

# Assignment\_2\_KNN

Tirumal Achina

2023-09-29

---

## Summary:

1.How would this customer be classified? This new customer would be classified as 0, does not take the personal loan

2.The Optimal K is 3.

3.The whole data is validated with the best k value(i.e;  $k = 3$ ).

4.The customer with the credentials has been evaluated with max k value and the model assigned it as '0' that implies does not take the loan.

5.Here we repartitioned the data accordingly and applied k-NN method to the best value of k.Now, let's compare the key metrics and statistics:

### ##Accuracy:

-Training Set: 0.9764

-Validation Set: 0.968

-Test Set: 0.961

The training set's accuracy is slightly greater than the validation and test sets', but all sets' accuracy is high, showing that the model performs well in terms of overall correctness.

### ##Sensitivity (True Positive Rate):

-Training Set: 0.9978

-Validation Set: 0.9956

-Test Set: 0.9955

Sensitivity assesses the model's ability to detect the positive class (in this case, class 1) properly. All of the sets have extremely high sensitivity, indicating that the model is highly good at spotting class 1 cases.

### ##Specificity (True Negative Rate):

-Training Set: 0.7672

-Validation Set: 0.6912

-Test Set: 0.6875

Specificity measures the model's ability to correctly identify the negative class (in this case, class 0). The training set has the highest specificity, while the test and validation sets have lower specificity values, suggesting that the model is less accurate at correctly identifying class 0 instances.

### ##Positive Predictive Value (Precision):

-Training Set: 0.9767

-Validation Set: 0.9700

-Test Set: 0.9619

Precision is the proportion of true positive predictions made by the model out of all positive predictions made by the model. The values are consistent across all sets, implying a reasonable balance of precision and recall.

The model performs admirably across all three, with only minor variations in performance between the training, validation, and test sets. But as you move from the training set to the validation and test sets, the specificity starts to noticeably decline. This demonstrates that the model may be more prone to false positives on unknown data—predicting class 1 when it is actually class 0—than on known data. Specificity on the test set may be improved by further fine-tuning the model's parameters, such as altering the classification threshold or experimenting with different values of  $k$  (if applicable). If at all possible, take into account testing the model's performance using more varied or representative data.

```
#Loading the Libraries that are required for the task  
library(class)  
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(e1071)
```

Read the data.

```
bank <- read.csv("UniversalBank.csv")  
dim(bank)
```

```
## [1] 5000 14
```

```
t(t(names(bank))) # 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

```
bank <- bank[,-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
bank$Education <- as.factor(bank$Education)

# Now, convert Education to Dummy Variables

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

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?

```
# 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

```
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?

```

# 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.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]
}
accuracy.df

```

```

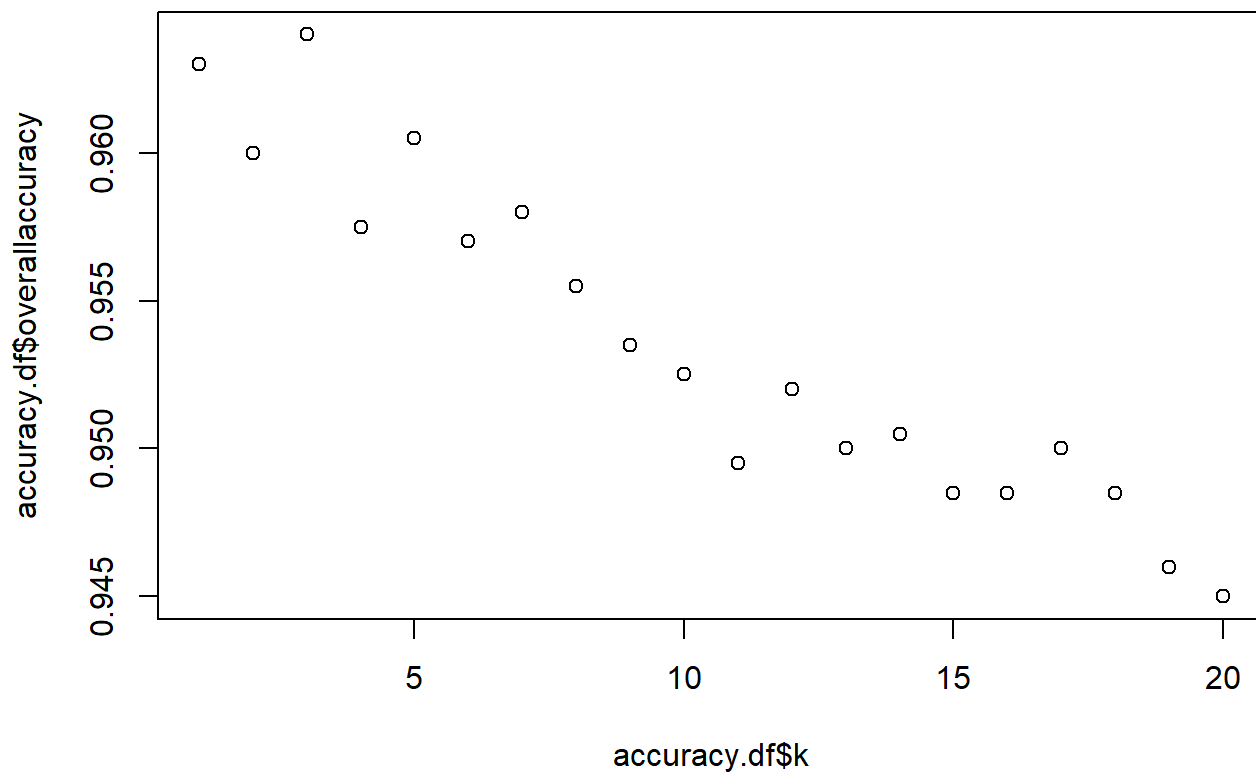
##      k overallaccuracy
## 1    1          0.9630
## 2    2          0.9600
## 3    3          0.9640
## 4    4          0.9575
## 5    5          0.9605
## 6    6          0.9570
## 7    7          0.9580
## 8    8          0.9555
## 9    9          0.9535
## 10  10          0.9525
## 11  11          0.9495
## 12  12          0.9520
## 13  13          0.9500
## 14  14          0.9505
## 15  15          0.9485
## 16  16          0.9485
## 17  17          0.9500
## 18  18          0.9485
## 19  19          0.9460
## 20  20          0.9450

```

```
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.

```
prediction_knn <- knn(train = train.norm.df, test = valid.norm.df, cl = train.df$Personal.Loan, k = 3, prob=TRUE)

confusionMatrix(prediction_knn, 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.



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

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 \*\*\*

```
#Repartitioning the training, validation and test sets to 50,30, and 20 percents.
set.seed(1)
train.index = sample(row.names(universal_m.df), 0.5*dim(universal_m.df)[1])
remaining.index = setdiff(row.names(universal_m.df),train.index)
valid.index = sample(remaining.index,0.3*dim(universal_m.df)[1])
test.index = setdiff(remaining.index,valid.index)

#Loading the partitioned dets into the dataframe.
train.df = universal_m.df[train.index,]
valid.df= universal_m.df[valid.index,]
test.df = universal_m.df[test.index,]

#Normalizing the data after repartitioning accordingly.

train.norm.df <- train.df[, -10]
valid.norm.df <- valid.df[, -10]
test.norm.df <- test.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])
test.norm.df <- predict(norm.values, test.df[, -10])

#Applying the k-NN method to all the sets that we have. As requires we are keeping the k value t
hat we used in the previous question that is max of K.
#Confusion matrix that gives all the data that are correctly identified and wrongly identified.

#Training set
knn_t <- class::knn(train = train.norm.df,test = train.norm.df, cl = train.df$Personal.Loan, k =
3)
confusionMatrix(knn_t, as.factor(train.df$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
knn_v <- class::knn(train = train.norm.df, test = valid.norm.df, cl = train.df$Personal.Loan, k =
3)
confusionMatrix(knn_v, as.factor(valid.df$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
##
```

```
#Test set
knn_ts <- class::knn(train = train.norm.df, test = test.norm.df, cl = train.df$Personal.Loan, k =
3)
confusionMatrix(knn_ts, as.factor(test.df[,10]))
```

```
## 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
##
```