Universal Bank Class Problem

Steve Spence

10/3/2019

First, we will load all of the packages that will be required for this problem. Specifically, “ISLR”, “caret”, “dplyr”, “FNN”, and “gmodels” will be loaded for this problem.

```
# Require all the packages that will be used in this problem

require(ISLR)

## Loading required package: ISLR

require(caret)

## Loading required package: caret

## Warning: package 'caret' was built under R version 3.4.4

## Loading required package: lattice

## Warning: package 'lattice' was built under R version 3.4.4

## Loading required package: ggplot2

require(dplyr)

## Loading required package: dplyr

## Warning: package 'dplyr' was built under R version 3.4.4

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

require(FNN)

## Loading required package: FNN

## Warning: package 'FNN' was built under R version 3.4.4

require(gmodels)
```

```
## Loading required package: gmodels

## Warning: package 'gmodels' was built under R version 3.4.4
```

Next, we will import the "UniversalBank" data set into the RStudio environment.

```r
# Import data set from BlackBoard into the RStudio environment

Bank <- read.csv("UniversalBank.csv")
```

A summary of the data set will be displayed to review the data set.

```r
# Investigate the structure of the data set

str(Bank)

## 'data.frame':    5000 obs. of  14 variables:
##  $ ID                : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Age               : int  25 45 39 35 35 37 53 50 35 34 ...
##  $ Experience        : int  1 19 15 9 8 13 27 24 10 9 ...
##  $ Income            : int  49 34 11 100 45 29 72 22 81 180 ...
##  $ ZIP.Code          : int  91107 90089 94720 94112 91330 92121 91711
93943 90089 93023 ...
##  $ Family            : int  4 3 1 1 4 4 2 1 3 1 ...
##  $ CCAvg             : num  1.6 1.5 1 2.7 1 0.4 1.5 0.3 0.6 8.9 ...
##  $ Education         : int  1 1 1 2 2 2 2 3 2 3 ...
##  $ Mortgage          : int  0 0 0 0 0 155 0 0 104 0 ...
##  $ Personal.Loan     : int  0 0 0 0 0 0 0 0 0 1 ...
##  $ Securities.Account: int  1 1 0 0 0 0 0 0 0 0 ...
##  $ CD.Account        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Online            : int  0 0 0 0 0 1 1 0 1 0 ...
##  $ CreditCard        : int  0 0 0 0 1 0 0 1 0 0 ...

# Investigate summary statistics for the data set

summary(Bank)

##        ID             Age          Experience       Income
##  Min.   :   1   Min.   :23.00   Min.   :-3.0   Min.   :  8.00
##  1st Qu.:1251   1st Qu.:35.00   1st Qu.:10.0   1st Qu.: 39.00
##  Median :2500   Median :45.00   Median :20.0   Median : 64.00
##  Mean   :2500   Mean   :45.34   Mean   :20.1   Mean   : 73.77
##  3rd Qu.:3750   3rd Qu.:55.00   3rd Qu.:30.0   3rd Qu.: 98.00
##  Max.   :5000   Max.   :67.00   Max.   :43.0   Max.   :224.00
##     ZIP.Code         Family          CCAvg          Education
##  Min.   : 9307   Min.   :1.000   Min.   : 0.000   Min.   :1.000
##  1st Qu.:91911   1st Qu.:1.000   1st Qu.: 0.700   1st Qu.:1.000
##  Median :93437   Median :2.000   Median : 1.500   Median :2.000
##  Mean   :93152   Mean   :2.396   Mean   : 1.938   Mean   :1.881
##  3rd Qu.:94608   3rd Qu.:3.000   3rd Qu.: 2.500   3rd Qu.:3.000
##  Max.   :96651   Max.   :4.000   Max.   :10.000   Max.   :3.000
```

```
##     Mortgage     Personal.Loan   Securities.Account   CD.Account  
##  Min.   :  0.0   Min.   :0.000   Min.   :0.0000     Min.   :0.0000  
##  1st Qu.:  0.0   1st Qu.:0.000   1st Qu.:0.0000     1st Qu.:0.0000  
##  Median :  0.0   Median :0.000   Median :0.0000     Median :0.0000  
##  Mean   : 56.5   Mean   :0.096   Mean   :0.1044     Mean   :0.0604  
##  3rd Qu.:101.0   3rd Qu.:0.000   3rd Qu.:0.0000     3rd Qu.:0.0000  
##  Max.   :635.0   Max.   :1.000   Max.   :1.0000     Max.   :1.0000  
##      Online         CreditCard   
##  Min.   :0.0000   Min.   :0.000  
##  1st Qu.:0.0000   1st Qu.:0.000  
##  Median :1.0000   Median :0.000  
##  Mean   :0.5968   Mean   :0.294  
##  3rd Qu.:1.0000   3rd Qu.:1.000  
##  Max.   :1.0000   Max.   :1.000
```

We will remove the "ID" and "ZIP.Code" variables from the data set, as stated in problem statement.

```
# Create a new data set with "ID" and "ZIP.Code" variables removed

Bank_1 <- Bank[ ,-c(1,5)]

# Review the structure of the data set

str(Bank_1)

## 'data.frame':    5000 obs. of  12 variables:
##  $ Age               : int  25 45 39 35 35 37 53 50 35 34 ...
##  $ Experience        : int  1 19 15 9 8 13 27 24 10 9 ...
##  $ Income            : int  49 34 11 100 45 29 72 22 81 180 ...
##  $ Family            : int  4 3 1 1 4 4 2 1 3 1 ...
##  $ CCAvg             : num  1.6 1.5 1 2.7 1 0.4 1.5 0.3 0.6 8.9 ...
##  $ Education         : int  1 1 1 2 2 2 2 3 2 3 ...
##  $ Mortgage          : int  0 0 0 0 0 155 0 0 104 0 ...
##  $ Personal.Loan     : int  0 0 0 0 0 0 0 0 0 1 ...
##  $ Securities.Account: int  1 1 0 0 0 0 0 0 0 0 ...
##  $ CD.Account        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Online            : int  0 0 0 0 0 1 1 0 1 0 ...
##  $ CreditCard        : int  0 0 0 0 1 0 0 1 0 0 ...
```

For categorical variables with more than two categories, we will need to create dummy variables. For this data set, the "Family" and "Education" variables would require dummy variables.

```
# Convert "Education" and "Family" variables to categorical character variables

Bank_1$Education <- as.factor(Bank_1$Education)
Bank_1$Family <- as.factor(Bank_1$Family)
```

```
# Create the dummy model for "Education" and "Family"

dummy_model1 <- dummyVars(~Family + Education, data = Bank_1)

# Add the dummy variables to "Bank_1" and remove the original "Education" and
"Family" variables.

dv <- as.data.frame(predict(dummy_model1, Bank_1))
Bank_1 <- as.data.frame(c(Bank_1, dv))
Bank_1 <- Bank_1[, -c(4,6)]
```

Per the problem statement, we will now split the data set into 60% training and 40% test data via the "createDataPartition" function.

```
# Set the seed for randomized functions

set.seed(100319)

# Split the data into 60% training data and 40% test data

Bank_1_Index <- createDataPartition(Bank_1$Age, p=0.4, list = F)

Bank_1_Test <- Bank_1[Bank_1_Index,]

Bank_1_Train <- Bank_1[-Bank_1_Index,]
```

Next, we will have to normalize the training and test data sets via the "preProcess" function.

```
# Create a copy of the data set for normalization

Bank_1_Train_Norm <- Bank_1_Train
Bank_1_Test_Norm <- Bank_1_Test

# Use preProcess function to create a model for centering and scaling the
data

Norm_Values <- preProcess(Bank_1_Train[, c(1:5)], method = c("center",
"scale"))

# Replace the numeric variables with normalized and centered data

Bank_1_Train_Norm[, c(1:5)] <- predict(Norm_Values, Bank_1_Train[, c(1:5)])
Bank_1_Test_Norm[, c(1:5)] <- predict(Norm_Values, Bank_1_Test[, c(1:5)])
```

Now, the KNN function can be utilized.

```
# Create the KNN model with K = 1 and only training and test data

knn_model2 <- knn(train = Bank_1_Train_Norm[, -6], test = Bank_1_Test_Norm[,
```

```
-6],
          cl = Bank_1_Train_Norm[, 6], k = 1, prob = TRUE)

head(Bank_1)

##   Age Experience Income CCAvg Mortgage Personal.Loan Securities.Account
## 1  25          1     49   1.6        0             0                  1
## 2  45         19     34   1.5        0             0                  1
## 3  39         15     11   1.0        0             0                  0
## 4  35          9    100   2.7        0             0                  0
## 5  35          8     45   1.0        0             0                  0
## 6  37         13     29   0.4      155             0                  0
##   CD.Account Online CreditCard Family.1 Family.2 Family.3 Family.4
## 1          0      0          0        0        0        0        1
## 2          0      0          0        0        0        1        0
## 3          0      0          0        1        0        0        0
## 4          0      0          0        1        0        0        0
## 5          0      0          1        0        0        0        1
## 6          0      1          0        0        0        0        1
##   Education.1 Education.2 Education.3
## 1           1           0           0
## 2           1           0           0
## 3           1           0           0
## 4           0           1           0
## 5           0           1           0
## 6           0           1           0

# Create the customer profile for the customer called out in question #1

customer <- data.frame("Age" = 40,
                       "Experience" = 10,
                       "Income" = 84,
                       "CCAvg" = 2,
                       "Mortgage" = 0,
                       "Securities.Account" = 0,
                       "CD.Account" = 0,
                       "Online" = 1,
                       "CreditCard" = 1,
                       "Family.1" = 0,
                       "Family.2" = 1,
                       "Family.3" = 0,
                       "Family.4" = 0,
                       "Education.1" = 0,
                       "Education.2" = 1,
                       "Education.3" = 0)

# Perform the same preProcessing steps on the customer profile as the model
was created on

customer[, c(1:5)] <- predict(Norm_Values, customer[, c(1:5)])
```

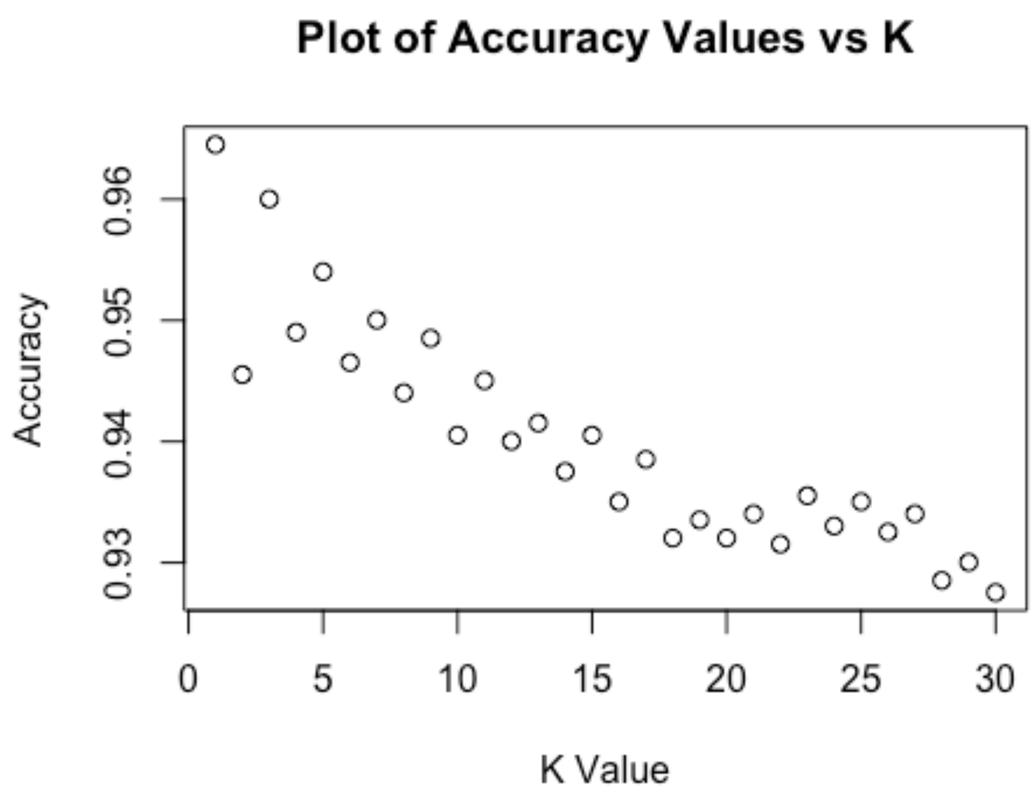
```r
# Run the KNN model on the customer profile

knn_model_customer <- knn(train = Bank_1_Train_Norm[, -6], test = customer,
          cl = Bank_1_Train_Norm[, 6], k = 1, prob = TRUE)

# Return the value predicted by the model

as.data.frame(knn_model_customer)

##   knn_model_customer
## 1                  0
```

1. According to the KNN model prediction, the customer in question would not accept the personal loan.

```r
# ICreate a data frame with two columns - k, and accuracy

accuracy.df <- data.frame(k = seq(1, 30, 1), accuracy = rep(0, 30))

# Perform predictions on the values at different K values with KNN

for(i in 1:30) {
  knn.pred <- knn(train = Bank_1_Train_Norm[, -6], test = Bank_1_Test_Norm[,
-6],
          cl = Bank_1_Train_Norm[, 6], k = i)
  accuracy.df[i, 2] <- confusionMatrix(as.factor(knn.pred),
as.factor(Bank_1_Test_Norm[, 6]))$overall[1]
}

# Display the results in a data frame

accuracy.df

##     k  accuracy
## 1   1 0.9645177
## 2   2 0.9455272
## 3   3 0.9600200
## 4   4 0.9490255
## 5   5 0.9540230
## 6   6 0.9465267
## 7   7 0.9500250
## 8   8 0.9440280
## 9   9 0.9485257
## 10 10 0.9405297
## 11 11 0.9450275
## 12 12 0.9400300
## 13 13 0.9415292
## 14 14 0.9375312
## 15 15 0.9405297
```

```
## 16 16 0.9350325
## 17 17 0.9385307
## 18 18 0.9320340
## 19 19 0.9335332
## 20 20 0.9320340
## 21 21 0.9340330
## 22 22 0.9315342
## 23 23 0.9355322
## 24 24 0.9330335
## 25 25 0.9350325
## 26 26 0.9325337
## 27 27 0.9340330
## 28 28 0.9285357
## 29 29 0.9300350
## 30 30 0.9275362
```

```r
# Rough plot of the accuracies to see the trend in data

plot(x = accuracy.df$k,y = accuracy.df$accuracy, main = "Plot of Accuracy
Values vs K", xlab = "K Value", ylab = "Accuracy")
```



2. The choice of K that balances between overfitting and ignoring predictor information appears to be K = 1, which results in the highest accuracy reading at 0.965.

```r
# Create the KNN model with K = 1 and only training and test data

knn_model2 <- knn(train = Bank_1_Train_Norm[, -6], test = Bank_1_Test_Norm[, 
-6],
          cl = Bank_1_Train_Norm[, 6], k = 1, prob = TRUE)

# Confusion Matrix

predicted <- as.factor(knn_model2)
actual <- as.factor(Bank_1_Test_Norm[, 6])

confusionMatrix(predicted, actual, positive = "1")

## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0 1792   63
##          1    8  138
##                                           
##                Accuracy : 0.9645          
##                  95% CI : (0.9555, 0.9722)
##     No Information Rate : 0.8996          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.7765          
##  Mcnemar's Test P-Value : 1.468e-10       
##                                           
##             Sensitivity : 0.68657         
##             Specificity : 0.99556         
##          Pos Pred Value : 0.94521         
##          Neg Pred Value : 0.96604         
##              Prevalence : 0.10045         
##          Detection Rate : 0.06897         
##    Detection Prevalence : 0.07296         
##       Balanced Accuracy : 0.84106         
##                                           
##        'Positive' Class : 1               
## 
```

3. Confusion matrix of the KNN model using K = 1.

```r
# Run the KNN model on the customer profile with the K = 1 value

knn_model_customer2 <- knn(train = Bank_1_Train_Norm[, -6], test = customer,
          cl = Bank_1_Train_Norm[, 6], k = 20, prob = TRUE)

# Return the value predicted by the model

as.data.frame(knn_model_customer2)
```

```
##   knn_model_customer2
## 1                   0
```

4.	The customer is this case is still predicted to not accept the personal loan.

Per the problem statement, we will now split the data set into 50% training data, 30% validation data, and 20% test data via the “createDataPartition” function.

```r
# Set the seed for randomized functions

set.seed(100619)

# Split the data into 50% training data, 30% validation data, and 20% test
data

Bank_2_Index <- createDataPartition(Bank_1$Age, p=0.2, list = F)

Bank_2_Test <- Bank_1[Bank_2_Index,]

Bank_2_Remaining <- Bank_1[-Bank_2_Index,]

Bank_2_Index <- createDataPartition(Bank_2_Remaining$Age, p=0.625, list = F)

Bank_2_Train <- Bank_2_Remaining[Bank_2_Index,]

Bank_2_Validation <- Bank_2_Train[-Bank_2_Index,]
```

The newly divided data will now need to be normalized, as we did before.

```r
# Create a copy of the data sets for normalization

Bank_2_Train_Norm <- Bank_2_Train
Bank_2_Test_Norm <- Bank_2_Test
Bank_2_Validation_Norm <- Bank_2_Validation

# Use preProcess function to create a model for centering and scaling the
data

Norm_Values <- preProcess(Bank_2_Train[, c(1:5)], method = c("center",
"scale"))

# Replace the numeric variables with normalized and centered data

Bank_2_Train_Norm[, c(1:5)] <- predict(Norm_Values, Bank_2_Train[, c(1:5)])
Bank_2_Test_Norm[, c(1:5)] <- predict(Norm_Values, Bank_2_Test[, c(1:5)])
Bank_2_Validation_Norm[, c(1:5)] <- predict(Norm_Values, Bank_2_Validation[,
c(1:5)])
```

We will now re-run the KNN model with the newly divided data sets.

5. The confusion matricies for the test data should have lower accuracy results, because it holds data that has not been seen by the model when training it. Therefore, the confusion matrix for training data should be more accurate, because it has already seen the data that it is predicting. Depending on exactly what metric we want to compare, there could be comparisons made about precision, recall, accuracy, specificity, etc.

The first confusion matrix is for the “test” data:

```
# Create the KNN model with K = 1

knn_model_test <- knn(train = Bank_2_Train_Norm[, -6], test =
Bank_2_Test_Norm[, -6],
          cl = Bank_2_Train_Norm[, 6], k = 1, prob = TRUE)

# Confusion Matrix

predicted_test <- as.factor(knn_model_test)
actual_test <- as.factor(Bank_2_Test_Norm[, 6])

confusionMatrix(predicted_test, actual_test, positive = "1")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 893  39
##          1   3  66
##
##                Accuracy : 0.958
##                  95% CI : (0.9437, 0.9696)
##     No Information Rate : 0.8951
##     P-Value [Acc > NIR] : 2.570e-13
##
##                   Kappa : 0.7367
##  Mcnemar's Test P-Value : 6.641e-08
##
##             Sensitivity : 0.62857
##             Specificity : 0.99665
##          Pos Pred Value : 0.95652
##          Neg Pred Value : 0.95815
##              Prevalence : 0.10490
##          Detection Rate : 0.06593
##    Detection Prevalence : 0.06893
##       Balanced Accuracy : 0.81261
##
##        'Positive' Class : 1
##
```

The second confusion matrix is for the “validation” data:

```r
# Create the KNN model with K = 1

knn_model_validation <- knn(train = Bank_2_Train_Norm[, -6], test =
Bank_2_Validation_Norm[, -6],
          cl = Bank_2_Train_Norm[, 6], k = 1, prob = TRUE)

# Confusion Matrix

predicted_validation <- as.factor(knn_model_validation)
actual_validation <- as.factor(Bank_2_Validation_Norm[, 6])

confusionMatrix(predicted_validation, actual_validation, positive = "1")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 852   0
##          1   0  81
##
##                Accuracy : 1
##                  95% CI : (0.9961, 1)
##     No Information Rate : 0.9132
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.00000
##             Specificity : 1.00000
##          Pos Pred Value : 1.00000
##          Neg Pred Value : 1.00000
##              Prevalence : 0.08682
##          Detection Rate : 0.08682
##    Detection Prevalence : 0.08682
##       Balanced Accuracy : 1.00000
##
##        'Positive' Class : 1
##
```

The third confusion matrix is for the “train” data:

```r
# Create the KNN model with K = 1

knn_model_train <- knn(train = Bank_2_Train_Norm[, -6], test =
Bank_2_Train_Norm[, -6],
          cl = Bank_2_Train_Norm[, 6], k = 1, prob = TRUE)

# Confusion Matrix
```

```r
predicted_train <- as.factor(knn_model_train)
actual_train <- as.factor(Bank_2_Train_Norm[, 6])

confusionMatrix(predicted_train, actual_train, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2254    0
##          1    0  247
##
##                Accuracy : 1
##                  95% CI : (0.9985, 1)
##     No Information Rate : 0.9012
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.00000
##             Specificity : 1.00000
##          Pos Pred Value : 1.00000
##          Neg Pred Value : 1.00000
##              Prevalence : 0.09876
##          Detection Rate : 0.09876
##    Detection Prevalence : 0.09876
##       Balanced Accuracy : 1.00000
##
##        'Positive' Class : 1
##
```

From the confusion matricies above, we can see that the predicted test data does have a lower accuracy reading than the training data. In this case, the validation data and training data both had an accuracy of 1.0; however, this is usually not the case with larger amounts of normalized data.