Homework 4

Group 7 - Suman Ravichandran & Harsh Shahdev

11/21/2020

```
# Import the required packages
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(readxl)
library(e1071)
library(FNN)
library(class)
```

```
##
## Attaching package: 'class'
```

```
## The following objects are masked from 'package:FNN':
##
##     knn, knn.cv
```

```
library(modelr)
library(caret)
```

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

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

```
uBank <- read_xlsx("UniversalBank.xlsx", sheet = "Data")

df_test <- data.frame(
  Education1 = 0, Education2 = 1, Education3 = 0, Age = 40, Experience = 10, Income = 84, Family = 2,
  CCAvg = 2, Mortgage = 0, Personal.Loan = "", `Securities Account` = 0,
  `CD Account` = 0, Online = 1, Creditcard = 1
)
```

```
<!-- Creating Dummy variables for Education type -->
```

```
attach(uBank)

X <- uBank[, 8]


X$Education <- paste0("Education", X$Education)
mm <- model.matrix(~ . + 0, data = X)
colnames(mm) <- c("Education1", "Education2", "Education3")

uBank <- cbind(mm, uBank[, -8])

detach(uBank)
```

```
<!-- Normalizing -->
```

```
train_index <- sample(rownames(uBank), 0.6 * dim(uBank)[1])
valid_index <- setdiff(rownames(uBank), train_index)

train_df <- uBank[train_index, -c(4, 8)]
valid_df <- uBank[valid_index, -c(4, 8)]

train_norm_df <- train_df
valid_norm_df <- valid_df
universal_norm_df <- uBank[, -c(4, 8)]

norm_values <- preProcess(train_df[, -10], method = c("center", "scale"))
train_norm_df <- predict(norm_values, train_df)
valid_norm_df <- predict(norm_values, valid_df)
universal_norm_df <- predict(norm_values, uBank[, -c(4, 8)])

colnames(df_test) <- colnames(uBank[, -c(4, 8)])

test_norm_df <- predict(norm_values, df_test)
```

```
<!-- Performing Knn -->
```

```
KN <- knn(train = train_norm_df[, -10], test = test_norm_df[, -10], cl = train_norm_df[, 10], k = 1)
row.names(train_df)[attr(KN, "KN.index")]
```

```
## character(0)
```

```
KN
```

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

```
accuracy_df <- data.frame(k = seq(1, 20), accuracy = rep(0, 20))

valid_norm_df[, 10] <- as.factor(valid_norm_df[, 10])


for (i in 1:20) {
  knn_pred <- knn(train_norm_df[, -10], valid_norm_df[, -10], cl = train_norm_df[, 10], k = i)
  accuracy_df[i, 2] <- confusionMatrix(knn_pred, valid_norm_df[, 10])$overall[1]
}

max(accuracy_df[, 2])
```

```
## [1] 0.9625
```

```
## (c)
```

```
KN1 <- knn(train_norm_df[, -10], valid_norm_df[, -10], cl = train_norm_df[, 10], k = 3)

valid_predict <- confusionMatrix(KN1, valid_norm_df[, 10])

valid_predict
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1791   67
##          1    7  135
##
##                Accuracy : 0.963
##                  95% CI : (0.9538, 0.9708)
##     No Information Rate : 0.899
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7653
##
##  Mcnemar's Test P-Value : 6.953e-12
##
##             Sensitivity : 0.9961
##             Specificity : 0.6683
##          Pos Pred Value : 0.9639
##          Neg Pred Value : 0.9507
##              Prevalence : 0.8990
##          Detection Rate : 0.8955
##    Detection Prevalence : 0.9290
##       Balanced Accuracy : 0.8322
##
##        'Positive' Class : 0
##
```

```
KN2 <- knn(train = train_norm_df[, -10], test = test_norm_df[, -10], cl = train_norm_df[, 10], k = 3)
row.names(train_df)[attr(KN, "KN.index")]
```

```
## character(0)
```

```
KN2
```

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

```
train_index1 <- sample(rownames(uBank), 0.5 * dim(uBank)[1])
index1 <- setdiff(rownames(uBank), train_index1)
valid_index1 <- sample(rownames(uBank[index1, ]), 0.6 * dim(uBank[index1, ])[1])
test_index1 <- setdiff(rownames(uBank[index1, ]), valid_index1)

train_df1 <- uBank[train_index1, -c(4, 8)]
valid_df1 <- uBank[valid_index1, -c(4, 8)]
test_df1 <- uBank[test_index1, -c(4, 8)]
```

```
<!-- normalizing the data  -->
```

```
train_norm_df1 <- train_df1
valid_norm_df1 <- valid_df1
test_norm_df1 <-  test_df1

train_norm_df1 <- predict(norm_values, train_df1)
valid_norm_df1 <- predict(norm_values, valid_df1)
test_norm_df1 <-  predict(norm_values, test_df1)

test_norm_df1[, 10] <- as.factor(test_norm_df1[, 10])
```

```
<!-- performing Knn for test set with training and validation sets -->
```

```
KN3 <- knn(train_norm_df1[, -10], test_norm_df1[, -10], cl = train_norm_df1[, 10], k = 3)

KN4 <- knn(valid_norm_df1[, -10], test_norm_df1[, -10], cl = valid_norm_df1[, 10], k = 3)


test_predict1 <- confusionMatrix(KN3, test_norm_df1[, 10])

test_predict2 <- confusionMatrix(KN4, test_norm_df1[, 10])

test_predict1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 902  43
##          1   4  51
##
##                Accuracy : 0.953
##                  95% CI : (0.938, 0.9653)
##     No Information Rate : 0.906
##     P-Value [Acc > NIR] : 1.832e-08
##
##                   Kappa : 0.661
##
##  Mcnemar's Test P-Value : 2.976e-08
##
##             Sensitivity : 0.9956
##             Specificity : 0.5426
##          Pos Pred Value : 0.9545
##          Neg Pred Value : 0.9273
##              Prevalence : 0.9060
##          Detection Rate : 0.9020
##    Detection Prevalence : 0.9450
##       Balanced Accuracy : 0.7691
##
##        'Positive' Class : 0
##
```

```
test_predict2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 898  53
##          1   8  41
##
##                Accuracy : 0.939
##                  95% CI : (0.9223, 0.953)
##     No Information Rate : 0.906
##     P-Value [Acc > NIR] : 9.902e-05
##
##                   Kappa : 0.5441
##
##  Mcnemar's Test P-Value : 1.765e-08
##
##             Sensitivity : 0.9912
##             Specificity : 0.4362
##          Pos Pred Value : 0.9443
##          Neg Pred Value : 0.8367
##              Prevalence : 0.9060
##          Detection Rate : 0.8980
##    Detection Prevalence : 0.9510
##       Balanced Accuracy : 0.7137
##
##        'Positive' Class : 0
##
```

```
rm(list = ls())
```

```
<!-- # Problem 2: Predicting Housing Median Prices k-NN -->
```

```
# Import the BostonHousing.xlsx file
Boston_Housing <- read_xlsx("BostonHousing.xlsx", sheet = "Data")

# Define the normalize function
normalize <- function(x) {
  return((x - min(x))) / (max(x) - min(x))
}

# Normalize the dataframe
df_norm <- as.data.frame(lapply(Boston_Housing[1:13], normalize))

# Generate the training data indices
indices <- sample(seq_len(nrow(df_norm)), size = floor(0.6 * nrow(df_norm)))

# Get training and validation data
train_data <- df_norm[indices, ]
validation_data <- df_norm[-indices, ]

# Create a dataframe to keep track of k vs error
error_df <- data_frame("k" = 1:5, "error" = rep(0, 5))

# Loop for K = 1 to 5
for (i in 1:5) {
  model <- knnreg(x = train_data[, 1:12], y = train_data[, 13], k = i)
  error_df[i, 2] <- RMSE(validation_data[, 13], predict(model, validation_data[, 1:12]))
}

# Get the K-value with the lowest RMSE error
best_k <- filter(error_df, error == min(error_df$error))$k
cat("The model with the best K is:", best_k, "\n")
```

```
## The model with the best K is: 4
```

```
<!-- ## (b) Predict the MEDV for a tract with the following information, using the best k: -->
```

```
# Let's get the model with the best K
model <- knnreg(x = train_data[, 1:12], y = train_data[, 13], k = best_k)

# Create a dataframe for the given record
df <- data.frame(
  "CRIM" = 0.2, "ZN" = 0, "INDUS" = 7,
  "CHAS" = 0, "NOX" = 0.538, "RM" = 6,
  "AGE" = 62, "DIS" = 4.7, "RAD" = 4,
  "TAX" = 307, "PTRATIO" = 21, "LSTAT" = 10
)

# Normalize the dataframe
df_norm <- as.data.frame(lapply(df[1:12], normalize))

# Predict the MEDV value for the new record.
prediction <- predict(model, df_norm)
cat("The MEDV prediction for the above record is:", prediction, "\n")
```

```
## The MEDV prediction for the above record is: 23.05
```

```
# Train the model with k = 1
model <- knnreg(x = train_data[, 1:12], y = train_data[, 13], k = 1)

# print
cat(
  "Error for Training Data at k = 1:",
  RMSE(train_data[, 13], predict(model, train_data[, 1:12])),
  "\n"
)
```

```
## Error for Training Data at k = 1: 0
```

```
# remove all env variables
rm(list = ls())
```

```
<!-- The algorithm goes through the following operations in order to produce each operation -->


<!-- For each record in the dataset, the algorithm : -->


<!-- 1. The euclidean distance of that record is calculated with every other record in the dataset. -->


<!-- 2.  The euclidean distances are sorted from the lowest to the highest. -->


<!-- 3. Takes the top K neighbors and then - -->
<!--    - If it is a classification problem, it takes the classes of each of the K neighbors and assigns the majo
rity class to the current record. -->
<!--    - If it is a regression problem, it takes the average of the output variable of each of the K neighbors a
nd assigns it to the current record. -->

<!-- --- -->


# Problem 3
```

```
# Import Accidents.xlsx
Accidents_df <- read_xlsx("Accidents.xlsx", sheet = "Data")

Accidents_df$INJURY <- Accidents_df$MAX_SEV_IR
Accidents_df <- Accidents_df %>%
  mutate_at(
    .vars = "INJURY",
    .funs = c(function(x) ifelse(x == "1" | x == "2", "Yes", "No"))
  )
Accidents_df$INJURY <- as.factor(Accidents_df$INJURY)

# Look at the summary of Injury
summary(Accidents_df$INJURY)
```

```
##    No   Yes
## 20721 21462
```

```
# Select the required columns and the 12 records
df <- select(Accidents_df, c("WEATHER_R", "TRAF_CON_R", "INJURY"))[1:12, ]

# Create the pivot table
pivot_table <- as.data.frame(table(df$WEATHER_R, df$TRAF_CON_R, df$INJURY,
  dnn = c("WEATHER_R", "TRAF_CON_R", "INJURY")
))

print(pivot_table)
```

```
##    WEATHER_R TRAF_CON_R INJURY Freq
## 1          1          0     No    1
## 2          2          0     No    5
## 3          1          1     No    1
## 4          2          1     No    1
## 5          1          2     No    1
## 6          2          2     No    0
## 7          1          0    Yes    2
## 8          2          0    Yes    1
## 9          1          1    Yes    0
## 10         2          1    Yes    0
## 11         1          2    Yes    0
## 12         2          2    Yes    0
```

```
# Function to calculate probability manually
calculate_prob <- function(weather_r, traf_con_r) {
  filter(pivot_table, WEATHER_R == weather_r & TRAF_CON_R == traf_con_r & INJURY == "Yes")$Freq /
    sum(filter(pivot_table, WEATHER_R == weather_r & TRAF_CON_R == traf_con_r)$Freq)
}


cat("P(INJURY = Yes | WEATHER_R = 1, TRAF_CON_R = 0):", calculate_prob(1, 0), "\n")
```

```
## P(INJURY = Yes | WEATHER_R = 1, TRAF_CON_R = 0): 0.6666667
```

```
cat("P(INJURY = Yes | WEATHER_R = 2, TRAF_CON_R = 0):", calculate_prob(2, 0), "\n")
```

```
## P(INJURY = Yes | WEATHER_R = 2, TRAF_CON_R = 0): 0.1666667
```

```
cat("P(INJURY = Yes | WEATHER_R = 1, TRAF_CON_R = 1):", calculate_prob(1, 1), "\n")
```

```
## P(INJURY = Yes | WEATHER_R = 1, TRAF_CON_R = 1): 0
```

```
cat("P(INJURY = Yes | WEATHER_R = 2, TRAF_CON_R = 1):", calculate_prob(2, 1), "\n")
```

```
## P(INJURY = Yes | WEATHER_R = 2, TRAF_CON_R = 1): 0
```

```
cat("P(INJURY = Yes | WEATHER_R = 1, TRAF_CON_R = 2):", calculate_prob(1, 2), "\n")
```

```
## P(INJURY = Yes | WEATHER_R = 1, TRAF_CON_R = 2): 0
```

```
cat("P(INJURY = Yes | WEATHER_R = 2, TRAF_CON_R = 2):", calculate_prob(2, 2), "\n")
```

```
## P(INJURY = Yes | WEATHER_R = 2, TRAF_CON_R = 2): NaN
```

```
 <!-- (2b - 3) Classify the 12 accidents using these probabilities and a cutoff of 0.5. Since the cutoff is 0.5,
the only combination of attributes that has probability greater than 0.5 is when WEATHER_R = 1 and TRAF_CON_R = 0
-->
```

```
# add a predictions column
df <- df %>%
  mutate(predictions = ifelse(WEATHER_R == 1 & TRAF_CON_R == 0, "Yes", "No"))

cat("The predictions are: \n")
```

```
## The predictions are:
```

```
print(df)
```

```
## # A tibble: 12 x 4
##    WEATHER_R TRAF_CON_R INJURY predictions
##        <dbl>      <dbl> <fct>  <chr>
##  1         1          0 Yes    Yes
##  2         2          0 No     No
##  3         2          1 No     No
##  4         1          1 No     No
##  5         1          0 No     Yes
##  6         2          0 Yes    No
##  7         2          0 No     No
##  8         1          0 Yes    Yes
##  9         2          0 No     No
## 10         2          0 No     No
## 11         2          0 No     No
## 12         1          2 No     No
```

```
# Manually calculate the probability
prob <- ((3 / 12) * ((2 / 3) * (0 / 3))) / (((3 / 12) * ((2 / 3) * (0 / 3))) + ((9 / 12) * ((3 / 9) * (2 / 9
))))

cat("P(INJURY = Yes | WEATHER_R = 1, TRAF_CON_R = 1):", prob, "\n")
```

```
## P(INJURY = Yes | WEATHER_R = 1, TRAF_CON_R = 1): 0
```

```
 <!-- (2b - 5). Run a naive Bayes classifier on the 12 records and 2 predictors. Obtain probabilities and classif
ications for all 12 records. Compare this to the exact Bayes classification. Are the resulting classifications eq
uivalent? Is the ranking (= ordering) of observations equivalent? -->
```

```
# Train the Naive Bayes Classifier
nb <- naiveBayes(INJURY ~ WEATHER_R + TRAF_CON_R, data = df[, 1:3])

pred_prob <- predict(nb, newdata = df[, 1:3], type = "raw")

pred_class <- data.frame(ifelse(pred_prob[, 1] - pred_prob[2] < 0, "Yes", "No"))
colnames(pred_class) <- "class"

actual_vs_predicted <- data.frame("actual" = df$INJURY, "predicted" = pred_class$class)
actual_vs_predicted$exact_bayes <- df$predictions
actual_vs_predicted$no_prob <- pred_prob[, 1]
actual_vs_predicted$yes_prob <- pred_prob[, 2]

cat("Actual vs Predicted Probabilities are: \n")
```

```
## Actual vs Predicted Probabilities are:
```

```
print(actual_vs_predicted)
```

```
##    actual predicted exact_bayes     no_prob     yes_prob
## 1     Yes       Yes         Yes 0.001916916 0.9980830837
## 2      No        No          No 0.006129754 0.9938702459
## 3      No        No          No 0.999548668 0.0004513316
## 4      No        No          No 0.998552097 0.0014479028
## 5      No       Yes         Yes 0.001916916 0.9980830837
## 6     Yes        No          No 0.006129754 0.9938702459
## 7      No        No          No 0.006129754 0.9938702459
## 8     Yes       Yes         Yes 0.001916916 0.9980830837
## 9      No        No          No 0.006129754 0.9938702459
## 10     No        No          No 0.006129754 0.9938702459
## 11     No        No          No 0.006129754 0.9938702459
## 12     No        No          No 0.989399428 0.0106005719
```

```
<!-- The resulting classifications of the Naive Bayes Classifier and the Exact Bayes classifier are equivalent. T
heir ranking and ordering is also equivalent. -->
```

```
# Generate the training data indices
df <- Accidents_df[, c(1, 3, 5, 6, 7, 8, 11, 12, 14, 15, 16, 20, 25)]
indices <- sample(seq_len(nrow(df)), size = floor(0.6 * nrow(df)))

# Get training and validation data
train_data <- df[indices, ]
validation_data <- df[-indices, ]
```

```
nb <- naiveBayes(INJURY ~ ., train_data)

pred_class <- predict(nb, newdata = train_data)

cat("The Classification Matrix is :\n")
```

```
## The Classification Matrix is :
```

```
confusionMatrix(data = pred_class, reference = train_data$INJURY)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    No   Yes
##        No  12461   289
##        Yes     0 12559
##
##                Accuracy : 0.9886
##                  95% CI : (0.9872, 0.9899)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9772
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.9775
##          Pos Pred Value : 0.9773
##          Neg Pred Value : 1.0000
##              Prevalence : 0.4924
##          Detection Rate : 0.4924
##    Detection Prevalence : 0.5038
##       Balanced Accuracy : 0.9888
##
##        'Positive' Class : No
##
```

```
<!-- (3c -3) What is the overall error for the validation set? -->
```

```
pred_class <- predict(nb, newdata = validation_data)

cf <- confusionMatrix(data = pred_class, reference = validation_data$INJURY)

cat("The error rate is :", paste0(round(100 * (1 - cf$overall[[1]]), 4), "%"), "\n")
```

```
## The error rate is : 1.0549%
```

```
<!-- (3c- 4). What is the percent improvement relative to the naive rule (using the validation set)? -->
```

```
naive_cf <- confusionMatrix(
  data = as.factor(rep("Yes", dim(validation_data)[[1]])),
  reference = validation_data$INJURY
)

cat(
  "The difference between the Naive Bayes Classifier accuracy and the Naive Rule accuracy is:",
  paste0(round(100 * (cf$overall[[1]] - naive_cf$overall[[1]]), 4), "%"), "\n"
)
```

```
## The difference between the Naive Bayes Classifier accuracy and the Naive Rule accuracy is: 47.8962%
```

```
<!-- (3c - 5). Examine the conditional probabilities output. Why do we get a probability of zero for P(INJURY = N
o | SPD_LIM = 5)? -->
```

```
pivot_table <- as.data.frame(table(validation_data$INJURY, validation_data$SPD_LIM,
  dnn = c("INJURY", "SPD_LIM")
))


cat(
  "P(INJURY = No | SPD_LIM = 5):",
  filter(pivot_table, SPD_LIM == 5 & INJURY == "No")$Freq /
    sum(filter(pivot_table, SPD_LIM == 5)$Freq)
)
```

```
## P(INJURY = No | SPD_LIM = 5):
```

```
rm(list = ls())
```