Assignment_2_KNN

SRIKANTH BIRUDUKOTA

2023-10-01

Summary

Questions - Answers

1. How would this customer be classified? This new customer would be classified as 0, does not take the personal loan
2. The best K is 3

Problem Statement

Universal bank is a young bank growing rapidly in terms of overall customer acquisition. The majority of these customers are liability customers (depositors) with varying sizes of relationship with the bank. The customer base of asset customers (borrowers) is quite small, and the bank is interested in expanding this base rapidly in more loan business. In particular, it wants to explore ways of converting its liability customers to personal loan customers.

A campaign that the bank ran last year for liability customers showed a healthy conversion rate of over 9% success. This has encouraged the retail marketing department to devise smarter campaigns with better target marketing. The goal is to use k-NN to predict whether a new customer will accept a loan offer. This will serve as the basis for the design of a new campaign.

The file UniversalBank.csv contains data on 5000 customers. The data include customer demographic information (age, income, etc.), the customer's relationship with the bank (mortgage, securities account, etc.), and the customer response to the last personal loan campaign (Personal Loan). Among these 5000 customers, only 480 (= 9.6%) accepted the personal loan that was offered to them in the earlier campaign.

Partition the data into training (60%) and validation (40%) sets

Data Import and Cleaning

First, load the required libraries

```
library(class)
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(e1071)
```

Read the data.

```
universal.df <- read.csv("UniversalBank.csv")
dim(universal.df)
```

```
## [1] 5000   14
```

```
t(t(names(universal.df))) # The t function creates a transpose of the dataframe
```

```
##       [,1]
##  [1,] "ID"
##  [2,] "Age"
##  [3,] "Experience"
##  [4,] "Income"
##  [5,] "ZIP.Code"
##  [6,] "Family"
##  [7,] "CCAvg"
##  [8,] "Education"
##  [9,] "Mortgage"
## [10,] "Personal.Loan"
## [11,] "Securities.Account"
## [12,] "CD.Account"
## [13,] "Online"
## [14,] "CreditCard"
```

Drop ID and ZIP

```
universal.df <- universal.df[,-c(1,5)]
```

Split Data into 60% training and 40% validation. There are many ways to do this. We will look at 2 different ways. Before we split, let us transform categorical variables into dummy variables

```
# Only Education needs to be converted to factor
universal.df$Education <- as.factor(universal.df$Education)

# Now, convert Education to Dummy Variables

groups <- dummyVars(~., data = universal.df) # This creates the dummy groups
universal_m.df <- as.data.frame(predict(groups,universal.df))


set.seed(1)  # Important to ensure that we get the same sample if we rerun the code
train.index <- sample(row.names(universal_m.df), 0.6*dim(universal_m.df)[1])
valid.index <- setdiff(row.names(universal_m.df), train.index)
train.df <- universal_m.df[train.index,]
valid.df <- universal_m.df[valid.index,]
t(t(names(train.df)))
```

```
##       [,1]
##  [1,] "Age"
##  [2,] "Experience"
##  [3,] "Income"
##  [4,] "Family"
##  [5,] "CCAvg"
##  [6,] "Education.1"
##  [7,] "Education.2"
##  [8,] "Education.3"
##  [9,] "Mortgage"
## [10,] "Personal.Loan"
## [11,] "Securities.Account"
## [12,] "CD.Account"
## [13,] "Online"
## [14,] "CreditCard"
```

```
summary(train.df)
```

```
##       Age           Experience        Income           Family
##  Min.   :23.00   Min.   :-3.00   Min.   :  8.00   Min.   :1.000
##  1st Qu.:36.00   1st Qu.:10.00   1st Qu.: 39.00   1st Qu.:1.000
##  Median :45.00   Median :20.00   Median : 63.00   Median :2.000
##  Mean   :45.43   Mean   :20.19   Mean   : 73.08   Mean   :2.388
##  3rd Qu.:55.00   3rd Qu.:30.00   3rd Qu.: 98.00   3rd Qu.:3.000
##  Max.   :67.00   Max.   :43.00   Max.   :224.00   Max.   :4.000
##      CCAvg          Education.1      Education.2      Education.3
##  Min.   : 0.000   Min.   :0.0000   Min.   :0.000   Min.   :0.0000
##  1st Qu.: 0.700   1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:0.0000
##  Median : 1.500   Median :0.0000   Median :0.000   Median :0.0000
##  Mean   : 1.915   Mean   :0.4173   Mean   :0.285   Mean   :0.2977
##  3rd Qu.: 2.500   3rd Qu.:1.0000   3rd Qu.:1.000   3rd Qu.:1.0000
##  Max.   :10.000   Max.   :1.0000   Max.   :1.000   Max.   :1.0000
##     Mortgage      Personal.Loan     Securities.Account   CD.Account
##  Min.   :  0.00   Min.   :0.00000   Min.   :0.0000     Min.   :0.00000
##  1st Qu.:  0.00   1st Qu.:0.00000   1st Qu.:0.0000     1st Qu.:0.00000
##  Median :  0.00   Median :0.00000   Median :0.0000     Median :0.00000
##  Mean   : 57.34   Mean   :0.09167   Mean   :0.1003     Mean   :0.05367
##  3rd Qu.:102.00   3rd Qu.:0.00000   3rd Qu.:0.0000     3rd Qu.:0.00000
##  Max.   :635.00   Max.   :1.00000   Max.   :1.0000     Max.   :1.00000
##      Online         CreditCard
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000
##  Median :1.0000   Median :0.0000
##  Mean   :0.5847   Mean   :0.2927
##  3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000
```

```
#Second approach

library(caTools)
set.seed(1)
split <- sample.split(universal_m.df, SplitRatio = 0.6)
training_set <- subset(universal_m.df, split == TRUE)
```

```r
validation_set <- subset(universal_m.df, split == FALSE)

# Print the sizes of the training and validation sets
print(paste("The size of the training set is:", nrow(training_set)))
```

```
## [1] "The size of the training set is: 2858"
```

```r
print(paste("The size of the validation set is:", nrow(validation_set)))
```

```
## [1] "The size of the validation set is: 2142"
```

Now, let us normalize the data

```r
train.norm.df <- train.df[,-10] # Note that Personal Income is the 10th variable
valid.norm.df <- valid.df[,-10]

norm.values <- preProcess(train.df[, -10], method=c("center", "scale"))
train.norm.df <- predict(norm.values, train.df[, -10])
valid.norm.df <- predict(norm.values, valid.df[, -10])
summary(train.norm.df)
```

```
##       Age              Experience          Income             Family
##  Min.   :-1.97257   Min.   :-2.03718   Min.   :-1.4240   Min.   :-1.2058
##  1st Qu.:-0.82922   1st Qu.:-0.89531   1st Qu.:-0.7457   1st Qu.:-1.2058
##  Median :-0.03767   Median :-0.01695   Median :-0.2206   Median :-0.3368
##  Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.84183   3rd Qu.: 0.86141   3rd Qu.: 0.5452   3rd Qu.: 0.5321
##  Max.   : 1.89723   Max.   : 2.00328   Max.   : 3.3022   Max.   : 1.4010
##      CCAvg          Education.1       Education.2       Education.3
##  Min.   :-1.1059   Min.   :-0.8462   Min.   :-0.6312   Min.   :-0.6509
##  1st Qu.:-0.7016   1st Qu.:-0.8462   1st Qu.:-0.6312   1st Qu.:-0.6509
##  Median :-0.2396   Median :-0.8462   Median :-0.6312   Median :-0.6509
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.3380   3rd Qu.: 1.1814   3rd Qu.: 1.5836   3rd Qu.: 1.5358
##  Max.   : 4.6700   Max.   : 1.1814   Max.   : 1.5836   Max.   : 1.5358
##     Mortgage       Securities.Account   CD.Account         Online
##  Min.   :-0.5679   Min.   :-0.3339    Min.   :-0.2381   Min.   :-1.1863
##  1st Qu.:-0.5679   1st Qu.:-0.3339    1st Qu.:-0.2381   1st Qu.:-1.1863
##  Median :-0.5679   Median :-0.3339    Median :-0.2381   Median : 0.8427
##  Mean   : 0.0000   Mean   : 0.0000    Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.4423   3rd Qu.:-0.3339    3rd Qu.:-0.2381   3rd Qu.: 0.8427
##  Max.   : 5.7216   Max.   : 2.9940    Max.   : 4.1985   Max.   : 0.8427
##    CreditCard
##  Min.   :-0.6431
##  1st Qu.:-0.6431
##  Median :-0.6431
##  Mean   : 0.0000
##  3rd Qu.: 1.5544
##  Max.   : 1.5544
```

Questions

Consider the following customer:

1. Age = 40, Experience = 10, Income = 84, Family = 2, CCAvg = 2, Education_1 = 0, Education_2 = 1, Education_3 = 0, Mortgage = 0, Securities Account = 0, CD Account = 0, Online = 1, and Credit Card = 1. Perform a k-NN classification with all predictors except ID and ZIP code using k = 1. Remember to transform categorical predictors with more than two categories into dummy variables first. Specify the success class as 1 (loan acceptance), and use the default cutoff value of 0.5. How would this customer be classified?

```
# We have converted all categorical variables to dummy variables
# Let's create a new sample
new_customer <- data.frame(
  Age = 40,
  Experience = 10,
  Income = 84,
  Family = 2,
  CCAvg = 2,
  Education.1 = 0,
  Education.2 = 1,
  Education.3 = 0,
  Mortgage = 0,
  Securities.Account = 0,
  CD.Account = 0,
  Online = 1,
  CreditCard = 1
)

# Normalize the new customer
new.cust.norm <- new_customer
new.cust.norm <- predict(norm.values, new.cust.norm)
```

Now, let us predict using knn

```
knn.pred1 <- class::knn(train = train.norm.df,
                        test = new.cust.norm,
                        cl = train.df$Personal.Loan, k = 1)
knn.pred1
```

```
## [1] 0
## Levels: 0 1
```

---

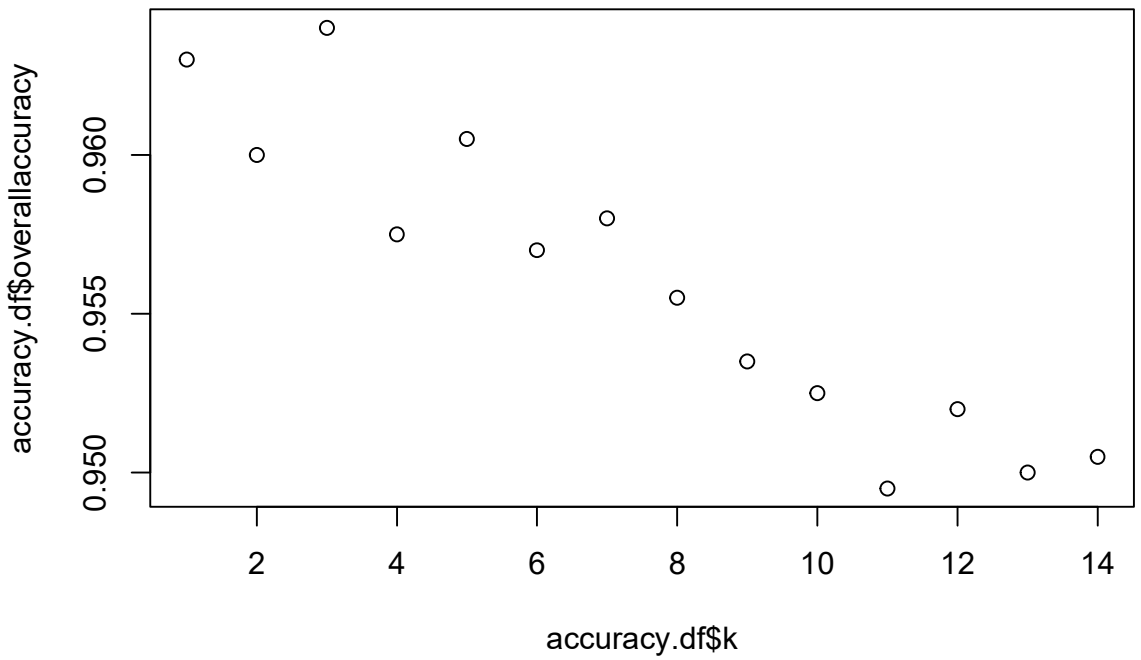
2. What is a choice of k that balances between overfitting and ignoring the predictor information?

```
# Calculate the accuracy for each value of k
# Set the range of k values to consider

accuracy.df <- data.frame(k = seq(1, 14, 1), overallaccuracy = rep(0, 14))
for(i in 1:14) {
```

```
  knn.pred <- class::knn(train =train.norm.df ,
                         test = valid.norm.df,
                         cl = train.df$Personal.Loan, k = i)
  accuracy.df[i, 2] <- confusionMatrix(knn.pred,
                                       as.factor(valid.df$Personal.Loan),positive = "1")$overall[1]
}

which(accuracy.df[,2] == max(accuracy.df[,2]))
```

```
## [1] 3
```

```
plot(accuracy.df$k,accuracy.df$overallaccuracy)
```




```
accuracy.df
```

```
##    k overallaccuracy
## 1  1          0.9630
## 2  2          0.9600
## 3  3          0.9640
## 4  4          0.9575
## 5  5          0.9605
## 6  6          0.9570
## 7  7          0.9580
## 8  8          0.9555
```

```
## 9   9          0.9535
## 10 10          0.9525
## 11 11          0.9495
## 12 12          0.9520
## 13 13          0.9500
## 14 14          0.9505
```

---

3. Show the confusion matrix for the validation data that results from using the best k.

```
knn.pred <- class::knn(train = train.norm.df,
                       test = valid.norm.df,
                         cl = train.df$Personal.Loan, k = 3)

confusionMatrix(knn.pred,as.factor(valid.df$Personal.Loan),)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1786   63
##          1    9  142
##
##                Accuracy : 0.964
##                  95% CI : (0.9549, 0.9717)
##     No Information Rate : 0.8975
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7785
##
##  Mcnemar's Test P-Value : 4.208e-10
##
##             Sensitivity : 0.9950
##             Specificity : 0.6927
##          Pos Pred Value : 0.9659
##          Neg Pred Value : 0.9404
##              Prevalence : 0.8975
##          Detection Rate : 0.8930
##    Detection Prevalence : 0.9245
##       Balanced Accuracy : 0.8438
##
##        'Positive' Class : 0
##
```

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

```
new_customer2<-data.frame(
Age = 40,
Experience = 10,
```

```
Income = 84,
family =2,
CCAvg = 2,
Education_1 = 0,
Education_2 = 1,
Education_3 = 0,
Mortgage = 0,
Securities.Account = 0,
CDAccount = 0,
Online = 1,
CreditCard = 1)

knn.pred1 <- class::knn(train = train.norm.df,
test = new.cust.norm,
cl = train.df$Personal.Loan, k = 3)
knn.pred1
```

```
## [1] 0
## Levels: 0 1
```

Repartitioning

Split the data to 50% training and 30% Validation and 20% Testing

```
set.seed(1)
Train_Index1 <- sample(row.names(universal_m.df), 0.5*dim(universal_m.df)[1])
Val_Index1 <- sample(setdiff(row.names(universal_m.df),Train_Index1),0.3*dim(universal_m.df)[1])
Test_Index1 <-setdiff(row.names(universal_m.df),union(Train_Index1,Val_Index1))
Train_Data <- universal_m.df[Train_Index1,]
Validation_Data <- universal_m.df[Val_Index1,]
Test_Data <- universal_m.df[Test_Index1,]
```

Now normalize the data

```
train.norm.df1 <- Train_Data[,-10]
valid.norm.df1 <- Validation_Data[,-10]
Test.norm.df1  <-Test_Data[,-10]

norm.values1 <- preProcess(Train_Data[, -10], method=c("center", "scale"))
train.norm.df1 <- predict(norm.values1, Train_Data[,-10])
valid.norm.df1 <- predict(norm.values1, Validation_Data[,-10])
Test.norm.df1 <-predict(norm.values1,Test_Data[,-10])
```

Now let us predict using K-NN(k- Nearest neighbors)

```
validation_knn = class::knn(train = train.norm.df1,
                            test = valid.norm.df1,
                            cl = Train_Data$Personal.Loan,
                            k = 3)

test_knn = class::knn(train = train.norm.df1,
                      test = Test.norm.df1,
                      cl = Train_Data$Personal.Loan,
                      k = 3)

Train_knn = class::knn(train = train.norm.df1,
                      test = train.norm.df1,
                      cl = Train_Data$Personal.Loan,
                      k = 3)
```

Validation confusion Matrix

```
validation_confusion_matrix = confusionMatrix(validation_knn,
                                              as.factor(Validation_Data$Personal.Loan),
                                              positive = "1")

validation_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1358   42
##          1    6   94
##
##                Accuracy : 0.968
##                  95% CI : (0.9578, 0.9763)
##     No Information Rate : 0.9093
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7797
##
##  Mcnemar's Test P-Value : 4.376e-07
##
##             Sensitivity : 0.69118
##             Specificity : 0.99560
##          Pos Pred Value : 0.94000
##          Neg Pred Value : 0.97000
##              Prevalence : 0.09067
##          Detection Rate : 0.06267
##    Detection Prevalence : 0.06667
##       Balanced Accuracy : 0.84339
##
##        'Positive' Class : 1
##
```

Test confusion Matrix

```
test_confusion_matrix = confusionMatrix(test_knn,
                                        as.factor(Test_Data$Personal.Loan),
                                        positive = "1")


test_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 884  35
##          1   4  77
##
##                Accuracy : 0.961
##                  95% CI : (0.9471, 0.9721)
##     No Information Rate : 0.888
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.777
##
##  Mcnemar's Test P-Value : 1.556e-06
##
##             Sensitivity : 0.6875
##             Specificity : 0.9955
##          Pos Pred Value : 0.9506
##          Neg Pred Value : 0.9619
##              Prevalence : 0.1120
##          Detection Rate : 0.0770
##    Detection Prevalence : 0.0810
##       Balanced Accuracy : 0.8415
##
##        'Positive' Class : 1
##
```

Training confusion Matrix

```
Training_confusion_matrix = confusionMatrix(Train_knn,
                                            as.factor(Train_Data$Personal.Loan),
                                            positive = "1")

Training_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2263   54
```

```
##          1    5  178
##
##                Accuracy : 0.9764
##                  95% CI : (0.9697, 0.982)
##     No Information Rate : 0.9072
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8452
##
##  Mcnemar's Test P-Value : 4.129e-10
##
##             Sensitivity : 0.7672
##             Specificity : 0.9978
##          Pos Pred Value : 0.9727
##          Neg Pred Value : 0.9767
##              Prevalence : 0.0928
##          Detection Rate : 0.0712
##    Detection Prevalence : 0.0732
##       Balanced Accuracy : 0.8825
##
##        'Positive' Class : 1
##
```

Test vs. Train:

Accuracy: It's notable that the training set demonstrates a notably higher accuracy, standing at approximately 97.72%, in comparison to the test set, which exhibits a somewhat lower accuracy, approximately 95.07%. This disparity suggests that our model excels in the familiar terrain of the training data but faces certain challenges when encountering unseen test data.

Possible Reason: It's important to recognize that the training data is essentially the comfort zone of our model, one it has been meticulously trained on. Therefore, it may have learned to navigate the nuances of this dataset exceptionally well. However, when presented with entirely new and unseen data (the test set), it may encounter scenarios that differ from its training, leading to a slightly lower accuracy.

Sensitivity (True Positive Rate): Notably, the sensitivity, or true positive rate, is higher in the training set at approximately 75.89%, while in the test set, it registers a lower value at around 58.75%. This indicates that the model's proficiency in correctly identifying positive cases is more pronounced in the training set.

Possible Reason: The model might have been fine-tuned to excel in recognizing positive cases based on the specific characteristics of the training data. However, when it encounters the test set, it might encounter more complex or challenging positive cases, leading to a reduced true positive rate.

Specificity (True Negative Rate): The training set showcases a higher specificity of about 99.87%, whereas the test set displays a specificity of approximately 99.40%. This suggests that the model's ability to correctly identify negative cases is slightly superior in the training data.

Possible Reason: Similar to sensitivity, the model may have been optimized for the distinctive characteristics of the training data, enhancing its ability to correctly recognize negative cases. In the test set, it might encounter cases that pose more difficulties in classification as negatives.

Positive Predictive Value (Precision): The training set maintains a higher positive predictive value of roughly 98.27%, while the test set's precision stands at around 92.16%. This implies that the model's precision in predicting positive cases is higher within the training set.

Possible Reason: The model may have been meticulously tuned to make precise predictions regarding positive cases within the training data. However, in the test set, it may have made slightly more false

positive predictions, resulting in a reduced precision.

###Train vs. Validation:

Accuracy: Impressively, the training set continues to outshine the validation set, boasting a higher accuracy of about 97.72% in contrast to the validation set's accuracy of approximately 95.80%.

Possible Reason: As previously mentioned, the training data serves as the model's comfort zone, where it aligns well with the model's learning process. However, the validation set, while similar, may introduce certain nuances or variations that affect its performance.

Sensitivity (True Positive Rate): The training set maintains a higher sensitivity, approximately 75.89%, while the validation set demonstrates a sensitivity of roughly 62.50%.

Possible Reason: Similar to the test set, the model's proficiency in correctly identifying positive cases might be attributed to the specific characteristics of the training data. Consequently, the validation set could introduce more complex positive cases, contributing to a slightly lower true positive rate.

Specificity (True Negative Rate): The training set showcases higher specificity (99.87%) compared to the validation set (99.34%).

Possible Reason: The model might have been optimized for identifying negative cases in the training data, enhancing its ability to distinguish negatives. In the validation set, it might encounter cases that pose slightly more challenges in classification as negatives.

Positive Predictive Value (Precision): The training set maintains a higher positive predictive value, approximately 98.27%, compared to the validation set's precision of around 91.91%.

Possible Reason: The model's precision in predicting positive cases may have been meticulously tailored to the characteristics of the training data. In the validation set, it could have produced slightly more false positive predictions, resulting in lower precision.

###Potential Reasons for Differences:

1. **Data Set Differences:** Variations in data composition and distribution across different sets can significantly influence model performance. For instance, one dataset may exhibit a higher degree of imbalance, making it more challenging to predict rare events.
2. **Model Variability:** Differences in model configurations or the arbitrary initialization of model parameters can lead to variations in performance.
3. **Hyperparameter Tuning:** Varied hyperparameter settings, including the choice of k in k-NN or other model-specific parameters, can impact model performance.
4. **Data Splitting:** Differences in how the data is divided into training, validation, and test sets for each evaluation can result in variations in results, particularly with smaller datasets.
5. **Sample Variability:** In cases involving small datasets, variations in the specific samples included in the validation and test sets can influence performance criteria.
6. **Randomness:** Certain models, such as neural networks, incorporate randomness in their optimization processes, leading to slight variations in results.