Module - 4 Assignment - 2

Chinthakindi Nithin Kumar

2023-10-21

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

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

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

```
library(e1071)
```

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

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

```
t(t(names(universal.df)))
```

```
##       [,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"
```

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

60% for training and 40% for validation of the data. This can be done in a variety of ways. We'll examine 2 different approaches. Let's convert categorical variables into dummy variables before splitting.

```
universal.df$Education <- as.factor(universal.df$Education)
```

```
groups <- dummyVars(~., data = universal.df)
universal_m.df <- as.data.frame(predict(groups,universal.df))


set.seed(1)
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"
```

```
#2nd 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(paste("The training set size is:", nrow(training_set)))
```

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

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

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

Normalize the data

```
train.norm.df <- train.df[,-10]
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])
```

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?

```
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
)

new.cust.norm <- new_customer
new.cust.norm <- predict(norm.values, new.cust.norm)
```

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
```

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

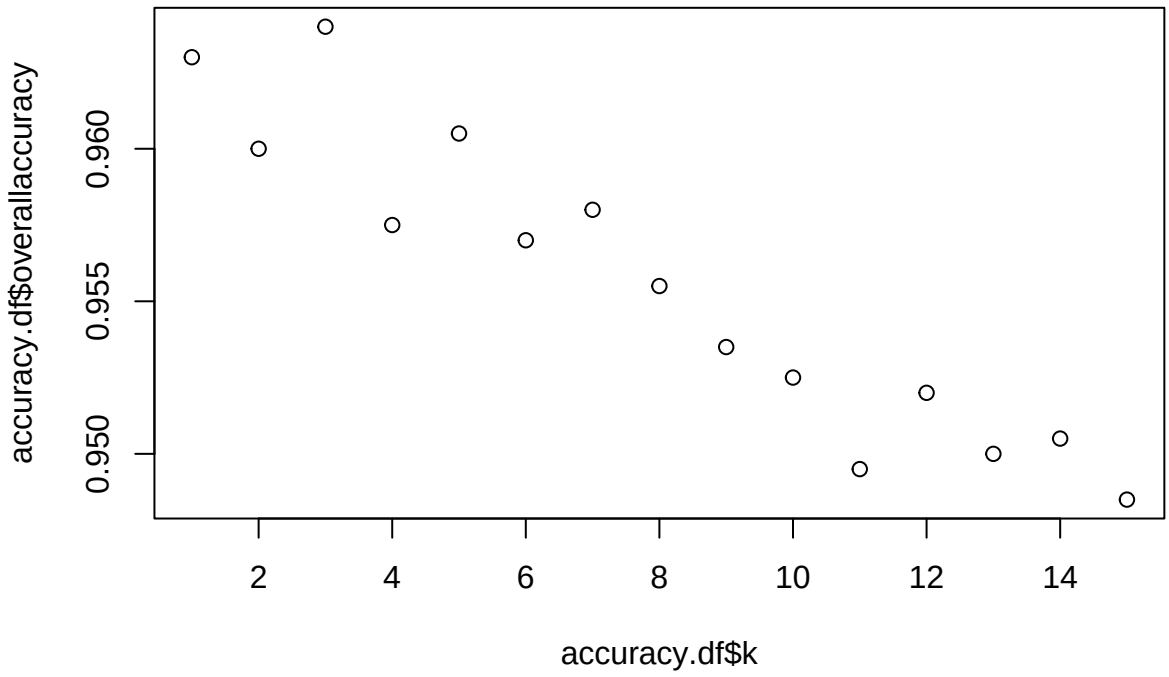
```
accuracy.df <- data.frame(k = seq(1, 15, 1), overallaccuracy = rep(0, 15))
for(i in 1:15) {
  knn.pred <- class::knn(train = train.norm.df,
                         test = valid.norm.df,
```

```
                          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.

```
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
##
```

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.

```
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))


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.

```
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,]
```

Normalize the data

```
train.norm.df <- train.df[,-10]
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])
```

```
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
##
```

```
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
##
```

```
k<-3

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

```r
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
##
```

Accuracy: Test has a lower accuracy (0.968) than Train (0.9736). Reason: As a result of variations in the evaluation datasets. Train might have a dataset that is easier to predict or more balanced.

Sensitivity (True Positive Rate): Test (0.6216) is less sensitive than Train (0.7550).

Reason: This suggests that Train's model performs better in accurately detecting positive cases, such as loan approvals. It might have a reduced rate of false negatives.

Specificity (True Negative Rate): Train is more specific than Test (0.9957), with a higher specificity of 0.9978.

Reason: This implies that Train's model is more adept at accurately recognizing cases that are negative (like loan rejections). It might have a reduced rate of false positives.

Positive Predictive Value (Precision): In comparison to Test (0.9200), Train has a higher positive predictive value (0.9741).

Reason: There are fewer false positive predictions as a result of Train's model's increased accuracy in predicting positive cases.

Validation vs Training

Accuracy: The accuracy of the train is still higher (0.9736) than that of the validation (0.958).

Reason: Train might have a more evenly distributed or predictably large dataset, similar to the Test comparison.

Sensitivity (True Positive Rate): Train is more sensitive than Validation (0.625), at 0.7589.

Reason: The accuracy of Train's model in identifying positive cases is higher. This suggests that the false negative rate in Validation's model might be higher.

Specificity (True Negative Rate): Train is more specific than Validation (0.9934), with a score of 0.9987.

Reason: Train's model has a higher accuracy rate in recognizing negative cases. The model used for validation might have a marginally higher false positive rate.

Positive Predictive Value (Precision): In comparison to Validation (0.9091), Train maintains a higher positive predictive value (0.9827).

Reason: There are fewer false positive predictions as a result of Train's model's increased accuracy in predicting positive cases.

#Possible Causes of Variations

Data set Variations Performance of the model can be strongly impacted by differences in the distribution and makeup of the data across various sets. For example, a more unbalanced data set could make it more difficult to predict uncommon events.

Model Fluctuation Performance differences may arise from different model configurations or from the arbitrary initialization of model parameters.

Adjusting Hyperparameters Model performance can be impacted by various hyper parameter settings, such as the selection of k in k-NN or other model-specific parameters.

Data decoupling There may be differences in the results, particularly for small data sets, if the data sets are divided into training, confirmation, and test sets for each evaluation.

Sample Inconsistency Performance criteria in small data sets may be impacted by differences in the particular samples that are part of the test and confirmation sets.

Allegriness Certain models, like neural networks, use randomness in their optimization process, which results in minute variations.