DSCI5340_HW3_Group2

November 8, 2023

[ ]:
```
pip install dmba
```

[ ]:
```
pip install ISLP
```

1 Importing Libraries

[ ]:
```
# Import required packages
from pathlib import Path
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier
import matplotlib.pylab as plt
from sklearn.metrics import confusion_matrix
from ISLP import load_data, confusion_table
import dmba
import warnings
warnings.filterwarnings("ignore", message="X has feature names, but␣
 ↪KNeighborsClassifier was fitted without feature names")

%matplotlib inline
```

2 Importing Dataset

[ ]:
```
univ_df = dmba.load_data('UniversalBank.csv')
univ_df['Number'] = univ_df.index+1
univ_df = univ_df.drop(columns = ['ID','ZIP Code'])
univ_df.head()
```

[ ]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal Loan	\
0	25	1	49	4	1.6	1	0	0	
1	45	19	34	3	1.5	1	0	0	
2	39	15	11	1	1.0	1	0	0	
3	35	9	100	1	2.7	2	0	0	

```
4   35           8      45       4    1.0          2         0              0

   Securities Account  CD Account  Online  CreditCard  Number
0                   1           0       0           0       1
1                   1           0       0           0       2
2                   0           0       0           0       3
3                   0           0       0           0       4
4                   0           0       0           1       5
```

[ ]:
```
univ_df = pd.get_dummies(univ_df, columns=['Education'], prefix='Education')
univ_df.head()
```

[ ]:
```
   Age  Experience  Income  Family  CCAvg  Mortgage  Personal Loan  \
0   25           1      49       4    1.6         0              0
1   45          19      34       3    1.5         0              0
2   39          15      11       1    1.0         0              0
3   35           9     100       1    2.7         0              0
4   35           8      45       4    1.0         0              0

   Securities Account  CD Account  Online  CreditCard  Number  Education_1  \
0                   1           0       0           0       1            1
1                   1           0       0           0       2            1
2                   0           0       0           0       3            1
3                   0           0       0           0       4            0
4                   0           0       0           1       5            0

   Education_2  Education_3
0            0            0
1            0            0
2            0            0
3            1            0
4            1            0
```

2.1 1. Partition the data into training (75%) and validation (25%) sets.

[ ]:
```
import numpy as np
trainData, validData = train_test_split(univ_df, test_size=0.25,␣
 ↪random_state=123)
print(trainData.shape, validData.shape)
```

```
(3750, 15) (1250, 15)
```

[ ]:
```
trainData.head()
```

[ ]:
```
      Age  Experience  Income  Family  CCAvg  Mortgage  Personal Loan  \
2413   60          34      31       2    1.0         0              0
1471   52          26     180       1    1.0         0              0
1196   37          13      71       2    2.7        94              0
```

```
1509   56          26      92       2    4.5         0              0
4110   66          41      59       3    2.4         0              0

      Securities Account  CD Account  Online  CreditCard  Number  Education_1  \
2413                   0           0       0           0    2414            0
1471                   0           0       1           1    1472            1
1196                   0           0       1           0    1197            1
1509                   1           0       0           1    1510            0
4110                   0           0       0           0    4111            1

      Education_2  Education_3
2413            0            1
1471            0            0
1196            0            0
1509            0            1
4110            0            0
```

2. Consider the following customer for classification: Age = 40, Experience = 10, Income = 84, Family = 2, CCAvg = 2, Education_1 = 0, Education_2 = 1, Education_3 = 0, Mortgage = 0, Securities Account = 1, CD Account = 1, Online = 1, and Credit Card = 1.

3 Record to be classified

Consider the following customer for classification: Age = 40, Experience = 10, Income = 84, Family = 2, CCAvg = 2, Education_1 = 0, Education_2 = 1, Education_3 = 0, Mortgage = 0, Securities Account = 1, CD Account = 1, Online = 1, and Credit Card = 1.

[ ]:
```
new_univ_rec = pd.DataFrame([{'Age' : 40, 'Experience' : 10, 'Income' : 84,␣
 ↪'Family' : 2, 'CCAvg' : 2, 'Mortgage' : 0, 'Securities Account' : 1, 'CD␣
 ↪Account' : 1, 'Online' : 1, 'CreditCard' : 1,  'Education_1' : 0,␣
 ↪'Education_2' : 1, 'Education_3' : 0}])
new_univ_rec
```

[ ]:
```
   Age  Experience  Income  Family  CCAvg  Mortgage  Securities Account  \
0   40          10      84       2      2         0                   1

   CD Account  Online  CreditCard  Education_1  Education_2  Education_3
0           1       1           1            0            1            0
```

4 3. Standardize all the data sets using mean and standard deviations.

[ ]:
```
scaler = preprocessing.StandardScaler()
scaler.fit(trainData[['Age', 'Experience', 'Income', 'Family', 'CCAvg',␣
 ↪'Mortgage', 'Securities Account',
```

```
'CD Account', 'Online', 'CreditCard']])

# Transform the full dataset
univNorm = pd.concat([pd.DataFrame(scaler.transform(univ_df[['Age',␣
 ↪'Experience', 'Income', 'Family', 'CCAvg',
'Mortgage', 'Securities Account', 'CD Account', 'Online', 'CreditCard']]),
                                    columns=['zAge', 'zExperience', 'zIncome',␣
 ↪'zFamily', 'zCCAvg', 'zMortgage',
                                    'zSecurities Account', 'zCD Account',␣
 ↪'zOnline', 'zCreditCard']),
                                    univ_df[['Education_1', 'Education_2',␣
 ↪'Education_3','Personal Loan', 'Number']]], axis=1)

trainNorm = univNorm.iloc[trainData.index]
validNorm = univNorm.iloc[validData.index]


new_univ_rec_scaled = pd.DataFrame(scaler.transform(new_univ_rec[['Age',␣
 ↪'Experience', 'Income', 'Family', 'CCAvg',
                                                                    'Mortgage',␣
 ↪'Securities Account', 'CD Account', 'Online', 'CreditCard']]),
                                   columns=['zAge', 'zExperience', 'zIncome',␣
 ↪'zFamily', 'zCCAvg',
                                            'zMortgage', 'zSecurities Account',␣
 ↪'zCD Account', 'zOnline', 'zCreditCard'])

newunivrecNorm = pd.concat([new_univ_rec_scaled, new_univ_rec[['Education_1',␣
 ↪'Education_2', 'Education_3']]], axis=1)
```

[ ]:
```
univNorm.head(2)
```

[ ]:
```
       zAge  zExperience   zIncome   zFamily    zCCAvg  zMortgage  \
0 -1.787106    -1.677628 -0.541291  1.411536 -0.196898  -0.554029
1 -0.026988    -0.093654 -0.867678  0.536183 -0.253888  -0.554029

   zSecurities Account  zCD Account   zOnline  zCreditCard  Education_1  \
0             2.926829    -0.256801 -1.222027    -0.639308            1
1             2.926829    -0.256801 -1.222027    -0.639308            1

   Education_2  Education_3  Personal Loan  Number
0            0            0              0       1
1            0            0              0       2
```

[ ]:
```
trainNorm.head(2)
```

[ ]:
```
          zAge  zExperience   zIncome   zFamily    zCCAvg  zMortgage  \
2413  1.293100     1.226325 -0.932955 -0.339170 -0.538838  -0.554029
1471  0.589053     0.522336  2.309152 -1.214523 -0.538838  -0.554029

      zSecurities Account  zCD Account   zOnline  zCreditCard  Education_1  \
2413            -0.341667    -0.256801 -1.222027    -0.639308            0
1471            -0.341667    -0.256801  0.818312     1.564190            1

      Education_2  Education_3  Personal Loan  Number
2413            0            1              0    2414
1471            0            0              0    1472
```

[ ]:
```
validNorm.head(2)
```

[ ]:
```
          zAge  zExperience   zIncome  zFamily    zCCAvg  zMortgage  \
2648 -1.699100    -1.765627  1.765174 -0.33917  2.994538  -0.554029
2456  0.765064     0.874330 -0.758882 -0.33917 -0.652817  -0.554029

      zSecurities Account  zCD Account   zOnline  zCreditCard  Education_1  \
2648            -0.341667    -0.256801 -1.222027    -0.639308            1
2456            -0.341667    -0.256801 -1.222027     1.564190            1

      Education_2  Education_3  Personal Loan  Number
2648            0            0              0    2649
2456            0            0              0    2457
```

[ ]:
```
newunivrecNorm
```

[ ]:
```
       zAge  zExperience   zIncome  zFamily    zCCAvg  zMortgage  \
0 -0.467018    -0.885641  0.220278 -0.33917  0.031062  -0.554029

   zSecurities Account  zCD Account   zOnline  zCreditCard  Education_1  \
0             2.926829     3.894072  0.818312      1.56419            0

   Education_2  Education_3
0            1            0
```

5 4. Perform a k-NN classification with all predictors except ID and ZIP code using k = 1. How would this customer be classified?

[ ]:
```
knn = NearestNeighbors(n_neighbors=1)
knn.fit(trainNorm[['zAge', 'zExperience', 'zIncome', 'zFamily', 'zCCAvg',␣
 ↪'zMortgage', 'zSecurities Account', 'zCD Account',
                    'zOnline', 'zCreditCard',  'Education_1','Education_2',␣
 ↪'Education_3']])
```

```
distances, indices = knn.kneighbors(newunivrecNorm)
print(trainNorm.iloc[indices[0], :])
```

```
          zAge  zExperience  zIncome  zFamily    zCCAvg  zMortgage  \
1322 -1.171065    -1.325634 -0.56305 -0.33917 -0.157005  -0.554029

      zSecurities Account  zCD Account   zOnline  zCreditCard  Education_1  \
1322             2.926829     3.894072  0.818312      1.56419            0

      Education_2  Education_3  Personal Loan  Number
1322            1            0              0    1323
```

For K=1, KNN classification has given the personal Loan value = 0 for the classified customer, which is against the class of interest i.e., 1. Hence the classified customer will not take the personal loan.

6 5. Now find the optimal value of k using the validation data set. What is the optimal k?

[ ]:
```
train_X = trainNorm[['zAge', 'zExperience', 'zIncome', 'zFamily', 'zCCAvg',␣
 ↪'zMortgage', 'zSecurities Account',
'zCD Account','zOnline', 'zCreditCard',  'Education_1','Education_2',␣
 ↪'Education_3']]
train_y = trainNorm['Personal Loan']
valid_X = validNorm[['zAge', 'zExperience', 'zIncome', 'zFamily', 'zCCAvg',␣
 ↪'zMortgage', 'zSecurities Account',
'zCD Account', 'zOnline', 'zCreditCard',  'Education_1','Education_2',␣
 ↪'Education_3']]
valid_y = validNorm['Personal Loan']

# Train a classifier for different values of k
results = []
for k in range(1, 15):
    knn = KNeighborsClassifier(n_neighbors=k).fit(train_X.values, train_y.
 ↪values)
    results.append({
        'k': k,
        'accuracy': accuracy_score(valid_y.values, knn.predict(valid_X.values))
    })

# Convert results to a pandas data frame
results = pd.DataFrame(results)
print(results)
```

```
    k  accuracy
0   1    0.9624
1   2    0.9504
```

```
2    3    0.9624
3    4    0.9512
4    5    0.9608
5    6    0.9552
6    7    0.9584
7    8    0.9464
8    9    0.9520
9   10    0.9472
10  11    0.9496
11  12    0.9440
12  13    0.9464
13  14    0.9440
```

From the above output, the highest accuracy is 0.9624 for which K value is 1. Hence the Optimal K value is 1.

7 6. Print the confusion matrix for the validation data that results from using the optimal k.

[ ]:
```
print("Confusion Matrix:")
conf_matrix = confusion_matrix(knn.predict(valid_X.values),valid_y.values)
print(conf_matrix)
```

```
Confusion Matrix:
[[1124   69]
 [   1   56]]
```

[ ]:
```
confusion_table(knn.predict(valid_X.values),valid_y.values)
```

[ ]:
```
Truth         0   1
Predicted
0          1124  69
1             1  56
```

[ ]:
```
accuracy_score(valid_y.values, knn.predict(valid_X.values))
```

[ ]:
```
0.944
```

8 7. Classify the customer specified in Question 2 using the best k.

[ ]:
```
knn = NearestNeighbors(n_neighbors=1)
knn.fit(trainNorm[['zAge', 'zExperience', 'zIncome', 'zFamily', 'zCCAvg',␣
 ↪'zMortgage', 'zSecurities Account',
'zCD Account','zOnline', 'zCreditCard',  'Education_1','Education_2',␣
 ↪'Education_3']])
distances, indices = knn.kneighbors(newunivrecNorm)
```

```
print(trainNorm.iloc[indices[0], :])
```

```
          zAge  zExperience  zIncome  zFamily    zCCAvg  zMortgage  \
1322 -1.171065    -1.325634 -0.56305 -0.33917 -0.157005  -0.554029

      zSecurities Account  zCD Account   zOnline  zCreditCard  Education_1  \
1322             2.926829     3.894072  0.818312      1.56419            0

      Education_2  Education_3  Personal Loan  Number
1322            1            0              0    1323
```

9 For optimal value of K = 1

the value of the personal loan for 1 neighbour is 0. Hence, the given input customer is not willing to take personal loan.

10 8. Now repartition the data into three parts: training, validation, and test sets (50%, 30%, and 20%).

[ ]:
```
re_trainData, tempData = train_test_split(univNorm, test_size=0.5,␣
 ↪random_state=123)
re_validData, re_testData = train_test_split(tempData, test_size=0.4,␣
 ↪random_state=123)
```

[ ]:
```
re_trainData.shape,re_validData.shape,re_testData.shape
```

[ ]:
```
((2500, 15), (1500, 15), (1000, 15))
```

10.1 9. Apply the k-NN method with the optimal k chosen above.

Opitmal Value of k is 1

[ ]:
```
# Train the k-NN classifier on the training set
re_train_x = re_trainData[['zAge', 'zExperience', 'zIncome', 'zFamily',␣
 ↪'zCCAvg', 'zMortgage',
'zSecurities Account', 'zCD Account','zOnline', 'zCreditCard', ␣
 ↪'Education_1','Education_2', 'Education_3']]
re_train_y = re_trainData['Personal Loan']
knn = KNeighborsClassifier(n_neighbors=1).fit(re_train_x.values, re_train_y.
 ↪values)
```

[ ]:
```
# Evaluate the model on the validation set
re_valid_x = re_validData[['zAge', 'zExperience', 'zIncome', 'zFamily',␣
 ↪'zCCAvg', 'zMortgage',
'zSecurities Account', 'zCD Account','zOnline', 'zCreditCard', ␣
 ↪'Education_1','Education_2', 'Education_3']]
```

```
re_valid_y = re_validData['Personal Loan']
```

[ ]:
```
re_valid_accuracy = accuracy_score(re_valid_y, knn.predict(re_valid_x))
re_valid_accuracy
```

[ ]: 0.958

[ ]:
```
# Test the model on the test set
re_test_x = re_testData[['zAge', 'zExperience', 'zIncome', 'zFamily', 'zCCAvg',␣
 ↪'zMortgage',
'zSecurities Account', 'zCD Account','zOnline', 'zCreditCard', ␣
 ↪'Education_1','Education_2','Education_3']]
re_test_y = re_testData['Personal Loan']
```

[ ]:
```
re_test_accuracy = accuracy_score(re_test_y, knn.predict(re_test_x))
re_test_accuracy
```

[ ]: 0.961

10. Compare the confusion matrix of the test set with that of the training and validation sets. Comment on the differences and their reason

[ ]:
```
confusion_table_re_test = confusion_table(knn.predict(re_test_x),re_test_y)
print("Confusion Table of Test Set:")
confusion_table_re_test
```

```
Confusion Table of Test Set:
```

[ ]:
```
Truth        0   1
Predicted
0          893  23
1           16  68
```

[ ]:
```
re_test_accuracy = accuracy_score(re_test_y, knn.predict(re_test_x))
print("Accuracy of test set :", re_test_accuracy)
```

```
Accuracy of test set : 0.961
```

[ ]:
```
confusion_table_re_valid = confusion_table(knn.predict(re_valid_x),re_valid_y)
print("Confusion Table of Validation Set:")
confusion_table_re_valid
```

```
Confusion Table of Validation Set:
```

[ ]:
```
Truth         0    1
Predicted
0          1332   42
1            21  105
```

[ ]:
```
re_valid_accuracy = accuracy_score(re_valid_y, knn.predict(re_valid_x))
print("Accuracy of Validation set :", re_valid_accuracy)
```

```
Accuracy of Validation set : 0.958
```

[ ]:
```
confusion_table_re_train = confusion_table(knn.predict(re_train_x),re_train_y)
print("Confusion Table of Train Set:")
confusion_table_re_train
```

```
Confusion Table of Train Set:
```

[ ]:
```
Truth         0    1
Predicted
0          2258    0
1             0  242
```

[ ]:
```
re_train_accuracy = accuracy_score(re_train_y, knn.predict(re_train_x))
print("Accuracy of Train set :", re_train_accuracy)
```

```
Accuracy of Train set : 1.0
```

Conclusions by comparing the confusion matrix of the test set with that of the training and validation we observed:The K-NN classification model is performing well in training,validation and test data sets. By oberving the results the model has a good generalization ability. The model has 100% acuracy on training set has expected because the model is trained on it. Where as, Their is slight drop in the accuracy for the validation(95.80%) and test(96.10%) datasets because the model has not seen the data while training.

Since k=1 means relying on a single nearest neighbor, if that neighbor happens to be an outlier or a mislabeled data point which can lead to incorrect predictions.

[ ]:
```
!pip install nbconvert
```