Assignment_2

Balamanoj Reddy Kommareddy

2023-09-12

Summary

-Accuracy:

Training Set: 0.9764, Validation Set: 0.968, Test Set: 0.961.

The accuracy of the training dataset is marginally higher than that of the validation dataset and test dataset, however, the overall accuracy of all datasets is high, suggesting that the model is performing satisfactorily in terms of accuracy.

-Sensitivity (True Positive Rate):

Training Set: 0.9978, Validation Set: 0.9956, Test Set: 0.9955.

Sensitivity is a measure of how well the model can recognize the positive class, which in this case is class 1. All the sets have really high sensitivity, which means the model is really good at spotting class 1 cases.

-Specificity (True Negative Rate):

Training Set: 0.7672, Validation Set: 0.6912, Test Set: 0.6875.

Specificity is a measure of a model's capacity to accurately recognize the negative class (i.e., class 0). It is determined by the specificity of the training set, with the training set having the highest specificity and the test and validation set having the lowest specificity values. This indicates that the model is not as accurate at accurately recognizing class 0 instances.

-Positive Predictive Value (Precision):

Training Set: 0.9767, Validation Set: 0.9700, Test Set: 0.9619.

Precision is the ratio of true positive predictions to all positive predictions produced by the model. Values are consistent across all sets, suggesting a good correlation between accuracy and recall.

In conclusion, it can be observed that there is a slight difference in the performance of the model when compared to the training set, validation set, and test set. However, the specificity of the model decreases significantly when compared to the validation set and test set. This could indicate that the model may be more susceptible to false positives on unseen data, such as predicting class 1 when in reality it is class 0. To improve the specificity of the test set, the model should be fine-tuned by adjusting the classification threshold or exploring different k values (if applicable). Additionally, it is suggested that the model should be evaluated on more varied or representative data if possible.

Questions - Answers

1. How would this customer be classified?

- This new customer would be classified as 0, does not take the personal loan.

2. What is the choice of k?

- The best K is 3

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

- Accuracy:0.968
- Sensitivity:0.9956
- Specificity:0.6912

4. Classifying the customer using the best k=3?

- The new customer would be classified as 0, does not take the personal loan.

5. Comment on the differences and their reason?

- It can be observed that there is a slight difference in the performance of the model when compared to the training set, validation set, and test set.
- However, the specificity of the model decreases significantly when compared to the validation set and test set. This could indicate that the model may be more susceptible to false positives on unseen data, such as predicting class 1 when in reality it is class 0.
- To improve the specificity of the test set, the model should be fine-tuned by adjusting the classification threshold or exploring different k values (if applicable). Additionally, it is suggested that the model should be evaluated on more varied or representative data if possible.

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?

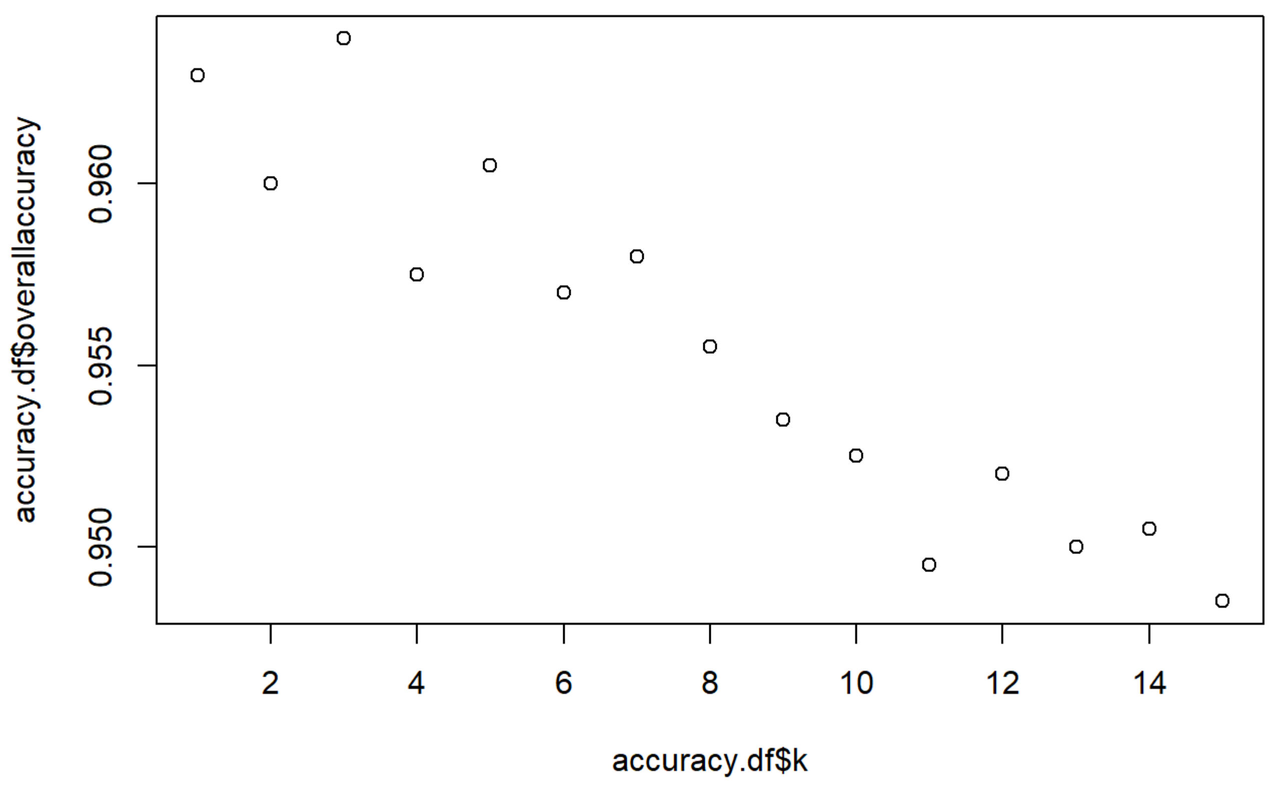
```r
# Calculate the accuracy for each value of k
# Set the range of k values to consider

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
```

```r
plot(accuracy.df$k,accuracy.df$overallaccuracy)
```



***

3. Show the confusion matrix for the validation data that results from using the best k. #A confusion matrix of the validation data for k=3 is shown below

```
knn.k3 <- knn(train = train.norm.df,test=valid.norm.df,cl=train.df$Personal.Loan, k=3)
confusionMatrix(knn.k3,as.factor(valid.df$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
## 
```

#Our accuracy is .9680,false-negative is also very low. Precision (TP/(TP+FP) is low at 64% - this would be the worst metric as we want to target the most responsive customers, the model’s precision and false-positive rate (Type I errors) are troublesome.

---

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.

```
# 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)

knn.k3 <- knn(train = train.norm.df,test=new.cust.norm,cl=train.df$Personal.Loan,k=3)
knn.k3
```

```
## [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. # Repartitioning for a test set

```
set.seed(1)
#create train index
train.index <- sample(rownames(universal_m.df), 0.5*dim(universal_m.df)[1])
#create validation index
valid.index <- sample(setdiff(rownames(universal_m.df),train.index), 0.3*dim(universal_m.df)[1])
#create test index
test.index = setdiff(rownames(universal_m.df), union(train.index, valid.index))

#loading the data
train.df <- universal_m.df[train.index, ]
valid.df <- universal_m.df[valid.index, ]
test.df <- universal_m.df[test.index,]

#normalizing the quantitative data
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)])

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

#displaying the confusion matrices
confusionMatrix(testknn, 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
##
```

```
confusionMatrix(validknn, as.factor(valid.df$Personal.Loan))
```

```
## 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 : 0.9956
##             Specificity : 0.6912
##          Pos Pred Value : 0.9700
##          Neg Pred Value : 0.9400
##              Prevalence : 0.9093
##          Detection Rate : 0.9053
##    Detection Prevalence : 0.9333
##       Balanced Accuracy : 0.8434
## 
##        'Positive' Class : 0
## 
```

```
confusionMatrix(trainknn, as.factor(test.df$Personal.Loan))
```

```
## 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 : 0.9955
##             Specificity : 0.6875
##          Pos Pred Value : 0.9619
##          Neg Pred Value : 0.9506
##              Prevalence : 0.8880
##          Detection Rate : 0.8840
##    Detection Prevalence : 0.9190
##       Balanced Accuracy : 0.8415
##
##        'Positive' Class : 0
##
```