Assignment\_2

2025-06-09

# Step 1: Load libraries  
if (!require("dplyr")) install.packages("dplyr", dependencies = TRUE)

## Loading required package: dplyr

## Warning: package 'dplyr' was built under R version 4.4.3

##   
## 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

if (!require("caret")) install.packages("caret", dependencies = TRUE)

## Loading required package: caret

## Warning: package 'caret' was built under R version 4.4.3

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.4.3

## Loading required package: lattice

## Warning: package 'lattice' was built under R version 4.4.3

if (!require("class")) install.packages("class", dependencies = TRUE)

## Loading required package: class

## Warning: package 'class' was built under R version 4.4.3

if (!require("readr")) install.packages("readr", dependencies = TRUE)

## Loading required package: readr

library(dplyr)  
library(caret)  
library(class)  
library(readr)  
  
# Step 2: Load and clean data  
# Load raw data and remove ID and ZIP Code since they are not predictive features  
bank\_raw <- read\_csv("C:/Users/arkha/Downloads/UniversalBank.csv")

## Rows: 5000 Columns: 14

## ── Column specification ────────────────────────────────────────────────────────  
## Delimiter: ","  
## dbl (14): ID, Age, Experience, Income, ZIP Code, Family, CCAvg, Education, M...  
##   
## ℹ Use `spec()` to retrieve the full column specification for this data.  
## ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

bank\_cleaned <- bank\_raw %>% select(-ID, -`ZIP Code`)  
  
# Step 3: Dummy encoding  
# Convert the 'Education' categorical variable with 3 levels into dummy variables  
bank\_cleaned <- cbind(bank\_cleaned, model.matrix(~ factor(Education) - 1, bank\_cleaned))  
bank\_cleaned <- bank\_cleaned %>% select(-Education)  
  
# Step 4: Normalize numeric columns  
# Min-max scale numerical variables to bring them into the range [0,1], helping distance-based models like k-NN  
minmax\_scale <- function(x) (x - min(x)) / (max(x) - min(x))  
bank\_scaled <- bank\_cleaned %>% mutate(  
 Age = minmax\_scale(Age),  
 Experience = minmax\_scale(Experience),  
 Income = minmax\_scale(Income),  
 CCAvg = minmax\_scale(CCAvg),  
 Mortgage = minmax\_scale(Mortgage)  
)  
  
# Step 5: Partition (60% train, 40% validation split)  
# Split the data into training and validation sets, maintaining class proportions  
set.seed(42)  
split\_index <- createDataPartition(bank\_scaled$`Personal Loan`, p = 0.6, list = FALSE)  
training\_data <- bank\_scaled[split\_index, ]  
validation\_data <- bank\_scaled[-split\_index, ]  
X\_train\_set <- training\_data %>% select(-`Personal Loan`)  
Y\_train\_set <- training\_data$`Personal Loan`  
X\_valid\_set <- validation\_data %>% select(-`Personal Loan`)  
Y\_valid\_set <- validation\_data$`Personal Loan`  
  
# Step 6: Normalize new customer  
# Normalize new applicant's data using training data's min/max to ensure consistent scale  
applicant\_info <- data.frame(  
 Age = 40, Experience = 10, Income = 84, Family = 2,  
 CCAvg = 2, Mortgage = 0, `Securities Account` = 0,  
 `CD Account` = 0, Online = 1, CreditCard = 1,  
 `factor(Education)1` = 0, `factor(Education)2` = 1, `factor(Education)3` = 0  
)  
scale\_relative <- function(x, ref) (x - min(ref)) / (max(ref) - min(ref))  
applicant\_info$Age <- scale\_relative(applicant\_info$Age, training\_data$Age)  
applicant\_info$Experience <- scale\_relative(applicant\_info$Experience, training\_data$Experience)  
applicant\_info$Income <- scale\_relative(applicant\_info$Income, training\_data$Income)  
applicant\_info$CCAvg <- scale\_relative(applicant\_info$CCAvg, training\_data$CCAvg)  
applicant\_info$Mortgage <- scale\_relative(applicant\_info$Mortgage, training\_data$Mortgage)  
  
# Step 7: Predict with k = 1  
# Perform k-NN classification for the applicant using k = 1 (most similar neighbor)  
knn\_pred\_k1 <- knn(train = X\_train\_set, test = applicant\_info, cl = Y\_train\_set, k = 1)  
cat("Prediction with k = 1:", knn\_pred\_k1, "\n")

## Prediction with k = 1: 2

# Answer to Q1: The applicant is classified as 2 (loan \*\*not\*\* accepted)  
  
# Step 8: Best k using validation  
# Loop through k = 1 to 20 to determine the best-performing value of k on validation data  
k\_scores <- data.frame(k = 1:20, accuracy = NA)  
for (k\_val in 1:20) {  
 pred\_k <- knn(X\_train\_set, X\_valid\_set, cl = Y\_train\_set, k = k\_val)  
 k\_scores$accuracy[k\_val] <- mean(pred\_k == Y\_valid\_set)  
}  
optimal\_k\_val <- k\_scores$k[which.max(k\_scores$accuracy)]  
cat("Best k found:", optimal\_k\_val, "\n")

## Best k found: 1

# Answer to Q2: The best k balancing overfitting and generalization was found to be 1  
  
# Step 9: Confusion matrix for best k  
# Create a confusion matrix using the optimal k value from the previous step on validation data  
validation\_predictions <- knn(X\_train\_set, X\_valid\_set, cl = Y\_train\_set, k = optimal\_k\_val)  
print(table(Predicted = validation\_predictions, Actual = Y\_valid\_set))

## Actual  
## Predicted 0 1  
## 0 1778 71  
## 1 30 121

# Answer to Q3: Confusion matrix (k = 1)  
# Actual  
# Predicted 0 1  
# 0 1778 71  
# 1 30 121  
  
# Step 10: Predict customer again  
# Classify the same applicant using the optimal k value  
final\_applicant\_pred <- knn(X\_train\_set, applicant\_info, cl = Y\_train\_set, k = optimal\_k\_val)  
cat("Prediction with best k:", final\_applicant\_pred, "\n")

## Prediction with best k: 2

# Answer to Q4: The applicant is classified as 2 (loan \*\*not\*\* accepted)  
  
# Step 11: Repartition (50% train, 30% validation, 20% test)  
# Perform a 3-way split for more comprehensive model evaluation: 50% train, 30% validation, 20% test  
set.seed(123)  
reshuffled\_data <- bank\_scaled[sample(nrow(bank\_scaled)), ]  
total\_rows <- nrow(reshuffled\_data)  
train\_part <- reshuffled\_data[1:floor(0.5 \* total\_rows), ]  
valid\_part <- reshuffled\_data[(floor(0.5 \* total\_rows) + 1):floor(0.8 \* total\_rows), ]  
test\_part <- reshuffled\_data[(floor(0.8 \* total\_rows) + 1):total\_rows, ]  
  
X\_part\_train <- train\_part %>% select(-`Personal Loan`)  
Y\_part\_train <- train\_part$`Personal Loan`  
X\_part\_valid <- valid\_part %>% select(-`Personal Loan`)  
Y\_part\_valid <- valid\_part$`Personal Loan`  
X\_part\_test <- test\_part %>% select(-`Personal Loan`)  
Y\_part\_test <- test\_part$`Personal Loan`  
  
# Step 12: Confusion matrices  
# Evaluate model performance across train, validation, and test splits using best k  
train\_predicted <- knn(X\_part\_train, X\_part\_train, cl = Y\_part\_train, k = optimal\_k\_val)  
valid\_predicted <- knn(X\_part\_train, X\_part\_valid, cl = Y\_part\_train, k = optimal\_k\_val)  
test\_predicted <- knn(X\_part\_train, X\_part\_test, cl = Y\_part\_train, k = optimal\_k\_val)  
  
cat("Training Confusion Matrix:\n")

## Training Confusion Matrix:

print(table(Predicted = train\_predicted, Actual = Y\_part\_train))

## Actual  
## Predicted 0 1  
## 0 2271 0  
## 1 0 229

# Training matrix indicates near-perfect fit, suggesting overfitting at k=1  
  
cat("Validation Confusion Matrix:\n")

## Validation Confusion Matrix:

print(table(Predicted = valid\_predicted, Actual = Y\_part\_valid))

## Actual  
## Predicted 0 1  
## 0 1329 48  
## 1 19 104

# Indicates generalization capability to unseen data (Validation)  
  
cat("Test Confusion Matrix:\n")

## Test Confusion Matrix:

print(table(Predicted = test\_predicted, Actual = Y\_part\_test))

## Actual  
## Predicted 0 1  
## 0 894 40  
## 1 7 59