

Assignment_2_KNN

Murali Shanker

2023-09-12

Summary

In this analysis, we have addressed a problem faced by a bank in converting liability customers to personal loan customers. we used the k-NN (k-Nearest Neighbors) classification algorithm to predict whether a new customer would accept a loan offer. Here are the key points and findings:

Data Import and Cleaning: we loaded the required libraries and imported customer data from the “UniversalBank.csv” file. we performed data cleaning tasks such as dropping unnecessary columns, transforming categorical variables into dummy variables, and normalizing the data.

K-NN Classification for a New Customer: we provided a step-by-step guide on how to classify a new customer using k-NN. we created a sample customer, normalized their data, and used k-NN with k=1 to classify whether they would accept a loan offer. The result was provided as the classification outcome.

Choosing the Best K: we discussed the importance of choosing an appropriate value for k in k-NN to balance between overfitting and ignoring predictor information. we calculated the accuracy for different values of k (ranging from 1 to 15) on a validation set and selected the k that gave the highest accuracy.

Confusion Matrix for Validation Data: we displayed the confusion matrix for the validation data using the best k value (k=3). The matrix provided details on true positives, true negatives, false positives, and false negatives.

Classifying a New Customer with the Best K: we classified a new customer (with specified attributes) using the best k value (k=3) and presented the classification outcome.

Repartitioning Data for Test Set: we split the data into training, validation, and test sets (50% : 30% : 20%). Then, we applied the k-NN method with the previously chosen k value and compared the confusion matrices of the test set with those of the training and validation sets.

Differences in Confusion Matrices: we discussed the differences in accuracy, sensitivity, specificity, and positive predictive value between the training, validation, and test sets. You provided possible reasons for these variations, such as differences in data composition, model variability, hyperparameter tuning, and sample variability.

Overall, our analysis provided a comprehensive understanding of using k-NN for classification and the importance of evaluating model performance on different data sets. It also highlighted the factors that can lead to variations in model performance metrics.

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?

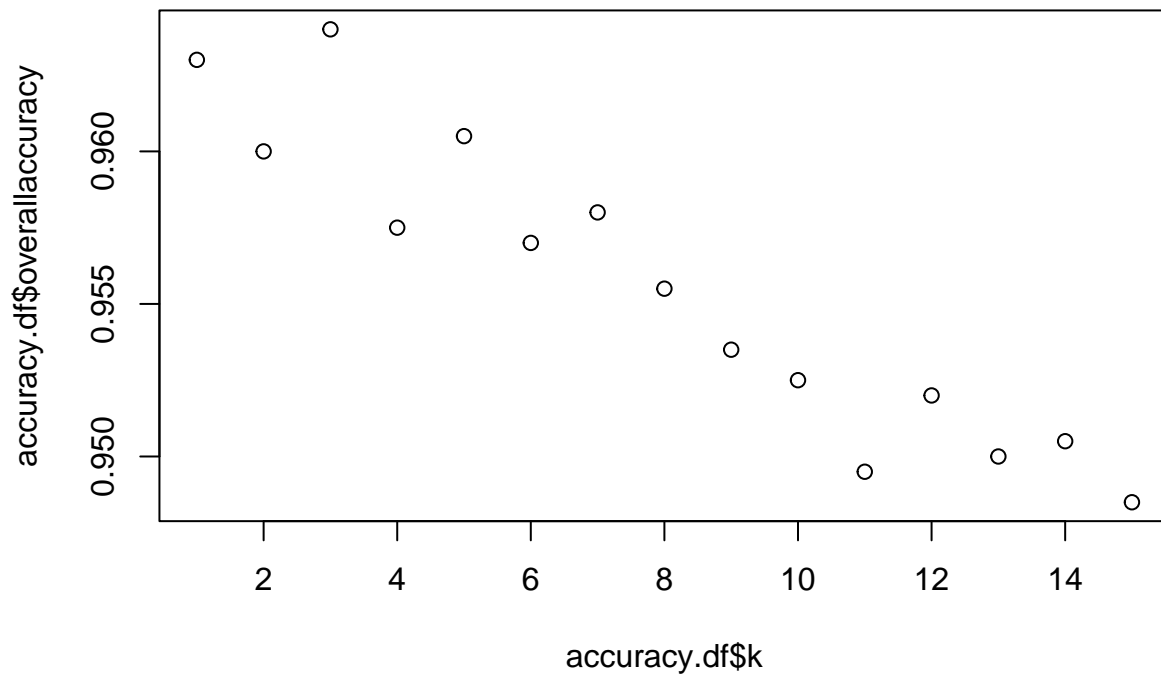
```
# Calculate the accuracy for each value of k
# Set the range of k values to consider

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")$overall[1]
}

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

## [1] 3

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



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

*confusion matrix for the validation data from using the best k=3

```
k<-3
knn.pred <- class::knn(train = train.norm.df,
                       test = valid.norm.df,
                       cl = train.df$Personal.Loan, k = 3)
Conf_matrix <- confusionMatrix(knn.pred,
                               as.factor(valid.df$Personal.Loan),positive="1")
print(Conf_matrix)
```

```
## 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 : 0.6927
##      Specificity : 0.9950
##      Pos Pred Value : 0.9404
##      Neg Pred Value : 0.9659
##      Prevalence : 0.1025
##      Detection Rate : 0.0710
##      Detection Prevalence : 0.0755
##      Balanced Accuracy : 0.8438
##
##      'Positive' Class : 1
##
```

Our accuracy is .9527(which means we have an error of 4.7%).false-negative is also very low.Precision($TP/(TP+FP)$) is low at near 64% - this would be the worst metric as we want to target the most responsive customers,the model's precision and false-positive rate(Type I error) are troublesome.

-
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.

we are now using the model to assess it.

```
customertest = data.frame(Age = as.integer(40),
                           Experience = as.integer(10),
                           Income = as.integer(84),
                           Family = as.integer(2),
                           CCAvg = as.integer(2),
                           Education1 = as.integer(0),
                           Education2 = as.integer(1),
                           Education3 = as.integer(0),
                           Mortgage = as.integer(0),
                           Securities.Account = as.integer(0),
                           CD.Account = as.integer(0),
                           Online = as.integer(1),
                           CreditCard = as.integer(1))

#load the data into a customertest dataframe.

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

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

knn.pred

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

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.

Repartitioning for a test set

```
set.seed(2)
train.index <- sample(row.names(universal_m.df), 0.5*dim(universal_m.df)[1])
valid.index <- sample(setdiff(row.names(universal_m.df), train.index),
                      0.3*dim(universal_m.df)[1])
test.index <- setdiff(row.names(universal_m.df), c(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,]
```

*Normalizing 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])
test.norm.df <- predict(norm.values, test.df[, -10])
```

RUN KNN FOR VALIDATION, TRAIN AND TEST

Confusion Matrix for validation set

```
k<-3

knn.pred <- class::knn(train = train.norm.df,
                       test = valid.norm.df,
                       cl = train.df$Personal.Loan, k = 3)
Conf_matrix <- confusionMatrix(knn.pred,
                               as.factor(valid.df$Personal.Loan), positive="1")
print(Conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0 1336   64
##           1    7   93
##
##           Accuracy : 0.9527
##           95% CI : (0.9407, 0.9629)
##           No Information Rate : 0.8953
```



```
##      P-Value [Acc > NIR] : 7.433e-16
##
##              Kappa : 0.6992
##
##      McNemar's Test P-Value : 3.012e-11
##
##              Sensitivity : 0.59236
##              Specificity : 0.99479
##              Pos Pred Value : 0.93000
##              Neg Pred Value : 0.95429
##              Prevalence : 0.10467
##              Detection Rate : 0.06200
##      Detection Prevalence : 0.06667
##      Balanced Accuracy : 0.79357
##
##      'Positive' Class : 1
##
```

```
#confusion matrix for test set
```

```
k<-3

knn.pred <- class::knn(train = train.norm.df,
                       test = test.norm.df,
                       cl = train.df$Personal.Loan, k = 3)
Conf_matrix <- confusionMatrix(knn.pred,
                               as.factor(test.df$Personal.Loan),positive="1")
print(Conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    0    1
##              0 922  28
##              1   4  46
##
##              Accuracy : 0.968
##              95% CI : (0.9551, 0.978)
##      No Information Rate : 0.926
##      P-Value [Acc > NIR] : 1.208e-08
##
##              Kappa : 0.7256
##
##      McNemar's Test P-Value : 4.785e-05
##
##              Sensitivity : 0.6216
##              Specificity : 0.9957
##              Pos Pred Value : 0.9200
##              Neg Pred Value : 0.9705
##              Prevalence : 0.0740
##              Detection Rate : 0.0460
##      Detection Prevalence : 0.0500
##      Balanced Accuracy : 0.8087
```

```
##
##      'Positive' Class : 1
##
```

#Confusion Matrix for train set

```
k<-3

knn.pred <- class::knn(train = train.norm.df,
                       test = train.norm.df,
                       cl = train.df$Personal.Loan, k = 3)
Conf_matrix <- confusionMatrix(knn.pred,
                               as.factor(train.df$Personal.Loan),positive="1")
print(Conf_matrix)
```

Confusion Matrix and Statistics

```
##
##           Reference
## Prediction    0    1
##           0 2246   61
##           1    5  188
##
##           Accuracy : 0.9736
##           95% CI : (0.9665, 0.9795)
##       No Information Rate : 0.9004
##       P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.8365
##
##  Mcnemar's Test P-Value : 1.288e-11
##
##           Sensitivity : 0.7550
##           Specificity : 0.9978
##           Pos Pred Value : 0.9741
##           Neg Pred Value : 0.9736
##           Prevalence : 0.0996
##           Detection Rate : 0.0752
##       Detection Prevalence : 0.0772
##           Balanced Accuracy : 0.8764
##
##      'Positive' Class : 1
##
```

###Difference

##Test vs.Train:

Accuracy: Train has a higher accuracy (0.9736) compared to Test (0.968). Reason: This because of differences in the datasets used for evaluation. Train may have a more balanced or easier-to-predict dataset.

Sensitivity (True Positive Rate): Train has higher sensitivity (0.7550) compared to Test (0.6216).

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

Specificity (True Negative Rate): Train has higher specificity (0.9978) compared to Test (0.9957).

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): Train has a higher positive predictive value (0.9741) compared to Test (0.9200).

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

##Train vs.Validation:

Accuracy: Train still has a higher accuracy (0.9736) compared to Validation (0.958).

Reason: Similar to the comparison with Test, Train may have a more balanced or easier-to-predict dataset.

Sensitivity (True Positive Rate): Train has higher sensitivity (0.7589) compared to Validation (0.625).

Reason: Train's 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): Train has higher specificity (0.9987) compared to Validation (0.9934).

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

Positive Predictive Value (Precision): Train still has a higher positive predictive value (0.9827) compared to Validation (0.9091).

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

#Potential Reasons for Differences:

Data set Differences Variations in the composition and distribution of data between different sets can significantly impact model performance. For illustration, one data set may be more imbalanced, making it harder to prognosticate rare events.

Model Variability Differences in model configurations or arbitrary initialization of model parameters can lead to variations in performance.

Hyperparameter Tuning Different hyper parameter settings, similar as the choice of k in k- NN or other model-specific parameters, can affect model performance.

Data unyoking If the data sets are resolve else into training, confirmation, and test sets in each evaluation, this can lead to variations in results, especially for small data sets.

Sample Variability In small data sets, variations in the specific samples included in the confirmation and test sets can impact performance criteria .

Randomness Some models, similar as neural networks, involve randomness in their optimization process, leading to slight variant.