]*math.log(probability33[i][j])


   cr_entropy2.append(cross_entropy)
sum =0.0
for i  in range(len(cr_entropy2)):
    sum += cr_entropy2[i]
mean = sum/len(y_val11)
print("Mean Cross Entropy on Validation Dataset, Model 3")
print(mean)
print("_____")
print("Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 3")
print(log_loss(y_val11, probability3))
print("_____")
```

Output

The cross entropy from the function I wrote matches that calculated from log_loss function from Sk Learn.

Mean Cross Entropy on Validation Dataset, Model 1

0.6418350552597027

Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 1

0.6418350552597037

Mean Cross Entropy on Validation Dataset, Model 2

0.6591349289029242

Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 2

0.6591349289029239

Mean Cross Entropy on Validation Dataset, Model 3

0.6738041118677622

Log_Loss from Sk Learn In Built Function on Validation Dataset, Model 3

0.6738041118677613

Part c

Model 1 has the lowest cross entropy of all the models. Model 1 has 20 base classifiers.

Code:

```
ada_clf1.fit(X_train, y_train)
y_test1 = y_test.to_numpy()
probability4 = ada_clf2.predict_proba(X_test)
cr_entropy1 = []
actual = []
probability44 = np.zeros((len(y_test) ,2))
for i in range(len(y_test)):
    a = [0 for j in range(2)]
    a[y_test1[i]] = 1
    actual.append(a)
actual = np.array(actual)
for i in range(len(probability4)):
   cross_entropy =0.0
   for j in range(len(probability4[i])):
       if probability4[i][j] ==0:
           probability44[i][j] = probability4[i][j] + 1e-15
       elif probability4[i][j] ==1:
           probability44[i][j] = probability4[i][j] - 1e-15
       else:
           probability44[i][j] = probability4[i][j]
       cross_entropy += - actual[i][j]*math.log(probability44[i][j])

   cr_entropy1.append(cross_entropy)
sum =0.0
for i  in range(len(cr_entropy1)):
    sum += cr_entropy1[i]
```

```
mean = sum/len(y_test1)

print("Mean Cross Entropy on Test Dataset, Model 1")

print(mean)

print("_____")

print("Log_Loss from Sk Learn In Built Function on Test Dataset, Model 1")

print(log_loss(y_test1, probability4))

print("_____")
```

Mean Cross Entropy on Test Dataset, Model 1

0.6603635609800996

Log_Loss from Sk Learn In Built Function on Test Dataset, Model 1

0.6603635609800995

Exercise 5 – ROC Curve (30 points). For this exercise **you must write your own code; no scikit-learn, except maybe to compute AUC.**
For each model produced in Exercises 2-4 do the following:

(a) Determinize the testing predictions made above, using 1001 different probability thresholds (0.000, 0.001, 0.002, . . ., 0.999, 1.000). "Determinization" means converting the probability to a deterministic class label (0 or 1). Use (1) below for determinization. We have that p^* is the critical threshold; p_i is the predicted probability for example i; and P_i is the resulting deterministic prediction:

$$P_i = \begin{cases} 1, & \text{if } p_i \geq p^* \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

(b) At each of the 1001 probability thresholds, compute the true positive rate (TPR) and false positive rate (FPR). Recall that these values are easily computed from the confusion matrix. (You would have to re-calculate the confusion matrix for each one of these thresholds, for each model.)

(c) Plot the ROC (receiver operating characteristic) curve, using the 1001 points created in part 6b. If you have forgotten what a ROC curve looks like, see our notes on model evaluation. The ROC curve **must** contain a point at the bottom left (0, 0) and top right (1, 1). Also, it must contain the dashed grey line, indicating the performance of a random predictor. Include the ROC curve for each model in your write-up.

(d) Find the probability threshold yielding the highest Youden index (TPR - FPR). Report the Youden index and the corresponding probability threshold for each model.

(e) Compute the AUC (area under the curve) for each model. You may use the function `sklearn.metrics.roc_auc_score` for this part.

Part a:

Example code for one of the models is provided. Rest can be found in the text files.

```
# Model 1 Prediction of Labels Based on Probability Thresholds
predictions1 = []
for i in range(0,1001,1):
    k = float(i/1000)
    label1 = []
    for j in range(len(probability1)):

        if probability1[j][1] > k:
            label1.append(1)
        else:
            label1.append(0)
    predictions1.append(label1)
```

Part b:

Example code for one of the models is provided. Rest can be found in the text files.

```
# compute True Positive Rate and True Negative Rate
# Model 1 Prediction of Labels Based on Probability Thresholds on Validation Dataset

TPR1 = []
FPR1 = []
Youden_Index1 = 0
for i in range(0,1001, 1):
    k = float(i/1000)
    tn = 0
    tp = 0
    fn = 0
    fp = 0
    for j in range(len(predictions1[i])):
```

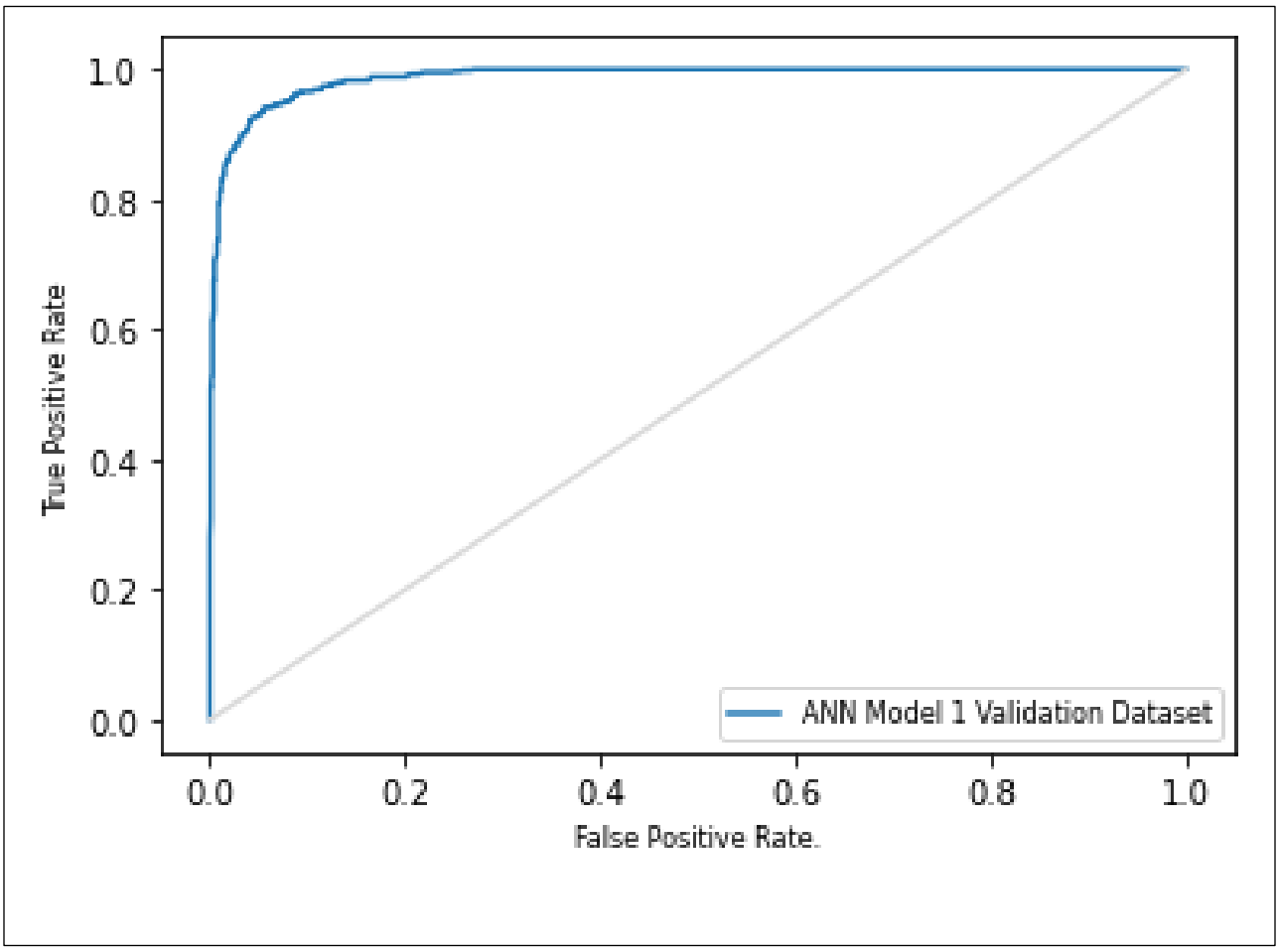
```
        if predictions1[i][j] ==0:
            if y_val11[j] == 0:
                tn += 1
            else:
                fn += 1

        if predictions1[i][j] == 1:
            if y_val11[j]  == 1:
                tp += 1
            else:
                fp += 1

    if (tp+fn) != 0:
        true_positive_rate1 = tp/(tp+fn)
    else:
        true_positive_rate1 =0




    if (tn +fp) !=0:
        false_positive_rate1 = fp/(tn+fp)
    else:
        false_positive_rate1 = 0

    if (true_positive_rate1 - false_positive_rate1) > Youden_Index1:
        Youden_Index1 = (true_positive_rate1 - false_positive_rate1)



    TPR1.append(true_positive_rate1)
    FPR1.append(false_positive_rate1)

plt.plot(FPR1, TPR1)
plt.plot(RX, RY, c='0.85')
```

```
plt.xlabel("False Positive Rate.", size = 8,)
plt.ylabel("True Positive Rate", size = 8)
plt.legend(["Model 1 Validation Dataset"], loc ="lower right", prop = {'size': 8})
plt.show()
print("Highest Youden_Index for Model 1 Validation Dataset is:")
print(Youden_Index1)
```

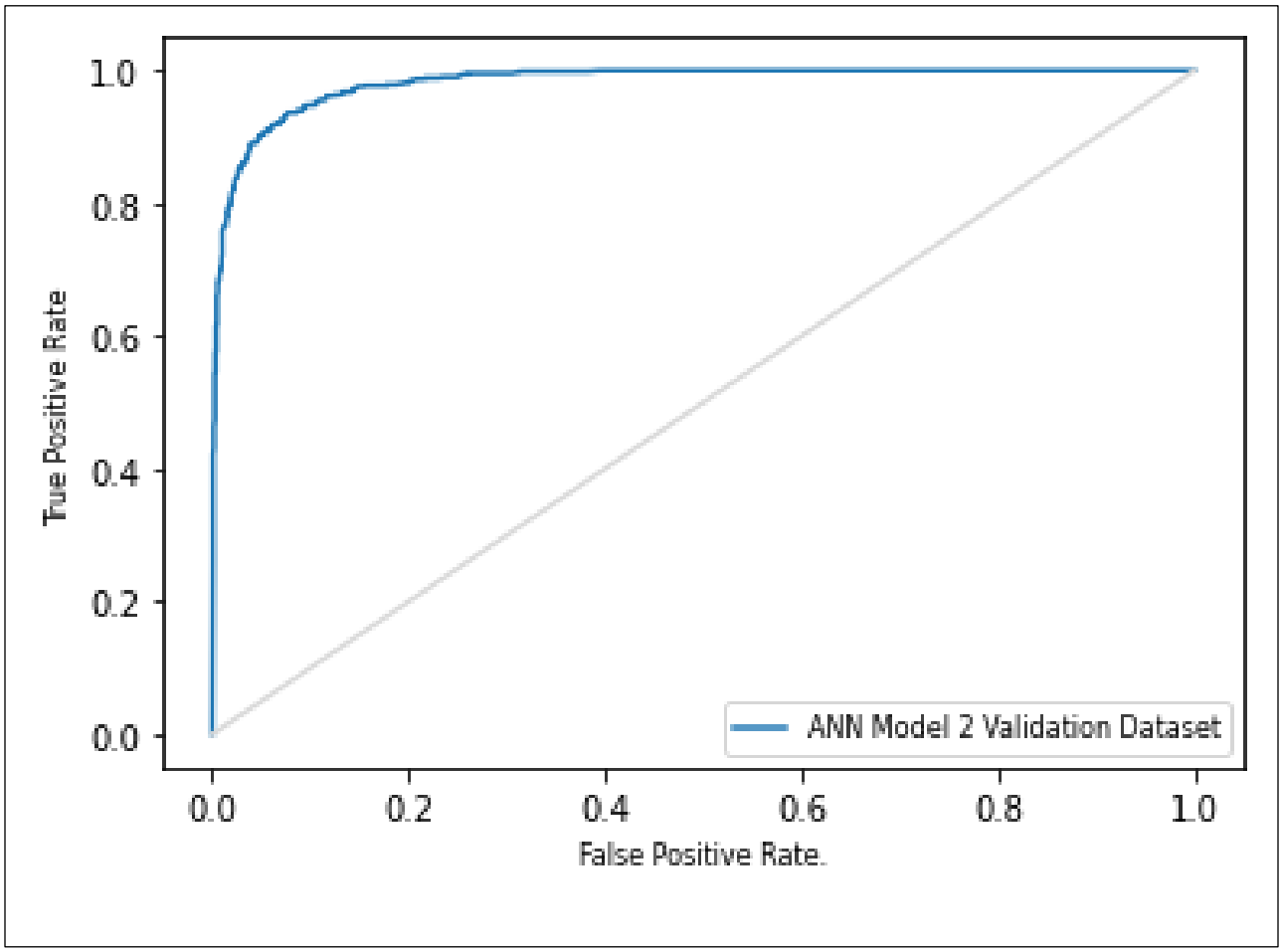
Part c:

Artificial Neural Networks

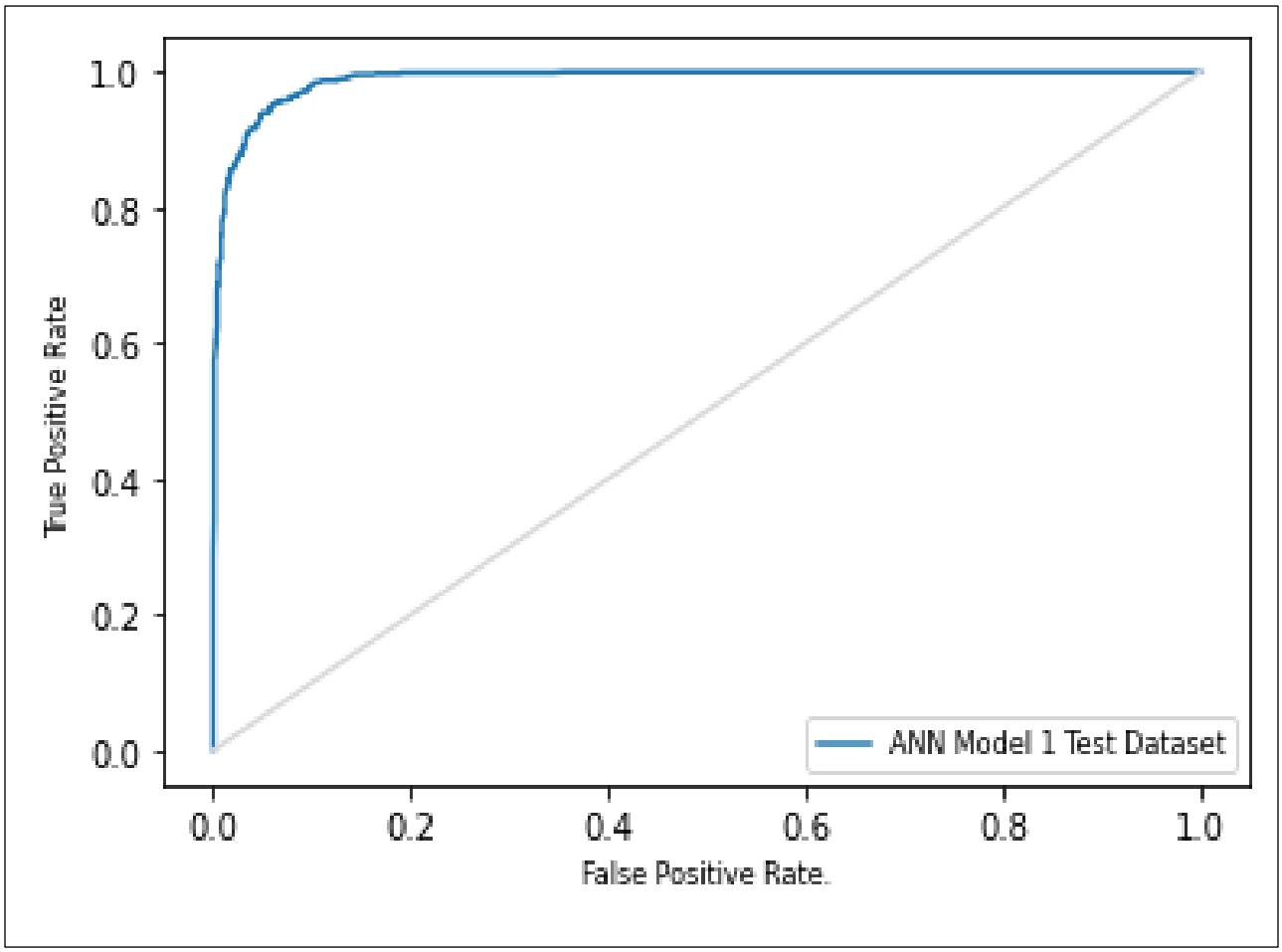
 # Model 1 on Validation Examples, Hidden Layer with 20 Units



 # Model 2 on Validation Examples, 2 Hidden Layers with 10 Units Each

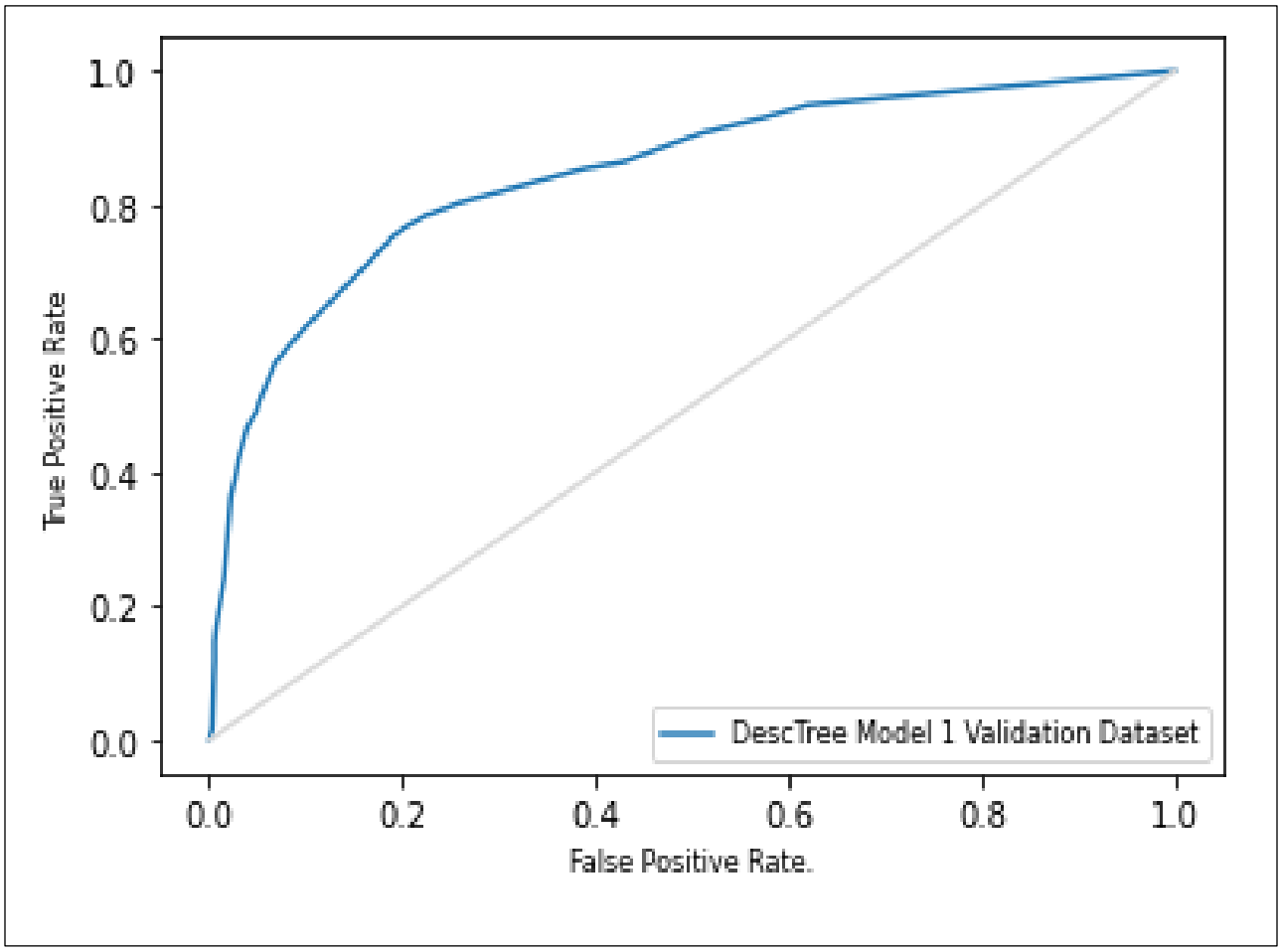


```
# Model 1 Test Dataset
```

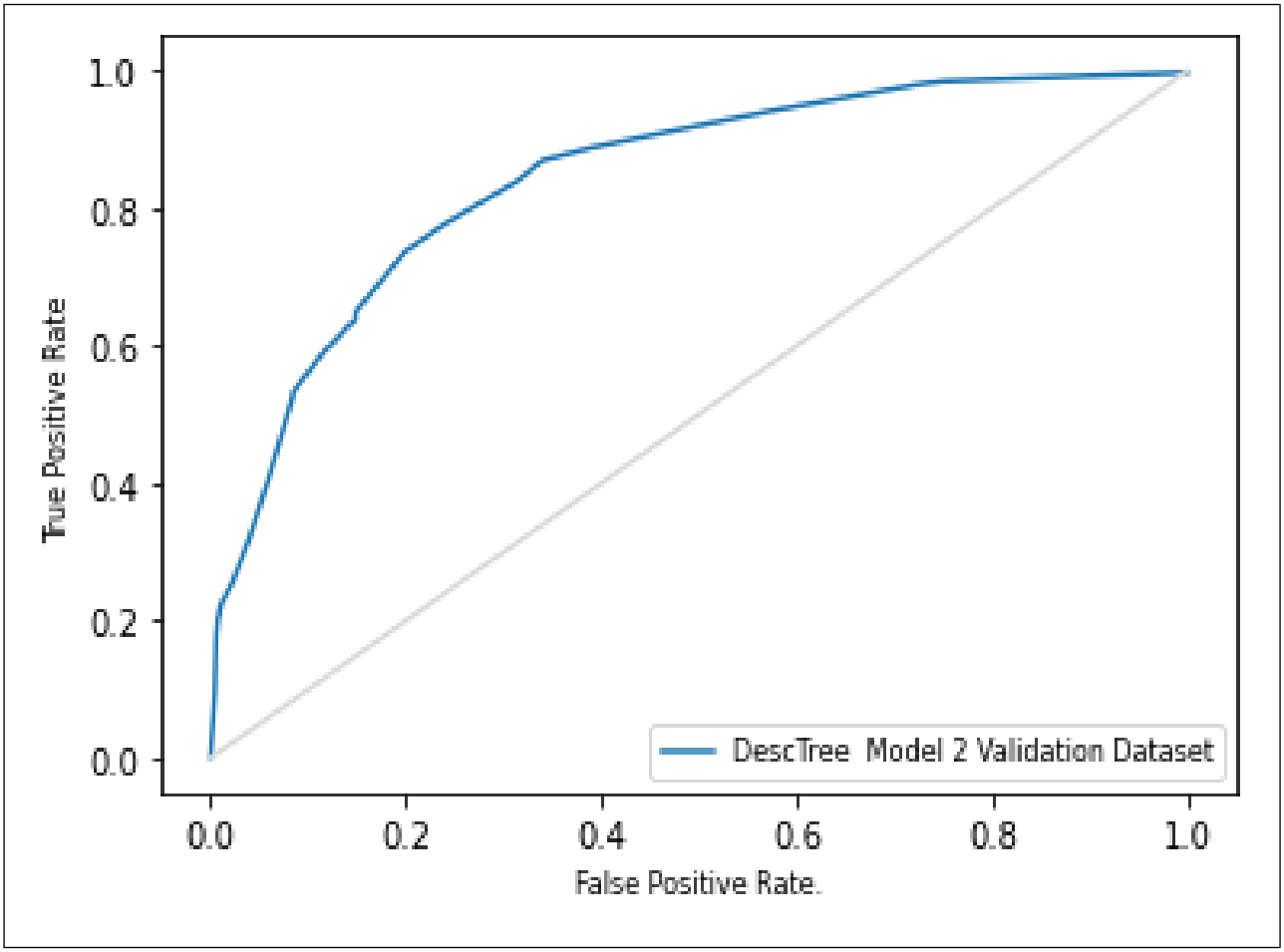


Decision Trees

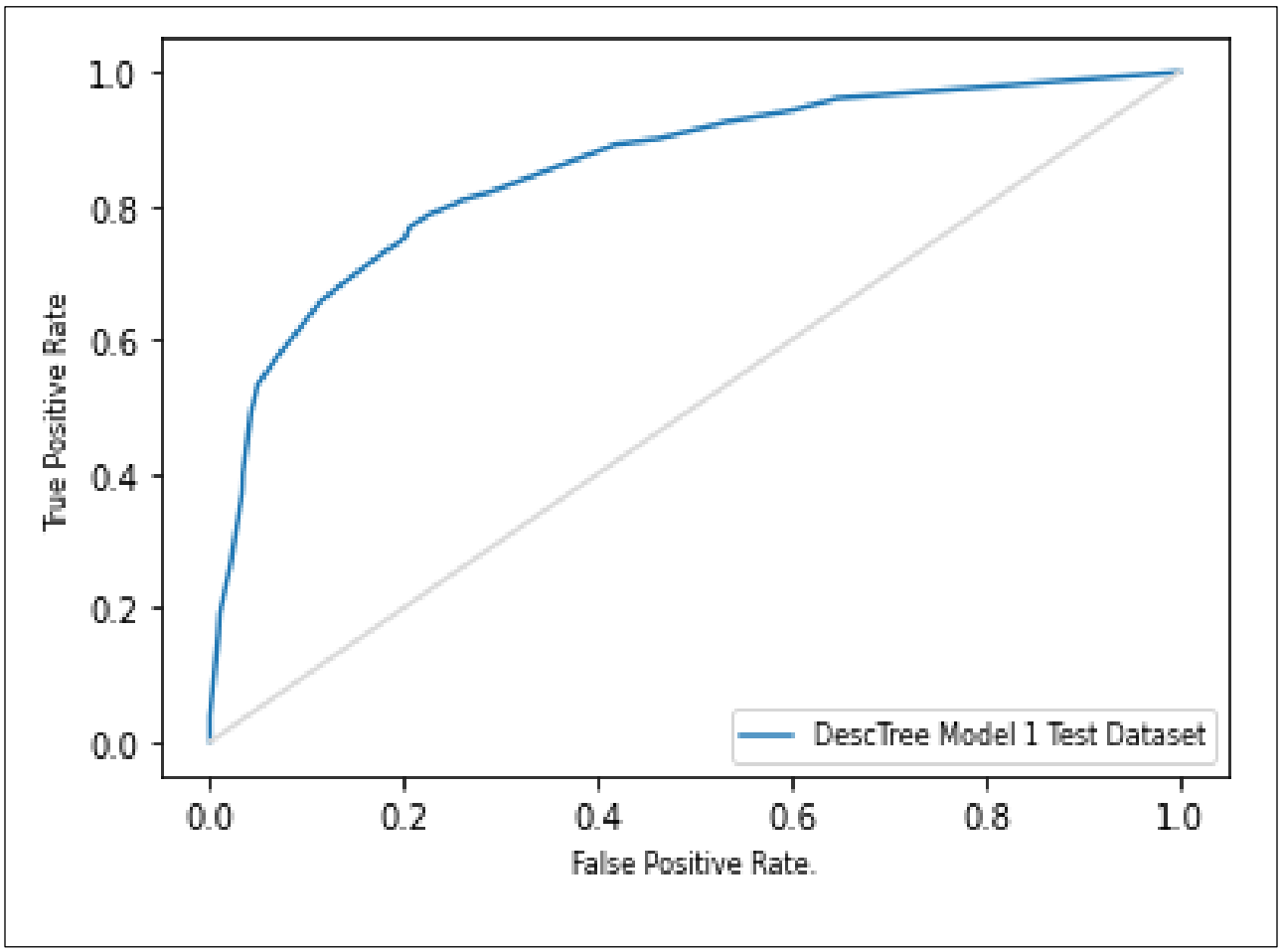
```
# Model 1 on Validation Examples, Gini Index, Max_Tree Depth 5
```



 # Model 2 on Validation Examples, Information Gain, Max_Tree Depth 5



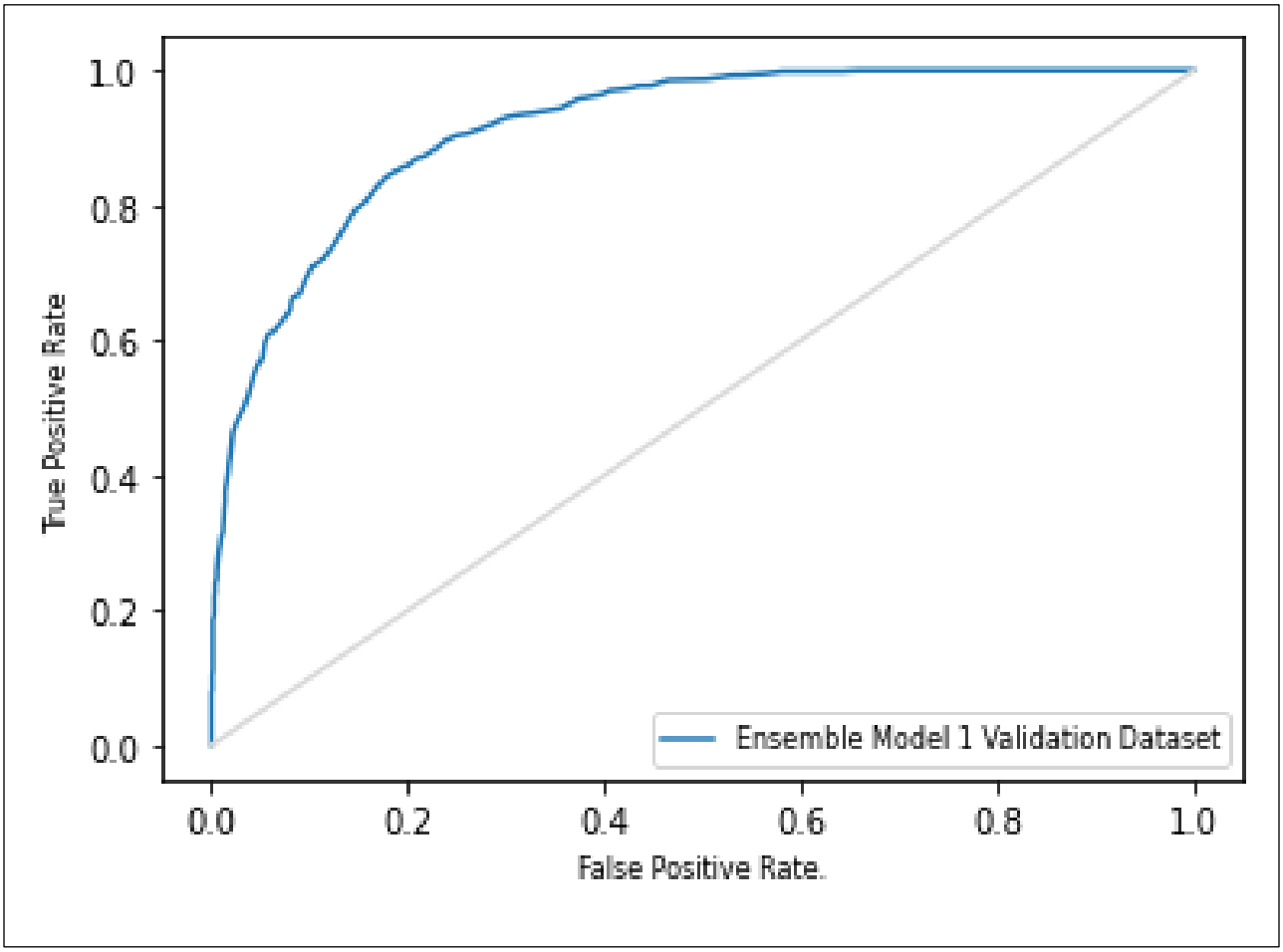
 # Model 1 on Test Dataset, Gini Index, Max_Tree Depth 5



Ensembles

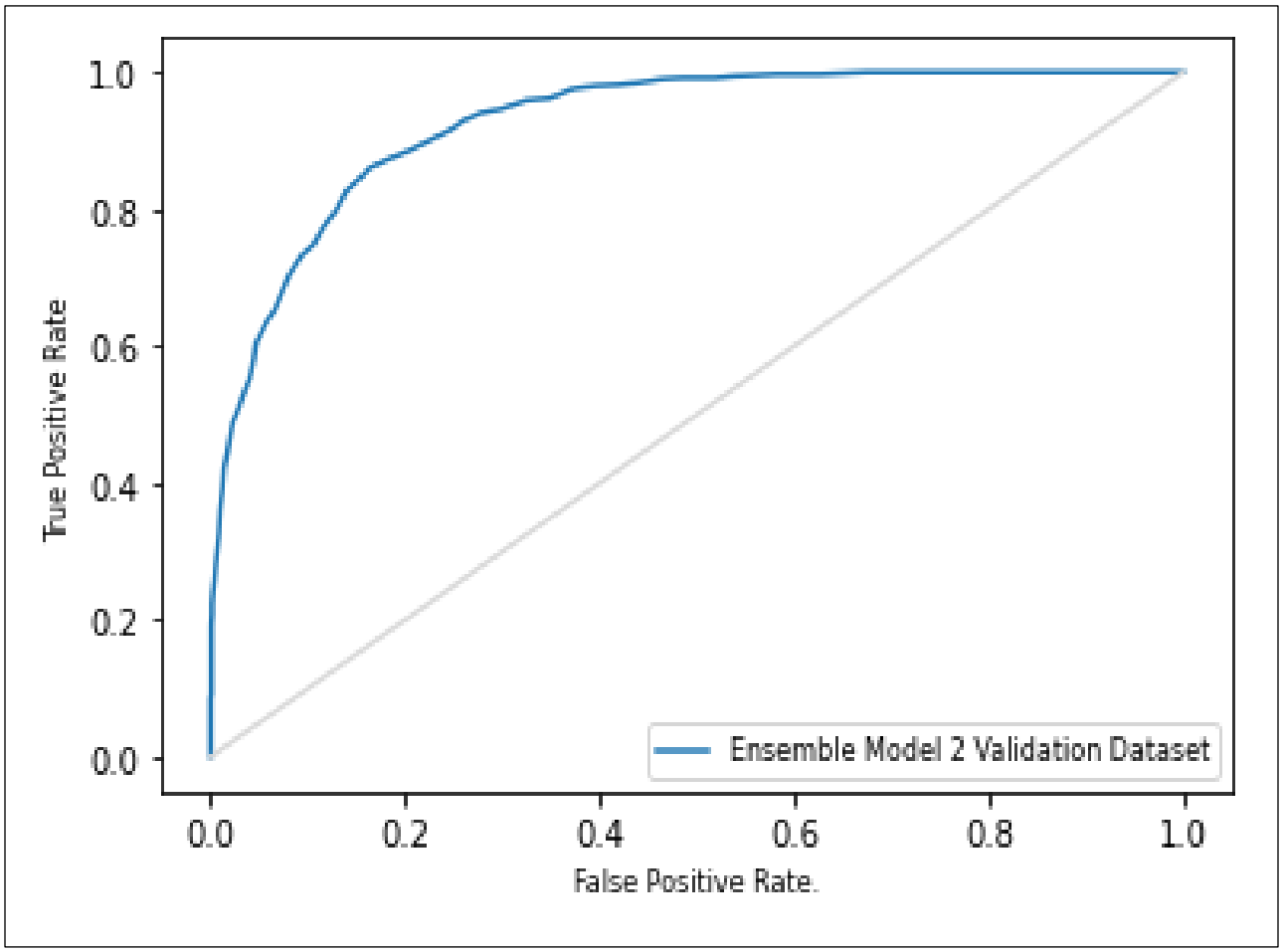
 # Model 1

 # Max_depth = 1, n_estimators = 20



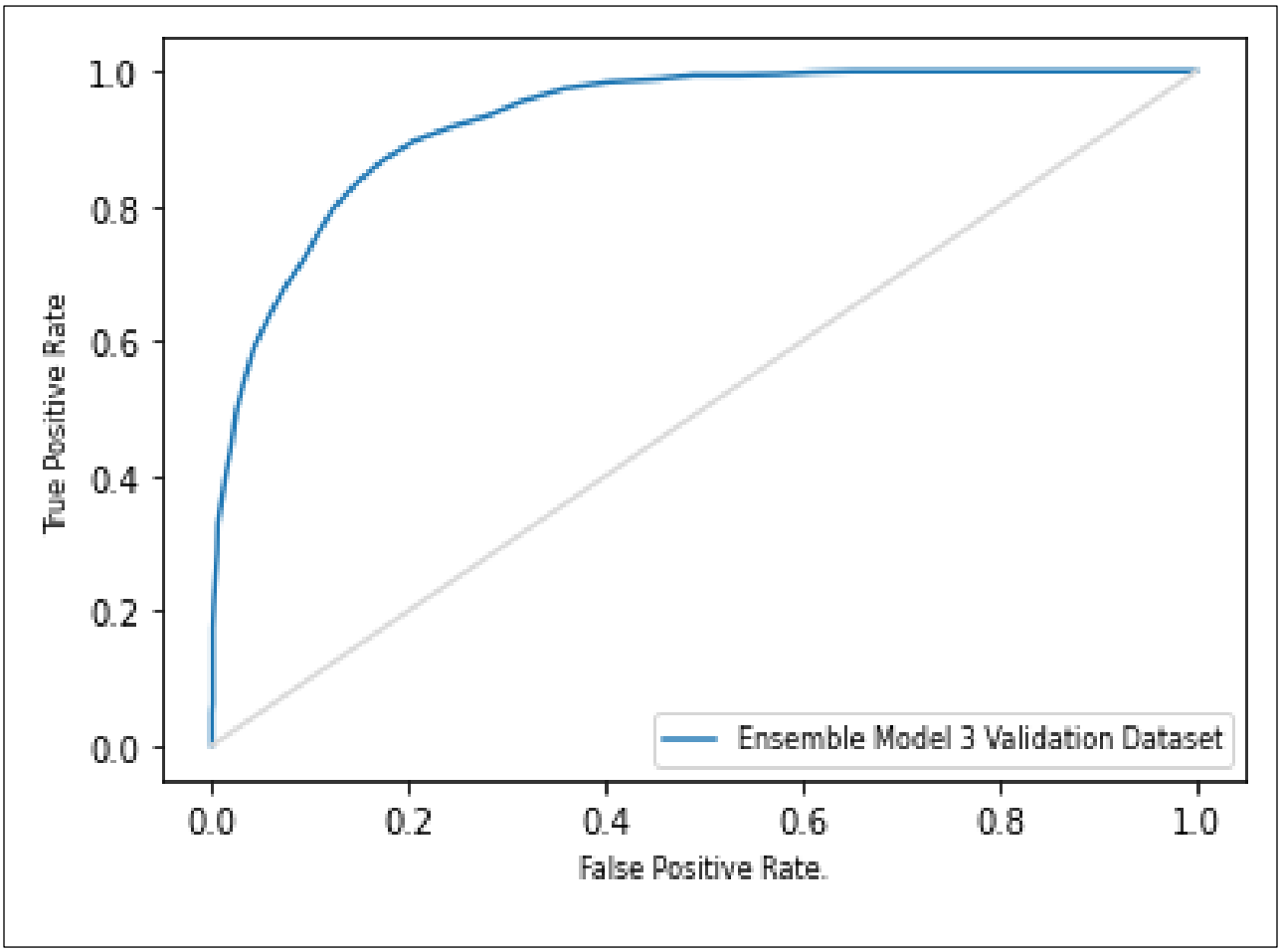
 # Model 2

 # Max_depth = 1, n_estimators = 40



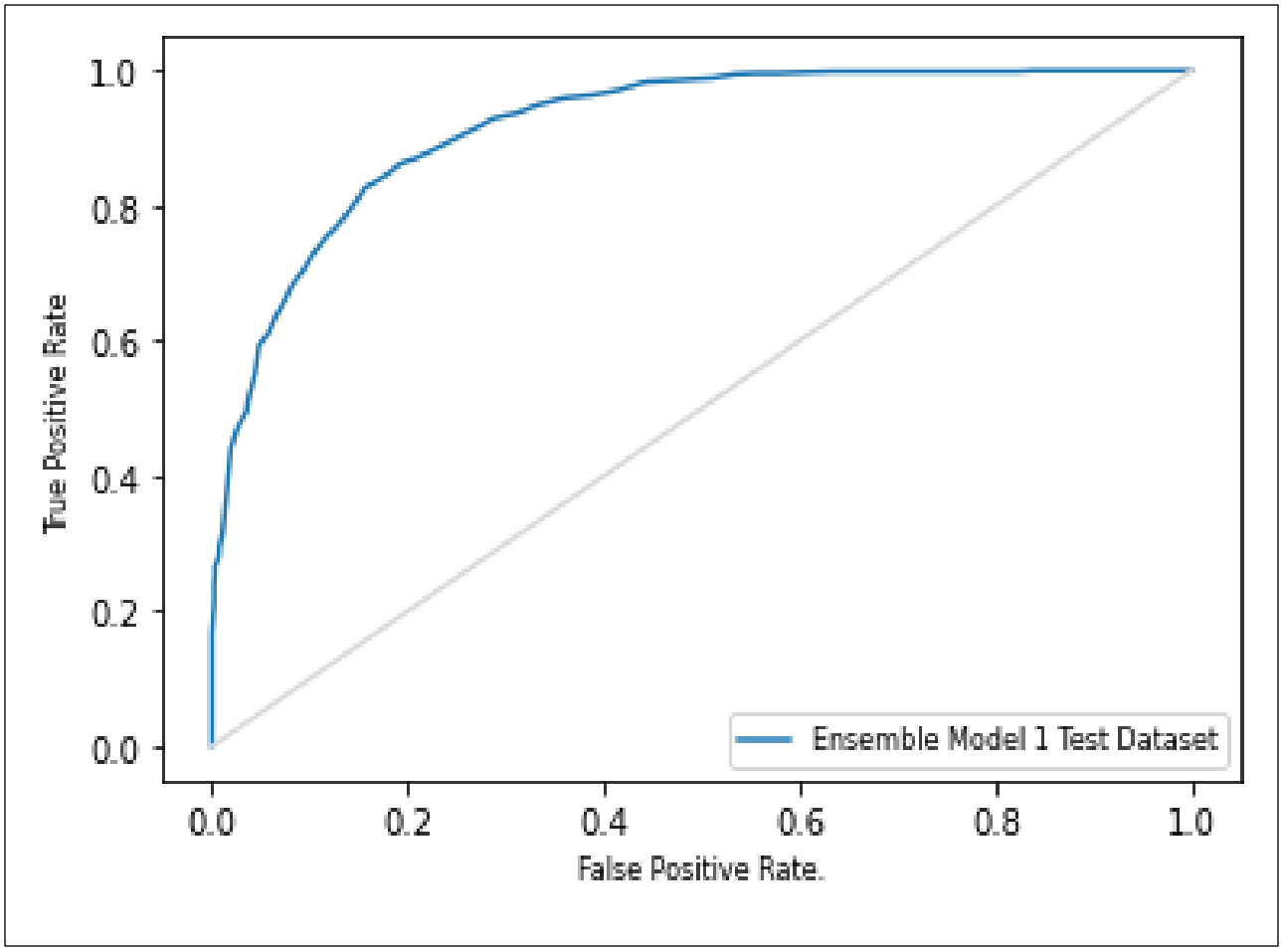
 # Model 3

 # Max_depth = 1, n_estimators = 80



 # Model 1

 # Max_depth = 1, n_estimators = 20, Test Dataset



Part d

Artificial Neural Networks

Probability Threshold with Highest Index 0.409

Highest Youden_Index for Model 1 Validation Dataset is:

0.8851106148370438

Probability Threshold with Highest Index 0.435

Highest Youden_Index for Model 2 Validation Dataset is:

0.8599343987717217

Probability Threshold with Highest Index 0.368

Highest Youden_Index for Model 1 Test Dataset is:

0.8924645823155838

Decision Trees

Probability Threshold with Highest Index 0.288

Highest Youden_Index for Model 1 Validation Dataset is:

0.5635860841649801

Probability Threshold with Highest Index 0.411

Highest Youden_Index for Model 2 Validation Dataset is:

0.53719729220462

Probability Threshold with Highest Index 0.407

Highest Youden_Index for Model 1 Test Dataset is:

0.5618849884848909

Ensembles

Probability Threshold with Highest Index 0.492

Highest Youden_Index for Model 1 Validation Dataset is:

0.6642735012910881

Probability Threshold with Highest Index 0.495

Highest Youden_Index for Model 2 Validation Dataset is:

0.6965070835368832

Probability Threshold with Highest Index 0.497

Highest Youden_Index for Model 3 Validation Dataset is:

0.6940644846116267

Probability Threshold with Highest Index 0.494

Highest Youden_Index for Model 1 Test Dataset is:

0.6698129667108661

Part e

Example code provided here

```
# Area Under Curve Determinations


# for Model 1, Validation Dataset
AUC1 = 0.0
for i in range(len(FPR1)):
    if i == 0:


        prev_coordinate = FPR1[i]
        AUC1 =0.0
    if i >0:


        AUC1 += (1/2)* (TPR1[i] +TPR1[i])*(-FPR1[i] + prev_coordinate)
    prev_coordinate = FPR1[i]
print("AUC from Fuction Built from Raw FPR and TPR Data, Model 1 Validation Dataset")


print(AUC1)
print("_____________________________________________________________________________")
print("AUC from In Built Function, Model 1 Validation Dataset")
auc = roc_auc_score(y_val11, probability1[:,1])
print(auc)
print("_____________________________________________________________________________")
```

Artificial Neural Networks

AUC from Function Built from Raw FPR and TPR Data, Model 1 Validation Dataset

0.9873159327238473

AUC from In Built Function, Model 1 Validation Dataset

0.9873715454672343

AUC from Function Built from Raw FPR and TPR Data, Model 2 Validation Dataset

0.9820556214669574

AUC from In Built Function, Model 2 Validation Dataset

0.9822671679810174

AUC from Function Built from Raw FPR and TPR Data, Model 1 Test Dataset

0.9889439685253697

AUC from In Built Function, Model 1 Test Dataset

0.9890224806336799

Decision Trees

AUC from Function Built from Raw FPR and TPR Data, Model 1 Validation Dataset

0.8257870838858259

AUC from In Built Function, Model 1 Validation Dataset

0.8465513556424035

AUC from Function Built from Raw FPR and TPR Data, Model 2 Validation Dataset

0.8266485361853584

AUC from Function Built from Raw FPR and TPR Data, Model 2 Validation Dataset

0.8432756560122827

AUC from Function Built from Raw FPR and TPR Data, Model 1 Test Dataset

0.8344615814083326

AUC from Function Built from Raw FPR and TPR Data, Model 1 Test Dataset

0.8531932619163933

Ensembles

AUC from Function Built from Raw FPR and TPR Data, Model 1 Validation Dataset

0.9117098628655176

AUC from In Built Function, Model 1 Validation Dataset

0.9141104796217461

AUC from Function Built from Raw FPR and TPR Data, Model 2 Validation Dataset

0.9229981593272384

AUC from In Built Function, Model 2 Validation Dataset

0.92669149975574

AUC from Function Built from Raw FPR and TPR Data, Model 3 Validation Dataset

0.9213952037825391

AUC from In Built Function, Model 3 Validation Dataset

0.9271974666759717

AUC from Function Built from Raw FPR and TPR Data, Model 1 Test Dataset

0.9139169254658381

AUC from In Built Function, Model 1 Test Dataset

0.9176735117593692
