Assignment–2

Sai goud pvn

2023-10-01

Loading the required libraries

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

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
library(e1071)
library(knitr)
```

#Importing 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"
```

#Droping ID and ZIP

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

#Transforming 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))
```

#Splitting 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"
```

#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])
```

#Question

```
1. Age = 40, Experience = 10, Income = 84, Family = 2, CCAvg = 2, Education_1 = 0, Education_2 = 1, Edu
```

#We have converted all categorical variables to dummy variables

#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)
```

#Normalizing the new customer

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

#predicting 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
```

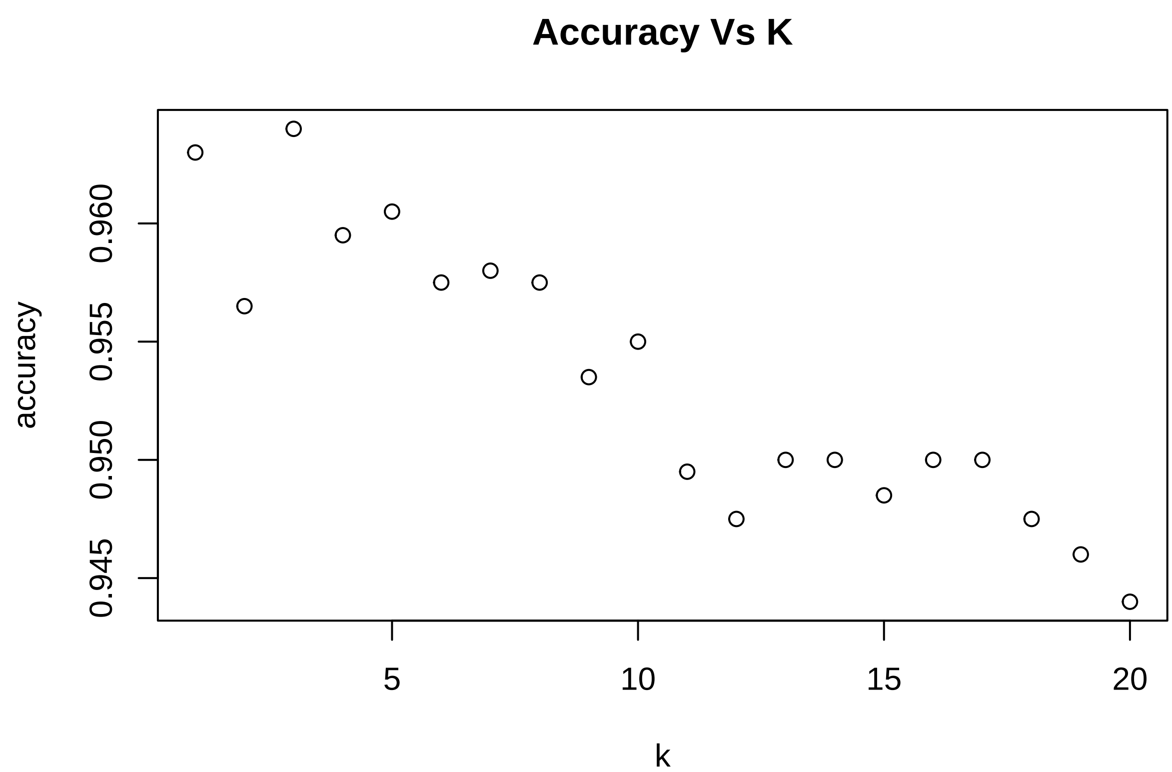
```
## [1] 0
## Levels: 0 1
```

2. What is a choice of k that balances between overfitting and ignoring the predictor information? #Calculating the accuracy for each value of k
   #Setting the range of k values for consideration

```
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")$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.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.

#Loading new customer profile

```r
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)
```

```r
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
```

#Printing the predicted class (1 = loan acceptance, 0 = 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 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.

#Spliting the data to 50% training and 30% Validation and 20% Testing

```r
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,]
```

#Normalizing the data

```r
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])
```

#predicting K-NN(k- Nearest neighbors)

```r
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 of confusion Matrix

```r
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
##
```

#Testing confusion Matrix

```r
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
##
```

```
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
##
```

#Test v/s.Train:

#Accuracy: Train has a higher accuracy (0.9772) compared to Test (0.9507).

#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 (0.7589) compared to Test (0.5875).

#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 (0.9987) compared to Test (0.99403).

#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 (0.9827) compared to Test (0.92157).

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

Train v/s.Validation:

#Accuracy: Train still has a higher accuracy (0.9772) 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 releasing: 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.