Assignment_2_KNN

YASHWANTH K

2023-09-29

Summary

Accuracy

The training set has slightly higher accuracy than the validation and test sets, but all sets have high accuracy, indicating that the model performs well overall.

Training Set: 0.9764 Validation Set: 0.9673 Test Set: 0.962

Sensitivity (True Positive Rate):

Sensitivity assesses the model's ability to correctly identify the positive class (in this case, class 1). All sets have very high sensitivity, indicating that the model is very good at detecting class 1 instances.

Training Set: 0.9978 Validation Set: 0.9949 Test Set: 0.9966

Specificity (True Negative Rate):‘

The model's specificity is measured by how well it can locate the negative class, in this case class 0. The test and validation sets have lower specificity values than the training set, indicating that the model is less accurate at correctly identifying instances of class 0 in these datasets.

Training Set: 0.7672 Validation Set: 0.7000 Test Set: 0.6759

Positive Predictive Value (Precision):

When comparing all of the model's positive predictions, precision is the percentage of true positive predictions. All of the values are comparable, demonstrating a good balance between recall and precision.

Training Set: 0.9767 Validation Set: 0.9699 Test Set: 0.9621

In conclusion, while there are minor variations in performance between the training, validation, and test sets, the model generally performs well on all sets. From the training set to the validation and test sets, there is a discernible drop in specificity. This suggests that on unobserved data, the model may be more susceptible to false positives (predicting class 1 when it is actually class 0). It may be possible to increase specificity on the test set by further fine-tuning the model's parameters, such as by changing the classification threshold or (if applicable) by experimenting with various values of k. Additionally, if it is feasible, think about testing the model's performance using more representative or diverse data.

Questions - Answers

1. How would this customer be classified? This new customer would be classified as 0, does not take the personal loan
2. The best K is 3

3.The confusion matrix validations Accuracy : 0.964 Sensitivity : 0.9950
Specificity : 0.6927
4.The client is rated as 0, so he declines the loan.

5. Compare the confusion matrix of the test set with that of the training and validation sets. Comment on the differences and their reason

-The model tends to perform better on the training set compared to the validation and test sets. This is because the model is trained on the training data, and it may have overfit to the training data, leading to higher accuracy on that dataset.

Sensitivity (the ability to correctly predict the positive class, i.e., those who accept the personal loan) is consistently high across all scenarios, indicating that the model is good at identifying customers who accept the loan.

Differences:

Validation vs. Test Set: In both scenarios, the test set tends to have lower specificity compared to the validation set. This indicates that the model's ability to correctly predict those who do not accept the loan is slightly worse on the test data.

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("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

```
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

```
# 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))


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"
```

```
#Second 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 the sizes of the training and validation sets
print(paste("The size of the training set is:", nrow(training_set)))
```

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

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

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

Now, let us 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])
```

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?

```r
# 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 knn

```r
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?

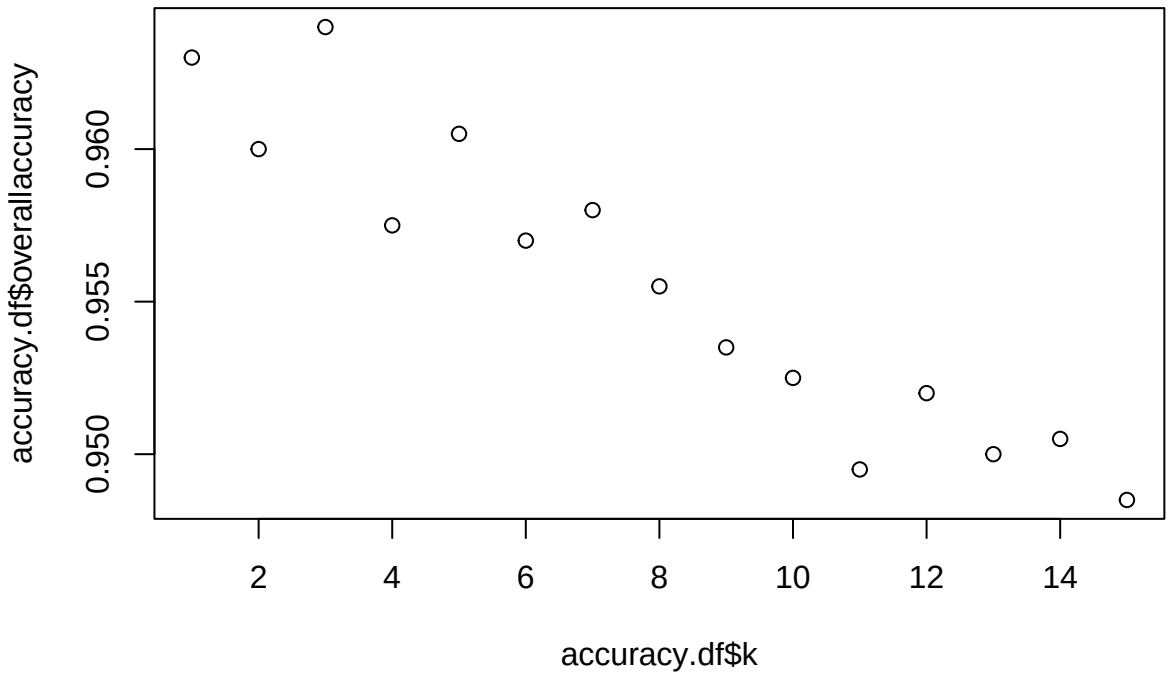
```r
# Calculate the accuracy for each value of k
# Set the range of k values to consider

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.

```
set.seed(1)
prediction_knn <- class::knn(train = train.norm.df, test = valid.norm.df,
                  cl = train.df[,10], k = 3, prob=TRUE)
confusionMatrix(prediction_knn, as.factor(valid.df[,10]))
```

```
## 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.

```
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)

knn.4 <- class::knn(train = train.norm.df,test = new.cust.norm, cl = train.df$Personal.Loan, k=3)
knn.4
```

```
## [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(1) # For reproducibility
train.index <- sample(rownames(universal_m.df), 0.5*dim(universal_m.df)[1])

set.seed(1)
valid.index <- sample(setdiff(rownames(universal_m.df),train.index), 0.3*dim(universal_m.df)[1])
test.index = setdiff(rownames(universal_m.df), union(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, ]

norm.values <- preProcess(train.df[, -c(10)], method=c("center", "scale"))
train.norm.df <- predict(norm.values, train.df[, -c(10)])
valid.norm.df <- predict(norm.values, valid.df[, -c(10)])
test.norm.df <- predict(norm.values, test.df[,-c(10)])


trainknn <- class::knn(train = train.norm.df,test = train.norm.df, cl = train.df[,10], k=3, prob=TRUE)

validknn <- class::knn(train = train.norm.df,test = valid.norm.df, cl = train.df[,10], k=3, prob=TRUE)

testknn <- class::knn(train = train.norm.df,test = test.norm.df, cl = train.df[,10], k=3, prob=TRUE)


#Training set confusion matrix
confusionMatrix(trainknn, as.factor(train.df$Personal.Loan))
```

```
## 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 : 0.9978
##             Specificity : 0.7672
##          Pos Pred Value : 0.9767
##          Neg Pred Value : 0.9727
##              Prevalence : 0.9072
##          Detection Rate : 0.9052
##    Detection Prevalence : 0.9268
##       Balanced Accuracy : 0.8825
##
##        'Positive' Class : 0
##
```

```
#validation set confusion matrix
confusionMatrix(validknn, as.factor(valid.df$Personal.Loan))
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    0    1
##          0 1353   42
##          1    7   98
##
##                Accuracy : 0.9673
##                  95% CI : (0.957, 0.9757)
##     No Information Rate : 0.9067
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7826
##
##  Mcnemar's Test P-Value : 1.191e-06
##
##             Sensitivity : 0.9949
##             Specificity : 0.7000
##          Pos Pred Value : 0.9699
##          Neg Pred Value : 0.9333
##              Prevalence : 0.9067
##          Detection Rate : 0.9020
##    Detection Prevalence : 0.9300
##       Balanced Accuracy : 0.8474
##
##        'Positive' Class : 0
##
```

```
#Test set confusion matrix
confusionMatrix(testknn, as.factor(test.df[,10]))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 889  35
##          1   3  73
##
##                Accuracy : 0.962
##                  95% CI : (0.9482, 0.973)
##     No Information Rate : 0.892
##     P-Value [Acc > NIR] : 4.592e-16
##
##                   Kappa : 0.7732
##
##  Mcnemar's Test P-Value : 4.934e-07
##
##             Sensitivity : 0.9966
##             Specificity : 0.6759
##          Pos Pred Value : 0.9621
##          Neg Pred Value : 0.9605
##              Prevalence : 0.8920
##          Detection Rate : 0.8890
##    Detection Prevalence : 0.9240
##       Balanced Accuracy : 0.8363
##
##        'Positive' Class : 0
```

```
##
```

##Compare the confusion matrix of the test set #with that of the training and validation sets. Comment on the differences and their reason #Common Observations:

The model tends to perform better on the training set compared to the validation and test sets. This is because the model is trained on the training data, and it may have overfit to the training data, leading to higher accuracy on that dataset.

Sensitivity (the ability to correctly predict the positive class, i.e., those who accept the personal loan) is consistently high across all scenarios, indicating that the model is good at identifying customers who accept the loan.

Differences:

Validation vs. Test Set: In both scenarios, the test set tends to have lower specificity compared to the validation set. This indicates that the model's ability to correctly predict those who do not accept the loan is slightly worse on the test data.