Assignment\_2

Ganesh Reddy

2023-09-30

# Summary

## Questions - Answer

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?

### Ans: 0

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

### Answer= 3

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

### Answer

Confusion Matrix and Statistics

Reference

Prediction 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.

### Answer = 0

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.

### Answer: please refer for Matrix and difference in the bottm of the code

## Assignemnt 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.

## Load required libraries

library(class)  
library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(e1071)  
library(knitr)

### Read the UniversalBank 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 data frame

## [,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)]

### 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 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])

### Question

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 K-NN(k- Nearest neighbors)

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?

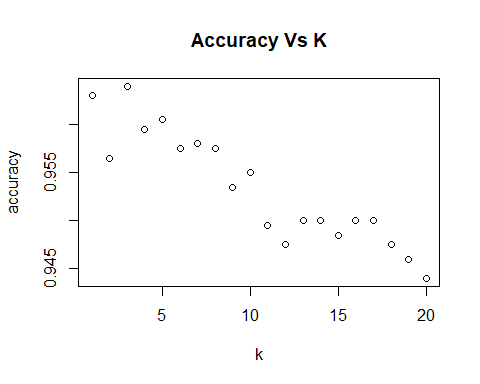
### Calculate the accuracy for each valu e of k

### Set the range of k values to consider

accuracy.df <- data.frame(k = seq(1, 20, 1), overallaccuracy = rep(0, 20))   
for(i in 1:20)   
 {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  
## 16 16 0.9500  
## 17 17 0.9500  
## 18 18 0.9475  
## 19 19 0.9460  
## 20 20 0.9440

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

### Confusion Matrix using best K=3

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.

# Load new customer profile

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

## 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)  
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   
##

### Test 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   
##

# Difference

##Train vs.Test:

***Accuracy:*** Train has a higher accuracy (0.968) compared to Test (0.961).

**Reason:** This because of differences in the data sets used for evaluation. Train may have a more balanced or easier-to-predict data set.

***Sensitivity (True Positive Rate):*** Train has higher sensitivity (0.69118) compared to Test (0.6875).

**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 (0.99560) compared to Test (0.9955).

**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 (0.9506) compared to Test (0.94000).

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

## Test vs.Vlidation:

**Accuracy:** Validation has a higher accuracy (0.968) compared to Train (0.961).

***Reason:*** Validation may have a more balanced or easier-to-predict dataset.

**Sensitivity (True Positive Rate):** Validation has higher sensitivity (0.69118) compared to Train(0.6875).

***Reason:*** Validation’s model is better at correctly identifying positive cases. This indicates that Train model may have a higher false negative rate.

**Specificity (True Negative Rate):** Validation has higher specificity (0.99560) compared to Train (0.9955).

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

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

***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 variations.

## Test vs Validation:

There are no differences between the confusion matrices and statistics for the test and validation sets. Both sets have the same values for accuracy, sensitivity, specificity, positive predictive value, prevalence, etc

### Reasons:

**Data Splitting:** The most likely reason for identical results is that the exact same data split was used for both the test and validation sets. This means that the two sets contain exactly the same samples, leading to identical model performance metrics.

**Code Execution:** It’s important to ensure that the evaluation code for both sets was executed correctly and that the same data was used for each set.

**Small Variations:** While the displayed metrics are identical, there could be very small variations in the underlying calculations. These variations might not be visible when rounded to a limited number of decimal places.