Computer Assignment 6 - Classification

Machine Learning, Spring 2020

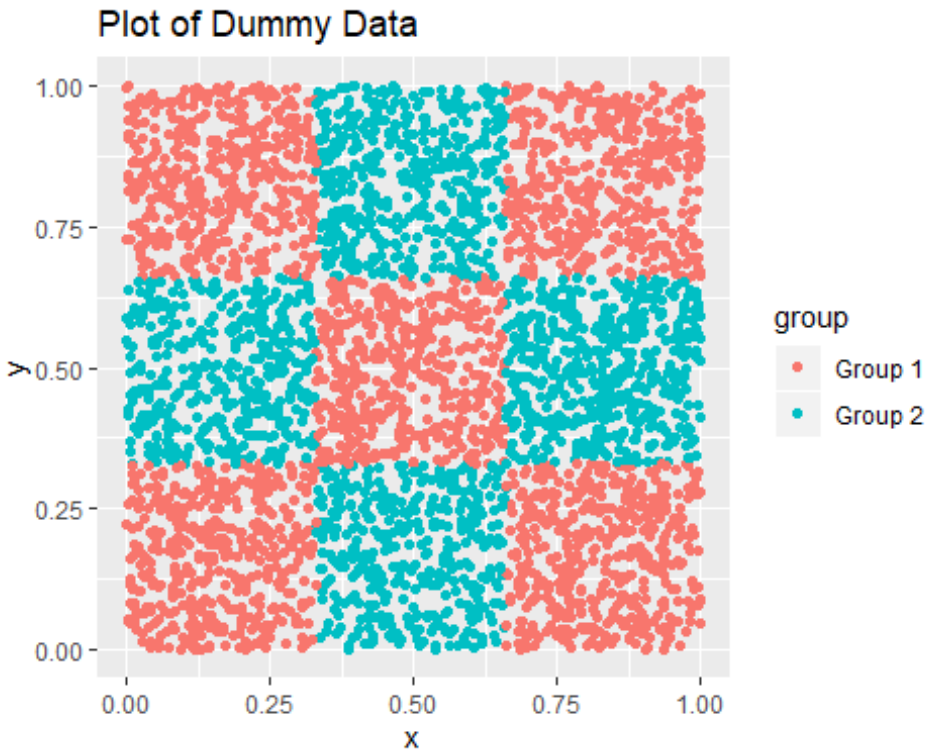
Rui Li

Thinking further about inital cluster centers

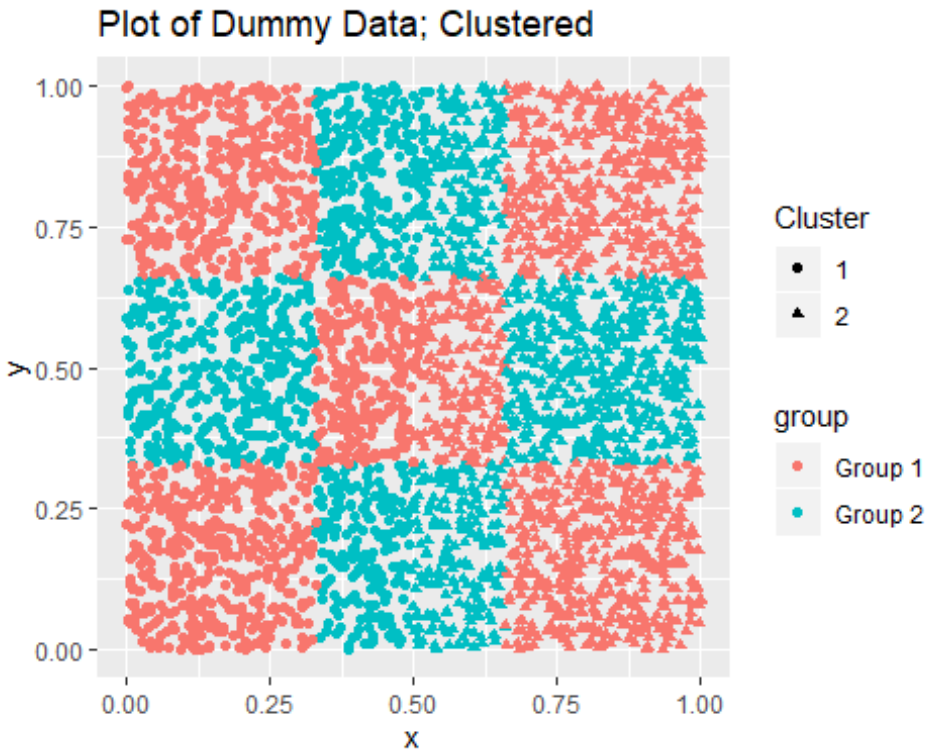
Greetings! The last problem on CA04 was met with some difficulty – many of you were unable to see the total within-cluster sum of squares change with different cluster centers. While it worked for some of you (and, for me, when I wrote the problem), ultimately it seems the kmeans algorithm worked too well even though we restricted it to a single updating step, and it found the optimal clustering regardless. The important punchline is that exsaustively finding the optimal cluster centers is a HARD problem and that we instead find the local maximum. The following is a repeat of the last problem on CA04, except with a new data set (and a set.seed() command for extra security) I created to truly show a change in total within-cluster sum of squares. -Murph

When passing a number x as the argument for `centers` to the kmeans algorithm, x cluster centers are chosen randomly, points are assigned to a cluster based on these centers, and then the cluster centers are iteratively updated in an attempt to find the centers that minimize total within-cluster sum of squares. The number of times `kmeans` updates is controlled by the argument `iter.max`. To examine how the choice of initial centers affects the algorithm, we will disable the updating steps via `iter.max` and choose some initial cluster centers of our own.

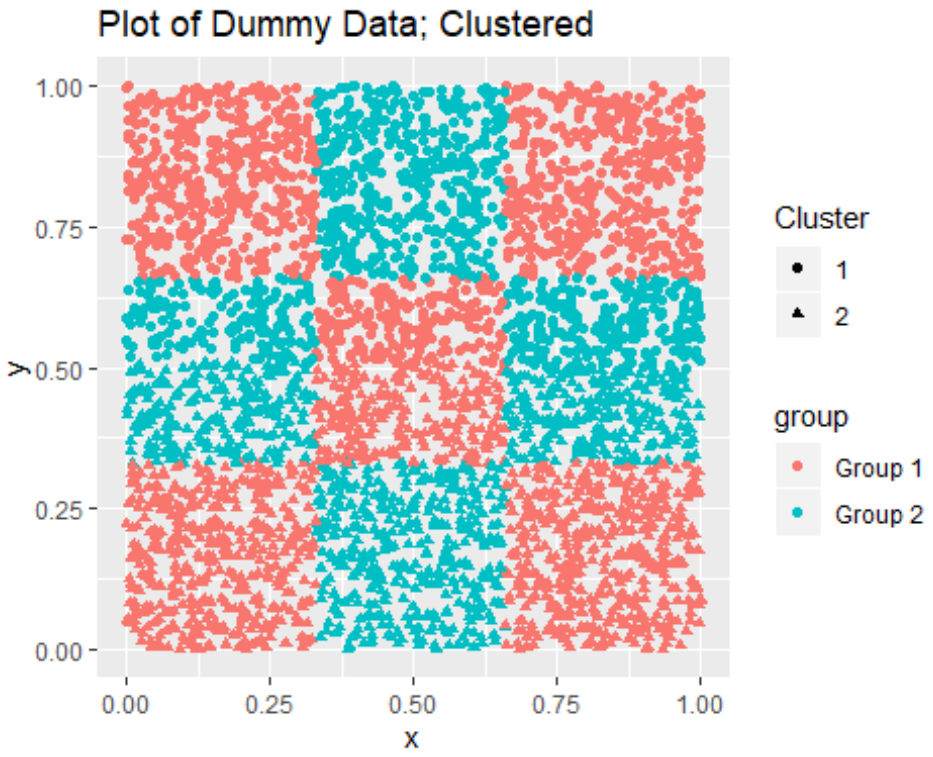
```
library(ggplot2)
# First, we create a dummy data set
set.seed(10)
random_data = data.frame(x = runif(4000),
                         y = runif(4000),
                         group = rep(NA, times = 8000))
random_data[which(random_data$x < 0.33 & random_data$y < 0.33 |
                  random_data$x > 0.66 & random_data$y < 0.33 |
                  random_data$x < 0.33 & random_data$y > 0.66 |
                  random_data$x > 0.66 & random_data$y > 0.66 |
                  (random_data$x > 0.33 & random_data$x < 0.66 &
                     random_data$y > 0.33 & random_data$y < 0.66)),]$group =
"Group 1"
random_data[which(is.na(random_data$group)),]$group = "Group 2"

# This data set looks like:
ggplot(random_data, aes(x = x, y = y, col = group)) + geom_point() +
  ggtitle("Plot of Dummy Data")
```




```
# We'll perform the kmeans with two random centers:
random.km = kmeans(random_data[,1:2], centers = 2, iter.max = 1)
ggplot(random_data, aes(x = x, y = y, col = group, pch = as.factor(random.km
$cluster))) +
  geom_point() +
  scale_shape_discrete(name = "Cluster") +
  ggtitle("Plot of Dummy Data; Clustered")
```



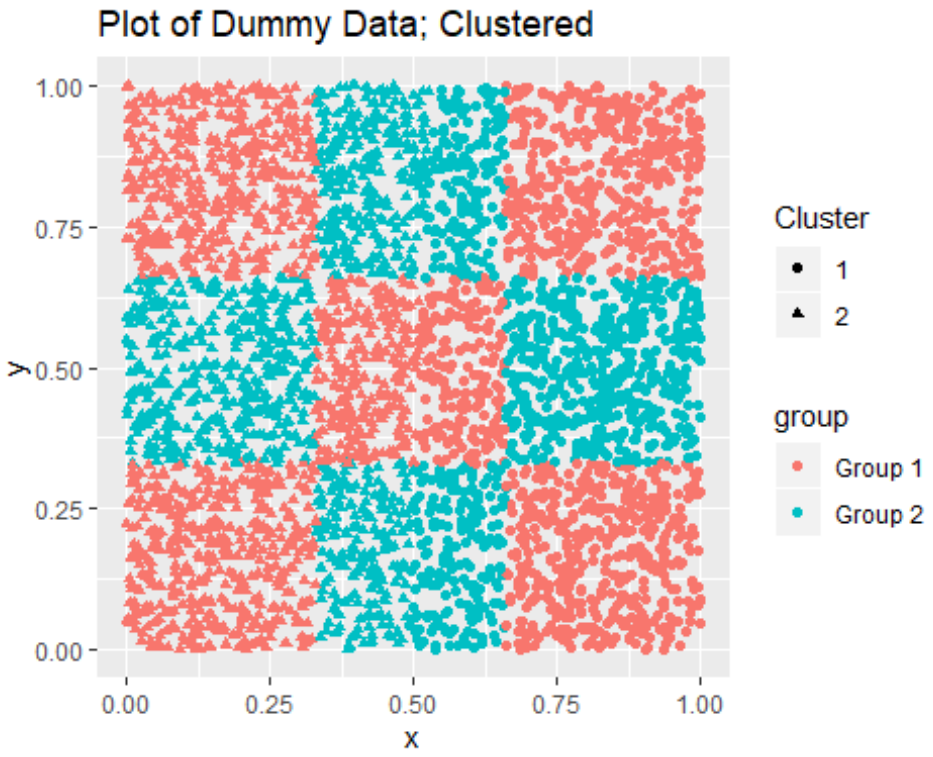

```
# Which gives the following within-cluster sum of squares:
random.km$tot.withinss

## [1] 829.424

my_centers = random_data[1:2, 1:2]
random.km = kmeans(random_data[,1:2], centers = my_centers, iter.max = 1)
ggplot(random_data, aes(x = x, y = y, col = group, pch = as.factor(random.km
$cluster))) +
  geom_point() +
  scale_shape_discrete(name = "Cluster") +
  ggtitle("Plot of Dummy Data; Clustered")
```




```
random.km$tot.withinss

## [1] 847.1953
```

Don't just look at the number!! Can you see how the clusters themselves changed? Now, choose your own cluster centers! These *do not* need to be actual observed points from our data. Try something weird, and report the total within-cluster sum of squares. Did it change? Was it better or worse than the random choice? Comment as to why you think that is.

```
my_centers = random_data[1000:1001, 1:2]
random.km = kmeans(random_data[,1:2], centers = my_centers, iter.max = 1)
ggplot(random_data, aes(x = x, y = y, col = group, pch = as.factor(random.km
$cluster))) +
  geom_point() +
  scale_shape_discrete(name = "Cluster") +
  ggtitle("Plot of Dummy Data; Clustered")
```




```
random.km$tot.withinss
```

```
## [1] 829.424
```

```
The total within-cluster sum of squares is 829.424 which is smaller, showing
that the center choice is better. But according to the graph above, upper hal
f of the points are defined as cluster 1, while the lower half as cluster 2,
which is different to the groups scatter.
```

Bayes Rule and Univariate Normal Simulations

Notice, in the above problem, how we created the `random_data` variable using both the dataframe the `runif` functions. This latter function (`runif`) draws random values from a uniform(0,1) distribution. As mentioned in our first few Computing Assignments, `R` can simulate a number of distributions, including the normal distributions

$$\mathbb{W} \sim \mathcal{N}(-2,1) \quad \& \quad \mathbb{V} \sim \mathcal{N}(2,1).$$

For this next exercise, we are going to simulate 300 observations from $\mathbb{W}$ and 200 observations from $\mathbb{V}$, and create another variable $\mathbb{Y}$ that classifies from which normal distribution each observation came. Indeed,

```
W_obs = rnorm(300, mean = -1)
V_obs = rnorm(200, mean = 1)
Y_class = c(rep(0, times = length(W_obs)), rep(1, times = length(V_obs)))
train_data = data.frame(X = c(W_obs, V_obs), Y = Y_class)
```

Question: Why did we not specify the value for standard deviation in the `rnorm` function?

```
Because if `sd` is not specified, it assumes the default values of 1, which i
s the exact value that we want for W_obs and V_obs.
```

Since we have specified ourselves the model for our data (here: two different normals), we can assess the performance of any classification technique we use on this data. We will illustrate this by calculating the Bayes rule for our `train_data`. To do so, fill in the following quantities

```
pi_0 = sum(train_data$Y==0)/nrow(train_data)
pi_1 = sum(train_data$Y==1)/nrow(train_data)
```

To start, let's just calculate the Bayes' rule for the first observation of `train_data`. To find the conditional probability, you may use the `density` function in `R` to estimate the PDF and get the appropriate probability mass for x. That is, let $P(x|Y = 0)$ be the probability mass of x in the sub-population for which $Y = 0$. The following is an example of how to get a probability mass from an estimated density function using the full `training_data`. For your purposes, you will need to use a subset of the `training_data` instead of the full.

```
# Example of PDF estimation
x_obs = train_data[1,1]
full_density = density(train_data$X)
index_of_density = sum(full_density$x <= x_obs)
pdf_value_of_x_obs = full_density$y[index_of_density]
pdf_value_of_x_obs

x_obs = train_data[1,1]
#prob_x_given_0
given0_density = density(train_data$X[train_data$Y==0])
given0_index = sum(given0_density$x <= x_obs)
prob_x_given_0 = given0_density$y[given0_index]
#prob_x_given_1
given1_density = density(train_data$X[train_data$Y==1])
given1_index = sum(given1_density$x <= x_obs)
prob_x_given_1 = given1_density$y[given1_index]
Bayes_rule_for_x = prob_x_given_1 * pi_1 / (prob_x_given_1 * pi_1 + prob_x_gi
ven_0 * pi_0)
```

What hypothesis does this Bayes Rule test? Based on our calculation, in which distribution should we classify `x_obs`?

```
This Bayes Rule test ~ H0: `x_obs` is more likely belong to the distribution
V; HA: `x_obs` is more likely belong to the distribution W.

Since the value of Bayes_rule_for_x is 0.5564234 > 1/2, `x_obs` should be cla
ssified as distribution V.
```

Now, calculate the Bayes Rule for every value, and use them to compute a classifier for every observation in `train_data`. (DO NOT do this exhaustively. You should be using built-

in features in R and/or a for loop.) Compare these classifiers with the true classifiers. Calculate, and report, the Bayes' Risk.

```
#Construct a new column
train_data$'new_Y' = NA

#Compute Classifiers
for (i in 1:nrow(train_data)) {
  x_obs = train_data[i,1]
  #prob_x_given_0
  given0_density = density(train_data$X[train_data$Y==0])
  given0_index = sum(given0_density$x <= x_obs)
  prob_x_given_0 = given0_density$y[given0_index]
  #prob_x_given_1
  given1_density = density(train_data$X[train_data$Y==1])
  given1_index = sum(given1_density$x <= x_obs)
  prob_x_given_1 = given1_density$y[given1_index]
  Bayes_rule_for_x = prob_x_given_1 * pi_1 / (prob_x_given_1 * pi_1 + prob_x_
given_0 * pi_0)

  #Assign value to classifier
  if(length(Bayes_rule_for_x)==0){
    train_data$`new_Y`[i]=0
  }else if(Bayes_rule_for_x>=1/2){
    train_data$'new_Y'[i]=1
  }else{train_data$'new_Y'[i]=0}
}

#Calculate and Report the Bayes' Risk
Bayes_Risk = sum(train_data$Y!=train_data$new_Y)/nrow(train_data)
Bayes_Risk

## [1] 0.15
```

k-nearest Neighbors and LDA

Using the same train_data from the last exercise, fit a k-nearest neighbors model for $k \in \{1,3,10\}$. The code for $k = 1$ is provided.

```
# Take special care to seperate the classifier from the rest of the data when
 fitting a
# knn model.  Consider the following code to give you an idea as to how one d
oes this.
# Note, however, that this code is based off of my naming practices, and may
require editing
# depending on your previous code.
library(class)
train_data_classifiers = as.factor(train_data$Y)
train_data_observations = data.frame(train_data$X)
```

```
#k=1
knn.1 <-  knn(train_data_observations, train_data_observations, cl = train_da
ta_classifiers, k=1)
R_knn_1 = 100 * sum(train_data_classifiers != knn.1)/length(knn.1)
R_knn_1
```

```
## [1] 0
```

```
#k=3
knn.3 <-  knn(train_data_observations, train_data_observations, cl = train_da
ta_classifiers, k=3)
R_knn_3 = 100 * sum(train_data_classifiers != knn.3)/length(knn.3)
R_knn_3
```

```
## [1] 9.8
```

```
#k=10
knn.10 <-  knn(train_data_observations, train_data_observations, cl = train_d
ata_classifiers, k=10)
R_knn_10 = 100 * sum(train_data_classifiers != knn.10)/length(knn.10)
R_knn_10
```

```
## [1] 13.2
```

Comment on the performance for the different values of k. Why does `k = 1` do so well? What is it doing that gives it such great performance?

```
According to the calculation, the ranking performance is `k=1` > `k=3` > `k=1
0`. The `k=1` does so well is because it uses only the training point closest
 to the query point, the bias of the 1-nearest neighbor estimate is low.
```

Now let's do the same thing with Fisher's Linear Discriminate Analysis (LDA). We have provided the follow code as an example. Assess the risk using the derived predictions (you will need to grab the `class` attribute from this variable).

```
library(MASS)
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

# Fit the model
model <- lda(Y~X, data = train_data)
# Make predictions
predictions <- model %>% predict(train_data)

risk_predictions = sum(predictions$class!=train_data$Y)/length(train_data)
risk_predictions

## [1] 25.66667
```

Method Evaluation

Now that we have explored Bayes' Rule, k-nearest neighbors, and LDA, we will see how each method performs on data that was NOT used to originally set them up. Consider the following new data set drawn from the same random variables $\mathbb{W}$ & $\mathbb{V}$.

```
W_obs = rnorm(150, mean = -2)
V_obs = rnorm(50, mean = 2)
Y_class_test = c(rep(0, times = length(W_obs)), rep(1, times = length(V_obs)))
test_data = data.frame(X = c(W_obs, V_obs), Y = Y_class_test)
```

Use the information from `train_data` to classify values in the `test_data`, then compare these calculated classes with the true classes found in `Y_class_test`. Report the Bayes' Risk, and compare it to the same metric calculated from k-nearest neighbors and LDA.

Hints:

1. In the case of Bayes rule, you will use the same π_0 & π_1 and calculate the conditional probabilities in the same way, except this time your `x_obs` will be from `test_data`. DO NOT use the class labels from `test_data` ANYWHERE in this calculation (until you evaluate at the end).

2. Use the manual page `?knn` to see how one inputs a different dataset for the "test" parameter. You may have to separate the observations X from the class labels Y.

3. In a similar fashion, you are expected to read the manual pages for the functions used in the LDA process.

```
##Calculate the Bayes Rule and Bayes Risk
#Construct a new column
test_data$'new_Y' = NA

#Compute Classifiers
for (i in 1:nrow(test_data)) {
  x_obs = test_data[i,1]
  #prob_x_given_0
```

```r
  given0_density = density(train_data$X[train_data$Y==0])
  given0_index = sum(given0_density$x <= x_obs)
  prob_x_given_0 = given0_density$y[given0_index]
  #prob_x_given_1
  given1_density = density(train_data$X[train_data$Y==1])
  given1_index = sum(given1_density$x <= x_obs)
  prob_x_given_1 = given1_density$y[given1_index]
  Bayes_rule_for_x = prob_x_given_1 * pi_1 / (prob_x_given_1 * pi_1 + prob_x_
given_0 * pi_0)

  #Assign value to classifier
  if(length(Bayes_rule_for_x)==0){
    test_data$`new_Y`[i]=0
  }else if(Bayes_rule_for_x>=1/2){
    test_data$'new_Y'[i]=1
  }else{test_data$'new_Y'[i]=0}
}

#Calculate and Report the Bayes' Risk
Bayes_Risk = sum(test_data$Y!=test_data$new_Y)/nrow(test_data)

#knn risk
library(class)
train_data_classifiers = as.factor(train_data$Y)
train_data_observations = data.frame(train_data$X)
test_data_observations = data.frame(test_data$X)
test_data_classifiers = as.factor(test_data$Y)

#k=1
knn.1 <-  knn(train_data_observations, test_data_observations, cl = train_dat
a_classifiers, k=1)
R_knn_1 = 100 * sum(test_data_classifiers != knn.1)/length(knn.1)

#k=3
knn.3 <-  knn(train_data_observations, test_data_observations, cl = train_dat
a_classifiers, k=3)
R_knn_3 = 100 * sum(test_data_classifiers != knn.3)/length(knn.3)

#k=10
knn.10 <-  knn(train_data_observations, test_data_observations, cl = train_da
ta_classifiers, k=10)
R_knn_10 = 100 * sum(test_data_classifiers != knn.10)/length(knn.10)

#LDA
library(MASS)
library(dplyr)
# Fit the model
model <- lda(Y~X, data = train_data)
# Make predictions
predictions <- model %>% predict(test_data)
```

```
risk_predictions = sum(predictions$class!=test_data$Y)/length(test_data$X)
```

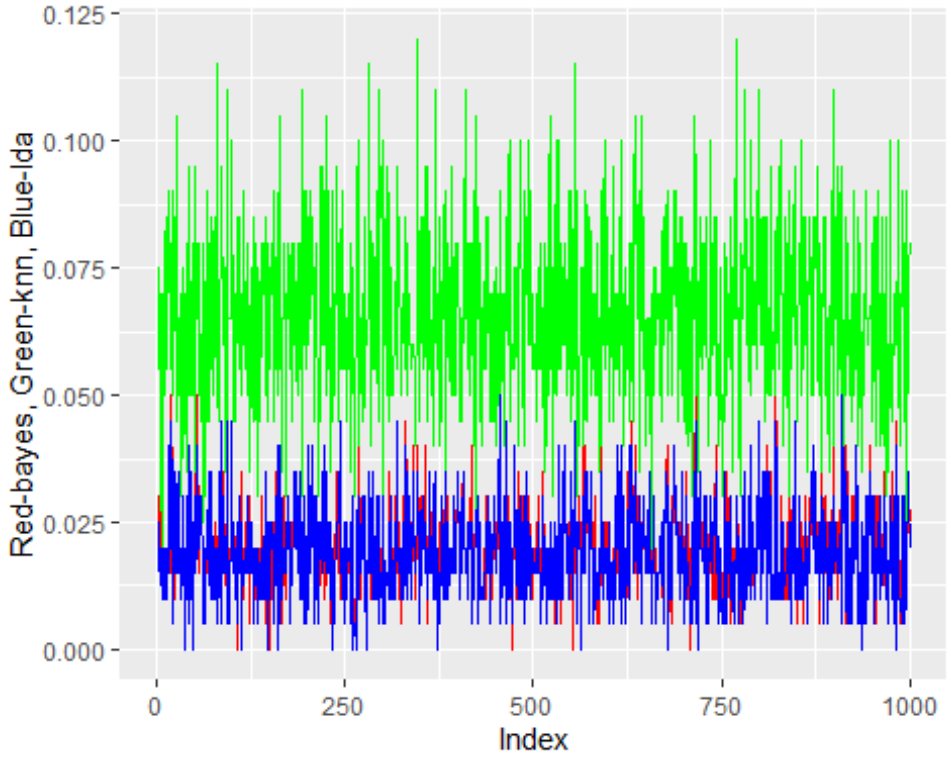
Now, let's do this 1000 more times! During each iteration of the following for loop, use the models you have created to calculate classifiers for the `test_data` and calculate the risk for each.

```
set.seed(13)
all_bayes_risks = c()
all_knn_risks = c()
all_lda_risks = c()

for(iteration in 1:1000){
  W_obs = rnorm(150, mean = -2)
  V_obs = rnorm(50, mean = 2)
  Y_class_test = c(rep(0, times = length(W_obs)), rep(1, times = length(V_ob
s)))
  test_data = data.frame(X = c(W_obs, V_obs), Y = Y_class_test)

##Calculate the Bayes Rule and Bayes Risk
#Construct a new column
test_data$'new_Y' = NA

#Compute Classifiers
for (i in 1:nrow(test_data)) {
  x_obs = test_data[i,1]
  #prob_x_given_0
  given0_density = density(train_data$X[train_data$Y==0])
  given0_index = sum(given0_density$x <= x_obs)
  prob_x_given_0 = given0_density$y[given0_index]
  #prob_x_given_1
  given1_density = density(train_data$X[train_data$Y==1])
  given1_index = sum(given1_density$x <= x_obs)
  prob_x_given_1 = given1_density$y[given1_index]
  Bayes_rule_for_x = prob_x_given_1 * pi_1 / (prob_x_given_1 * pi_1 + prob_x_
given_0 * pi_0)

  #Assign value to classifier
  if(length(Bayes_rule_for_x)==0){
    test_data$`new_Y`[i]=0
  }else if(Bayes_rule_for_x>=1/2){
    test_data$'new_Y'[i]=1
  }else{test_data$'new_Y'[i]=0}
}

#knn risk
library(class)
train_data_classifiers = as.factor(train_data$Y)
train_data_observations = data.frame(train_data$X)
```

```
test_data_observations = data.frame(test_data$X)
test_data_classifiers = as.factor(test_data$Y)

#k=1
knn.1 <-  knn(train_data_observations, test_data_observations, cl = train_dat
a_classifiers, k=1)

#LDA
library(MASS)
library(dplyr)
# Fit the model
model <- lda(Y~X, data = train_data)
# Make predictions
predictions <- model %>% predict(test_data)

  bayes_risk = sum(test_data$Y!=test_data$new_Y)/nrow(test_data)
  knn_risk = sum(test_data_classifiers != knn.1)/length(knn.1)
  lda_risk = sum(predictions$class!=test_data$Y)/length(test_data$X)

  all_bayes_risks = c(all_bayes_risks, bayes_risk)
  all_knn_risks = c(all_knn_risks, knn_risk)
  all_lda_risks = c(all_lda_risks, lda_risk)

}
```

If done correctly, you should have three vectors of risk values, each from a different classification method. Create an intuitive plot that compares these three values. Make sure this plot compares the values AT THE SAME INTERATION. Our suggestion would be a line plot with the x-axis as the iteration number and the y-axis as the risk value, colored by classification method.

```
df = as.data.frame(cbind(as.matrix(c(1:1000)),as.matrix(all_bayes_risks),as.m
atrix(all_knn_risks),as.matrix(all_lda_risks)))

ggplot(df, aes(V1)) +                    # basic graphical object
  geom_line(aes(y=V2), colour="red") +   # first layer
  geom_line(aes(y=V3), colour="green")+  # second layer
  geom_line(aes(y=V4), colour="blue") +  # Final layer
  ylab("Red-bayes, Green-knn, Blue-lda")+xlab("Index")
```



Comment on your model

```
From the graph above, we can notice that the risks generated by lda and bayes
 rule are largely the same and are less than the risk generated by knn.
```