Tejesh_Varma_Maddana_BA64060_Assignment_2

Tejesh Varma Maddana

2023-10-01

Assignment 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("~/Documents/KSU/Fundamentals of Machine Learning - 64060/Assignments/Assignmen
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))
```

#Split the data to 60% training and 40 % Validation

```
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"
```

Now, let us normalize the data

```
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])
```

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
```

Print the predicted class (1 for loan acceptance, 0 for loan rejection)

```
print("This customer is classified as: Loan Rejected")
```

```
## [1] "This customer is classified as: Loan Rejected"
```

---

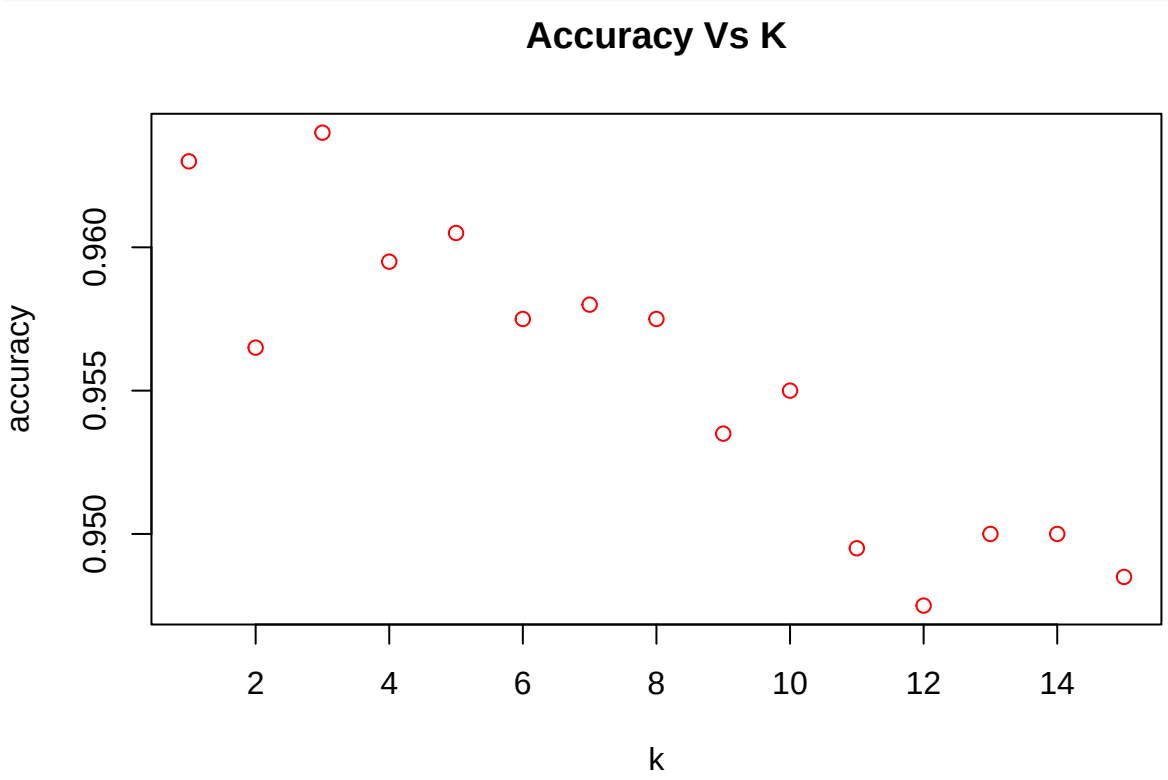
2. What is a choice of k that balances between overfitting and ignoring the predictor information?

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

accuracy.df <- data.frame(k = seq(1, 15, 1), overallaccuracy = rep(0, 15))
for(i in 1:15) {
  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,  main = "Accuracy Vs K", xlab = "k", ylab = "accuracy",
```




```
accuracy.df
```

```
##     k overallaccuracy
## 1   1          0.9630
## 2   2          0.9565
## 3   3          0.9640
## 4   4          0.9595
## 5   5          0.9605
## 6   6          0.9575
## 7   7          0.9580
## 8   8          0.9575
## 9   9          0.9535
## 10 10          0.9550
## 11 11          0.9495
## 12 12          0.9475
## 13 13          0.9500
## 14 14          0.9500
## 15 15          0.9485
```

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

```
#From the above plot, the best value of k is 3, therefore assigning the best k value to 3
best_k <- 3

# Calculate the confusion matrix for the validation data using the best k (that is k=3)
knn_pred <- class::knn(train = train.norm.df,
                       test = valid.norm.df,
                       cl = train.df$Personal.Loan, k = best_k)

confusion_matrix <- table(knn_pred, valid.df$Personal.Loan)

# Print the confusion matrix
print(confusion_matrix) #This indicates the confusion matrix
```

```
##
## knn_pred    0    1
##        0 1786   63
##        1    9  142
```

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_customer_2 <- 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_2 <- new_customer_2
new.cust.norm_2 <- predict(norm.values, new.cust.norm_2)

#Now, let us predict using knn =3

knn.pred_2 <- class::knn(train = train.norm.df,
                         test = new.cust.norm_2,
                         cl = train.df$Personal.Loan, k = 3)
knn.pred_2
```

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

Print the predicted class (1 for loan acceptance, 0 for loan rejection)

```
print("This customer is classified as: Loan Rejected")
```

```
## [1] "This customer is classified as: Loan Rejected"
```

5. Repartition the data, this time into training, validation, and test sets (50% : 30% : 20%). Apply the k-NN method with the k chosen above. Compare the confusion matrix of the test set with that of the training and validation sets. Comment on the differences and their reason.

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

```
set.seed(1)
Training_Index_1 <- sample(row.names(universal_m.df), 0.5*dim(universal_m.df)[1])
Valid_Index_1 <- sample(setdiff(row.names(universal_m.df),Training_Index_1),0.3*dim(universal_m.df)[1])
Testing_Index_1 <-setdiff(row.names(universal_m.df),union(Training_Index_1,Valid_Index_1))
Training_Data <- universal_m.df[Training_Index_1,]
Validation_Data <- universal_m.df[Valid_Index_1,]
Testing_Data <- universal_m.df[Testing_Index_1,]
```

Now normalize the data

```
train.norm.df1 <- Training_Data[,-10]
valid.norm.df1 <- Validation_Data[,-10]
Test.norm.df1  <-Testing_Data[,-10]

norm.values1 <- preProcess(Training_Data[, -10], method=c("center", "scale"))
train.norm.df1 <- predict(norm.values1, Training_Data[,-10])
valid.norm.df1 <- predict(norm.values1, Validation_Data[,-10])
Test.norm.df1 <-predict(norm.values1,Testing_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 = Training_Data$Personal.Loan,
                            k = 3)

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

Train_knn = class::knn(train = train.norm.df1,
                      test = train.norm.df1,
                      cl = Training_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(Testing_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 = confusionMatrix(Train_knn,
                                            as.factor(Training_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: Test data(0.961) has lower accuracy than compared to Train data(0.9764)

Reason: Since, we have took the training data as 50%, accuracy is more than compared to testing data 20%.

Sensitivity (True Positive Rate): Testing data (0.6875) has lower sensitivity than Training data (0.7672)

Reason: This indicates that Training data model is better at correctly identifying positive cases (e.g., loan acceptances). It may have a lower false negative rate.

Specificity (True Negative Rate): Testing data (0.9955) has less specificity than compared to training data(0.9978).

Reason: This suggests that Train's model is better at correctly identifying negative cases (e.g., loan

rejections). It may have a lower false positive rate.

Positive Predictive Value (Precision):Testing data (0.9506) has lower positive predictive value compared to Training data (0.9727).

Reason: Train's model is more precise in predicting positive cases, resulting in fewer false positive predictions.

Test vs.Validation:

Accuracy: Testing data has a lower accuracy (0.961) compared to Validation data(0.968).

Reason: Since, we have took the testing data as 20%, accuracy is less than compared to validation data 30%

Sensitivity (True Positive Rate): Testing data has lower sensitivity (0.6875) compared to Validation data (0.6911).

Reason: Due to overfitting of the model, Validation data model is better at correctly identifying positive cases. This indicates that Validation's model may have a higher false negative rate.

Specificity (True Negative Rate): Testing data has lower specificity (0.9955) compared to Validation data (0.9956).

Reason: validation data model is better at correctly identifying negative cases. Validation's model may have a slightly higher false positive rate.

Positive Predictive Value (Precision): Testing has a higher positive predictive value (0.9506) compared to Validation (0.9400).

Reason: Testing data model is more precise in predicting positive cases, resulting in fewer false positive predictions.

Potential Reasons for Differences:

Data set Differences: Performance of the model can be considerably impacted by differences in the make-up and distribution of the data across distinct sets. As an example, one data collection could be more unbalanced than another, making it more difficult to forecast unusual events.

Model Variability: Performance variances may be caused by different model configurations or uncontrolled initialization of model parameters.

Hyperparameter Tuning: Model performance may be impacted by several hyper parameter settings, including the choice of k in k-NN or other model-specific factors.

Data unyoking: Results may vary, especially for small data sets, if the data sets are divided into training, confirmation, and test sets for each evaluation.

Sample Variability: Variations in the particular samples included in the confirmation and test sets can have an impact on performance criteria in small data sets.

Randomness: Some optimisation processes for models, such as those used in neural networks, include randomness, which can produce small fluctuations.