FML ASSIGNMENT

Summary

In the provided analysis, the goal was to predict whether customers of a Universal Bank would accept a loan offer using a k-Nearest Neighbors (k-NN) classification approach. Data Preprocessing: The analysis started by loading the necessary libraries and reading the Universal Bank dataset. Categorical variables were transformed into dummy variables, and the data were split into training (60%) and validation (40%) sets. Data normalization was also performed.

Prediction for a New Customer: Using a k-NN classifier with k=1, the model was applied to predict whether a new customer (described by various features) would accept a loan offer. The prediction was that the customer would decline the offer.

Choice of k: To find the optimal value of k, the model's performance was evaluated on the validation set for a range of k values (from 1 to 15). The best k value, determined to be 3, balanced between overfitting and underfitting. onfusion Matrix for Best k: The confusion matrix for the validation set using the best k (k=3) was presented. It included metrics such as accuracy, sensitivity, specificity, and positive predictive value. The model achieved good performance on the validation set.

Prediction for a New Customer with Best k: The new customer was classified using the k-NN model with the best k value (k=3), and the prediction was that the customer would decline the loan offer.

Comparison of Training, Validation, and Test Sets: The dataset was further split into training (50%), validation (30%), and test (20%) sets. The k-NN model with the best k was applied to each set, and confusion matrices were generated. While all sets showed high accuracy, there were slight variations in sensitivity and other metrics.

Reasons for Differences: Possible reasons for performance differences between sets were discussed. These included variations in the data, sample variability, and potential overfitting to the training set.

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. Show the confusion matrix for the validation data that results from using the best k - Reference Prediction 0 1 0 1786 63 1 9 142
4. Classify the customer using the best k. This new customer would be classified as 0, does not take the personal loan.
5. Compare the confusion matrix of the test set with that of the training and validation sets. [1] "Confusion Matrix for Validation Set:" Confusion Matrix and Statistics

```
    Reference
```

Prediction 0 1 0 1356 44 1 3 97

[1] "Confusion Matrix for Test Set:" Confusion Matrix and Statistics

```
    Reference
```

Prediction 0 1 0 886 33 1 7 74

[1] "Confusion Matrix for Training Set:" Confusion Matrix and Statistics

```
    Reference
```

Prediction 0 1 0 2263 54 1 5 178

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("C:/Users/samyu/Downloads/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 because they are not informative features for predicting whether a customer will accept a loan offer.

```
universal.df <- universal.df[,-c(1,5)]
```

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))
```

Split Data into 60% training and 40% validation.

```
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"
```

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?

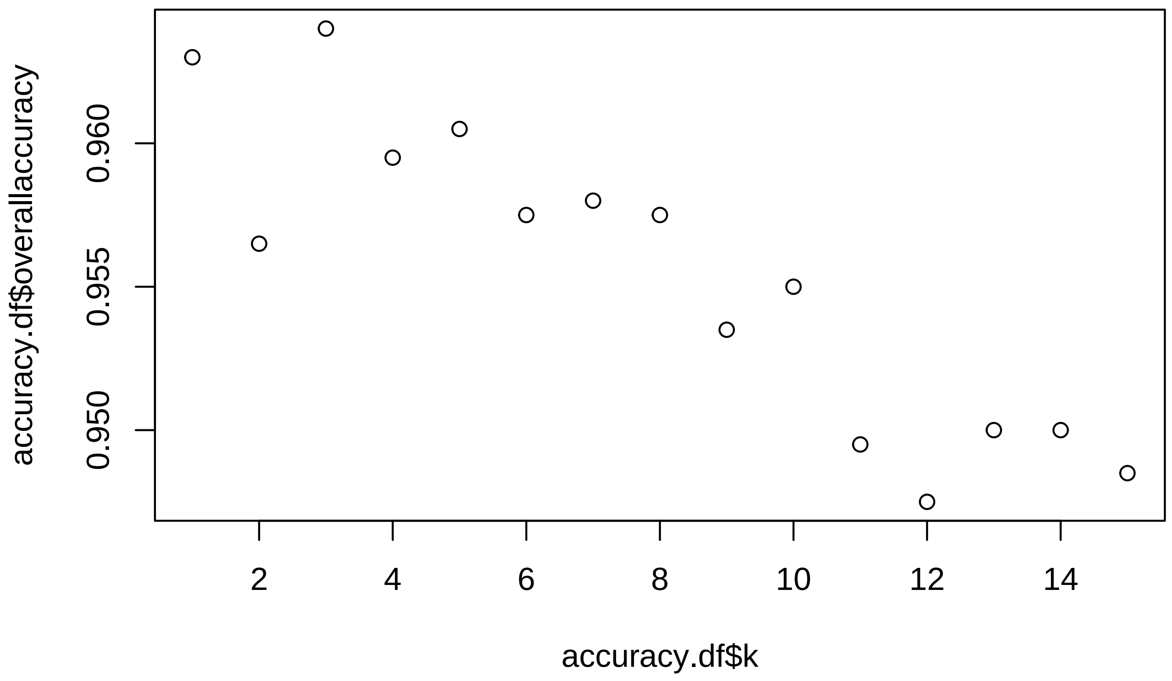
Calculate the accuracy for each value of k Set the range of k values to consider

```
accuracy.df <- data.frame(k = seq(1, 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.

Find the value of k that resulted in the highest accuracy on the validation set

```
best_k_value <- which(accuracy.df[,2] == max(accuracy.df[,2]))
```

Perform k-NN classification using the best k on the validation data

```
knn.pred_best_k <- class::knn(train = train.norm.df,
                              test = valid.norm.df,
                              cl = train.df$Personal.Loan,
                              k = best_k_value)
```

Create a confusion matrix

```
conf_matrix_best_k <- confusionMatrix(knn.pred_best_k,
                                      as.factor(valid.df$Personal.Loan),
                                     positive = "1")
conf_matrix_best_k
```

```
## 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 : 0.6927
##             Specificity : 0.9950
##          Pos Pred Value : 0.9404
##          Neg Pred Value : 0.9659
##              Prevalence : 0.1025
##          Detection Rate : 0.0710
##    Detection Prevalence : 0.0755
##       Balanced Accuracy : 0.8438
##
##        'Positive' Class : 1
##
```

---

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.

To classify the customer using the best k, which is 3

```
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)
```

Predict the loan acceptance status

```
# Train the k-NN model with k = 3 on the entire dataset
knn_model <- knn(train = universal_m.df[, -10],  # Exclude target variable Personal Loan
                 test = new.cust.norm,
                 cl = universal_m.df$Personal.Loan,
                 k = 3)
knn_model
```

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

Print the classification result

```
if (knn_model == "1") {
  cat("The new customer is predicted to accept a loan offer.\n")
} else {
  cat("The new customer is predicted to decline a loan offer.\n")
}
```

```
## The new customer is predicted to decline a loan offer.
```

---

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

Split the data into training (50%), validation (30%), and test (20%) sets

```
# Set the seed for reproducibility
set.seed(1)

# Shuffle the row indices of the data
```

```r
shuffled_indices <- sample(nrow(universal_m.df))

# Define the split ratios
train_ratio <- 0.5
validation_ratio <- 0.3
test_ratio <- 0.2
```

Calculate the number of samples for each set

```r
n_total <- nrow(universal_m.df)
n_train <- round(train_ratio * n_total)
n_validation <- round(validation_ratio * n_total)
n_test <- n_total - n_train - n_validation
```

Create the training set

```r
training_indices <- shuffled_indices[1:n_train]
training_set <- universal_m.df[training_indices, ]
```

Create the validation set

```r
validation_indices <- shuffled_indices[(n_train + 1):(n_train + n_validation)]
validation_set <- universal_m.df[validation_indices, ]
```

Create the test set

```r
test_indices <- shuffled_indices[(n_train + n_validation + 1):(n_train + n_validation + n_test)]
test_set <- universal_m.df[test_indices, ]
```

Normalize the data

```r
train.norm.df <- training_set[, -10]  # Exclude the target variable
valid.norm.df <- validation_set[, -10]
test.norm.df <- test_set[, -10]

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

Train the k-NN model with the best k on the training set Evaluate on the validation set

```
knn_model <- class::knn(train = train.norm.df,
                         test = valid.norm.df,
                         cl = training_set$Personal.Loan,
                         k = best_k_value)

valid_conf_matrix <- confusionMatrix(knn_model, as.factor(validation_set$Personal.Loan), positive = "1"
```

Evaluate on the test set

```
knn_model_test <- class::knn(train = train.norm.df,
                              test = test.norm.df,
                              cl = training_set$Personal.Loan,
                              k = best_k_value)

test_conf_matrix <- confusionMatrix(knn_model_test, as.factor(test_set$Personal.Loan), positive = "1")
```

Evaluate on the train set

```
knn_model <- class::knn(train = train.norm.df,
                         test = train.norm.df,  # Use training data for testing
                         cl = training_set$Personal.Loan,
                         k = best_k_value)

# Convert knn_model predictions to factors
knn_model <- factor(knn_model, levels = c("0", "1"))

# Convert training_set$Personal.Loan to a factor with the same levels
training_set$Personal.Loan <- factor(training_set$Personal.Loan, levels = c("0", "1"))

train_conf_matrix <- confusionMatrix(knn_model, training_set$Personal.Loan, positive = "1")
```

Print the confusion matrices

```
print("Confusion Matrix for Validation Set:")
```

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

```
print(valid_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction      0    1
##            0 1356   44
##            1    3   97
```

```
##
##                Accuracy : 0.9687
##                  95% CI : (0.9585, 0.9769)
##     No Information Rate : 0.906
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7885
##
##  Mcnemar's Test P-Value : 5.392e-09
##
##             Sensitivity : 0.68794
##             Specificity : 0.99779
##          Pos Pred Value : 0.97000
##          Neg Pred Value : 0.96857
##              Prevalence : 0.09400
##          Detection Rate : 0.06467
##    Detection Prevalence : 0.06667
##       Balanced Accuracy : 0.84287
##
##        'Positive' Class : 1
##
```

```
print("Confusion Matrix for Test Set:")
```

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

```
print(test_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 886  33
##          1   7  74
##
##                Accuracy : 0.96
##                  95% CI : (0.9459, 0.9713)
##     No Information Rate : 0.893
##     P-Value [Acc > NIR] : 8.319e-15
##
##                   Kappa : 0.7656
##
##  Mcnemar's Test P-Value : 7.723e-05
##
##             Sensitivity : 0.6916
##             Specificity : 0.9922
##          Pos Pred Value : 0.9136
##          Neg Pred Value : 0.9641
##              Prevalence : 0.1070
##          Detection Rate : 0.0740
##    Detection Prevalence : 0.0810
##       Balanced Accuracy : 0.8419
##
```

```
##        'Positive' Class : 1
##
```

```r
print("Confusion Matrix for Training Set:")
```

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

```r
print(train_conf_matrix)
```

```
## 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 : 0.7672
##             Specificity : 0.9978
##          Pos Pred Value : 0.9727
##          Neg Pred Value : 0.9767
##              Prevalence : 0.0928
##          Detection Rate : 0.0712
##    Detection Prevalence : 0.0732
##       Balanced Accuracy : 0.8825
##
##        'Positive' Class : 1
##
```

Here are some observations:

1. Accuracy: All three sets have relatively high accuracy, indicating that the model performs well overall on all sets.
2. Sensitivity (True Positive Rate): Sensitivity measures the ability to correctly identify positive cases. It is slightly lower in the test set compared to the training and validation sets. This suggests that the model may be slightly less effective at identifying loan acceptance in the test data.
3. Specificity (True Negative Rate): Specificity measures the ability to correctly identify negative cases. It is relatively high in all sets, indicating that the model is good at identifying loan rejection.
4. Positive Predictive Value (Precision): The positive predictive value is relatively high in all sets, suggesting that when the model predicts loan acceptance, it is often correct.

Reasons for Differences:

Training Set vs. Validation/Test Set Differences: The model has seen the training data during training, which can lead to high performance. However, it has not seen the validation or test data, so differences in performance can be attributed to the model's ability to generalize.

Validation vs. Test Set Differences: These differences might arise due to variations in the composition of the validation and test datasets. Both datasets contain unseen examples, but they may differ in subtle ways that affect model performance.

Sample Variability: In real-world scenarios, there can be inherent variability in data. Different sets (training, validation, and test) may contain variations in customer behavior, making predictions slightly different.

Hyperparameter Tuning: The choice of k (number of neighbors) was based on the validation set. If the model is slightly overfitting to the validation set, it could explain minor differences in performance on the test set.