Computer Assignment 6 - Classification

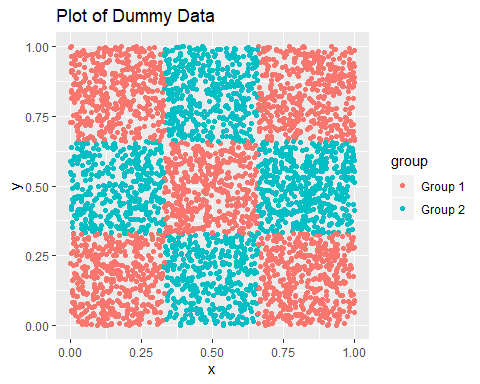
Rui Li

# Thinking further about inital cluster centers

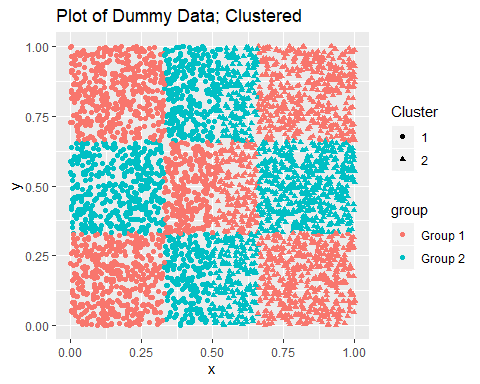
Greetings! The last problem on CA04 was met with some difficulty – many of you were unable to see the total within-cluster sum of squares change with different cluster centers. While it worked for some of you (and, for me, when I wrote the problem), ultimately it seems the kmeans algorithm worked too 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 as the argument for centers to the kmeans algorithm, 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")



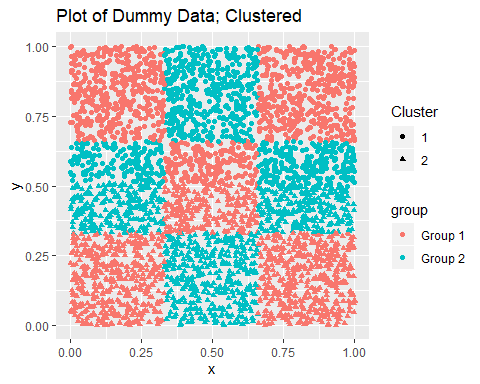
# We'll perform the kmeans with two random centers:  
random.km = kmeans(random\_data[,1:2], centers = 2, iter.max = 1)  
ggplot(random\_data, aes(x = x, y = y, col = group, pch = as.factor(random.km$cluster))) +   
 geom\_point() +   
 scale\_shape\_discrete(name = "Cluster") +   
 ggtitle("Plot of 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")

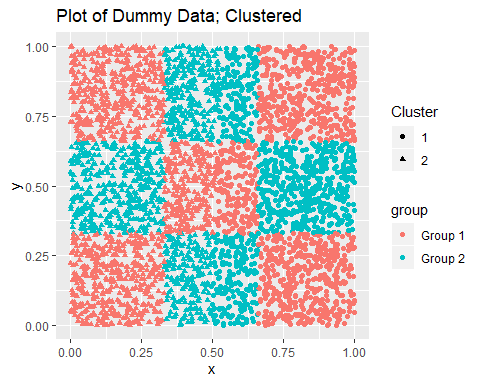


random.km$tot.withinss

## [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 half 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 distribution. As mentioned in our first few Computing Assignments, R can simulate a number of distributions, including the normal distributions

For this next exercise, we are going to simulate 300 observations from and 200 observations from , and create another variable 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 is 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 . That is, let be the probability mass of in the sub-population for which . 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\_given\_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 classified 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 -nearest neighbors model for . The code for 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 does 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\_data\_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\_data\_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\_data\_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 . 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=10`. 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 .

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 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 from the class labels .
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  
 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\_data\_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\_data\_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\_data\_classifiers, k=10)  
R\_knn\_10 = 100 \* sum(test\_data\_classifiers != knn.10)/length(knn.10)

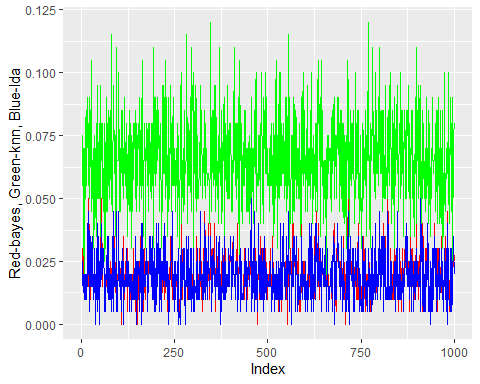
#LDA  
library(MASS)  
library(dplyr)  
# Fit the model  
model <- lda(Y~X, data = train\_data)  
# Make predictions  
predictions <- model %>% predict(test\_data)  
  
risk\_predictions = sum(predictions$class!=test\_data$Y)/length(test\_data$X)

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\_obs)))  
 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\_data\_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 -axis as the iteration number and the -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.matrix(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.