STOR 565 Fall 2019 Homework 5

Hunter Finger

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 are supposed to finish the Computational Part in a readable report manner for each dataset-based analysis problem. Other than what you are asked to do, you can decide any details on your own discretion. Also goto R demonstrations in the Lecture 6 folder on Sakai. There you will find a working example for LDA, QDA and k-nn for the titanic data as well as a much more extensive demonstartion on k-nn in the folder under classification-knn in the same folder in Sakai.

You may need some of these packages:

```r
library(MASS)
library(class)
library(dplyr)
```

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

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

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

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

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------------------------------- ti

## v ggplot2 3.1.0     v readr   1.3.1
## v tibble  2.0.0     v purrr   0.2.5
## v tidyr   0.8.2     v stringr 1.4.0
## v ggplot2 3.1.0     v forcats 0.3.0

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

In particular, the MASS package for doing LDA, QDA and the class package for doing K-nn.

The following data set is coming from a Kaggle competition that came out on November 12, 2015. Here is the description from the competition:

Time magazine noted Mashable as one of the 25 best blogs in 2009, and described it as a "one stop shop" for social media. As of November 2015, [Mashable] had over 6,000,000 Twitter followers and over 3,200,000 fans on Facebook. In this problem, you'll use data from thousands of articles published by Mashable in a period of two years to see which variables predict the popularity of an article.

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.

```
NewsPopTrain = read.csv("OnlineNewsPopularityTraining.csv")
NewsPopTest = read.csv("OnlineNewsPopularityTest.csv")
```

- 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.

```
NewsPopTrain = dplyr::select(NewsPopTrain, -c(shares, url, timedelta))
NewsPopTest = dplyr::select(NewsPopTest, -c(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.
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).

```
removeLDAissue = function(data){
  #remove variables that are discrete, binary, or non-normal.
  out = data %>% dplyr::select(-c(contains("week"))) %>% dplyr::select(-c(contains("data"))) %>% dplyr:
  return(out)
}

train = removeLDAissue(NewsPopTrain)
test = removeLDAissue(NewsPopTest)
confusion <- function(yhat, y, quietly = FALSE){
  if(!quietly)
    message("yhat is the vector of predicted outcomes, possibly a factor.\n
          Sensitivity = (first level predicted) / (first level actual) \n
          Specificity = (second level predicted) / (second level actual)")

  if(!is.factor(y) & is.factor(yhat))
    y <- as.factor(y)

  if(!all.equal(levels(yhat), levels(y)))
    stop("Factor levels of yhat and y do not match.")

  confusion_mat <- table(yhat, y, deparse.level = 2)
  stats <- data.frame(sensitivity = confusion_mat[1, 1]/sum(confusion_mat[, 1]),
                              specificity = confusion_mat[2, 2]/sum(confusion_mat[, 2]))

  return(list(confusion_mat = confusion_mat, stats = stats))
}
```

I removed all variables except for 12 variables that appeared normal to me. These varaibles can be seen in the code chunk below. All other variables were either discrete, skewed, or binary which was the reason for their removal.

```
names(train)
```

```
##  [1] "n_unique_tokens"             "average_token_length"
##  [3] "kw_avg_max"                  "kw_avg_avg"
##  [5] "global_subjectivity"         "global_sentiment_polarity"
##  [7] "global_rate_positive_words" "global_rate_negative_words"
##  [9] "rate_positive_words"         "rate_negative_words"
## [11] "avg_positive_polarity"       "avg_negative_polarity"
## [13] "popular"
```

(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).

2) Briefly discuss your results.

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

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

```
## Call:
## lda(popular ~ ., data = train)
##
## Prior probabilities of groups:
##        0        1
## 0.797074 0.202926
##
## Group means:
##   n_unique_tokens average_token_length kw_avg_max kw_avg_avg
## 0       0.5328523             4.567123   255841.8   3013.687
## 1       0.6324086             4.472787   274281.7   3618.107
##   global_subjectivity global_sentiment_polarity global_rate_positive_words
## 0           0.4403618                 0.1183681                 0.03945965
## 1           0.4554528                 0.1230559                 0.04046900
##   global_rate_negative_words rate_positive_words rate_negative_words
## 0                 0.01658678           0.6835300           0.2899273
## 1                 0.01677807           0.6777887           0.2797936
##   avg_positive_polarity avg_negative_polarity
## 0             0.3532167            -0.2570702
## 1             0.3571710            -0.2684284
##
## Coefficients of linear discriminants:
##                                      LD1
## n_unique_tokens             1.575560e-02
## average_token_length       -3.261939e-01
## kw_avg_max                 -1.021587e-06
## kw_avg_avg                  6.809144e-04
## global_subjectivity         3.560820e+00
## global_sentiment_polarity   1.771757e-01
## global_rate_positive_words  4.321171e-01
## global_rate_negative_words  4.403983e+00
## rate_positive_words        -5.636772e-01
## rate_negative_words        -1.184069e+00
## avg_positive_polarity      -5.247082e-01
## avg_negative_polarity      -7.536105e-01
```

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

```
## Call:
## qda(popular ~ ., data = train)
##
## Prior probabilities of groups:
##        0        1
## 0.797074 0.202926
##
## Group means:
##   n_unique_tokens average_token_length kw_avg_max kw_avg_avg
## 0       0.5328523             4.567123   255841.8   3013.687
## 1       0.6324086             4.472787   274281.7   3618.107
##   global_subjectivity global_sentiment_polarity global_rate_positive_words
## 0           0.4403618                 0.1183681                 0.03945965
## 1           0.4554528                 0.1230559                 0.04046900
##   global_rate_negative_words rate_positive_words rate_negative_words
## 0                 0.01658678           0.6835300           0.2899273
## 1                 0.01677807           0.6777887           0.2797936
##   avg_positive_polarity avg_negative_polarity
## 0             0.3532167            -0.2570702
## 1             0.3571710            -0.2684284
```

```
knn_models <- list()
ktrain <- dplyr::select(train, -popular)
ktest <- dplyr::select(test, -popular)

for (i in 1:50){

  knn_models[[i]] <- knn(ktrain, ktest, cl = train$popular, k = i)

}
```

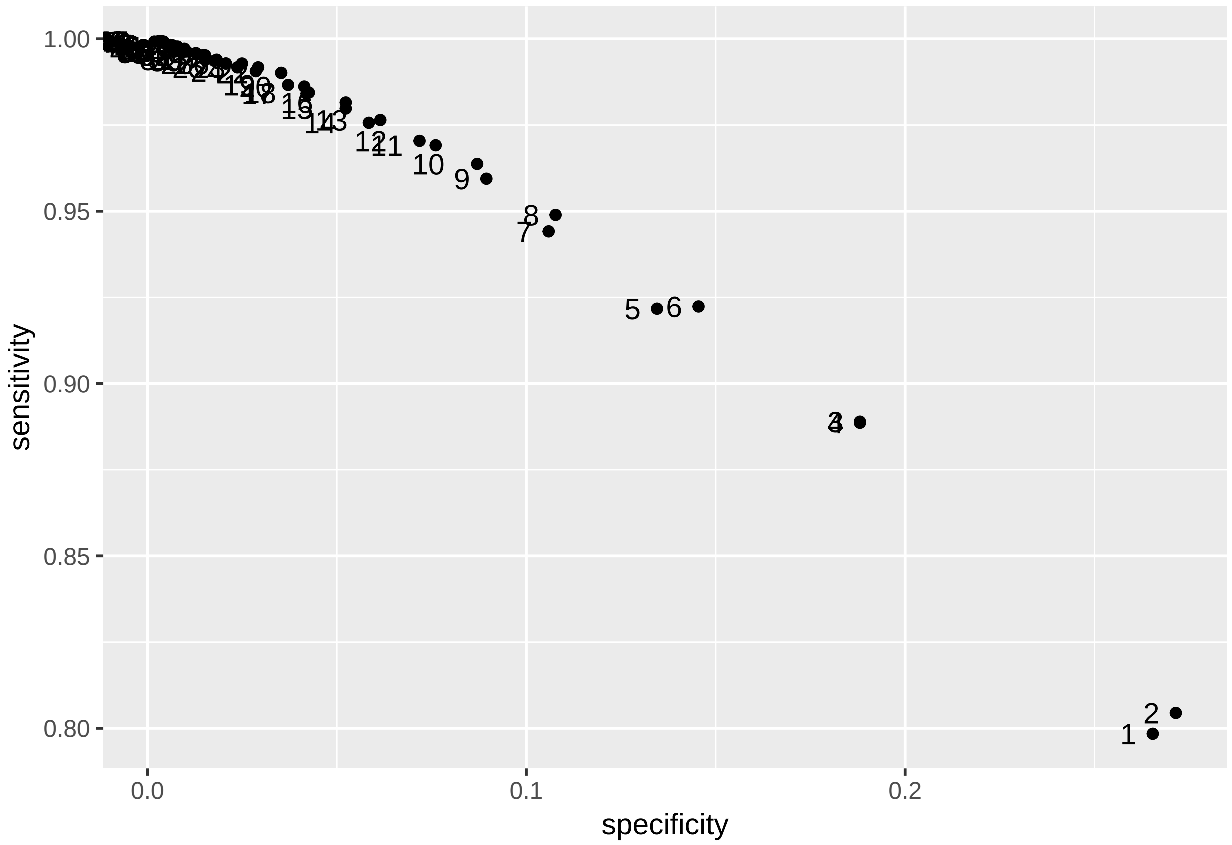
```r
knn_results <- lapply(knn_models, FUN = function(x){

  return(confusion(x, test$popular, quietly = TRUE)$stats)

  })

knn_results <- bind_rows(knn_results)
knn_results$K <- 1:50

ggplot(knn_results, aes(x = specificity, y = sensitivity, label = K)) + geom_point() + geom_text(hjust
```



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

```r
lda.pred = predict(lda.mod, newdata = test)
lda.class = lda.pred$class
table(lda.class, test$popular)
```

```
##
## lda.class    0    1
##         0 6248 1615
##         1   37   28
```

```r
mean(lda.class == test$popular)
```

```
## [1] 0.7916246
```

```r
lda.pred$posterior[1:20, 1]
```

```
##         1         2         3         4         5         6         7 
## 0.8400572 0.7987660 0.7954848 0.8612642 0.8438833 0.8338079 0.8067097 
##         8         9        10        11        12        13        14 
## 0.8376274 0.7910236 0.8575347 0.6826626 0.8290980 0.8409415 0.7179474 
##        15        16        17        18        19        20 
## 0.8549169 0.8778771 0.8627990 0.8548220 0.8807878 0.8257492
```

```r
lda.class[1:20]
```

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

```r
confusion(yhat = lda.pred$class, y = test$popular,  quietly = T)
```

```
## $confusion_mat
##     y
## yhat    0    1
##    0 6248 1615
##    1   37   28
## 
## $stats
##   sensitivity specificity
## 1    0.994113    0.017042
```

```r
qda.pred = predict(qda.mod, newdata = test)
qda.class = qda.pred$class
table(qda.class, test$popular)
```

```
##          
## qda.class    0    1
##         0 6055 1564
##         1  230   79
```

```r
mean(qda.class == test$popular)
```

```
## [1] 0.7737134
```

```r
qda.pred$posterior[1:20, 1]
```

```
##         1         2         3         4         5         6         7 
## 0.9987872 0.9875674 0.9974714 0.9976667 0.9973937 0.9982678 0.9982206 
##         8         9        10        11        12        13        14 
## 0.9987649 0.4601215 0.9937417 0.9853134 0.9911803 0.9985577 0.9949633 
##        15        16        17        18        19        20 
## 0.9988654 0.9982417 0.9985323 0.9878233 0.9954160 0.9986004
```

```r
qda.class[1:20]
```

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

```r
confusion(yhat = qda.pred$class, y = test$popular,  quietly = T)
```

```
## $confusion_mat
##     y
## yhat    0    1
##    0 6055 1564
##    1  230   79
##
## $stats
##   sensitivity specificity
## 1   0.9634049  0.04808278
```

```
knn = knn(train = ktrain, test = ktest, cl = train$popular, k = 50)
table(knn, test$popular)
```

```
##
## knn    0    1
##   0 6281 1638
##   1    4    5
```

```
mean(knn == test$popular)
```

```
## [1] 0.792886
```

```
confusion(yhat = knn, y = test$popular, quietly = T)
```

```
## $confusion_mat
##     y
## yhat    0    1
##    0 6281 1638
##    1    4    5
##
## $stats
##   sensitivity specificity
## 1   0.9993636 0.003043214
```

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).

KNN and LDA are the two most accurate classifiers. KNN is has an accuracy of 79.3% while LDA has an accuracy of 79.1%. QDA was the worst in all catergories so I am going to exclude it from future conversation. LDA has a sensitivity of .994 and specificity of .017. KNN is vastly better with a k selected to be 50 (the largest number in my range which was chosen with time in mind) with sensitivity of .999 and a specificity of .001.

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

If I was aiming for a model that is easier to explain to someone without statistical knowledge or time constraints were a concern, I would chose LDA because of its simplicity and relative performance compared to KNN. However, if the goal was to create the best model possible, I would chose the KNN model but run possible K's until the global solution was found. This would create the best possible model for the data given.

Question 2

You may need the following packages for this problem:

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

The aim is to understand the performance of different classification schemes.

Data simulation

(**Important: I have essentially adapted this from Gaston Sanchez's HW. For the data simulation portion, In the Computational zip folder, you will find the corresponding lab which essentially has full running code for part a of this problem. You will need to modify that code as I am asking you to simulate fewer scenarios etc**)

You are going to simulate 2 different sorts of data sets. Each data set has two classes, class 0 and class 1. Further in each scenario you will have 50 points belonging to class 0 and 50 belonging to class 1.

Scenario 1

Class 0: 50 points from bivariate normal:

$$\boldsymbol{\mu}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \Sigma_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Class 1: 50 points from Bivariate normal:

$$\boldsymbol{\mu}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Note that here we are in the setting where both classes have features that are multivariate normal with same Σ but different means. Further the form of Σ implies the two features for any individual are independent.

Scenario 2

Class 0: 50 points from bivariate normal:

$$\boldsymbol{\mu}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \Sigma_0 = \begin{pmatrix} 1 & -.5 \\ -.5 & 1 \end{pmatrix}.$$

Class 1: 50 points from Bivariate normal:

$$\boldsymbol{\mu}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Note that here we are in the setting where both classes have features that are multivariate normal with different Σ and different means.

[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.

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.

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

```
data_creation = 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 = id)))
  list(df1, df2)
}

knn_cv = function(data){
  data = data[sample(nrow(data)),]
  folds = cut(seq(1,nrow(data)), breaks = 5, labels = F)
  knn_model <- list()
  for(i in 1:5){
    testIDs = which(folds == i, arr.ind = T)
    test = data[testIDs,]
    ktest = dplyr::select(test, -y)
    train = data[-testIDs,]
    ktrain = dplyr::select(train, -y)
    knn_model[[i]] <- knn(ktrain, ktest, cl = train$y, k = i)


  }

knn_results <- lapply(knn_model, FUN = function(x){

  return(confusion(x, test$y, quietly = TRUE)$stats)

  })

knn_results <- bind_rows(knn_results)
knn_results$K <- 1:5

error = matrix(rep(NA, 1, 5))
for(i in 1:5){
testtab = table(knn_model[[i]], test$y)
  error[i] = (testtab["0", "1"] + testtab["1", "0"])/20
}
error


}
data = data_creation()
scenario1 = data[[1]]
scenario2 = data[[2]]
print("Scenario 1 Error Rate")
```

```
## [1] "Scenario 1 Error Rate"
```

```
knn_cv(scenario1)
```

```
##      [,1]
## [1,] 0.50
## [2,] 0.40
## [3,] 0.35
## [4,] 0.60
## [5,] 0.35
```

```
print("Scenario 2 Error Rate")
```

```
## [1] "Scenario 2 Error Rate"
```

```
knn_cv(scenario2)
```

```
##      [,1]
## [1,] 0.55
## [2,] 0.50
## [3,] 0.40
## [4,] 0.45
## [5,] 0.20
```

The optimal K for both scenarios is 5.

[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).

```
partb = function(){
  data = data_creation()
  scenario1tr = data[[1]]
  scenario2tr = data[[2]]

  testdata = data_creation()
  scenario1te = testdata[[1]]
  scenario2te = testdata[[2]]
  ktest1 = dplyr::select(scenario1te, -y)
  ktest2 = dplyr::select(scenario2te, -y)
  ktrain1 = dplyr::select(scenario1tr, -y)
  ktrain2 = dplyr::select(scenario2tr, -y)

  lda.mod1 = lda(y ~ ., data = scenario1tr)
  qda.mod1 = qda(y ~ ., data = scenario1tr)
  knn1_1 = knn(ktrain1, ktest1, cl = scenario1te$y, k = 1)
```

```r
  knn5_1 = knn(ktrain1, ktest1, cl = scenario1te$y, k = 5)

  lda.mod2 = lda(y~., data = scenario2tr)
  qda.mod2 = qda(y~., data = scenario2tr)

  knn1_2 = knn(ktrain2, ktest2, cl = scenario2te$y, k = 1)
  knn5_2 = knn(ktrain2, ktest2, cl = scenario2te$y, k = 5)

  lda.pred1 = predict(lda.mod1, newdata = scenario1te)
  lda.pred2 = predict(lda.mod2, newdata = scenario2te)
  ldatab1 = table(lda.pred1$class, scenario1te$y)
  ldatab2 = table(lda.pred2$class, scenario2te$y)
  error_1 = (ldatab1["0", "1"] + ldatab1["1", "0"])/100
  error_2 = (ldatab2["0", "1"] + ldatab2["1", "0"])/100


  qda.pred1 = predict(qda.mod1, newdata = scenario1te)
  qda.pred2 = predict(qda.mod2, newdata = scenario2te)
  qdatab1 = table(qda.pred1$class, scenario1te$y)
  qdatab2 = table(qda.pred2$class, scenario2te$y)
  errorqda1 = (qdatab1["0", "1"] + qdatab1["1", "0"])/100
  errorqda2 = (qdatab2["0", "1"] + qdatab2["1", "0"])/100

  knn1_1tab = table(knn1_1, scenario1te$y)
  knn1_2tab = table(knn1_2, scenario2te$y)
  errorknn1_1 = (knn1_1tab["0", "1"] + knn1_1tab["1", "0"])/100
  errorknn1_2 = (knn1_2tab["0", "1"] + knn1_2tab["1", "0"])/100

  knn5_1tab = table(knn5_1, scenario1te$y)
  knn5_2tab = table(knn5_2, scenario2te$y)
  errorknn5_1 = (knn5_1tab["0", "1"] + knn5_1tab["1", "0"])/100
  errorknn5_2 = (knn5_2tab["0", "1"] + knn5_2tab["1", "0"])/100

  row1 = cbind("LDA", error_1,error_2)
  row2 = cbind("QDA", errorqda1,errorqda2)
  row3 = cbind("KNN1", errorknn1_1,errorknn1_2)
  row4 = cbind("KNN5", errorknn5_1,errorknn5_1)
  rbind(row1, row2, row3, row4)

}
partb()
```

```
##              error_1 error_2
## [1,] "LDA"  "0.24"  "0.13"
## [2,] "QDA"  "0.26"  "0.11"
## [3,] "KNN1" "0.28"  "0.34"
## [4,] "KNN5" "0.29"  "0.29"
```

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.

```r
result = replicate(100, partb(), simplify=FALSE)
dim(result)
```

```
## NULL
```

```r
for(i in 1:3){
  print(result[[i]])
}
```

```
##              error_1 error_2
## [1,] "LDA"  "0.29"  "0.2"
## [2,] "QDA"  "0.27"  "0.24"
## [3,] "KNN1" "0.31"  "0.31"
## [4,] "KNN5" "0.33"  "0.33"
##              error_1 error_2
## [1,] "LDA"  "0.23"  "0.23"
## [2,] "QDA"  "0.23"  "0.21"
## [3,] "KNN1" "0.32"  "0.32"
## [4,] "KNN5" "0.24"  "0.24"
##              error_1 error_2
## [1,] "LDA"  "0.17"  "0.24"
## [2,] "QDA"  "0.19"  "0.2"
## [3,] "KNN1" "0.31"  "0.29"
## [4,] "KNN5" "0.22"  "0.22"
```

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

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