Assignment2

asif

2024-02-25

```
#install.packages("class")
library(class)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
bank_df<-read.csv("C:/Users/asifs/OneDrive/Desktop/UniversalBank.csv")
names(bank_df)
```

```
##  [1] "ID"                 "Age"                "Experience"
##  [4] "Income"             "ZIP.Code"           "Family"
##  [7] "CCAvg"              "Education"          "Mortgage"
## [10] "Personal.Loan"      "Securities.Account" "CD.Account"
## [13] "Online"             "CreditCard"
```

```
#drop id and zipcode columns
bank_df<-bank_df[,-c(1,5)]
print("updated dataframe by removing the id and zipcode columns")
```

```
## [1] "updated dataframe by removing the id and zipcode columns"
```

```
names(bank_df)
```

```
##  [1] "Age"                "Experience"         "Income"
##  [4] "Family"             "CCAvg"              "Education"
##  [7] "Mortgage"           "Personal.Loan"      "Securities.Account"
## [10] "CD.Account"         "Online"             "CreditCard"
```

```
head(bank_df)
```

```
##   Age Experience Income Family CCAvg Education Mortgage Personal.Loan
## 1  25          1     49      4   1.6         1        0             0
## 2  45         19     34      3   1.5         1        0             0
## 3  39         15     11      1   1.0         1        0             0
## 4  35          9    100      1   2.7         2        0             0
## 5  35          8     45      4   1.0         2        0             0
## 6  37          13     29      4   0.4          2      155              0
##   Securities.Account CD.Account Online CreditCard
## 1                  1          0      0          0
## 2                  1          0      0          0
## 3                  0          0      0          0
## 4                  0          0      0          0
## 5                  0          0      0          1
## 6                  0          0      1          0
```

```
#identifying the categorial variables from the dataframe.
categorical_vars <- sapply(bank_df, function(x) is.factor(x) || is.character(x))
categorical_var_names <- names(categorical_vars)[categorical_vars]
#converting education column into  factor
bank_df$Education <- as.factor(bank_df$Education)
# Now, convert Education to Dummy Variables
groups <- dummyVars(~., data = bank_df)
#combing the dummy variables and original dataframe
bank_df<- as.data.frame(predict(groups,bank_df))

#splitting the data
set.seed(1) # Important to ensure that we get the same sample if we rerun the code
train.index <- sample(row.names(bank_df), 0.6*dim(bank_df)[1])
valid.index <- setdiff(row.names(bank_df), train.index)
training_data <- bank_df[train.index,]
validation_data <- bank_df[valid.index,]
names(training_data)
```

```
##  [1] "Age"                "Experience"         "Income"
##  [4] "Family"             "CCAvg"              "Education.1"
##  [7] "Education.2"        "Education.3"        "Mortgage"
## [10] "Personal.Loan"      "Securities.Account" "CD.Account"
## [13] "Online"             "CreditCard"
```

```
nrow(training_data)
```

```
## [1] 3000
```

```
#next step is normalising the data
#copy the original data removing personal loan column
training_norm<-training_data[,-10]
names(training_norm)
```

```
##  [1] "Age"                "Experience"         "Income"
##  [4] "Family"             "CCAvg"              "Education.1"
##  [7] "Education.2"        "Education.3"        "Mortgage"
## [10] "Securities.Account" "CD.Account"         "Online"
## [13] "CreditCard"
```

```
validation_norm<-validation_data[,-10]
#using preprocess from caret package to mormalise
norm_values <- preProcess(training_data[, -10], method=c("center", "scale"))
```

```
training_norm <- predict(norm_values, training_data[, -10])
validation_norm <- predict(norm_values, validation_data[, -10])
# create a new sample using Age = 40, Experience = 10, Income = 84, Family = 2, CCAvg = 2, Education_1
new_sample<-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)
#normalising the new sample
sample_norm <- new_sample
sample_norm <- predict(norm_values,sample_norm)
#its time tp perform knn using k=1
knn.value1 <- class::knn(train = training_norm,
                        test = sample_norm,
                        cl = training_data$Personal.Loan, k = 1)
knn.value1
```

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

```
#What is a choice of k that balances between overfitting and ignoring the predictor information? varyin
accuracy.df <- data.frame(k = seq(1, 25, 1), overallaccuracy = rep(0, 25))
for(i in 1:25)
{knn.pred <- class::knn(train = training_norm,
test = validation_norm,
cl = training_data$Personal.Loan, k = i)
accuracy.df[i, 2] <- confusionMatrix(knn.pred,as.factor(validation_data$Personal.Loan),positive = "1")$
}
which(accuracy.df[,2] == max(accuracy.df[,2]))
```

```
## [1] 3
```

```
#Show the confusion matrix for the validation data that results from using the best k.
#using the k value from previous output.
knn.value2 <- class::knn(train = training_norm,
                        test = validation_norm,
                        cl = training_data$Personal.Loan, k = 3)
confusionMatrix(knn.value2,as.factor(validation_data$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
##
```

```r
#Consider the following customer: Age = 40, Experience = 10, Income = 84,Family = 2, CCAvg = 2, Educati

#creating smaple2 with above values
sample2<--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)

#Now, performing the knn using k=3 i.e best choice of k.
#new sample and sample2 is same, no need of seperating calculation of normalisation. just printing the
knn.value3 <- class::knn(train = training_norm,
                        test = sample_norm,
                        cl = training_data$Personal.Loan, k = 3)
knn.value3
```

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

```r
#as per question, if personalloan is 1, then customers accept the loan otherwise rejecting.
if (knn.value3 == 1) {
  print("This customer would be classified as accepting the loan")
} else {
  print("This customer would be classified as rejecting the loan")
}
```

```
## [1] "This customer would be classified as rejecting the loan"
```

```r
#Repartition the data, this time into training, validation, and test sets (50% : 30% : 20%). Applythe k

#partitioning of data
set.seed(123)  # Set seed for reproducibility
# dividing the dataser into halves, training data 50 and rem_data 50
train_index <- createDataPartition(bank_df$Personal.Loan, p = 0.5, list = FALSE)
train_data<-bank_df[train_index, ]
rem_data <- bank_df[-train_index, ]
#using the rem_data, validation and testing data is splitted.
valid_index <- createDataPartition(rem_data$Personal.Loan, p = 0.6, list = FALSE)
valid_data <- rem_data[-valid_index, ]
test_data <- rem_data[valid_index, ]
names(train_data)
```

```
##  [1] "Age"                "Experience"         "Income"
##  [4] "Family"             "CCAvg"              "Education.1"
##  [7] "Education.2"        "Education.3"        "Mortgage"
## [10] "Personal.Loan"      "Securities.Account" "CD.Account"
## [13] "Online"             "CreditCard"
```

```r
#normalising the data
train_norm1 <- train_data[,-10]
valid_norm1 <- valid_data[,-10]
test_norm1 <-test_data[,-10]
norm.values1 <- preProcess(train_data[, -10], method=c("center", "scale"))
train_norm1 <- predict(norm.values1, train_data[,-10])
valid_norm1 <- predict(norm.values1, valid_data[,-10])
test_norm1 <-predict(norm.values1,test_data[,-10])

#knn classification for train data
train_knn = class::knn(train = train_norm1,
                        test = train_norm1,
                        cl = train_data$Personal.Loan,k = 3)

#confusion matrix for training data
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 2267   57
##          1    4  172
##
##                Accuracy : 0.9756
##                  95% CI : (0.9688, 0.9813)
##     No Information Rate : 0.9084
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8364
##
##  Mcnemar's Test P-Value : 2.777e-11
##
##             Sensitivity : 0.7511
##             Specificity : 0.9982
##          Pos Pred Value : 0.9773
##          Neg Pred Value : 0.9755
##              Prevalence : 0.0916
##          Detection Rate : 0.0688
##    Detection Prevalence : 0.0704
##       Balanced Accuracy : 0.8747
##
##        'Positive' Class : 1
##
```

```
#knn classification for validation data
validation_knn = class::knn(train = train_norm1,
                            test = valid_norm1,
                            cl = train_data$Personal.Loan,k = 3)

#confusion matrix for validation data
validation_confusion_matrix =confusionMatrix(validation_knn,as.factor(valid_data$Personal.Loan),
                                             positive = "1")
validation_confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 890  38
##          1   2  70
##
##                Accuracy : 0.96
##                  95% CI : (0.9459, 0.9713)
##     No Information Rate : 0.892
##     P-Value [Acc > NIR] : 4.095e-15
##
##                   Kappa : 0.7568
##
##  Mcnemar's Test P-Value : 3.130e-08
##
##             Sensitivity : 0.6481
##             Specificity : 0.9978
##          Pos Pred Value : 0.9722
##          Neg Pred Value : 0.9591
##              Prevalence : 0.1080
##          Detection Rate : 0.0700
##    Detection Prevalence : 0.0720
##       Balanced Accuracy : 0.8230
##
##        'Positive' Class : 1
##
```

```
#knn classification for testing data
test_knn = class::knn(train = train_norm1,
                      test = test_norm1,
                      cl = train_data$Personal.Loan,k = 3)

#confusion matrix for testing data
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 1350   58
##          1    7   85
##
##                Accuracy : 0.9567
##                  95% CI : (0.9451, 0.9664)
##     No Information Rate : 0.9047
##     P-Value [Acc > NIR] : 2.347e-14
##
##                   Kappa : 0.7011
##
##  Mcnemar's Test P-Value : 5.584e-10
##
##             Sensitivity : 0.59441
##             Specificity : 0.99484
##          Pos Pred Value : 0.92391
##          Neg Pred Value : 0.95881
##              Prevalence : 0.09533
##          Detection Rate : 0.05667
##    Detection Prevalence : 0.06133
##       Balanced Accuracy : 0.79462
##
##        'Positive' Class : 1
##
```

Reasons for Differences:

1.Differences in the data set, the performance of a model can be greatly impacted by differences in the size and composition of the data within several sets. It could be more difficult to predict unusual events.

2.Performance variances may be caused by different model configurations or by arbitrary model parameter initialization.

3.Model performance can be affected by several hyper parameter settings, such as the selection of k in k-NN or other model-specific factors.

Differences between Train and validation data:

1.Accuracy: Training data has accuracy 0.97 and validation data 0.96. Training dataset is more balanced and easy to predict.

2.specificity: training data has 0.99 and validation data has 0.9978. validation dataset has higher false negative rate.

3.precision: Training data has 0.9773 and validation data has 0.9722. trains model is high in predicting positive cases. it results fewer false positive predictions.

4.Sensitivity: Training model have 0.7511 compared to validation model 0.6481. the values itself states that validation model have false negative rate. Train model have correctly identifying the positive cases.

Differences between train and test model:

1.Accuracy: Train model have accuracy of 0.97 and test model have 0.9567. Train model is more precise and balanced data and easy to predict.

2.Specificity: Train model have 0.9982 and test model have 0.9948. train model may have lower false positive cases.

3.Sensitivity: Train model have 0.7511 and test model have 0.594. train model may have lower false negative cases.

4.precision: train model have 0.977 and test model have 0.9239. Train model is more precise in predicting positive cases, results in less false positive cases.