Assignment-2

Pete Cuppernull

1/27/2020

Load Packages

```
library(tidyverse)
library(broom)
library(rsample)
library(patchwork)
library(corrplot)
library(ISLR)
library(caret)
library(class)
```

The Bayes Classifier

Question 1

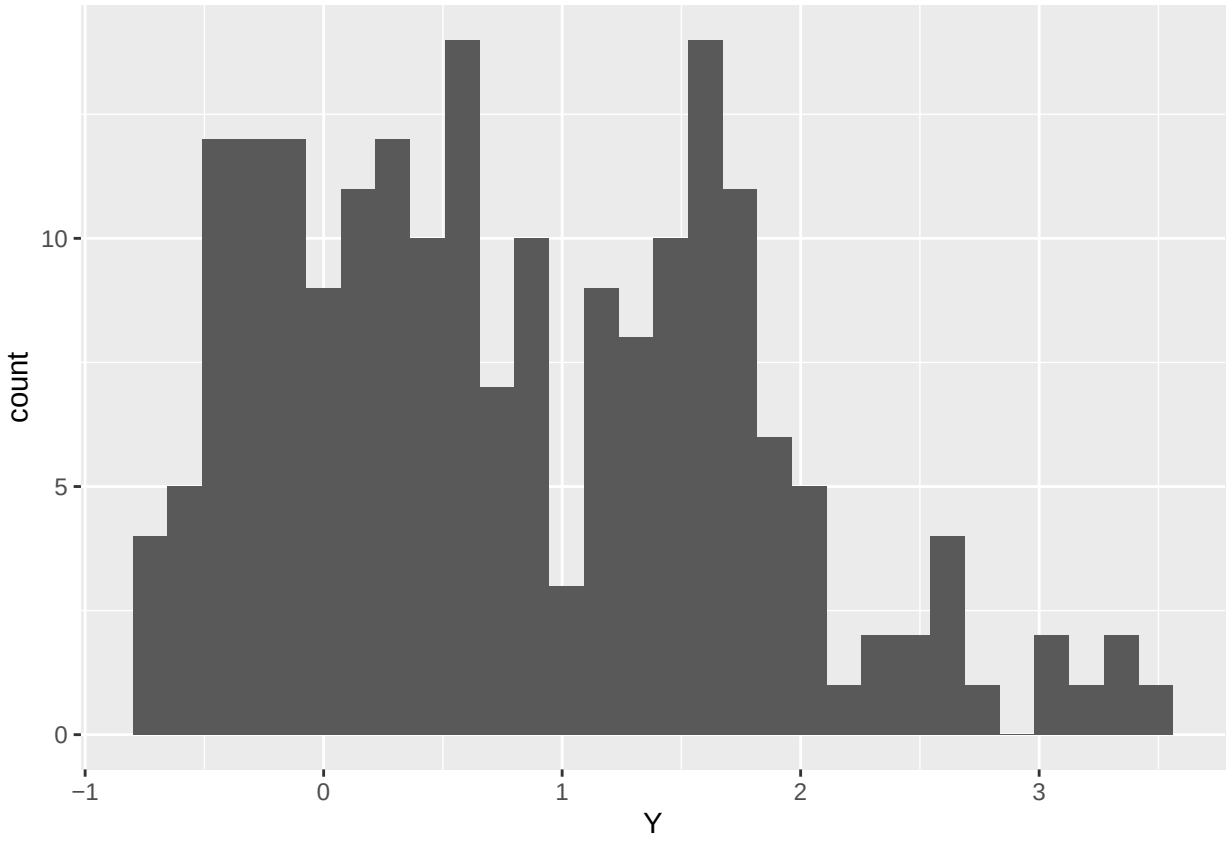
Parts A-C - create data

```
##Set Seed
set.seed(1414)

##Create Dataset
#Create x1 and x2
x1 <- runif(200, -1, 1)
x2 <- runif(200, -1, 1)

#Turn into DFs
data <- as.data.frame(x1)
x2 <- as.data.frame(x2)

#Create ID columns
data$obs <- seq.int(nrow(data))
x2$obs <- seq.int(nrow(x2))

#Join columns
data <- data %>%
  left_join(x2) %>%
  select(obs, x1, x2)

head(data)
```

```
##   obs          x1         x2
## 1   1 -0.54688097  0.9100521
## 2   2 -0.30589359  0.3513716
## 3   3 -0.06945728  0.8095712
```

```
## 4   4 -0.66379941 -0.6563678
## 5   5  0.77036681  0.9617783
## 6   6 -0.47114861  0.7875170
```

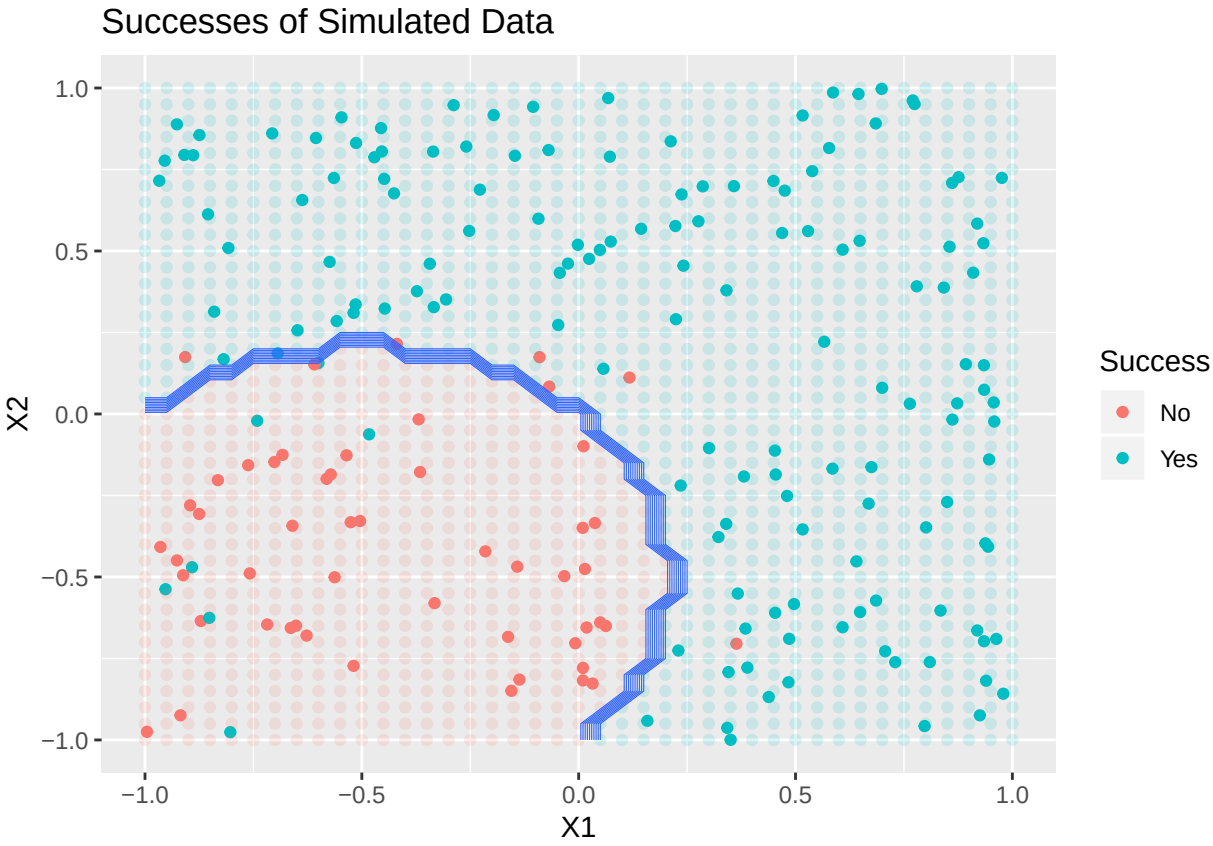
```r
##Calculate Y = ...
#Create error column
error <- rnorm(200, 0, 0.25)
error <- as.data.frame(error)
error$obs <- seq.int(nrow(error))

#Join error and calculate Y
data <- data %>%
  left_join(error) %>%
  mutate(Y = x1 + x1^2 + x2 + x2^2 + error)

ggplot(data, aes(Y)) +
  geom_histogram()
```



Parts D-H – calculate probability of success and create plot

```r
#Convert Ys to log-odds
probability_function <- function(model){
  odds <- exp(model)
  probability <- odds / (1 + odds)
  return(probability)
}
```

```r
##Create success column
data_success <- data %>%
  mutate(prob_success = probability_function(Y)) %>%
  mutate(Success = as.factor(if_else(prob_success > 0.5, "Yes", "No")))

###Create background grid
bg_1 <- as.data.frame(rep(seq(-1, 1, 0.05), 41))
bg_2 <- as.data.frame(rep(seq(-1, 1, 0.05), each = 41))

bg_grid <- cbind(bg_1, bg_2)
colnames(bg_grid) <- c("X1", "X2")
bg_grid <- bg_grid %>%
  mutate(Y = X1 + X1^2 + X2 + X2^2) %>%
  mutate(prob_success = probability_function(Y)) %>%
  mutate(Success = as.factor(if_else(prob_success > 0.5, "Yes", "No")))

bg_grid_cont <- bg_grid %>%
  mutate(Y = X1 + X1^2 + X2 + X2^2) %>%
  mutate(prob_success = probability_function(Y)) %>%
  mutate(Success = if_else(prob_success > 0.5, 1, 0))


ggplot() +
  geom_point(data = data_success, mapping = aes(x1, x2, color = Success)) +
  geom_point(data = bg_grid, mapping = aes(X1, X2, color = Success), alpha = .15) +
  geom_contour(data = bg_grid_cont, mapping = aes(X1, X2, z = Success), size = .2) +
  labs(x = "X1",
       y = "X2",
       title = "Successes of Simulated Data")
```



Differences between LDA and QDA

Question 2

A. Linear

If the Bayes deciison boundary is linear, LDA will generally perform better than QDA.

```
##CREATE FUNCTIONS FOR DATA CREATION AND PROCESSING
create_data <- function(){
x1 <- runif(1000, -1, 1)
x2 <- runif(1000, -1, 1)

#Turn into DFs
data <- as.data.frame(x1)
x2 <- as.data.frame(x2)

#Create ID columns
data$obs <- seq.int(nrow(data))
x2$obs <- seq.int(nrow(x2))

#Join columns
data <- data %>%
  left_join(x2) %>%
  select(obs, x1, x2)
```

```r
#Create error column
error <- rnorm(1000, 0, 1)
error <- as.data.frame(error)
error$obs <- seq.int(nrow(error))

#Join error and calculate Y
data1 <- data %>%
  left_join(error) %>%
  mutate(Y = x1 + x2 + error) %>%
  mutate(class = if_else(Y>=0, 1, 0))

data1
}

##PROCESSING FUNCTION
run_models <- function(iteration_number){
  iteration <- iteration_number
  ##create partitions
split <- initial_split(create_data(), prop = .7)
train <- training(split)
test <- testing(split)

##run LDA and save error rates
lda_m1 <- MASS::lda(class ~ x1 + x2, data = train)

pred_lda_test <- predict(lda_m1,
                newdata = test)
pred_lda_train <- predict(lda_m1,
                newdata = train)


table_lda_test <- table(test$class,pred_lda_test$class)
table_lda_train <- table(train$class,pred_lda_train$class)

error_lda_test <- sum(table_lda_test[row(table_lda_test) != col(table_lda_test)]) / sum(table_lda_test)
error_lda_train <- sum(table_lda_train[row(table_lda_train) != col(table_lda_train)]) / sum(table_lda_t

##run QDA and save error rates
qda_m1 <- MASS::qda(class ~ x1 + x2, data = train)

pred_qda_test <- predict(qda_m1,
                newdata = test)
pred_qda_train <- predict(qda_m1,
                newdata = train)

table_qda_test <- table(test$class,pred_qda_test$class)
table_qda_train <- table(train$class,pred_qda_train$class)

error_qda_test <- sum(table_qda_test[row(table_qda_test) != col(table_qda_test)]) / sum(table_qda_test)
error_qda_train <- sum(table_qda_train[row(table_qda_train) != col(table_qda_train)]) / sum(table_qda_t

#Collect results
results <- as.data.frame(rbind(c(iteration, error_lda_train, error_lda_test, error_qda_train, error_qda
```

```
colnames(results) <- c("iteration", "LDA Training Error", "LDA Test Error", "QDA Training Error", "QDA T
results
}

##Run 1000 sims
final_results_L <- map_dfr(1:1000, run_models)

final_results_L <- final_results_L %>%
  select(-iteration)
```

B. Summarize Results

```
skimr::skim(final_results_L)
```

Table 1: Data summary

Name	final_results_L
Number of rows	1000
Number of columns	4
Column type frequency:	
numeric	4
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
LDA Training Error	0	1	0.27	0.02	0.23	0.26	0.27	0.29	0.33	
LDA Test Error	0	1	0.28	0.03	0.19	0.26	0.28	0.30	0.37	
QDA Training Error	0	1	0.27	0.02	0.22	0.26	0.27	0.28	0.32	
QDA Test Error	0	1	0.28	0.03	0.19	0.26	0.28	0.30	0.37	

```
ggplot(final_results_L) +
  geom_density(aes(`LDA Training Error`, color = "LDA Training Error"), alpha = .25) +
  geom_vline(aes(xintercept = mean(`LDA Training Error`), color = "LDA Training Error"), size = 1.2) +
  geom_density(aes(`LDA Test Error`, color = "LDA Test Error"), alpha = .25) +
  geom_vline(aes(xintercept = mean(`LDA Test Error`), color = "LDA Test Error"), size = 1.2) +
  geom_density(aes(`QDA Training Error`, color = "QDA Training Error"), alpha = .25) +
  geom_vline(aes(xintercept = mean(`QDA Training Error`), color = "QDA Training Error"), size = 1.2) +
  geom_density(aes(`QDA Test Error`, color = "QDA Test Error"), alpha = .25) +
  geom_vline(aes(xintercept = mean(`QDA Test Error`), color = "QDA Test Error"), size = 1.2) +
  scale_color_brewer(type = "div",
                     name = NULL,
                     breaks = c("LDA Training Error", "LDA Test Error", "QDA Training Error", "QDA Test
  labs(title = "Distribution of LDA and QDA Training and Test Error Rates -- Linear",
       subtitle = "Mean denoted by vertical line",
       x = "Error Rate",
       y = "Frequency",
```

```
      legend = "Model")
```

We can observe that the LDA test error is slightly lower than the QDA test error, consistent with the notion that LDA models are prefered in cases on linear Bayes deicison boundaries. This is also a great example of the value of visualizing results – the tabular report does not allow us to observe our results with the same nuance as the graphical report.

Question 3

Part A

Conversely, in cases of non-linear Bayes decision boundaries, we expect QDA to generally perform better than LDA.

```
##Create functions for data and different number of sims

create_data_NL <- function(){
x1 <- runif(1000, -1, 1)
x2 <- runif(1000, -1, 1)

#Turn into DFs
data <- as.data.frame(x1)
x2 <- as.data.frame(x2)

#Create ID columns
data$obs <- seq.int(nrow(data))
x2$obs <- seq.int(nrow(x2))
```

```r
#Join columns
data <- data %>%
  left_join(x2) %>%
  select(obs, x1, x2)

#Create error column
error <- rnorm(1000, 0, 1)
error <- as.data.frame(error)
error$obs <- seq.int(nrow(error))

#Join error and calculate Y
data1 <- data %>%
  left_join(error) %>%
  mutate(Y = x1 + x1^2 + x2 + x2^2 + error) %>%
  mutate(class = if_else(Y>=0, 1, 0))

data1
}

run_models_NL <- function(iteration_number){
  iteration <- iteration_number
  ##create partitions
split <- initial_split(create_data_NL(), prop = .7)
train <- training(split)
test <- testing(split)

##run LDA and save error rates
lda_m1 <- MASS::lda(class ~ x1 + x1^2 + x2 + x2^2, data = train)

pred_lda_test <- predict(lda_m1,
                newdata = test)
pred_lda_train <- predict(lda_m1,
                newdata = train)


table_lda_test <- table(test$class,pred_lda_test$class)
table_lda_train <- table(train$class,pred_lda_train$class)

error_lda_test <- sum(table_lda_test[row(table_lda_test) != col(table_lda_test)]) / sum(table_lda_test)
error_lda_train <- sum(table_lda_train[row(table_lda_train) != col(table_lda_train)]) / sum(table_lda_t

##run DDA and save error rates
qda_m1 <- MASS::qda(class ~ x1 + x1^2 + x2 + x2^2, data = train)

pred_qda_test <- predict(qda_m1,
                newdata = test)
pred_qda_train <- predict(qda_m1,
                newdata = train)

table_qda_test <- table(test$class,pred_qda_test$class)
table_qda_train <- table(train$class,pred_qda_train$class)

error_qda_test <- sum(table_qda_test[row(table_qda_test) != col(table_qda_test)]) / sum(table_qda_test)
```

```
error_qda_train <- sum(table_qda_train[row(table_qda_train) != col(table_qda_train)]) / sum(table_qda_t

#Collect results
results <- as.data.frame(rbind(c(iteration, error_lda_train, error_lda_test, error_qda_train, error_qda

colnames(results) <- c("iteration", "LDA Training Error", "LDA Test Error", "QDA Training Error", "QDA
results
}

final_results_NL <- map_dfr(1:1000, run_models_NL)

final_results_NL <- final_results_NL %>%
  select(-iteration)
```

B.

```
skimr::skim(final_results_NL)
```

Table 3: Data summary

Name	final_results_NL
Number of rows	1000
Number of columns	4
Column type frequency:	
numeric	4
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
LDA Training Error	0	1	0.27	0.02	0.22	0.26	0.27	0.28	0.33	
LDA Test Error	0	1	0.27	0.03	0.20	0.26	0.27	0.29	0.36	
QDA Training Error	0	1	0.26	0.02	0.21	0.25	0.26	0.27	0.31	
QDA Test Error	0	1	0.26	0.03	0.18	0.25	0.26	0.28	0.34	

```
ggplot(final_results_NL) +
  geom_density(aes(`LDA Training Error`, color = "LDA Training Error"), alpha = .25) +
  geom_vline(aes(xintercept = mean(`LDA Training Error`), color = "LDA Training Error"), size = 1.2) +
  geom_density(aes(`LDA Test Error`, color = "LDA Test Error"), alpha = .25) +
  geom_vline(aes(xintercept = mean(`LDA Test Error`), color = "LDA Test Error"), size = 1.2) +
  geom_density(aes(`QDA Training Error`, color = "QDA Training Error"), alpha = .25) +
  geom_vline(aes(xintercept = mean(`QDA Training Error`), color = "QDA Training Error"), size = 1.2) +
  geom_density(aes(`QDA Test Error`, color = "QDA Test Error"), alpha = .25) +
  geom_vline(aes(xintercept = mean(`QDA Test Error`), color = "QDA Test Error"), size = 1.2) +
  scale_color_brewer(type = "div",
                     name = NULL,
                     breaks = c("LDA Training Error", "LDA Test Error", "QDA Training Error", "QDA Test
  labs(title = "Distribution of LDA and QDA Training and Test Error Rates -- Non-Linear",
```

```
        subtitle = "Mean denoted by vertical line",
        x = "Error Rate",
        y = "Frequency",
        legend = "Model")
```

This result is demonstrated more clearly. The graph supports the notion that QDA models are preferable in cases on non-linear Bayes decision boundaries, as evidenced by the lower error rates.

Question 4.

A.

We generally expect QDA to be preferable in large n cases, and the test error rate will improve compared to LDA. Even though QDA models are prone to higher variance, that concern is minimized with larger sample sizes.

```
create_data_NL_100 <- function(){
x1 <- runif(100, -1, 1)
x2 <- runif(100, -1, 1)

#Turn into DFs
data <- as.data.frame(x1)
x2 <- as.data.frame(x2)

#Create ID columns
data$obs <- seq.int(nrow(data))
x2$obs <- seq.int(nrow(x2))
```

```
#Join columns
data <- data %>%
  left_join(x2) %>%
  select(obs, x1, x2)

#Create error column
error <- rnorm(100, 0, 1)
error <- as.data.frame(error)
error$obs <- seq.int(nrow(error))

#Join error and calculate Y
data1 <- data %>%
  left_join(error) %>%
  mutate(Y = x1 + x1^2 + x2 + x2^2 + error) %>%
  mutate(class = if_else(Y>=0, 1, 0))

data1
}

create_data_NL_10000 <- function(){
x1 <- runif(10000, -1, 1)
x2 <- runif(10000, -1, 1)

#Turn into DFs
data <- as.data.frame(x1)
x2 <- as.data.frame(x2)

#Create ID columns
data$obs <- seq.int(nrow(data))
x2$obs <- seq.int(nrow(x2))

#Join columns
data <- data %>%
  left_join(x2) %>%
  select(obs, x1, x2)

#Create error column
error <- rnorm(10000, 0, 1)
error <- as.data.frame(error)
error$obs <- seq.int(nrow(error))

#Join error and calculate Y
data1 <- data %>%
  left_join(error) %>%
  mutate(Y = x1 + x1^2 + x2 + x2^2 + error) %>%
  mutate(class = if_else(Y>=0, 1, 0))

data1
}

create_data_NL_100000 <- function(){
x1 <- runif(100000, -1, 1)
x2 <- runif(100000, -1, 1)
```

```r
#Turn into DFs
data <- as.data.frame(x1)
x2 <- as.data.frame(x2)

#Create ID columns
data$obs <- seq.int(nrow(data))
x2$obs <- seq.int(nrow(x2))

#Join columns
data <- data %>%
  left_join(x2) %>%
  select(obs, x1, x2)

#Create error column
error <- rnorm(100000, 0, 1)
error <- as.data.frame(error)
error$obs <- seq.int(nrow(error))

#Join error and calculate Y
data1 <- data %>%
  left_join(error) %>%
  mutate(Y = x1 + x1^2 + x2 + x2^2 + error) %>%
  mutate(class = if_else(Y>=0, 1, 0))

data1
}

run_models_NL_100 <- function(iteration_number){
  iteration <- iteration_number
  ##create partitions
split <- initial_split(create_data_NL_100(), prop = .7)
train <- training(split)
test <- testing(split)

##run LDA and save error rates
lda_m1 <- MASS::lda(class ~ x1 + x1^2 + x2 + x2^2, data = train)

pred_lda_test <- predict(lda_m1,
                newdata = test)
pred_lda_train <- predict(lda_m1,
                newdata = train)


table_lda_test <- table(test$class,pred_lda_test$class)
table_lda_train <- table(train$class,pred_lda_train$class)

error_lda_test <- sum(table_lda_test[row(table_lda_test) != col(table_lda_test)]) / sum(table_lda_test)
error_lda_train <- sum(table_lda_train[row(table_lda_train) != col(table_lda_train)]) / sum(table_lda_t

##run QDA and save error rates
qda_m1 <- MASS::qda(class ~ x1 + x1^2 + x2 + x2^2, data = train)

pred_qda_test <- predict(qda_m1,
```

```
                newdata = test)
pred_qda_train <- predict(qda_m1,
                newdata = train)

table_qda_test <- table(test$class,pred_qda_test$class)
table_qda_train <- table(train$class,pred_qda_train$class)

error_qda_test <- sum(table_qda_test[row(table_qda_test) != col(table_qda_test)]) / sum(table_qda_test)
error_qda_train <- sum(table_qda_train[row(table_qda_train) != col(table_qda_train)]) / sum(table_qda_t

#Collect results
results <- as.data.frame(rbind(c(iteration, error_lda_train, error_lda_test, error_qda_train, error_qda

colnames(results) <- c("iteration", "LDA Training Error", "LDA Test Error", "QDA Training Error", "QDA
results
}

run_models_NL_10000 <- function(iteration_number){
  iteration <- iteration_number
  ##create partitions
split <- initial_split(create_data_NL_10000(), prop = .7)
train <- training(split)
test <- testing(split)

##run LDA and save error rates
lda_m1 <- MASS::lda(class ~ x1 + x1^2 + x2 + x2^2, data = train)

pred_lda_test <- predict(lda_m1,
                newdata = test)
pred_lda_train <- predict(lda_m1,
                newdata = train)


table_lda_test <- table(test$class,pred_lda_test$class)
table_lda_train <- table(train$class,pred_lda_train$class)

error_lda_test <- sum(table_lda_test[row(table_lda_test) != col(table_lda_test)]) / sum(table_lda_test)
error_lda_train <- sum(table_lda_train[row(table_lda_train) != col(table_lda_train)]) / sum(table_lda_t

##run DDA and save error rates
qda_m1 <- MASS::qda(class ~ x1 + x1^2 + x2 + x2^2, data = train)

pred_qda_test <- predict(qda_m1,
                newdata = test)
pred_qda_train <- predict(qda_m1,
                newdata = train)

table_qda_test <- table(test$class,pred_qda_test$class)
table_qda_train <- table(train$class,pred_qda_train$class)

error_qda_test <- sum(table_qda_test[row(table_qda_test) != col(table_qda_test)]) / sum(table_qda_test)
error_qda_train <- sum(table_qda_train[row(table_qda_train) != col(table_qda_train)]) / sum(table_qda_t
```

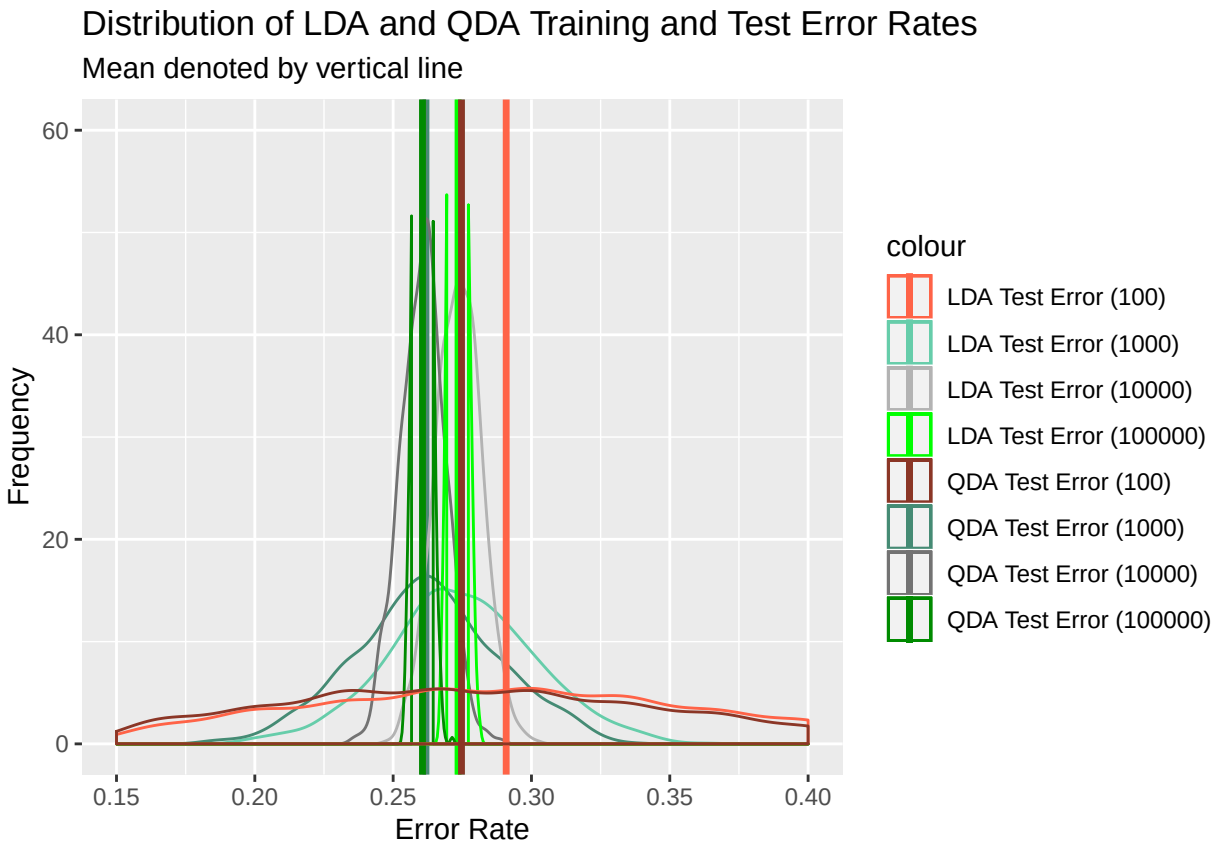
```r
#Collect results
results <- as.data.frame(rbind(c(iteration, error_lda_train, error_lda_test, error_qda_train, error_qda_

colnames(results) <- c("iteration", "LDA Training Error", "LDA Test Error", "QDA Training Error", "QDA T
results
}

run_models_NL_100000 <- function(iteration_number){
  iteration <- iteration_number
  ##create partitions
split <- initial_split(create_data_NL_100000(), prop = .7)
train <- training(split)
test <- testing(split)

##run LDA and save error rates
lda_m1 <- MASS::lda(class ~ x1 + x1^2 + x2 + x2^2, data = train)

pred_lda_test <- predict(lda_m1,
                newdata = test)
pred_lda_train <- predict(lda_m1,
                newdata = train)


table_lda_test <- table(test$class,pred_lda_test$class)
table_lda_train <- table(train$class,pred_lda_train$class)

error_lda_test <- sum(table_lda_test[row(table_lda_test) != col(table_lda_test)]) / sum(table_lda_test)
error_lda_train <- sum(table_lda_train[row(table_lda_train) != col(table_lda_train)]) / sum(table_lda_tr

##run QDA and save error rates
qda_m1 <- MASS::qda(class ~ x1 + x1^2 + x2 + x2^2, data = train)

pred_qda_test <- predict(qda_m1,
                newdata = test)
pred_qda_train <- predict(qda_m1,
                newdata = train)

table_qda_test <- table(test$class,pred_qda_test$class)
table_qda_train <- table(train$class,pred_qda_train$class)

error_qda_test <- sum(table_qda_test[row(table_qda_test) != col(table_qda_test)]) / sum(table_qda_test)
error_qda_train <- sum(table_qda_train[row(table_qda_train) != col(table_qda_train)]) / sum(table_qda_tr

#Collect results
results <- as.data.frame(rbind(c(iteration, error_lda_train, error_lda_test, error_qda_train, error_qda_

colnames(results) <- c("iteration", "LDA Training Error", "LDA Test Error", "QDA Training Error", "QDA T
results
}

##100 obs
final_results_NL_100 <- map_dfr(1:1000, run_models_NL_100)
```

```
 final_results_NL_100 <- final_results_NL_100 %>%
  select(-iteration)
##1000 obs

##10000 obs
final_results_NL_10000 <- map_dfr(1:1000, run_models_NL_10000)

final_results_NL_10000 <- final_results_NL_10000 %>%
 select(-iteration)
##100000 obs
final_results_NL_100000 <- map_dfr(1:1000, run_models_NL_100000)

final_results_NL_100000 <- final_results_NL_100000 %>%
  select(-iteration)
```

B.

```
cols <- c("LDA Test Error (1000)" = "aquamarine3", "QDA Test Error (1000)" = "aquamarine4",
          "LDA Test Error (100)" = "tomato1", "QDA Test Error (100)" = "tomato4",
          "LDA Test Error (10000)" = "gray70", "QDA Test Error (10000)" = "grey45",
          "LDA Test Error (100000)" = "green", "QDA Test Error (100000)" = "green4")
ggplot() +
  geom_density(data = final_results_NL, mapping = aes(`LDA Test Error`, color = "LDA Test Error (1000)"
  geom_vline(data = final_results_NL, mapping = aes(xintercept = mean(`LDA Test Error`), color = "LDA T
  geom_density(data = final_results_NL, mapping = aes(`QDA Test Error`, color = "QDA Test Error (1000)"
  geom_vline(data = final_results_NL, mapping = aes(xintercept = mean(`QDA Test Error`), color = "QDA T
  geom_density(data = final_results_NL_10000, mapping = aes(`LDA Test Error`, color = "LDA Test Error (
  geom_vline(data = final_results_NL_10000, mapping = aes(xintercept = mean(`LDA Test Error`), color =
  geom_density(data = final_results_NL_10000, mapping = aes(`QDA Test Error`, color = "QDA Test Error (
  geom_vline(data = final_results_NL_10000, mapping = aes(xintercept = mean(`QDA Test Error`), color =
  geom_density(data = final_results_NL_100000, mapping = aes(`LDA Test Error`, color = "LDA Test Error
  geom_vline(data = final_results_NL_100000, mapping = aes(xintercept = mean(`LDA Test Error`), color =
  geom_density(data = final_results_NL_100000, mapping = aes(`QDA Test Error`, color = "QDA Test Error
  geom_vline(data = final_results_NL_100000, mapping = aes(xintercept = mean(`QDA Test Error`), color =
  geom_density(data = final_results_NL_100, mapping = aes(`LDA Test Error`, color = "LDA Test Error (10
  geom_vline(data = final_results_NL_100, mapping = aes(xintercept = mean(`LDA Test Error`), color = "LD
  geom_density(data = final_results_NL_100, mapping = aes(`QDA Test Error`, color = "QDA Test Error (10
  geom_vline(data = final_results_NL_100, mapping = aes(xintercept = mean(`QDA Test Error`), color = "Q
  scale_color_manual(values = cols) +
  labs(title = "Distribution of LDA and QDA Training and Test Error Rates",
       subtitle = "Mean denoted by vertical line",
       x = "Error Rate",
       y = "Frequency",
       legend = "Model") +
  ylim(0,60) +
  xlim(.15, .4)
```

```
## Warning: Removed 112 rows containing non-finite values (stat_density).

## Warning: Removed 112 rows containing non-finite values (stat_density).
```



We can observe above that QDA performs better as the sample size increases. In the largest sample size above, the LDA model (light green) has a higher error rate than the QDA model (dark green).

Question 5. Voter Turnout

Import and Clean Data, Make Splits (a)

```
#Import data
mental_health <- read_csv("/Users/petecuppernull/Dropbox/UChicago/2019-20/Winter/Computatonal Modeling/

#remove na's
mental_health <- mental_health %>%
  select(vote96, mhealth_sum, age, educ, black, female, married, inc10) %>%
  na.omit() %>%
  mutate(vote96 = as.factor(vote96))

split_mh <- initial_split(mental_health, prop = .7)
train_mh <- training(split_mh)
test_mh <- testing(split_mh)
```

B. Run All Models

```
library(rcfss) # logit2prob function
library(tidymodels) # accuracy function
```

```r
##Logit
mh_logit <- glm(vote96 ~ ., data = train_mh, family = binomial)
mh_logit_error <- augment(mh_logit, newdata = test_mh) %>%
  as_tibble() %>%
  mutate(.prob = logit2prob(.fitted),
         .pred = factor(round(.prob))) %>%
  accuracy(truth = vote96, estimate = .pred)

error_logit <- 1 - mh_logit_error$.estimate

##LDA
lda_mh <- MASS::lda(vote96 ~ ., data = train_mh)

mh_lda_test <- predict(lda_mh,
                newdata = test_mh)
mh_lda_train <- predict(lda_mh,
                newdata = train_mh)


table_lda_test_mh <- table(test_mh$vote96,mh_lda_test$class)
table_lda_train_mh <- table(train_mh$vote96,mh_lda_train$class)

error_lda_test <- sum(table_lda_test_mh[row(table_lda_test_mh) != col(table_lda_test_mh)]) / sum(table_l
error_lda_train <- sum(table_lda_train_mh[row(table_lda_train_mh) != col(table_lda_train_mh)]) / sum(tal

##QDA
qda_mh <- MASS::qda(vote96 ~ ., data = train_mh)

mh_qda_test <- predict(qda_mh,
                newdata = test_mh)
mh_qda_train <- predict(qda_mh,
                newdata = train_mh)


table_qda_test_mh <- table(test_mh$vote96,mh_qda_test$class)
table_qda_train_mh <- table(train_mh$vote96,mh_qda_train$class)

error_qda_test <- sum(table_qda_test_mh[row(table_qda_test_mh) != col(table_qda_test_mh)]) / sum(table_q
error_qda_train <- sum(table_qda_train_mh[row(table_qda_train_mh) != col(table_qda_train_mh)]) / sum(tal

##Naive Bayes -- this isnt working for some reason
# create response and feature data
features <- setdiff(names(train_mh), "vote96") #setdiff(x,y) elements in x but not in y
x <- train_mh[, features]
y <- train_mh$vote96

# set up 10-fold cross validation procedure https://www.rdocumentation.org/packages/caret/versions/6.0-
train_control <- trainControl(
  method = "cv",
  number = 10
)

# train model
```

```r
mh_nb <- train(
  x = x,
  y = y,
  method = "nb",
  trControl = train_control
)

nb_matrix <- confusionMatrix(mh_nb)
nb_error <- 1 - nb_matrix$table[1, 1] + nb_matrix$table[2, 2]

##KNN
mse_knn_mh <- tibble(k = 1:10,
                  knn_train = map(k, ~ class::knn(select(train_mh, -vote96),
                                                  test = select(train_mh, -vote96),
                                                  cl = train_mh$vote96, k = .)),
                  knn_test = map(k, ~ class::knn(select(train_mh, -vote96),
                                                 test = select(test_mh, -vote96),
                                                 cl = train_mh$vote96, k = .)),
                  err_train = map_dbl(knn_train, ~ mean(test_mh$vote96 != .)),
                  err_test = map_dbl(knn_test, ~ mean(test_mh$vote96 != .)))
```

C.i. Error Rates

- Logit: 0.2979943
- LDA: 0.3123209
- QDA: 0.3352436
- Naive Bayes: 47.8137255
- KNN (nnumber of neighbors):
  - 1: 0.3180516
  - 2: 0.3553009
  - 3: 0.3323782
  - 4: 0.3237822
  - 5: 0.3266476
  - 6: 0.3266476
  - 7: 0.3209169
  - 8: 0.3065903
  - 9: 0.3037249
  - 10: 0.2951289

C.ii. ROC/AOC

```r
library(pROC)
#Logit

mh_logit_test <- augment(mh_logit, newdata = test_mh) %>%
  as_tibble() %>%
  mutate(.prob = logit2prob(.fitted),
         .pred = factor(round(.prob)))

logit_roc <- roc(as.numeric(test_mh$vote96), as.numeric(mh_logit_test$.pred))
logit_auc <- auc(logit_roc)
```

```
#LDA
lda_roc <- roc(as.numeric(test_mh$vote96), as.numeric(mh_lda_test$class))
lda_auc <- auc(lda_roc)

#QDA
qda_roc <- roc(as.numeric(test_mh$vote96), as.numeric(mh_qda_test$class))
qda_auc <- auc(qda_roc)

#Naive Bayes
nb_predict <- predict(mh_nb, test_mh)

nb_roc <- roc(as.numeric(test_mh$vote96), as.numeric(nb_predict))
nb_auc <- auc(nb_roc)

#KNN
knn_aoc <- function(x){
KNN_roc <- roc(as.numeric(mse_knn_mh$knn_test[[x]]), as.numeric(nb_predict))
knn_auc1 <- auc(KNN_roc)
return(knn_auc1)
}
```

Area under the Curves

- Logit: 0.6053865
- LDA: 0.5822384
- QDA: 0.5963001
- Naive Bayes: 0.5787229
- KNN (number of neighbors):
  - 1: 0.60388
  - 2: 0.6160594
  - 3: 0.6435248
  - 4: 0.6973458
  - 5: 0.7083219
  - 6: 0.6984665
  - 7: 0.7048444
  - 8: 0.6856318
  - 9: 0.705516
  - 10: 0.6789259

I would argue that the KNN model with 9 nearest neighbors performs the best, as indicated by its high AOC score and its low error rate (the lowest of all the models). By best, I am considering the predictive power of the model as indicated by both the error rate and AOC, and KNN with 9 neighbors performs well in both instances.