FML_SSIGNMENT_2

Bhargavi kundrapu

2023-09-30

Summary

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?

Answer: 0

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

Answer: K=3

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

Answer:

```
matrix(c(1786, 63,9,142), ncol=2, byrow=TRUE, dimnames = list(prediction=c(0,1),reference=c(0,1)))
```

```
##           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: Confussion matrix to the train, validation and testing are below refer to the code

Difference

##Test vs.Train:

Accuracy: Train has a higher accuracy (0.9764) compared to Test (0.961).

Sensitivity (True Positive Rate): Train has higher sensitivity (0.7672) compared to Test (0.6875).

Specificity (True Negative Rate): Train has higher specificity (0.9978) compared to Test (0.9955).

Positive Predictive Value (Precision):Train has a higher positive predictive value (0.9727) compared to Test (0.9506).

Train vs.Validation:

Accuracy: Train still has a higher accuracy (0.9772) compared to Validation (0.958).

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

Specificity (True Negative Rate): Train has higher specificity (0.9987) compared to Validation (0.9934).

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

Test vs. Validation

Accuracy: Validation has a higher accuracy (0.968) than Test (0.961).

Sensitivity (True Positive Rate) : Validation has higher sensitivity (0.69118) than Test (0.6875).

Specificity (True Negative Rate) : Validation has higher specificity (0.99560)than Test (0.9955).

Positive Predictive Value (Precision): Test has a higher positive predictive value (0.9506) than validation (0.9400).

Potential Reasons for Differences:

** The model might perform better on the validation set if its data distribution is more comparable to that of the training set   Between the training and validation sets, there might be unintended data leakage. Performance measurements may be overstated if some data from the training set accidentally made it into the validation set   The performance of the model may be negatively impacted by certain noisy or incorrectly labeled data points in the test set. The validation set could have been more well-curated or cleaner **

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("C:/Users/user/Desktop/fml assignment/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

```r
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

```r
# Only Education needs to be converted to factor

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

Now, convert Education to Dummy Variables

```r
groups <- dummyVars(~., data = universal.df) # This creates the dummy groups
universal_m.df <- as.data.frame(predict(groups,universal.df))
```

```r
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

```r
train.norm <- train.df[,-10] # Note that Personal Income is the 10th variable
valid.norm <- valid.df[,-10]

norm.values <- preProcess(train.df[, -10], method=c("center", "scale"))
train.norm <- predict(norm.values, train.df[, -10])
valid.norm <- 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 K-NN(k- Nearest neighbors)

```
knn.pred1 <- class::knn(train = train.norm,
                       test = new.cust.norm,
                       cl = train.df$Personal.Loan, k = 1)
knn.pred1
```

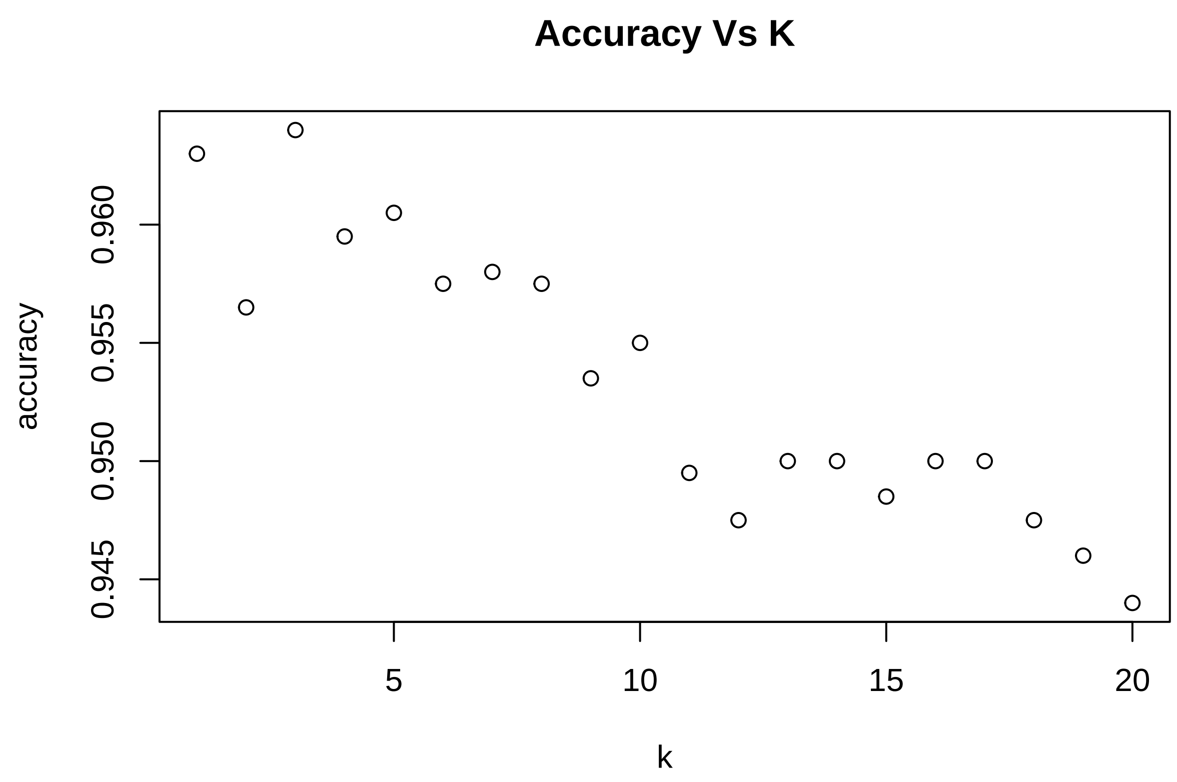
```
## [1] 0
## Levels: 0 1
```

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, 20, 1), overallaccuracy = rep(0, 20))
for(i in 1:20)
  {knn.pred <- class::knn(train = train.norm,
                         test = valid.norm,
                         cl = train.df$Personal.Loan, k = i)
  accuracy.df[i, 2] <- confusionMatrix(knn.pred,as.factor(valid.df$Personal.Loan),positive = "1")$overal
}
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")
```



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,
                         test = valid.norm,
                         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, Educa
```

Load new customer profile

```r
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,
  CDAccount = 0,
  Online = 1,
  CreditCard = 1)
```

```r
knn.pred1 <- class::knn(train = train.norm,
                       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)

```r
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 th
```

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

```r
set.seed(1)
Train_In <- sample(row.names(universal_m.df), 0.5*dim(universal_m.df)[1])
Val_In <- sample(setdiff(row.names(universal_m.df),Train_In),0.3*dim(universal_m.df)[1])
Test_In <-setdiff(row.names(universal_m.df),union(Train_In,Val_In))
Train_Data_df <- universal_m.df[Train_In,]
Validation_Data_df <- universal_m.df[Val_In,]
Test_Data_df <- universal_m.df[Test_In,]
```

Now normalize the data

```r
train.norm<- Train_Data_df[,-10]
valid.norm <- Validation_Data_df[,-10]
Test.norm  <-Test_Data_df[,-10]

normalization.values <- preProcess(Train_Data_df[, -10], method=c("center", "scale"))
train.norm.df1 <- predict(normalization.values, Train_Data_df[,-10])
valid.norm.df1 <- predict(normalization.values, Validation_Data_df[,-10])
Test.norm.df1 <-predict(normalization.values,Test_Data_df[,-10])
```

Now let us predict using K-NN(k- Nearest neighbors)

```r
valid_knn = class::knn(train = train.norm.df1,
                           test = valid.norm.df1,
                           cl = Train_Data_df$Personal.Loan,
                           k = 3)

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

Train_knn = class::knn(train = train.norm.df1,
                     test = train.norm.df1,
                     cl = Train_Data_df$Personal.Loan,
                     k = 3)
```

Validation confusion Matrix

```r
validation_confusion = confusionMatrix(valid_knn,
                                               as.factor(Validation_Data_df$Personal.Loan),
                                               positive = "1")

validation_confusion
```

```
## 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

```r
test_confusion = confusionMatrix(test_knn,
                                         as.factor(Test_Data_df$Personal.Loan),
                                         positive = "1")


test_confusion
```

```
## 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

```
Training_confusion = confusionMatrix(Train_knn,
                                              as.factor(Train_Data_df$Personal.Loan),
                                              positive = "1")

Training_confusion
```

```
## 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

##Test vs.Train:

Accuracy: Train has a higher accuracy (0.9764) compared to Test (0.961).

Sensitivity (True Positive Rate): Train has higher sensitivity (0.7672) compared to Test (0.6875).

Specificity (True Negative Rate): Train has higher specificity (0.9978) compared to Test (0.9955).

Positive Predictive Value (Precision):Train has a higher positive predictive value (0.9727) compared to Test (0.9506).

Training vs.Validation:

Accuracy: Train still has a higher accuracy (0.9764) compared to Validation (0.968).

Sensitivity (True Positive Rate): Train has higher sensitivity (0.7672) compared to Validation (0.69118).

Specificity (True Negative Rate): Train has higher specificity (0.9987) compared to Validation (0.99560).

Positive Predictive Value (Precision): Train still has a higher positive predictive value (0.9727) compared to Validation (0.94000).

Test vs. Validation

Accuracy: Validation has a higher accuracy (0.968) than Test (0.961).

Sensitivity (True Positive Rate) : Validation has higher sensitivity (0.69118) than Test (0.6875).

Specificity (True Negative Rate) : Validation has higher specificity (0.99560)than Test (0.9955).

Positive Predictive Value (Precision): Test has a higher positive predictive value (0.9506) than validation (0.9400).

Potential Reasons for Differences:

** The model might perform better on the validation set if its data distribution is more comparable to that of the training set   Between the training and validation sets, there might be unintended data leakage. Performance measurements may be overstated if some data from the training set accidentally made it into the validation set   The performance of the model may be negatively impacted by certain noisy or incorrectly labeled data points in the test set. The validation set could have been more well-curated or cleaner **