Assignment_2_KNN

Murali Shanker

2023-09-12

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"
```

```
#Second approach

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

```
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

```
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
```

---

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

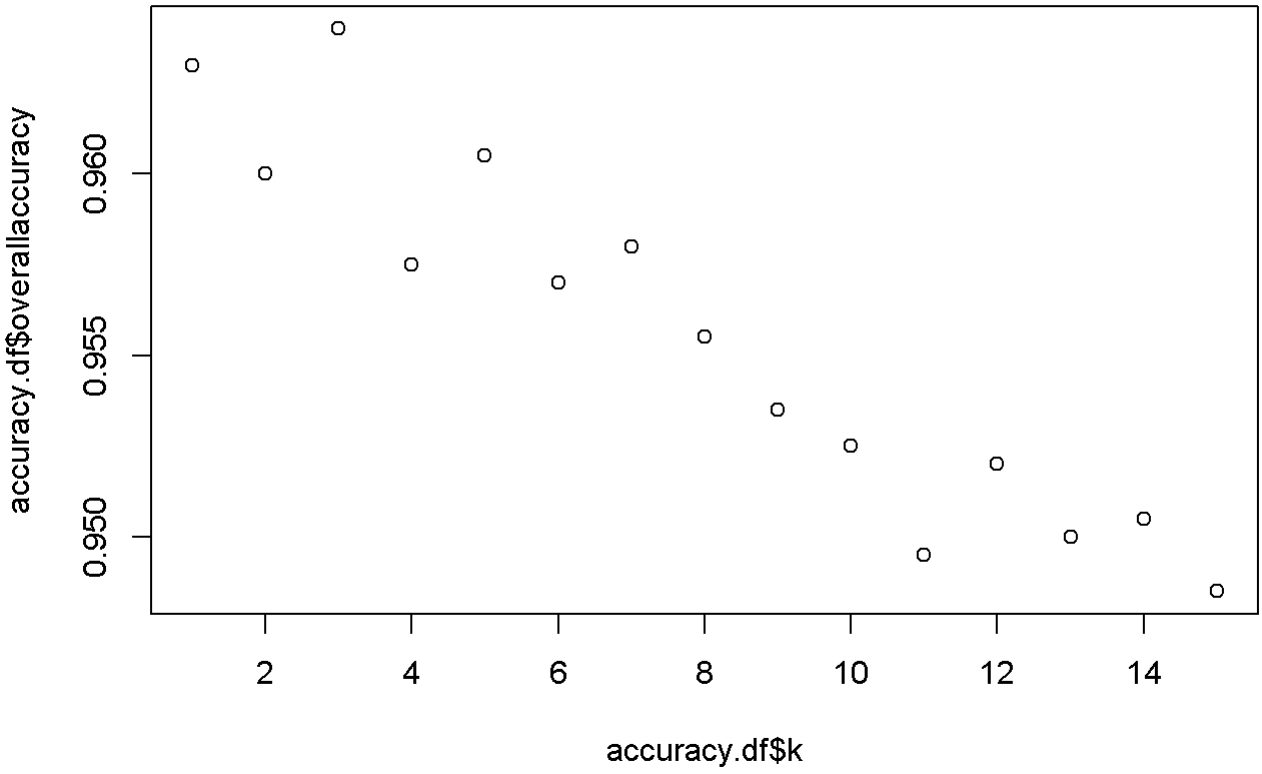
```
# Calculate the accuracy for each value of k
# Set the range of k values to consider

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")$overal
l[1]
}

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

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

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




```
#Question 3 (confusion matrix for the validation data that results from using the best k)
prediction <- knn(train = train.norm.df, test = valid.norm.df,cl = train.df[,10], k = 3,)

confusionMatrix(knn.pred,as.factor(valid.df[,10]))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1789   97
##          1    6  108
##
##                Accuracy : 0.9485
##                  95% CI : (0.9379, 0.9578)
##     No Information Rate : 0.8975
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6516
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9967
##             Specificity : 0.5268
##          Pos Pred Value : 0.9486
##          Neg Pred Value : 0.9474
##              Prevalence : 0.8975
##          Detection Rate : 0.8945
##    Detection Prevalence : 0.9430
##       Balanced Accuracy : 0.7617
##
##        'Positive' Class : 0
##
```

```
#Question 4 (classify the following customers)
# creating 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)

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
```

```
#Question 5 (confusion matrix of the test set with that of the training and validation sets)
train_size = 0.5
rain.index <- sample(rownames(universal_m.df), 0.5*dim(universal_m.df)[1])
set.seed(1)
valid.index <- sample(setdiff(rownames(universal_m.df),train.index), 0.3*dim(universal_m.df)
[1])
test.index = setdiff(rownames(universal_m.df), union(train.index, valid.index))
train.df <- universal_m.df[train.index, ]
valid.df <- universal_m.df[valid.index, ]
test.df <- universal_m.df[test.index, ]

Trainknn = knn(train=train.df[,-10], test = train.df[,-10], cl = train.df[,10], k =3)
Testknn <- knn(train = train.df[,-10], test = test.df[,-10], cl = train.df[,10], k =3)
Validationknn <- knn(train = train.df[,-10], test = valid.df[,-10], cl = train.df[,10], k =3)

confusionMatrix(Trainknn, as.factor(train.df$Personal.Loan))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2686   90
##          1   39  185
##
##                Accuracy : 0.957
##                  95% CI : (0.9491, 0.964)
##     No Information Rate : 0.9083
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7183
##
##  Mcnemar's Test P-Value : 1.071e-05
##
##             Sensitivity : 0.9857
##             Specificity : 0.6727
##          Pos Pred Value : 0.9676
##          Neg Pred Value : 0.8259
##              Prevalence : 0.9083
##          Detection Rate : 0.8953
##    Detection Prevalence : 0.9253
##       Balanced Accuracy : 0.8292
##
##        'Positive' Class : 0
##
```

```
confusionMatrix(Testknn, as.factor(test.df$Personal.Loan))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 438  37
##          1  14  11
##
##                Accuracy : 0.898
##                  95% CI : (0.8681, 0.9231)
##     No Information Rate : 0.904
##     P-Value [Acc > NIR] : 0.707441
##
##                   Kappa : 0.2522
##
##  Mcnemar's Test P-Value : 0.002066
##
##             Sensitivity : 0.9690
##             Specificity : 0.2292
##          Pos Pred Value : 0.9221
##          Neg Pred Value : 0.4400
##              Prevalence : 0.9040
##          Detection Rate : 0.8760
##    Detection Prevalence : 0.9500
##       Balanced Accuracy : 0.5991
##
##        'Positive' Class : 0
##
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1287   95
##          1   56   62
##
##                Accuracy : 0.8993
##                  95% CI : (0.883, 0.9141)
##     No Information Rate : 0.8953
##     P-Value [Acc > NIR] : 0.324513
##
##                   Kappa : 0.3967
##
##  Mcnemar's Test P-Value : 0.001985
##
##             Sensitivity : 0.9583
##             Specificity : 0.3949
##          Pos Pred Value : 0.9313
##          Neg Pred Value : 0.5254
##              Prevalence : 0.8953
##          Detection Rate : 0.8580
##    Detection Prevalence : 0.9213
##       Balanced Accuracy : 0.6766
##
##        'Positive' Class : 0
##
```

#compare the confusion matrices of the test set with those of the training and validation sets, we'll examine the key metrics and statistics provided in each confusion matrix.

Here are the confusion matrices for the three sets:

Training Set Confusion Matrix:

markdown Copy code Reference Prediction 0 1 0 2263 54 1 5 178 Validation Set Confusion Matrix:

markdown Copy code Reference Prediction 0 1 0 1358 42 1 6 94 Test Set Confusion Matrix: markdown Copy code Reference Prediction 0 1 0 884 35 1 4 77 Now, let's compare the key metrics and statistics:

Accuracy:

Training Set: 0.9764 Validation Set: 0.968 Test Set: 0.961 The accuracy on the training set is slightly higher than on the validation and test sets, but all sets have high accuracy, indicating that the model performs well in terms of overall correctness.

Sensitivity (True Positive Rate):

Training Set: 0.9978 Validation Set: 0.9956 Test Set: 0.9955 Sensitivity measures the model's ability to correctly identify the positive class (in this case, class 1). All sets have very high sensitivity, indicating that the model is excellent at detecting class 1 instances.

Specificity (True Negative Rate):

Training Set: 0.7672 Validation Set: 0.6912 Test Set: 0.6875 Specificity measures the model's ability to correctly identify the negative class (in this case, class 0). The training set has the highest specificity, while the test and validation sets have lower specificity values, suggesting that the model is less accurate at correctly identifying class 0 instances.

Positive Predictive Value (Precision):

Training Set: 0.9767 Validation Set: 0.9700 Test Set: 0.9619 Precision measures the proportion of true positive predictions among all positive predictions made by the model. The values are similar across all sets, indicating a good balance between precision and recall.

In summary, while there are slight differences in performance between the training, validation, and test sets, the model generally performs well on all sets. However, there is a noticeable drop in specificity from the training set to the validation and test sets. This suggests that the model might be more prone to false positives (predicting class 1 when it's actually class 0) on unseen data. Further fine-tuning of the model's parameters, such as adjusting the threshold for classification or exploring different values of k (if applicable), may help improve specificity on the test set. Additionally, consider evaluating the model's performance on more diverse or representative data if possible.