Computer Assignment 9 - Model Assumptions and Cross Validation

Machine Learning, Spring 2020

Rui Li

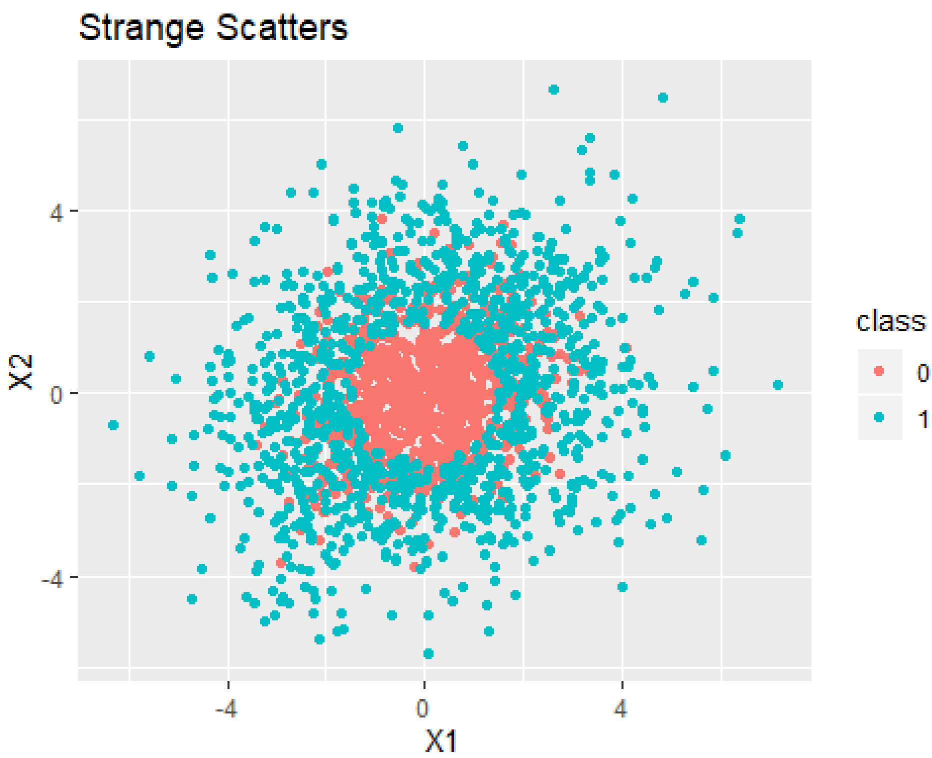
Not-So-Perfect Data for LDA

Recall on the previous CA we performed LDA on a data set that was generated from the LDA model. For that data set the two groups (classified 0 and 1) were generated from two multinormal distributions with the same variance but different means. LDA performed very well on that data because *the data were generated from the underlying LDA model* that we were trying to fit.

In this assignment, we will explore what happens when we apply our methods to a dataset that is generated from a distribution that is *very different* from a LDA model. We will start in two dimensions. Note that while we are still using the multinormal function to generate the following data, the variance of the marginals is different, the means are the same, and we manipulate `data_1` in such a way that it is no longer normal. Indeed,

```
# Make sure to install this package if you do not have it:
library(mvtnorm)
my_sigma_1 = t(matrix(c(2,0.2,0.2,2)*0.6, ncol=2)) %*% matrix(c(2,0.2,0.2,2)*
0.6, ncol=2)
my_sigma_2 = t(matrix(c(2,0.2,0.2,2), ncol=2)) %*% matrix(c(2,0.2,0.2,2), nco
l=2)
data_0 = rmvnorm(1000, mean = rep(0, times = 2), sigma = my_sigma_1)
data_1 = rmvnorm(1500, mean = rep(0, times = 2), sigma = my_sigma_2)
central_points = ( sqrt( data_1[,1]**2 + data_1[,2]**2) < 1.5 )
data_1 = data_1[!central_points,]

library(ggplot2)
temp_data = data.frame(rbind(data_0, data_1))
temp_data$class = as.factor(c(rep(0, times = 1000), rep(1, times = nrow(data_
1))))
ggplot(temp_data, aes(x = X1, y = X2, color = class)) + geom_point() +
  ggtitle("Strange Scatters")
```



Questions

1. The way these data were created breaks *two* of the assumptions for LDA. What are they? Do you predict LDA or KNN will have better performance on this data? Defend your choice.

```
These data breaks two assumptions. First of all, the independent variable of
group 1 is not normal. Also, the variance of two groups are different.

Therefore LDA would not perform well on this data. However, KNN will have bet
ter performance, because the k-nearest neighbor rules does not follow any ass
umption. What KNN does is to classify an object by a plurality vote of its ne
ighbors, with the object being assigned to the class most common among its k
nearest neighbors.
```

2. If you were to draw a decision boundary on this data, what shape would it be? Why might this not be a good thing for an LDA model?

```
The decision boundary would be an oval-shaped, quadric surface. Since the dec
ision boundary of an LDA model shold be a hyperplane, thus this shape is not
a good thing for an LDA model.
```

3. Split the data into a testing data set and a training data set. Build an LDA model on the training data. Use the training data again to predict the class labels and report the performance of your model. What is the error rate? (the proportion of times your model classified an observation incorrectly)

```r
library(MASS)
library(dplyr)
#Split the data set to testing data and training data
training_size <- round(.75 * nrow(temp_data))  # training set size
indices = sample(1:nrow(temp_data), training_size)
training_set <- temp_data[indices,]
testing_set <- temp_data[-(indices),]
#Build an LDA model
train_lda = lda(class~., data = training_set)
train_predictions <- train_lda %>% predict(training_set)
train_error = sum(train_predictions$class!=training_set$class)/length(trainin
g_set$class)
#Print the result
print(paste0("The error rate is ", train_error, "."))

## [1] "The error rate is 0.468710888610763."
```

4. Now predict the class labels using the testing data and report the performance of your model. What is the error rate?

```r
#Build an LDA model
test_predictions <- train_lda %>% predict(testing_set)
test_error = sum(test_predictions$class!=testing_set$class)/length(testing_se
t$class)
#Print the result
print(paste0("The error rate is ", test_error, "."))

## [1] "The error rate is 0.479323308270677."
```

5. Repeat steps 2-3 using a knn model for $k \in \{1,5,11\}$. Considering only the cases where you predicted labels on your testing data, compare the error rates between the LDA model and all of the knn models. Was your prediction in question 1 correct or incorrect?

```r
#knn risk
library(class)
train_data_classifiers = as.factor(training_set$class)
train_data_observations = training_set[,-3]
test_data_observations = testing_set[,-3]
test_data_classifiers = as.factor(testing_set$class)

#k=1
knn.1 <-  knn(train_data_observations, test_data_observations, cl = train_dat
a_classifiers, k=1)
knn.1_risk = sum(test_data_classifiers != knn.1)/length(knn.1)
print(paste0("The risk of knn.1 is ", knn.1_risk, "."))

## [1] "The risk of knn.1 is 0.257518796992481."

#k=5
knn.5 <-  knn(train_data_observations, test_data_observations, cl = train_dat
a_classifiers, k=5)
```

```
knn.5_risk = sum(test_data_classifiers != knn.5)/length(knn.5)
print(paste0("The risk of knn.5 is ", knn.5_risk, "."))

## [1] "The risk of knn.5 is 0.223684210526316."

#k=11
knn.11 <-  knn(train_data_observations, test_data_observations, cl = train_da
ta_classifiers, k=11)
knn.11_risk = sum(test_data_classifiers != knn.11)/length(knn.11)
print(paste0("The risk of knn.11 is ", knn.11_risk, "."))

## [1] "The risk of knn.11 is 0.214285714285714."

The average risk rate of the three knn models is 0.25, which is much smaller
than the risk rate of testing set 0.48, showing that knn has a better perform
ance on this data and my prediction on question 1 was correct.
```

Error Rate for Increasing Data Sizes

We will now examine how the error rate stabalizes for larger and larger data sizes. Manipulate the variable `data_size` in the following code to simulate different sizes of the data we created for the last problem.

```
data_size = 10
my_sigma_1 = t(matrix(c(2,0.2,0.2,2)*0.6, ncol=2)) %*% matrix(c(2,0.2,0.2,2)*
0.6, ncol=2)
my_sigma_2 = t(matrix(c(2,0.2,0.2,2), ncol=2)) %*% matrix(c(2,0.2,0.2,2), nco
l=2)
data_0 = data.frame(rmvnorm(data_size, mean = rep(0, times = 2), sigma = my_s
igma_1))
data_1 = data.frame(X1 = NA, X2 = NA)
count = 0
while(count < data_size){
  new_draw = rmvnorm(1, mean = rep(0, times = 2), sigma = my_sigma_2)
  if( sqrt( new_draw[,1]**2 + new_draw[,2]**2) >= 1.5 ) {
    data_1 = rbind(data_1, data.frame(new_draw))
    count = count + 1
  }
}
data_1 = data_1[-1,]
my_data = data.frame(rbind(data_0, data_1))
my_data$class = as.factor(c(rep(0, times = data_size), rep(1, times = data_si
ze)))
```

For `data_size` $\in \{5,10,25,50,100,200,500,1000,10000\}$ do the following:

- Generate the “Strange Scatters” data using the given code.
- Split the data into testing and training data sets.

- Build an LDA model and the KNN models for $k \in \{1,5,11\}$ on the training data, predict the labels on the training data, and calculate the training error rate.
- Build an LDA model and the KNN models for $k \in \{1,5,11\}$ on the training data, predict the labels on the testing data, and calculate the testing error rate.
- Save both error rates

```
#Define the data size vector
size = c(5,10,25,50,100,200,500,1000,10000)

#Initiate the error rate of training and testing
lda_train = c(); k.1_train = c(); k.5_train = c(); k.11_train = c()
lda_test = c(); k.1_test = c(); k.5_test = c(); k.11_test = c()

#Loop the data in vector size
for (data_size in size) {
  my_sigma_1 = t(matrix(c(2,0.2,0.2,2)*0.6, ncol=2)) %*% matrix(c(2,0.2,0.2,2)
*0.6, ncol=2)
  my_sigma_2 = t(matrix(c(2,0.2,0.2,2), ncol=2)) %*% matrix(c(2,0.2,0.2,2), n
col=2)
  data_0 = data.frame(rmvnorm(data_size, mean = rep(0, times = 2), sigma = my
_sigma_1))
  data_1 = data.frame(X1 = NA, X2 = NA)
  count = 0
  while(count < data_size){
    new_draw = rmvnorm(1, mean = rep(0, times = 2), sigma = my_sigma_2)
    if( sqrt( new_draw[,1]**2 + new_draw[,2]**2) >= 1.5 ) {
      data_1 = rbind(data_1, data.frame(new_draw))
      count = count + 1
    }
  }
  data_1 = data_1[-1,]
  my_data = data.frame(rbind(data_0, data_1))
  my_data$class = as.factor(c(rep(0, times = data_size), rep(1, times = data_
size)))

  # Generate the "Strange Scatters" data using the given code
  plot <- ggplot(my_data, aes(x = X1, y = X2, color = class)) + geom_point()
+
  ggtitle(paste0("Strange Scatters of data size ", data_size,"."))
  print(plot)

  # Split the data into testing and training data sets
  library(MASS); library(dplyr)
  training_size <- round(.75 * nrow(my_data))  # training set size
  indices = sample(1:nrow(my_data), training_size)
  training_set <- my_data[indices,]
  testing_set <- my_data[-(indices),]

  # Build an LDA model and the KNN models for k=1,5,11 on the training data,
```

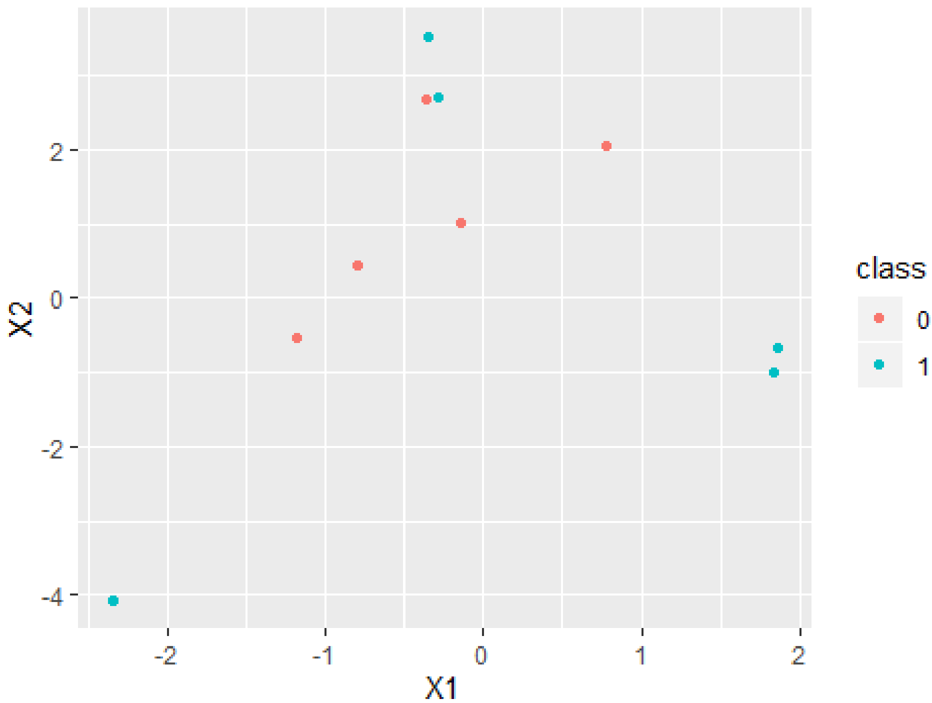
```
predict the labels on the training/testing data, and calculate the training/t
esting error rate
  ## Build an LDA model
  train_lda = lda(class~., data = training_set)
  train_predictions <- train_lda %>% predict(training_set)
  train_error = sum(train_predictions$class!=training_set$class)/length(train
ing_set$class)
  lda_train = c(lda_train,train_error)

  test_predictions <- train_lda %>% predict(testing_set)
  test_error = sum(test_predictions$class!=testing_set$class)/length(testing_
set$class)
  lda_test = c(lda_test,test_error)
  ##knn risk
  library(class)
  train_data_classifiers = as.factor(training_set$class)
  train_data_observations = training_set[,-3]
  test_data_observations = testing_set[,-3]
  test_data_classifiers = as.factor(testing_set$class)
  ###Train k=1
  knn.1 <-  knn(train_data_observations, train_data_observations, cl = train_
data_classifiers, k=1)
  knn.1_risk = sum(train_data_classifiers != knn.1)/length(knn.1)
  k.1_train = c(k.1_train,knn.1_risk)
  ###Train k=5
  knn.5 <-  knn(train_data_observations, train_data_observations, cl = train_
data_classifiers, k=5)
  knn.5_risk = sum(train_data_classifiers != knn.5)/length(knn.5)
  k.5_train = c(k.5_train,knn.5_risk)
  ###Train k=1
  knn.11 <-  knn(train_data_observations, train_data_observations, cl = train
_data_classifiers, k=11)
  knn.11_risk = sum(train_data_classifiers != knn.11)/length(knn.11)
  k.11_train = c(k.11_train,knn.11_risk)
  ###Test k=1
  knn.1 <-  knn(train_data_observations, test_data_observations, cl = train_d
ata_classifiers, k=1)
  knn.1_risk = sum(test_data_classifiers != knn.1)/length(knn.1)
  k.1_test = c(k.1_test,knn.1_risk)
  ###Test k=5
  knn.5 <-  knn(train_data_observations, test_data_observations, cl = train_d
ata_classifiers, k=5)
  knn.5_risk = sum(test_data_classifiers != knn.5)/length(knn.5)
  k.5_test = c(k.5_test,knn.5_risk)
  ###Test k=11
  knn.11 <-  knn(train_data_observations, test_data_observations, cl = train_
data_classifiers, k=11)
  knn.11_risk = sum(test_data_classifiers != knn.11)/length(knn.11)
  k.11_test = c(k.11_test,knn.11_risk)
}
```

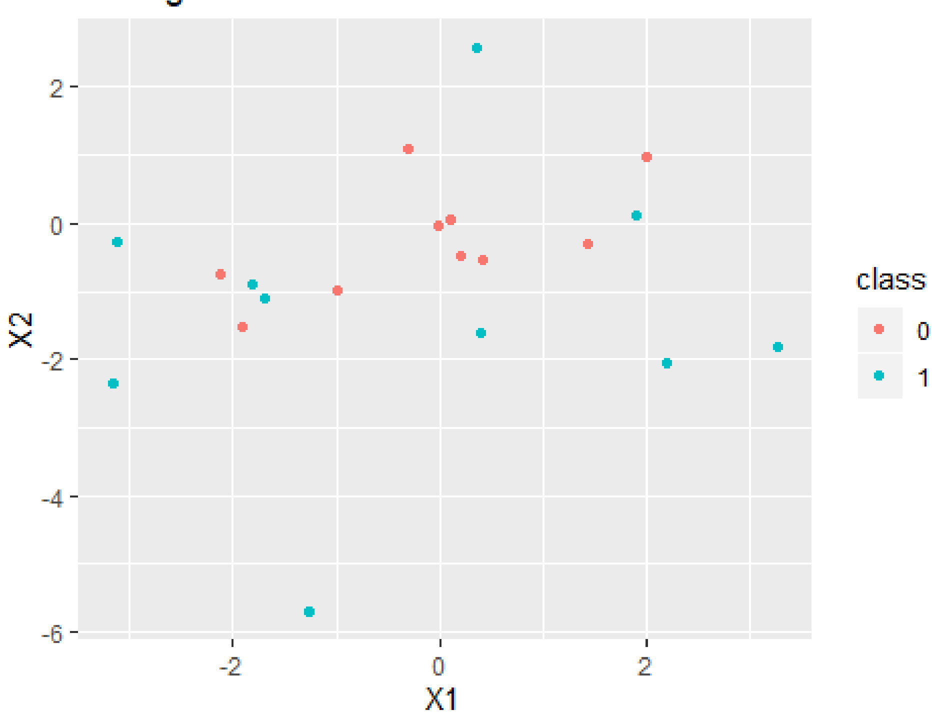
```
## Warning in knn(train_data_observations, train_data_observations, cl = trai
n_data_classifiers, : k =
## 11 exceeds number 8 of patterns
```

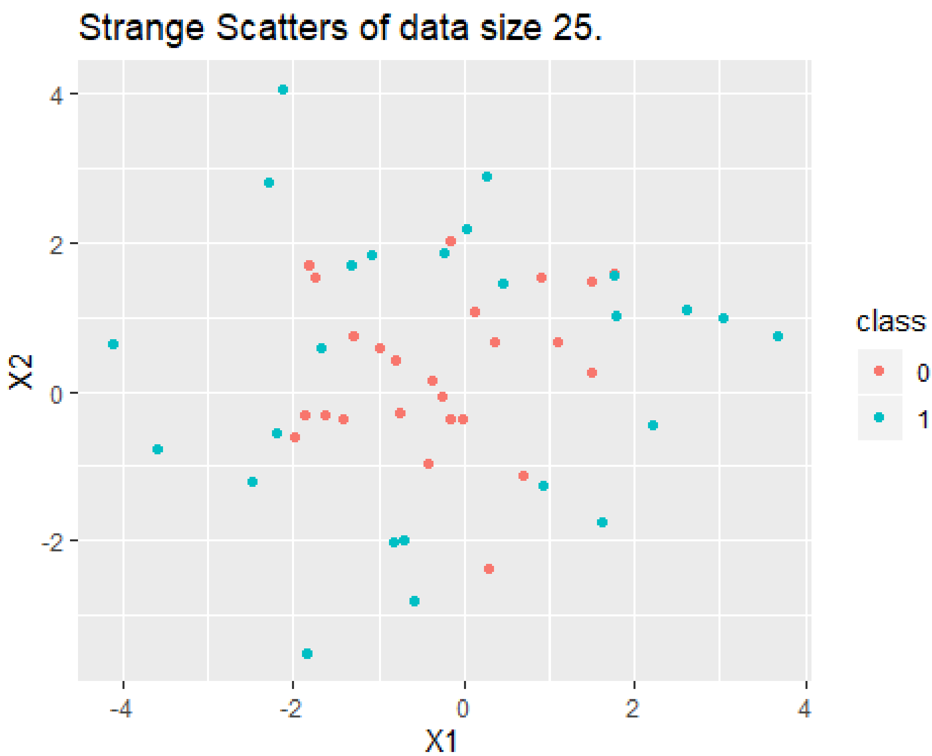
```
## Warning in knn(train_data_observations, test_data_observations, cl = train
_data_classifiers, : k =
## 11 exceeds number 8 of patterns
```

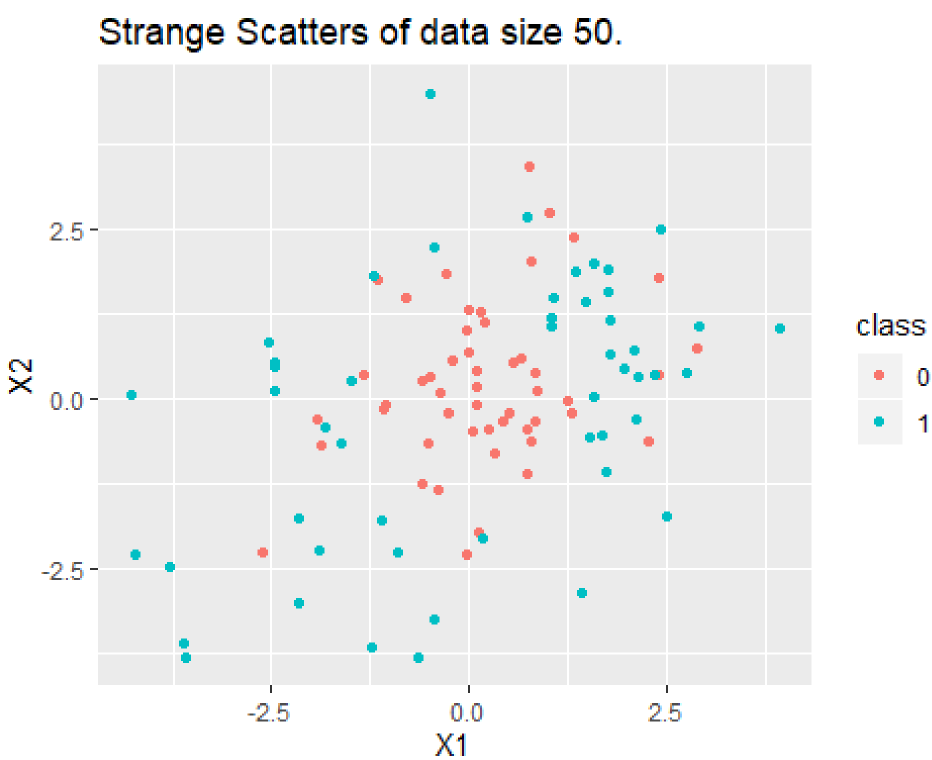
Strange Scatters of data size 5.



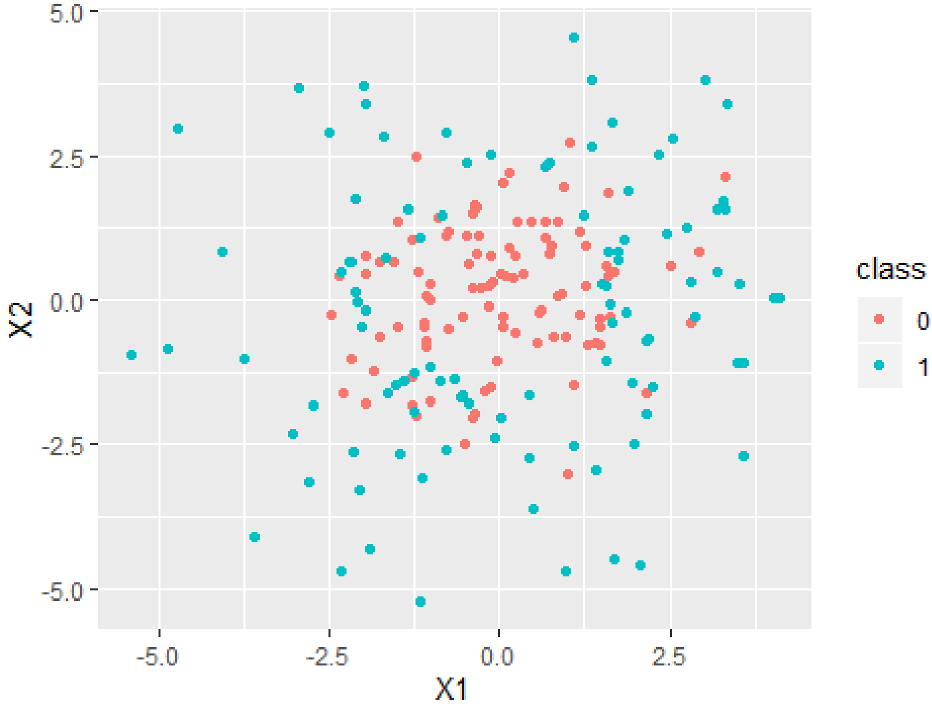
Strange Scatters of data size 10.



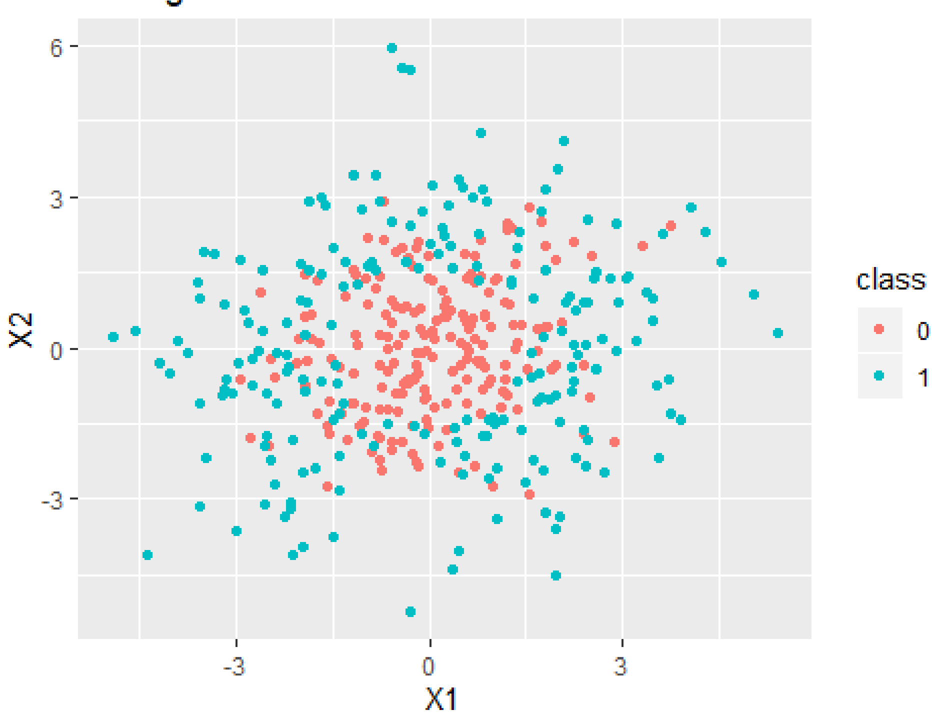

Strange Scatters of data size 25.
X2
X1
class
0
1


Strange Scatters of data size 50.
X2
X1
class
0
1

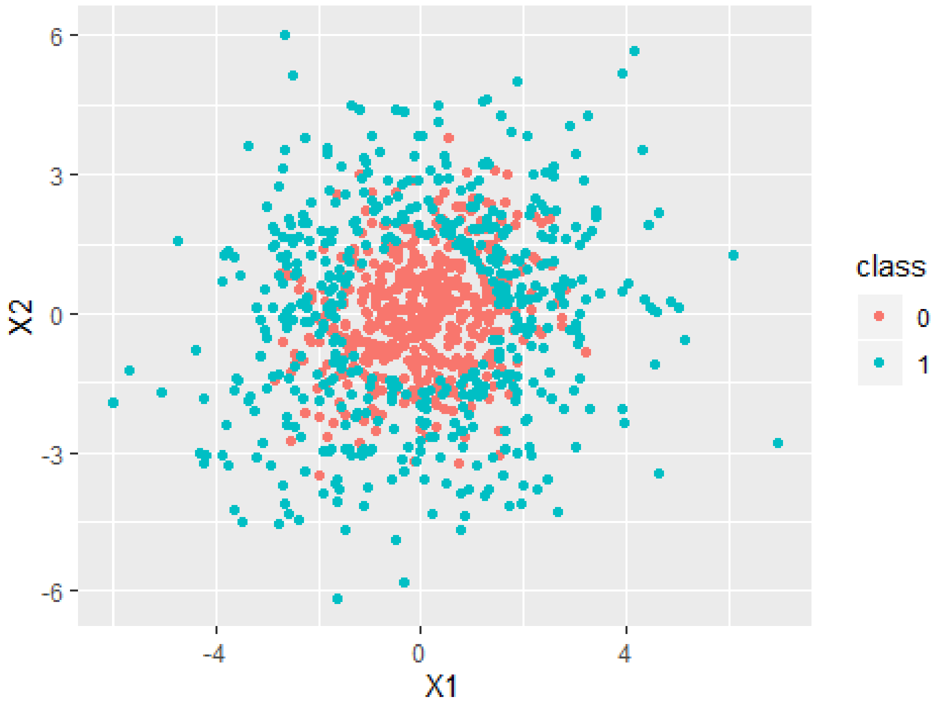
Strange Scatters of data size 100.



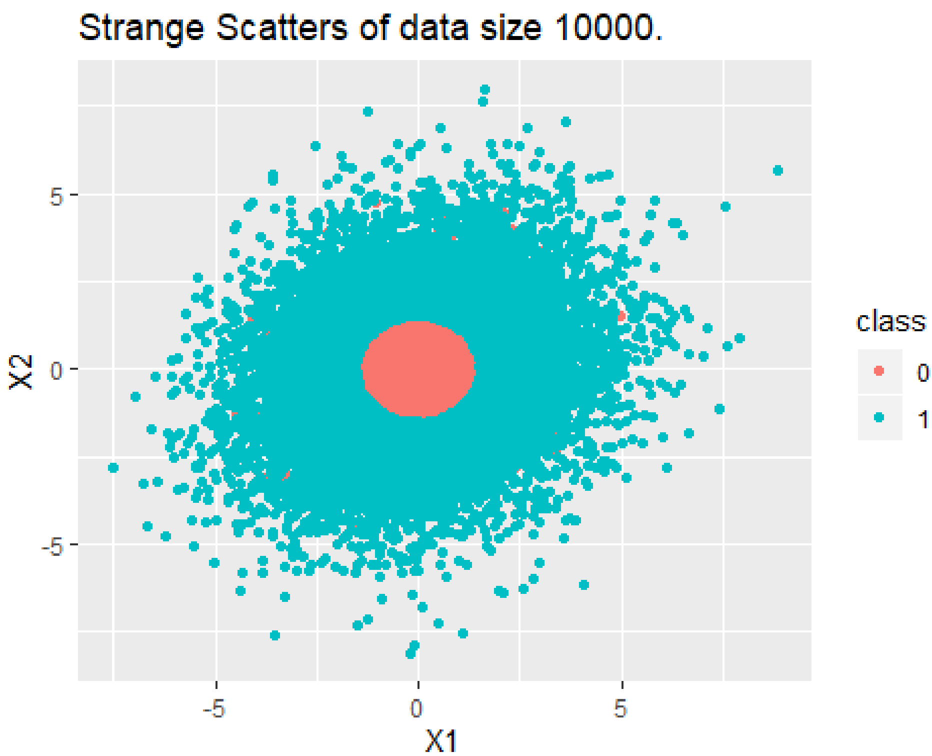
Strange Scatters of data size 200.



Strange Scatters of data size 500.



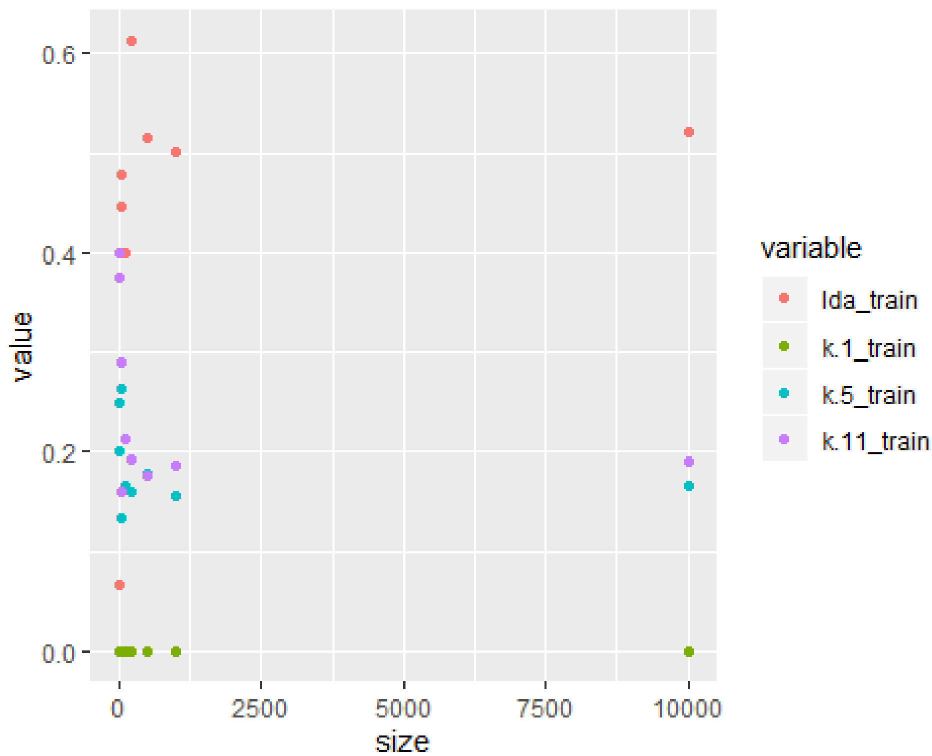
Strange Scatters of data size 1000.



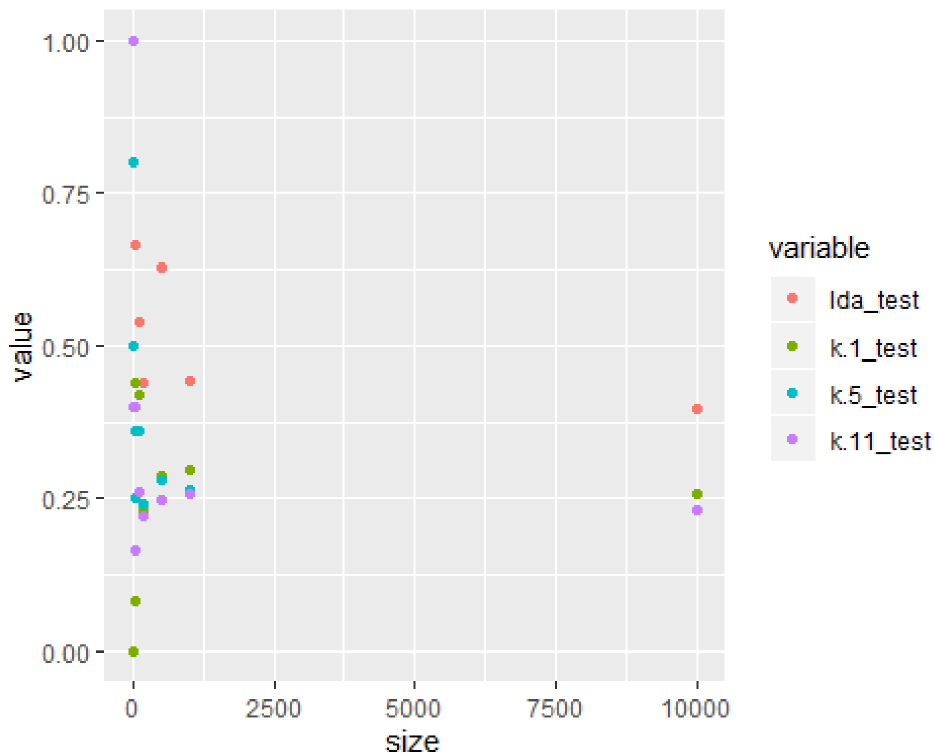
Plot two scatterplots: one for each error type with the data sizes on the x-axis, the error rate on the y-axis, and color by which model type you used.

Do you notice any trends as your data size increases? Do the error rates "stabilize"?

```
knitr::opts_chunk$set(error = TRUE)
library(ggplot2)
library(reshape)
#Plot train error rate
df_train <- data.frame(size, lda_train, k.1_train, k.5_train, k.11_train)
df_train.melted <- melt(df_train, id = "size")
ggplot(data = df_train.melted, aes(x = size, y = value, color = variable)) +
  geom_point()
```




```
#Plot test error rate
df_test <- data.frame(size, lda_test, k.1_test, k.5_test, k.11_test)
df_test.melted <- melt(df, id = "size")
ggplot(data = df_test.melted, aes(x = size, y = value, color = variable)) +
  geom_point()
```




```
From the two plots, we can see there is a clear trend in test error rate when
 data size increases. Each model stablilize the error rate between 0.25 to 0.
40. The train error rate plot also has the same trend but is not that obvious
 compared to the test one.
```

Now compare the testing error rates to the training error rates using boxplots. The modeled used should be on the x-axis, the value of the error rate should on the y-axis, and you should color by whether the error was calculated on the training or the testing data. Comment on any differences between the testing and the training errors.

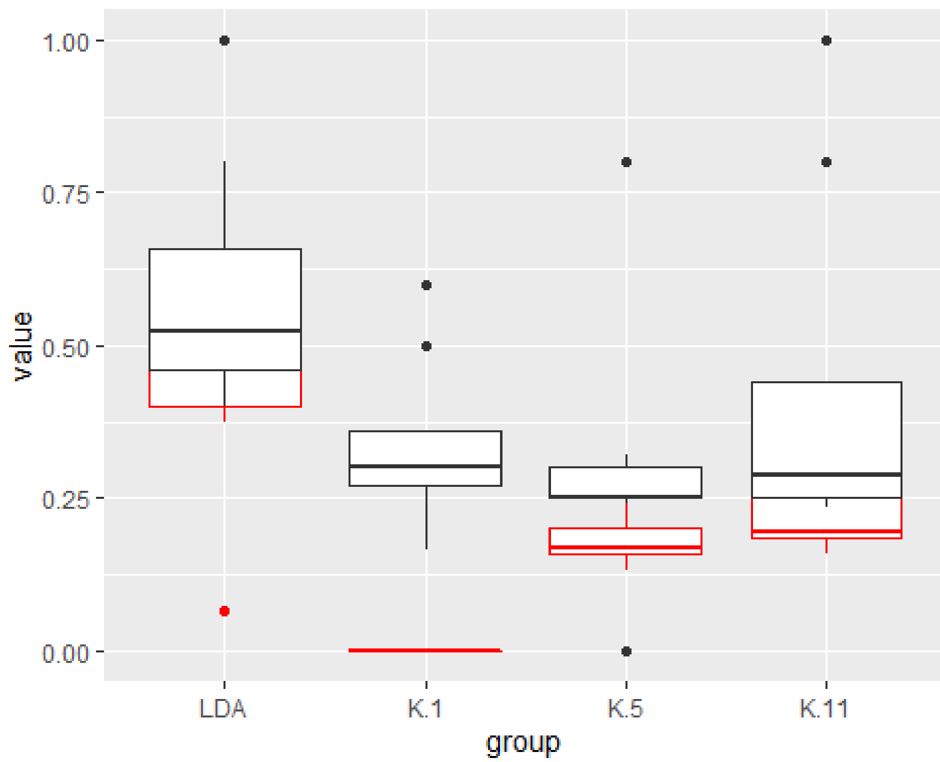
```
knitr::opts_chunk$set(error = TRUE)
a.ta = data.frame(group = "LDA", value = lda_train)
b.ta = data.frame(group = "K.1", value = k.1_train)
c.ta = data.frame(group = "K.5", value = k.5_train)
d.ta = data.frame(group = "K.11", value = k.11_train)

a.te = data.frame(group = "LDA", value = lda_test)
b.te = data.frame(group = "K.1", value = k.1_test)
c.te = data.frame(group = "K.5", value = k.5_test)
d.te = data.frame(group = "K.11", value = k.11_test)

plot.data.ta = rbind(a.ta, b.ta, c.ta, d.ta)
plot.data.te = rbind(a.te, b.te, c.te, d.te)

p <-ggplot(plot.data.ta, mapping = aes(x=group, y=value)) +
```

```
  geom_boxplot(col="red")
p + geom_boxplot(plot.data.te, mapping = aes(x=group, y=value))
```




```
From the boxplot above, red is for the train error rate, while black is for t
he test error rate. We can notice that generally, the test error rate is more
 stable and a little bit higher than then the train error rate.
```

Introduction to Cross Validation

Use the data generating procedure from the previous section to generate a dataset of size 1000. Perform a 10-fold cross validation based off of the procedure given in the lecture slides. You are **not** permitted to use `R`'s built in functions to perform this cross validation. Report cross validated (CV) error rate $\hat{R}^{10-CV}(\phi)$ for LDA and for the KNN models where $k \in \{1,5,11\}$. Compare this CV error rate with the error rates you derived in the previous problem for the same data size. What are these two quantities trying to estimate? Are they trying to estimate the same thing?

```
#Generate 1000 size data
data_size = 1000
my_sigma_1 = t(matrix(c(2,0.2,0.2,2)*0.6, ncol=2)) %*% matrix(c(2,0.2,0.2,2)*
0.6, ncol=2)
my_sigma_2 = t(matrix(c(2,0.2,0.2,2), ncol=2)) %*% matrix(c(2,0.2,0.2,2), nco
l=2)
data_0 = data.frame(rmvnorm(data_size, mean = rep(0, times = 2), sigma = my_s
igma_1))
```

```
data_1 = data.frame(X1 = NA, X2 = NA)
count = 0
while(count < data_size){
  new_draw = rmvnorm(1, mean = rep(0, times = 2), sigma = my_sigma_2)
  if( sqrt( new_draw[,1]**2 + new_draw[,2]**2) >= 1.5 ) {
    data_1 = rbind(data_1, data.frame(new_draw))
    count = count + 1
  }
}
data_1 = data_1[-1,]
cv_data = data.frame(rbind(data_0, data_1))
cv_data$class = as.factor(c(rep(0, times = data_size), rep(1, times = data_si
ze)))

#Cross validatio
cv_lda = c(); cv_k.1 = c(); cv_k.5 = c(); cv_k.11 = c();
for (i in 1:10) {
  indices = c((200*(i-1)+1):(200*i))
  training_set <- cv_data[-(indices),]
  testing_set <- cv_data[indices,]

  #LDA
  train_lda = lda(class~., data = training_set)
  test_predictions <- train_lda %>% predict(testing_set)
  test_error = sum(test_predictions$class!=testing_set$class)/length(testing_
set$class)
  cv_lda = c(cv_lda,test_error)

  #knn
  library(class)
  train_data_classifiers = as.factor(training_set$class)
  train_data_observations = training_set[,-3]
  test_data_observations = testing_set[,-3]
  test_data_classifiers = as.factor(testing_set$class)

  knn.1 <-  knn(train_data_observations, test_data_observations, cl = train_d
ata_classifiers, k=1)
  knn.1_risk = sum(test_data_classifiers != knn.1)/length(knn.1)
  cv_k.1 = c(cv_k.1, knn.1_risk)
  knn.5 <-  knn(train_data_observations, test_data_observations, cl = train_d
ata_classifiers, k=5)
  knn.5_risk = sum(test_data_classifiers != knn.5)/length(knn.5)
  cv_k.5 = c(cv_k.5, knn.5_risk)
  knn.11 <-  knn(train_data_observations, test_data_observations, cl = train_
data_classifiers, k=11)
  knn.11_risk = sum(test_data_classifiers != knn.11)/length(knn.11)
  cv_k.11 = c(cv_k.11, knn.11_risk)
}
```

```
# Print and compare the result
cv.lda = sum(cv_lda)/10; pre_lda = lda_test[8]
cv.k.1 = sum(cv_k.1)/10; pre_k.1 = k.1_test[8]
cv.k.5 = sum(cv_k.5)/10; pre_k.5 = k.5_test[8]
cv.k.11 = sum(cv_k.11)/10; pre_k.11 = k.11_test[8]

print(paste0("For LDA, the CV error rate is ", cv.lda, ", the previous error
rate is ", pre_lda, "."))

## [1] "For LDA, the CV error rate is 0.9985, the previous error rate is 0.52.
"

print(paste0("For Knn-1, the CV error rate is ", cv.k.1, ", the previous erro
r rate is ", pre_k.1, "."))

## [1] "For Knn-1, the CV error rate is 0.3165, the previous error rate is 0.
27."

print(paste0("For Knn-5, the CV error rate is ", cv.k.5, ", the previous erro
r rate is ", pre_k.5, "."))

## [1] "For Knn-5, the CV error rate is 0.306, the previous error rate is 0.2
5."

print(paste0("For Knn-11, the CV error rate is ", cv.k.11, ", the previous er
ror rate is ", pre_k.11, "."))

## [1] "For Knn-11, the CV error rate is 0.3115, the previous error rate is 0.
248."

Compared with the error rate, we can notice that the LDA model estimate the d
ata very different, because the CV error rate is much larger than the previou
s error rate. While the CV shows LDA modeling the data very bad, the non-CV s
hows LDA modeling quite well. However, the error rates between CV and non-CV
are quite close, indicating that they estimate the data similarly and quite w
ell.
```

What do the CV error rates for LDA and KNN tell us about which of these procedures is better for data generated from our given distribution?

```
Compared the CV error rate of LDA and KNN, we can see generally, knn has a mu
ch slower error rate of 0.3, and the LDA has a higher error rate of 0.98. Thu
s, Knn (KNN-11) is better for data generated from our given distribution.
```

In the previous problem, we derived the non-CV testing and training error rates for a data_size of 1000. What did those error rates tell us about the *classification rules* calculated for that particular data set? Can we use these non-CV results to say anything about the procedures we used?

```
lda.train = lda_train[8]; lda.test = lda_test[8]
k.1.train = k.1_train[8]; k.1.test = k.1_test[8]
k.5.train = k.5_train[8]; k.5.test = k.5_test[8]
```

```r
k.11.train = k.11_train[8]; k.11.test = k.11_test[8]

print(paste0("For LDA, the train error rate is ", lda.train, ", the test erro
r rate is ", lda.test, "."))

## [1] "For LDA, the train error rate is 0.502, the test error rate is 0.52."

print(paste0("For Knn-1, the train error rate is ", k.1.train, ", the test er
ror rate is ", k.1.test, "."))

## [1] "For Knn-1, the train error rate is 0, the test error rate is 0.27."

print(paste0("For Knn-5, the train error rate is ", k.5.train, ", the test er
ror rate is ", k.5.test, "."))

## [1] "For Knn-5, the train error rate is 0.156, the test error rate is 0.25.
"

print(paste0("For Knn-11, the train error rate is ", k.11.train, ", the test 
error rate is ", k.11.test, "."))

## [1] "For Knn-11, the train error rate is 0.186666666666667, the test error
 rate is 0.248."

These error rates tell us that for a specific data set, not all classificatio
n rule will apply. We should find the most satisfied one. From the non-CV res
ults, we can see that KNN can perform this data set the best with a lower err
or rate
```