Assignment 02

2023-10-01

```
library(readr)
data <- read_csv("UniversalBank.csv", show_col_types = FALSE)
```

Task 1

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. 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?

```
# Load necessary libraries
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
library(class)

# Read data from CSV file
data <- read_csv("UniversalBank.csv", show_col_types = FALSE)

# Data Preprocessing

# Create dummy variables for Education
data$Education_1 <- as.integer(data$Education == 1)
```

```r
data$Education_2 <- as.integer(data$Education == 2)
data$Education_3 <- as.integer(data$Education == 3)

# Remove unnecessary columns
data <- select(data, -c('ID', 'ZIP Code', 'Education'))

# Data Splitting

# Split the data into training (60%) and validation (40%) sets
index <- sample(1:nrow(data), nrow(data) * 0.6)
train_data <- data[index, ]
valid_data <- data[-index, ]

# New Customer Data
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
)

# Model Building and Prediction

# Perform k-NN classification with k = 1
knn_pred <- knn(train_data[, -7], valid_data[, -7], train_data$`Personal Loan` , k = 1)

# Display the classification result for the new customer
print("Classification Result for the New Customer:")
```

```
## [1] "Classification Result for the New Customer:"
```

```r
print(knn_pred)
```

```
##    [1] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##   [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
##   [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [112] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1
##  [149] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [186] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0
##  [223] 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0
##  [260] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [297] 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0
##  [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
```

```
##  [371] 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
##  [408] 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [445] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
##  [482] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [519] 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
##  [556] 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
##  [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1
##  [630] 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [667] 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [704] 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [741] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0
##  [778] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
##  [815] 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0
##  [852] 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [889] 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [926] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [963] 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
## [1000] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
## [1037] 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1074] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
## [1111] 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0
## [1148] 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1185] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
## [1222] 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
## [1259] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1
## [1296] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0
## [1333] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
## [1370] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
## [1407] 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 1
## [1444] 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
## [1481] 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1518] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0
## [1555] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0
## [1592] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1
## [1629] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
## [1666] 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1703] 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1740] 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1777] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
## [1814] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
## [1851] 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1888] 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1925] 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1962] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1999] 0 0
## Levels: 0 1
```

Task 2

```
# Use cross-validation to find the optimal k
k_values <- seq(1, 100, by = 2)  # or another range of your choice
error_rates <- sapply(k_values, function(k) {
```

```
  knn_pred <- knn(train_data[, -7], valid_data[, -7], cl=train_data$"Personal Loan", k=k)
  mean(knn_pred != valid_data$"Personal Loan")
})

# Find the k with the lowest error rate
optimal_k <- k_values[which.min(error_rates)]
optimal_k
```

```
## [1] 5
```

Task 3

c.Show the confusion matrix for the validation data that results from using the best k.

```
# Perform k-NN classification with the optimal k
knn_pred_optimal <- knn(train_data[, -7], valid_data[, -7], cl=train_data$"Personal Loan", k=optimal_k)

# Create a confusion matrix
confusion_matrix <- table(Predicted = knn_pred_optimal, Actual = valid_data$"Personal Loan")

# Show the confusion matrix
print(confusion_matrix)
```

```
##           Actual
## Predicted    0    1
##         0 1756  104
##         1   67   73
```

Task 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
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
)
```

```r
# Perform k-NN classification for the new customer using the optimal k
knn_pred_new_customer <- knn(train_data[, -7], new_customer, cl=train_data$"Personal Loan", k=optimal_k

# Show the classification for the new customer
print(knn_pred_new_customer)
```

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

Task 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.

```r
# Set the seed for reproducibility
set.seed(123)

# Repartition the data into training (50%), validation (30%), and test (20%) sets
index_train <- sample(1:nrow(data), nrow(data) * 0.5)
index_valid <- sample(setdiff(1:nrow(data), index_train), nrow(data) * 0.3)
index_test <- setdiff(1:nrow(data), c(index_train, index_valid))

train_data <- data[index_train, ]
valid_data <- data[index_valid, ]
test_data <- data[index_test, ]

# Perform k-NN classification with the optimal k on training set
knn_pred_train <- knn(train_data[, -7], train_data[, -7], cl=train_data$"Personal Loan", k=optimal_k)

# Perform k-NN classification with the optimal k on validation set
knn_pred_valid <- knn(train_data[, -7], valid_data[, -7], cl=train_data$"Personal Loan", k=optimal_k)

# Perform k-NN classification with the optimal k on test set
knn_pred_test <- knn(train_data[, -7], test_data[, -7], cl=train_data$"Personal Loan", k=optimal_k)

# Create confusion matrices
confusion_matrix_train <- table(Predicted = knn_pred_train, Actual = train_data$"Personal Loan")
confusion_matrix_valid <- table(Predicted = knn_pred_valid, Actual = valid_data$"Personal Loan")
confusion_matrix_test <- table(Predicted = knn_pred_test, Actual = test_data$"Personal Loan")

# Show the confusion matrices
print("Confusion Matrix - Training Set:")
```

```
## [1] "Confusion Matrix - Training Set:"
```

```r
print(confusion_matrix_train)
```

```
##          Actual
## Predicted    0    1
##         0 2234  114
##         1   37  115
```

```r
print("Confusion Matrix - Validation Set:")
```

```
## [1] "Confusion Matrix - Validation Set:"
```

```r
print(confusion_matrix_valid)
```

```
##           Actual
## Predicted    0    1
##         0 1305   99
##         1   52   44
```

```r
print("Confusion Matrix - Test Set:")
```

```
## [1] "Confusion Matrix - Test Set:"
```

```r
print(confusion_matrix_test)
```

```
##           Actual
## Predicted   0   1
##         0 865  76
##         1  27  32
```

Reason The model performs well on the training set but struggles to generalize on the test set, resulting in more false positives and false negatives in the test set's confusion matrix compared to the validation set. This indicates potential overfitting to the training data.