STOR 565 Fall 2019 Homework 5

Ted Henson

Remark. Credits for **Theoretical Part** and **Computational Part** are in total *100 pt* (40 pt for theoretical and 60pt for computational) please complete your computational report below in the **RMarkdown** file and submit your printed PDF homework created by it.

Computational Part

Question 1

You may need some of these packages:

```
library(MASS)
library(class)
```

Load and read more about the data

- Load the data *OnlineNewsPopularityTraining.csv*, which contains a large portion of the data set from the above competition.

```
library(readr)
training <- read_csv("OnlineNewsPopularityTraining.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   url = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

- Read the variable descriptions for the variables at this website: UCI website
- A binary label has been added to the data set `popular`, which specifies whether or not each website is considered a popular website (0 for popular and 1 for not popular).
- `popular` was created by assigning 1 to rows with `shares` values greater than 3300, and zero otherwise.

Prepare the data

- Remove the variables *shares*, *url* and *timedelta* from the dataset.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v ggplot2 3.2.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x dplyr::select() masks MASS::select()
```

```r
training = training %>% dplyr::select(-shares,
                                      -url,
                                      -timedelta)
```

Questions

(a) (10 points) The aim of this computational exercise is to prepare a classifier to predict whether or not a new website will be popular, i.e. classification by the `popular` variable in the dataset. You will do so using

- LDA
- QDA
- K-nearest neighbors

For each of the methods,

1) carefully describe how you choose any thresholds or tuning parameters.

For K-nearest neighbors, we would use cross validation to find the optimal number of k (neighbors).

2) list the predictors you would remove, if any, before fitting your models.

You must justify your answers by specifically naming concepts studied in this course. You also might want to justify your choices with summaries or plots of the data. Please do not print large amounts of data in the output.

I am being intentionally vague here because I want to see how you would handle such a data set in practice. All I ask is that you give proper justification for whatever you are doing. For example: the data contains indicator variables for different days of the week (weekday_is_monday etc). When doing LDA **I would remove these sorts of variables** as LDA inherently assumes that the features are continuous (and have a normal distribution).

```r
library(nortest)

my.func = function(x){


p.value = ad.test(x)$p.value
return(p.value)

}

ad.test.out = apply(training[,1:c(ncol(training)-1)], c(2), my.func)

print(ad.test.out)
```

```
##               n_tokens_title             n_tokens_content
##                      3.7e-24                      3.7e-24
##              n_unique_tokens             n_non_stop_words
##                      3.7e-24                      3.7e-24
##     n_non_stop_unique_tokens                    num_hrefs
##                      3.7e-24                      3.7e-24
##               num_self_hrefs                     num_imgs
##                      3.7e-24                      3.7e-24
##                   num_videos         average_token_length
##                      3.7e-24                      3.7e-24
##                 num_keywords    data_channel_is_lifestyle
##                      3.7e-24                      3.7e-24
```

```
## data_channel_is_entertainment           data_channel_is_bus
##                       3.7e-24                       3.7e-24
##        data_channel_is_socmed          data_channel_is_tech
##                       3.7e-24                       3.7e-24
##         data_channel_is_world                    kw_min_min
##                       3.7e-24                       3.7e-24
##                    kw_max_min                    kw_avg_min
##                       3.7e-24                       3.7e-24
##                    kw_min_max                    kw_max_max
##                       3.7e-24                       3.7e-24
##                    kw_avg_max                    kw_min_avg
##                       3.7e-24                       3.7e-24
##                    kw_max_avg                    kw_avg_avg
##                       3.7e-24                       3.7e-24
##     self_reference_min_shares     self_reference_max_shares
##                       3.7e-24                       3.7e-24
##    self_reference_avg_sharess             weekday_is_monday
##                       3.7e-24                       3.7e-24
##            weekday_is_tuesday          weekday_is_wednesday
##                       3.7e-24                       3.7e-24
##           weekday_is_thursday             weekday_is_friday
##                       3.7e-24                       3.7e-24
##           weekday_is_saturday             weekday_is_sunday
##                       3.7e-24                       3.7e-24
##                    is_weekend                        LDA_00
##                       3.7e-24                       3.7e-24
##                        LDA_01                        LDA_02
##                       3.7e-24                       3.7e-24
##                        LDA_03                        LDA_04
##                       3.7e-24                       3.7e-24
##           global_subjectivity     global_sentiment_polarity
##                       3.7e-24                       3.7e-24
##    global_rate_positive_words    global_rate_negative_words
##                       3.7e-24                       3.7e-24
##           rate_positive_words           rate_negative_words
##                       3.7e-24                       3.7e-24
##         avg_positive_polarity         min_positive_polarity
##                       3.7e-24                       3.7e-24
##         max_positive_polarity         avg_negative_polarity
##                       3.7e-24                       3.7e-24
##         min_negative_polarity         max_negative_polarity
##                       3.7e-24                       3.7e-24
##            title_subjectivity      title_sentiment_polarity
##                       3.7e-24                       3.7e-24
##        abs_title_subjectivity  abs_title_sentiment_polarity
##                       3.7e-24                       3.7e-24
```

```
bartlett.test(training[,1:c(ncol(training)-1)])
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  training[, 1:c(ncol(training) - 1)]
## Bartlett's K-squared = 34012017, df = 57, p-value < 2.2e-16
```

My Answer

After looking at the description of the predictors, none of them would intuitively seem to cause auto correlation with the response. Our Anderson Darling test also shows that all of our variables true distribution is most likely normal so the first LDA assumption is valid; however, our bartlett test shows that at least some of our variables probably have different variances, so LDA may not be valid as it assumes all variances are equal. Additionally, some of our variables are binary (all of the ones with is_ in the game), so we would need to remove them in order to do LDA and/or QDA. Below we discovered that some of our variables were almost perfectly collinear within a given class. We removed these redudant variables as a result. The built in knn.cv function is computationally taxing so we will choose k = 1 for now.

(b) (10 points)For **each of the methods** listed in (a):

1) Fit a model to predict `popular` class labels, consistent with your answer in (a).

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
# training$popular = as.factor(training$popular)
#
popular = training$popular


library(usdm)
```

```
## Loading required package: sp

## Loading required package: raster

##
## Attaching package: 'raster'

## The following object is masked from 'package:dplyr':
##
##     select

## The following object is masked from 'package:tidyr':
##
##     extract

## The following objects are masked from 'package:MASS':
##
##     area, select

## The following object is masked from 'package:e1071':
##
##     interpolate
```

```
#remove binary variables
mod.data = training[,-c(which(grepl('is_', colnames(training))))]

mod.data.pop = mod.data %>% dplyr::filter(popular == 1) %>% dplyr::select(-popular)
```

```r
cor.matrix = cor(mod.data.pop)


my.func = function(x){

  num = sum(abs(x) > .90)
  return(num)
}
perfect.collinear = apply(cor.matrix, 2,  my.func)



#remove almost perfect collinear variables

mod.data = mod.data %>% dplyr::select(-c(n_unique_tokens,
                                         n_non_stop_words,
                                         kw_max_min,
                                         LDA_00))


#checking other class
# mod.data.pop = mod.data %>% dplyr::filter(popular == 0) %>% dplyr::select(-popular)
#
# cor.matrix = cor(mod.data.pop)
#
#
# my.func = function(x){
#
#   num = sum(abs(x) > .90)
#   return(num)
# }
# apply(cor.matrix, 2,  my.func)

# mod.data = mod.data %>% dplyr::select(-popular)

qda.mod = qda(popular ~ .,
              data = mod.data)


lda.mod = lda(popular ~ .,
              data = mod.data)



smp_size <- floor(0.75 * nrow(mod.data))

## set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(mod.data)), size = smp_size)

train <- mod.data[train_ind, ]
test.data <- mod.data[-train_ind, ]
```

2) Briefly discuss your results.

You must show summary output of this model, along with plots and other documentation.

As shown below, the class means found by QDA and LDA are mostly the same when comparing the two model types. Aditionally, the means for a given variable are not that different when comparing different classes; however, these small differences are found for almost every variable. Some of the larger differences in the means between classes are for the variables relating to the "self_reference_shares". The K nearest neighbor model will be discussed below when evaluating the predictions, as the built in R knn does not provide any output to discuss aside from the predicted values.

QDA Model

```
qda.mod
```

```
## Call:
## qda(popular ~ ., data = mod.data)
##
## Prior probabilities of groups:
##        0        1
## 0.797074 0.202926
##
## Group means:
##   n_tokens_title n_tokens_content n_non_stop_unique_tokens num_hrefs
## 0       10.41942         543.8064                0.6767680  10.45024
## 1       10.34664         555.8751                0.7605938  12.42806
##   num_self_hrefs num_imgs num_videos average_token_length num_keywords
## 0       3.263331 4.233742   1.186392             4.567123     7.172271
## 1       3.362958 5.676818   1.516004             4.472787     7.416252
##   kw_min_min kw_avg_min kw_min_max kw_max_max kw_avg_max kw_min_avg
## 0   26.11195   303.9152   13103.95   751445.9   255841.8   1060.748
## 1   26.38222   353.8045   15430.51   755083.9   274281.7   1313.797
##   kw_max_avg kw_avg_avg self_reference_min_shares
## 0   5380.940   3013.687                  3384.996
## 1   6797.851   3618.107                  6373.200
##   self_reference_max_shares self_reference_avg_sharess    LDA_01    LDA_02
## 0                   9024.61                   5521.533 0.1442137 0.2333256
## 1                  15470.31                   9812.608 0.1290856 0.1480959
##      LDA_03    LDA_04 global_subjectivity global_sentiment_polarity
## 0 0.2059008 0.2324290           0.4403618                 0.1183681
## 1 0.2947011 0.2390947           0.4554528                 0.1230559
##   global_rate_positive_words global_rate_negative_words
## 0                 0.03945965                 0.01658678
## 1                 0.04046900                 0.01677807
##   rate_positive_words rate_negative_words avg_positive_polarity
## 0           0.6835300           0.2899273             0.3532167
## 1           0.6777887           0.2797936             0.3571710
##   min_positive_polarity max_positive_polarity avg_negative_polarity
## 0            0.09585667             0.7546777            -0.2570702
## 1            0.09496213             0.7642330            -0.2684284
##   min_negative_polarity max_negative_polarity title_subjectivity
## 0            -0.5182557            -0.1064141          0.2758016
## 1            -0.5340489            -0.1101093          0.3110648
##   title_sentiment_polarity abs_title_subjectivity
## 0               0.06594272              0.3418020
## 1               0.08855336              0.3409584
```

```
##   abs_title_sentiment_polarity
## 0                    0.1511620
## 1                    0.1758323
```

LDA Model

```
lda.mod
```

```
## Call:
## lda(popular ~ ., data = mod.data)
##
## Prior probabilities of groups:
##        0        1
## 0.797074 0.202926
##
## Group means:
##   n_tokens_title n_tokens_content n_non_stop_unique_tokens num_hrefs
## 0       10.41942         543.8064                0.6767680  10.45024
## 1       10.34664         555.8751                0.7605938  12.42806
##   num_self_hrefs num_imgs num_videos average_token_length num_keywords
## 0       3.263331 4.233742   1.186392             4.567123     7.172271
## 1       3.362958 5.676818   1.516004             4.472787     7.416252
##   kw_min_min kw_avg_min kw_min_max kw_max_max kw_avg_max kw_min_avg
## 0   26.11195   303.9152   13103.95   751445.9   255841.8   1060.748
## 1   26.38222   353.8045   15430.51   755083.9   274281.7   1313.797
##   kw_max_avg kw_avg_avg self_reference_min_shares
## 0   5380.940   3013.687                  3384.996
## 1   6797.851   3618.107                  6373.200
##   self_reference_max_shares self_reference_avg_sharess    LDA_01    LDA_02
## 0                   9024.61                   5521.533 0.1442137 0.2333256
## 1                  15470.31                   9812.608 0.1290856 0.1480959
##      LDA_03    LDA_04 global_subjectivity global_sentiment_polarity
## 0 0.2059008 0.2324290           0.4403618                 0.1183681
## 1 0.2947011 0.2390947           0.4554528                 0.1230559
##   global_rate_positive_words global_rate_negative_words
## 0                 0.03945965                 0.01658678
## 1                 0.04046900                 0.01677807
##   rate_positive_words rate_negative_words avg_positive_polarity
## 0           0.6835300           0.2899273             0.3532167
## 1           0.6777887           0.2797936             0.3571710
##   min_positive_polarity max_positive_polarity avg_negative_polarity
## 0            0.09585667             0.7546777            -0.2570702
## 1            0.09496213             0.7642330            -0.2684284
##   min_negative_polarity max_negative_polarity title_subjectivity
## 0            -0.5182557            -0.1064141          0.2758016
## 1            -0.5340489            -0.1101093          0.3110648
##   title_sentiment_polarity abs_title_subjectivity
## 0               0.06594272              0.3418020
## 1               0.08855336              0.3409584
##   abs_title_sentiment_polarity
## 0                    0.1511620
## 1                    0.1758323
##
```

```
## Coefficients of linear discriminants:
##                                         LD1
## n_tokens_title                 4.350522e-03
## n_tokens_content               1.284343e-04
## n_non_stop_unique_tokens       1.350914e-02
## num_hrefs                      1.335559e-02
## num_self_hrefs                -2.165433e-02
## num_imgs                       9.759884e-03
## num_videos                     5.432896e-03
## average_token_length          -2.851132e-01
## num_keywords                   6.914187e-02
## kw_min_min                     6.061239e-04
## kw_avg_min                    -1.196718e-04
## kw_min_max                    -3.222743e-07
## kw_max_max                    -4.336608e-07
## kw_avg_max                    -1.423588e-06
## kw_min_avg                    -1.563911e-04
## kw_max_avg                    -1.514723e-04
## kw_avg_avg                     1.214389e-03
## self_reference_min_shares      4.659648e-06
## self_reference_max_shares      5.438458e-07
## self_reference_avg_sharess     3.795993e-06
## LDA_01                        -1.055658e+00
## LDA_02                        -8.545650e-01
## LDA_03                        -4.844748e-01
## LDA_04                        -6.835289e-02
## global_subjectivity            1.602217e+00
## global_sentiment_polarity     -3.926878e-01
## global_rate_positive_words    -5.941723e-01
## global_rate_negative_words     2.399767e+00
## rate_positive_words            5.139836e-01
## rate_negative_words            2.208717e-01
## avg_positive_polarity         -7.240605e-01
## min_positive_polarity         -3.334176e-01
## max_positive_polarity         -4.636846e-02
## avg_negative_polarity         -8.507625e-01
## min_negative_polarity          1.584331e-01
## max_negative_polarity          2.488653e-01
## title_subjectivity             2.293782e-01
## title_sentiment_polarity       3.324164e-01
## abs_title_subjectivity         4.063724e-01
## abs_title_sentiment_polarity -5.365356e-03
```

(c) (10 points) Download the test data *OnlineNewsPopularityTest.csv*. Predict `popular` class labels using each of the models in (b). Then:

```r
library(caret)
library(DAAG)
```

```
##
## Attaching package: 'DAAG'

## The following object is masked from 'package:usdm':
##
##     vif
```

```
## The following object is masked from 'package:MASS':
##
##     hills
```

```r
test <- read_csv("OnlineNewsPopularityTest.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   url = col_character()
## )
## See spec(...) for full column specifications.
```

```r
lda.pred = predict(object = lda.mod,
                   newdata = test)$class
qda.pred = predict(qda.mod, test)$class


knn.pred = knn(train = train[,1:ncol(train)-1],
                 cl = factor(train$popular),
              test = test.data[,1:ncol(test.data)-1],
                 k = 1)




#lda predictions
# table(lda.pred, test$popular)
#
# mean(lda.pred == test$popular)
confusion(lda.pred, test$popular)
```

```
## Overall accuracy = 0.794
##
## Confusion matrix
##       Predicted (cv)
## Actual     0     1
##      0 0.801 0.199
##      1 0.477 0.523
```

```r
#qda predictions

# table(qda.pred, test$popular)
#
# mean(qda.pred == test$popular)

confusion(qda.pred, test$popular)
```

```
## Overall accuracy = 0.781
##
## Confusion matrix
##       Predicted (cv)
## Actual     0     1
##      0 0.799 0.201
##      1 0.648 0.352
```

```r
#knn predictions
# table(knn.pred, test$popular)
#
# mean(knn.pred == test$popular)


confusion(knn.pred, test.data$popular)
```

```
## Overall accuracy = 0.704
##
## Confusion matrix
##       Predicted (cv)
## Actual     0     1
##      0 0.812 0.188
##      1 0.735 0.265
```

c.1) Discuss the performance of each method using assessment measures such as MSPE, sensitivity, and specificity (see slide 68-69 for definitions of these objects; here popularity (class label 1) counts as "positives" and not popularity (class label 0) counts as negatives).

The LDA had the best overall accuracy and within each class. The QDA had better overall accruacy within each class compared to the KNN. In terms of false positives and false negatives, for each model, more misclassifications come from classifying text as unpopular, when they are actually popular. The LDA does this the least percentage of the time compared to the other models.

c.2) Discuss which classifier you prefer and why.

I would prefer the qda classifier since it has the best true classification rate across the board. In the future using cross validation to find the optimal number of K may make KNN superior; however, given the size of this dataset, the built in r knn.cv function is too computationally taxing to justify a small improvement from the LDA.

Question 2

You may need the following packages for this problem:

```r
library(MASS)
library(mvtnorm)
library(ggplot2)
library(e1071)
library(class)
```

Data simulation

[a.] ***(10 points)***

Simluate the 2 datasets above, one from each scenario. Write a function to find the optimal k value by 5-fold cross validation for each dataset, using the test error defined by the average number of misclassified points. The `knn.cv` function in the `class` package **DOES NOT** do this.

```r
library(MASS)
library(mvtnorm)
library(ggplot2)
library(caret)
library(e1071)
library(class)
```

```r
set.seed(100)
expit <- function(x) {
exp(x) / (1 + exp(x))
}
gen_datasets <- function() {
id <- diag(c(1, 1))
df1 <- data.frame(y=factor(rep(c(0, 1), each=50)),
rbind(rmvnorm(50, mean=c(0, 0), sigma = id),
rmvnorm(50, mean=c(1, 1), sigma = id)))
covmat <- matrix(c(1, -0.5, -0.5, 1), nrow=2)
df2 <- data.frame(y=factor(rep(c(0, 1), each=50)),
rbind(rmvnorm(50, mean=c(0, 0), sigma = covmat),
rmvnorm(50, mean=c(1, 1), sigma = covmat)))

mu <- c(0, 0); sigma <- matrix(c(1, -1/2, -1/2, 1), 2); nu <- 4
n <- 50 # Number of draws
x_first <- t(t(mvrnorm(n, rep(0, length(mu)), sigma)
* sqrt(nu / rchisq(n, nu))) + mu)
mu <- c(1, 1); sigma <- matrix(c(1, 0, 0, 1), 2); nu <- 4
n <- 50 # Number of draws
x_second <- t(t(mvrnorm(n, rep(0, length(mu)), sigma)
* sqrt(nu / rchisq(n, nu))) + mu)

list(df1, df2)
}

sim.data = gen_datasets()
```

You cannot use another built-in function to do the cross-validation though of course you will use built in functions to run the knn algorithm.

I suggest you write a general function that is intended for a single dataset, which you can then use repeatedly, rather than trying to do both data sets in one go.

Using code from previous lectures or homework, your function will need to perform the following steps:

- Randomly split the data into 5 folds of equal size.
- For a fixed k,
  - use `knn` in the `class` package to run the knn model, where the `train` argument is a data frame of your first 4 folds and `test` is your 5th fold
  - compute the classification error (and store it for output)
  - repeat the previous two steps, but with the 4th fold as your `test` argument, then the `3rd` etc.
- Repeat the previous step for k = 1, 2, 3, 4, 5.
- Return a data frame of the average classification error for each k.

```r
my.knn.func = function(df){

  results = data.frame(k = c(0),
                       avg.error = c(0))
data = df

  indices = createFolds(data$y,
                        k = 5)

 train = data[-c(indices$Fold5),]
```

```r
test = data[indices$Fold5,]

for(k in 1:5){



knn.out = knn(train = train[,2:3],
              test = test[,2:3],
              cl = train$y,
              k = k)

avg.error = mean(knn.out == test$y)

results[k,] = c(k, avg.error)
}


  return(results)
}


results.1 = my.knn.func(sim.data[[1]])
results.2 = my.knn.func(sim.data[[2]])
results.1
```

```
##   k avg.error
## 1 1      0.90
## 2 2      0.70
## 3 3      0.75
## 4 4      0.85
## 5 5      0.85
```

```r
results.2
```

```
##   k avg.error
## 1 1      0.70
## 2 2      0.65
## 3 3      0.70
## 4 4      0.65
## 5 5      0.70
```

In your response: **Show the output of running your function on the two simulated datasets, and state the optimal k value for each.**

The optimal value of k for the first and seconds simulated datasets are 4 and 5 respectively.

[b.] *(15 points)*

First: write a function to do the following:

1. **Training sets**: Simulate 2 data sets, one from each scenario above.
2. For each data set, fit LDA, QDA, k-NN with $k = 1$, k-NN with k chosen by the cross validation in part a.
3. **Test set**: Simulate another 2 data sets, one from each scenario above.
4. Using the 4 classification techniques you have estimated in Scenario 1 (Training set), apply this to the

Scenario 1 (Test set) and compute the test error rate (# of misclassified points in test set/100). Do the same for Scenario 2.

5. Return a 4 × 2 matrix of errors (first row consists of test errors for LDA on each of the 2 scenarios, 2nd row QDA test errors etc).

```
big.func = function(){
  .Random.seed

  the.matrix = matrix(nrow= 4, ncol = 2)

  expit <- function(x) {
    exp(x) / (1 + exp(x))
  }
  gen_datasets <- function() {
    id <- diag(c(1, 1))
    df1 <- data.frame(y=factor(rep(c(0, 1), each=50)),
                      rbind(rmvnorm(50, mean=c(0, 0), sigma = id),
                            rmvnorm(50, mean=c(1, 1), sigma = id)))
    covmat <- matrix(c(1, -0.5, -0.5, 1), nrow=2)
    df2 <- data.frame(y=factor(rep(c(0, 1), each=50)),
                      rbind(rmvnorm(50, mean=c(0, 0), sigma = covmat),
                            rmvnorm(50, mean=c(1, 1), sigma = covmat)))

    mu <- c(0, 0); sigma <- matrix(c(1, -1/2, -1/2, 1), 2); nu <- 4
    n <- 50 # Number of draws
    x_first <- t(t(mvrnorm(n, rep(0, length(mu)), sigma)
                 * sqrt(nu / rchisq(n, nu))) + mu)
    mu <- c(1, 1); sigma <- matrix(c(1, 0, 0, 1), 2); nu <- 4
    n <- 50 # Number of draws
    x_second <- t(t(mvrnorm(n, rep(0, length(mu)), sigma)
                  * sqrt(nu / rchisq(n, nu))) + mu)

    list(df1, df2)
  }

  train.data = gen_datasets()
  train.data$y = as.factor(train.data$y)
  # train1 = train.data[[1]]
  # train2 = train.data[[2]]
  #
  # train1$y = as.factor(train1$y)
  # train2$y = as.factor(train2$y)

  test.data = gen_datasets()
  # test1 = test.data[[1]]
  # test2 = test.data[[2]]
  #
  # test1$y = as.factor(train1$y)
  # test2$y = as.factor(train2$y)

  for(i in 1:2){
  qda.mod = qda(y ~ .,
                data = train.data[[i]])
  lda.mod = lda(y ~ .,
```

```r
                data = train.data[[i]])

  knn.1 = knn(train = train.data[[i]][,2:3],
              cl = train.data[[i]]$y,
              test = test.data[[i]][,2:3])

  if(i == 1){



  knn.opt = knn(train = train.data[[i]][,2:3],
              cl = train.data[[i]]$y,
              test = test.data[[i]][,2:3],
              k = 4)
  }

  if(i == 2){

      knn.opt = knn(train = train.data[[i]][,2:3],
              cl = train.data[[i]]$y,
              test = test.data[[i]][,2:3],
              k = 5)

  }


  lda.avg.error = 1-mean(lda.mod == test.data[[i]]$y)
  qda.avg.error = 1-mean(qda.mod == test.data[[i]]$y)
 knn.1.error = 1-mean(knn.1 == test.data[[i]]$y)
 knn.opt.error = 1-mean(knn.opt == test.data[[i]]$y)
 the.matrix[,i] = c(lda.avg.error,
                   qda.avg.error,
                   knn.1.error,
                   knn.opt.error
                   )

  }

  row.names(the.matrix) = c('LDA', 'QDA', 'KNN (k = 1)', 'K cv')
colnames(the.matrix) = c('Sim.Data.1', 'Sim.Data.2')
the.matrix
  return(the.matrix)

  }

big.func()
```

```
##             Sim.Data.1 Sim.Data.2
## LDA                1.0       1.00
## QDA                1.0       1.00
## KNN (k = 1)        0.3       0.25
## K cv               0.2       0.22
```

Second: Run your function 100 times, print the *dimension* of your function output using the `dim` function (you will have a 4 × 2 × 100 array), and print the **first three** matrices in the array only.

```
for(i in 1:100){

 results =  big.func()
 print(dim(results))
 if(i <= 3){

   print(results)
 }
}
```

```
## [1] 4 2
##              Sim.Data.1 Sim.Data.2
## LDA                1.00       1.00
## QDA                1.00       1.00
## KNN (k = 1)        0.35       0.23
## K cv               0.35       0.18
## [1] 4 2
##              Sim.Data.1 Sim.Data.2
## LDA                1.00       1.00
## QDA                1.00       1.00
## KNN (k = 1)        0.31       0.19
## K cv               0.25       0.21
## [1] 4 2
##              Sim.Data.1 Sim.Data.2
## LDA                1.00       1.00
## QDA                1.00       1.00
## KNN (k = 1)        0.21       0.18
## K cv               0.20       0.14
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
```

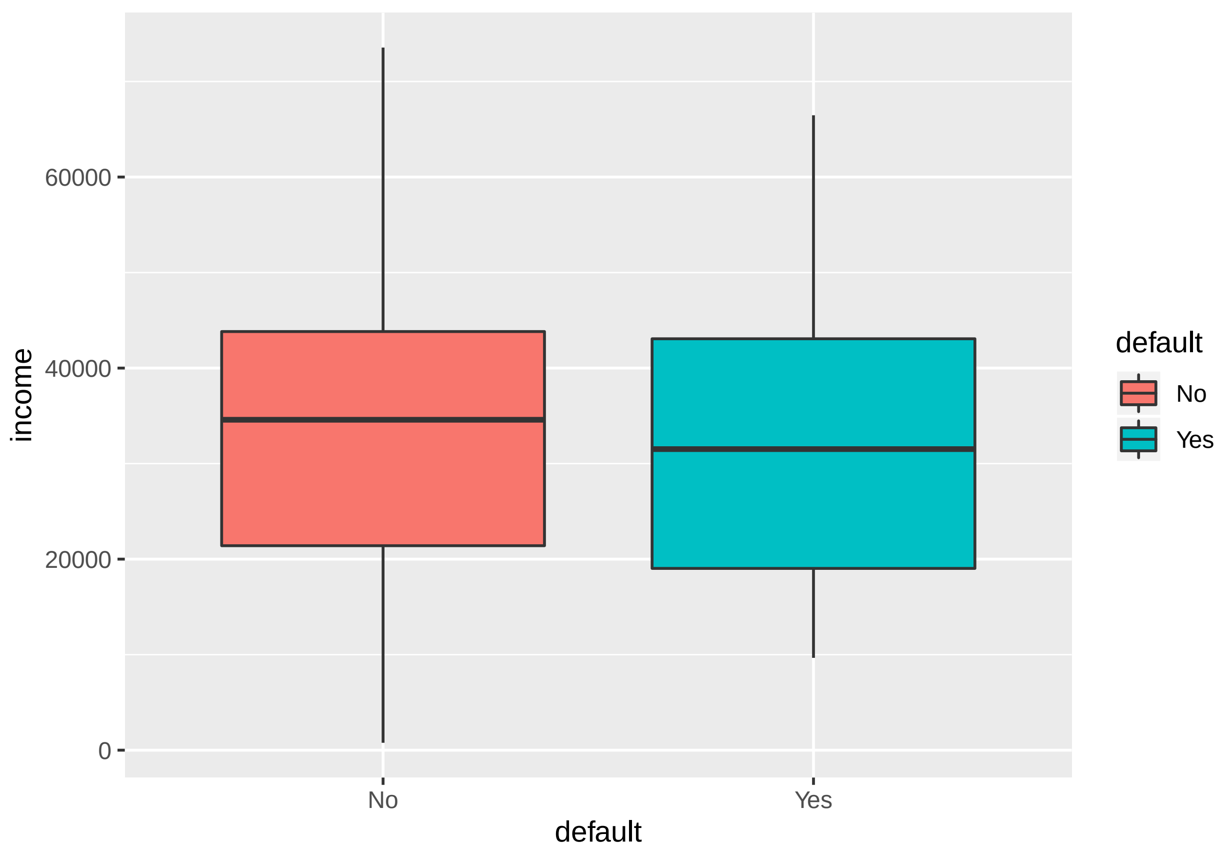
```
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
```

```
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
## [1] 4 2
```

[c.] ***(5 points)***

Make a box plot akin to Figure 4.10 and 4.11 in the ISL book.

```
ggplot(Default, aes(x = default, y = income,
                    fill = default)) + geom_boxplot()
```



```r
ggplot(Default, aes(x = default, y = balance,
                    fill = default)) + geom_boxplot()
```