Problem Set 2

Minyoung Do

2/2/2020

The Bayes Classifier

(20 points) For classification problems, the test error rate is minimized by a simple classifier that assigns each observation to the most likely class given its predictor values:

$$\Pr(Y = j|X = x_0)$$

where x_0 is the test observation and each possible class is represented by J. This is a conditional probability that $Y = j$, given the observed predictor vector x_0. This classifier is known as the Bayes classifier. If the response variable is binary (i.e. two classes), the Bayes classifier corresponds to predicting class one if $\Pr(Y = 1|X = x_0) > 0.5$, and class two otherwise.

Produce a graph illustrating this concept. Specifically, implement the following elements in your program:

a. Set your random number generator seed.

```
set.seed(420)
```

b. Simulate a dataset of $N = 200$ with X_1, X_2 where X_1, X_2 are random uniform variables between $[-1, 1]$.

```
data <- data.frame(replicate(2, runif(200, -1, 1)))
```

c. Calculate $Y = X_1 + X_1^2 + X_2 + X_2^2 + \epsilon$, where $\epsilon \sim N(\mu = 0, \sigma^2 = 0.25)$.

```
data$error <- rnorm(200, 0, .5)

data <- data %>%
  mutate(Y = X1 + X1^2 + X2 + X2^2 + error)
```

d. Y is defined in terms of the log-odds of success on the domain $[-\infty, +\infty]$. Calculate the probability of success bounded between $[0, 1]$.

Odds of Success could be written as:

$$Pr(Success) = \frac{Pr(Success)}{1 - Pr(Success)} = e^{\beta_0 + \beta_1 Success}$$

And as Y is defined as log-odds,

$$Y = \log(\frac{Pr(Success)}{1 - Pr(Success)})$$

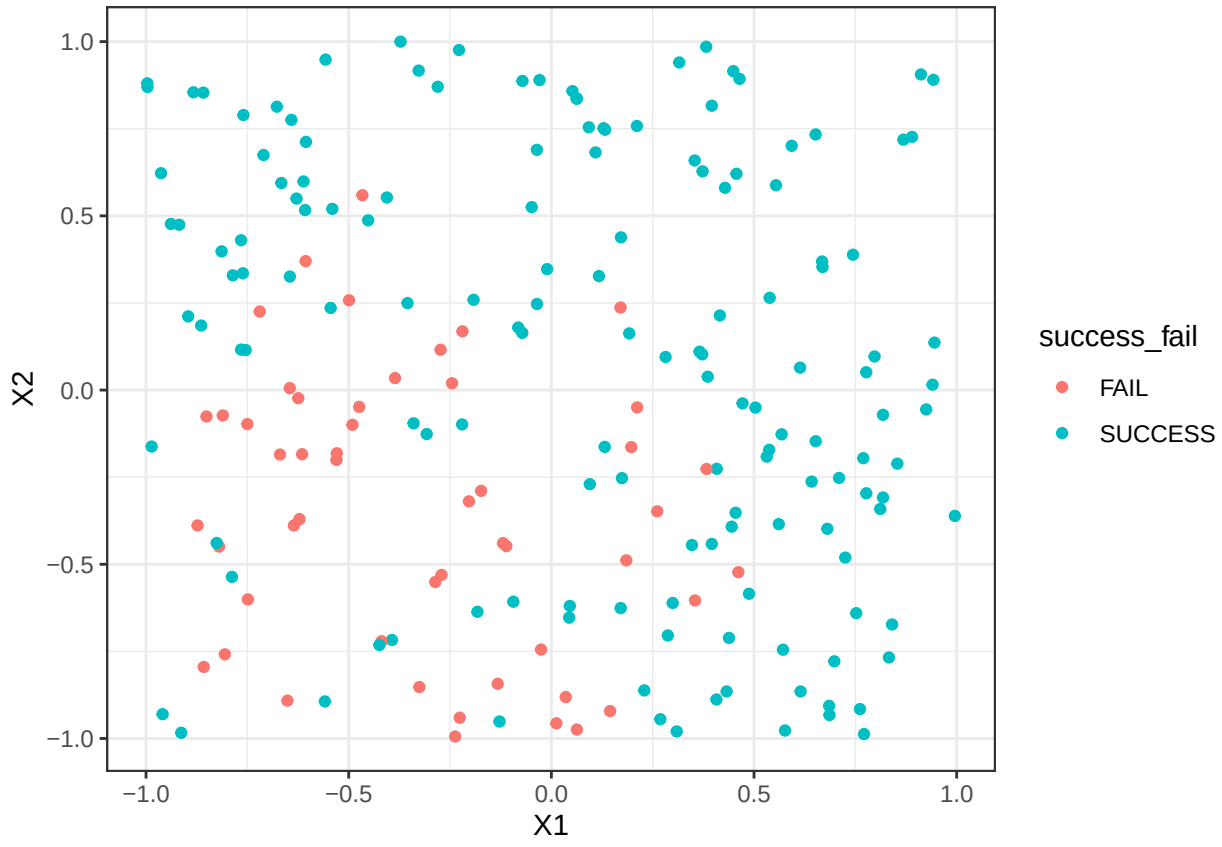
Now raise it to e to cancel out log, we have

$$Pr(Success) = \frac{e^Y}{1 + e^Y}$$

```
data$success <- exp(data$Y) / (1 + exp(data$Y))
```

e. Plot each of the data points on a graph and use color to indicate if the observation was a success or a failure.

```
data <- data %>%
  mutate(success_fail = ifelse(success > 0.5, "SUCCESS", "FAIL"))

data %>%
  ggplot(aes(X1, X2, color = success_fail)) +
  geom_point() +
  theme_bw()
```



f. Overlay the plot with Bayes decision boundary, calculated using X_1, X_2.

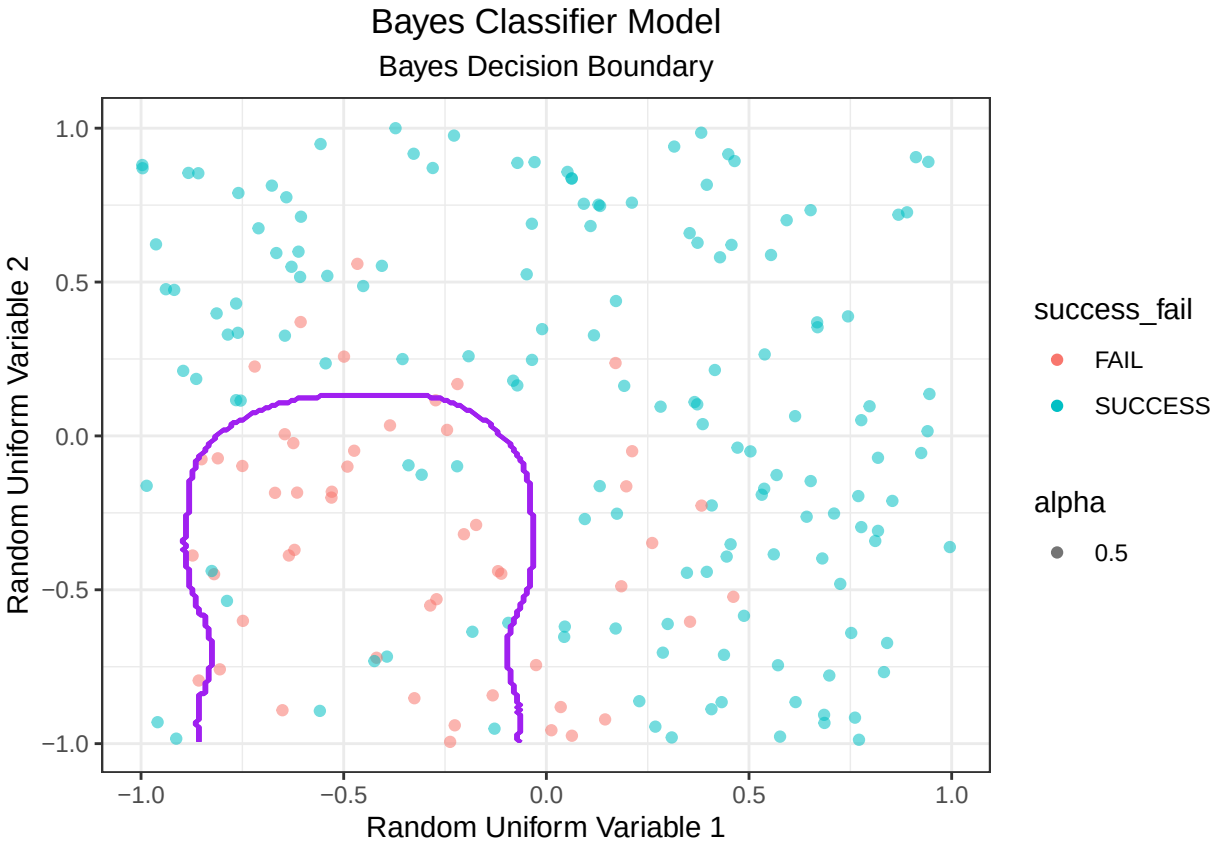
g. Give your plot a meaningful title and axis labels. The colored backgroound grid is optional

```
# setting up 10-fold cross validation procedure
train_control <- trainControl(method = "cv", number = 10)

# saving columns as objects
dat_points <- data[, 1:2]
binary <- data$success_fail

# training the model
matrix <- train(x = dat_points, y = binary, method = "nb", trControl = train_control)

# confusion matrix result
confusionMatrix(matrix)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction FAIL SUCCESS
##    FAIL    13.5     5.5
##    SUCCESS 12.0    69.0
##
##  Accuracy (average) : 0.825
```

```
# referenced Rpubs post here: https://rpubs.com/wilsonchua/MLR-Section16-NaiveBayes
# prepping Bayes boundary line
set_x1 <- seq(min(dat_points$X1), max(dat_points$X1), length = 250)
set_x2 <- seq(min(dat_points$X2), max(dat_points$X2), length = 250)

# tuning grid
grid <- expand.grid(X1 = set_x1, X2 = set_x2)

# obtaining predicted values
grid$success_fail <- predict(matrix, newdata = grid)

# changing the column class to dbl
grid$success_fail <- as.numeric(grid$success_fail)

# plotting the boundary line
data %>%
  ggplot(aes(X1, X2, color = success_fail)) +
  geom_point(aes(alpha = .5)) +
  geom_contour(data = grid, aes(z = success_fail), color = "purple") +
  labs(x = "Random Uniform Variable 1", y = "Random Uniform Variable 2",
```

```
      title = "Bayes Classifier Model", subtitle = "Bayes Decision Boundary") +
theme_bw() +
theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5))
```



Exploring Simulated Differences between LDA and QDA

Note: Unless otherwise specified, assume the number of observations $N = 1000$.

(20 points) If the Bayes decision boundary is linear, do we expect LDA or QDA to perform better on the training set? On the test set?

a. Repeat the following process 1000 times.

i. Simulate a dataset of 1000 observations with $X_1, X_2 \sim \text{Uniform}(-1, +1)$. Y is a binary response variable defined by a Bayes decision boundary of $f(X) = X_1 + X_2$, where values 0 or greater are coded TRUE and values less than 0 or coded FALSE. Whereas your simulated Y is a function of $X_1 + X_2 + \epsilon$ where $\epsilon \sim N(0, 1)$. That is, your simulated Y is a function of the Bayes decision boundary plus some irreducible error.

ii. Randomly split your dataset into 70/30% training/test sets.

iii. Use the training dataset to estimate LDA and QDA models.

iv. Calculate each model's training and test error rate.

```
set.seed(432)
model_error <- function(n) {
  # creating a data frame of two random uniform variable columns and error columns
  df <- data.frame(X1 = runif(n, -1, 1), X2 = runif(n, -1, 1), Error = runif(n, 0, 1)) %>%
    mutate(Y = ifelse(X1 + X2 + Error >= 0, TRUE, FALSE))
  split <- initial_split(df, prop = .7)
  train <- training(split)
  test <- testing(split)
  # spliting the original dataset into training and test sets by the proportion of 0.7
  lda_1 <- lda(Y ~ X1 + X2, data = train)
  qda_1 <- qda(Y ~ X1 + X2, data = train)
  # LDA results
  train_lda <- predict(lda_1, newdata = train)
  test_lda <- predict(lda_1, newdata = test)
  # LDA error rates
  error_train_lda <- mean(train$Y != train_lda$class)
  error_test_lda <- mean(test$Y != test_lda$class)
  # QDA results
  train_qda <- predict(qda_1, newdata = train)
  test_qda <- predict(qda_1, newdata = test)
  # QDA error rates
  error_train_qda <- mean(train$Y != train_qda$class)
  error_test_qda <- mean(test$Y != test_qda$class)
  # merging all the results into a data frame
  results <- data.frame(
    "Training LDA Error Rate" = error_train_lda,
    "Test LDA Error Rate" = error_test_lda,
    "Training QDA Error Rate" = error_train_qda,
    "Test QDA Error Rate" = error_test_qda)
  print(results)
}

m <- model_error(1000)
for(i in 1:999) {
  m <- bind_rows(m, model_error(1000))
}
```

b. Summarize all the simulations' error rates and report the results in tabular and graphical form. Use this evidence to support your answer.

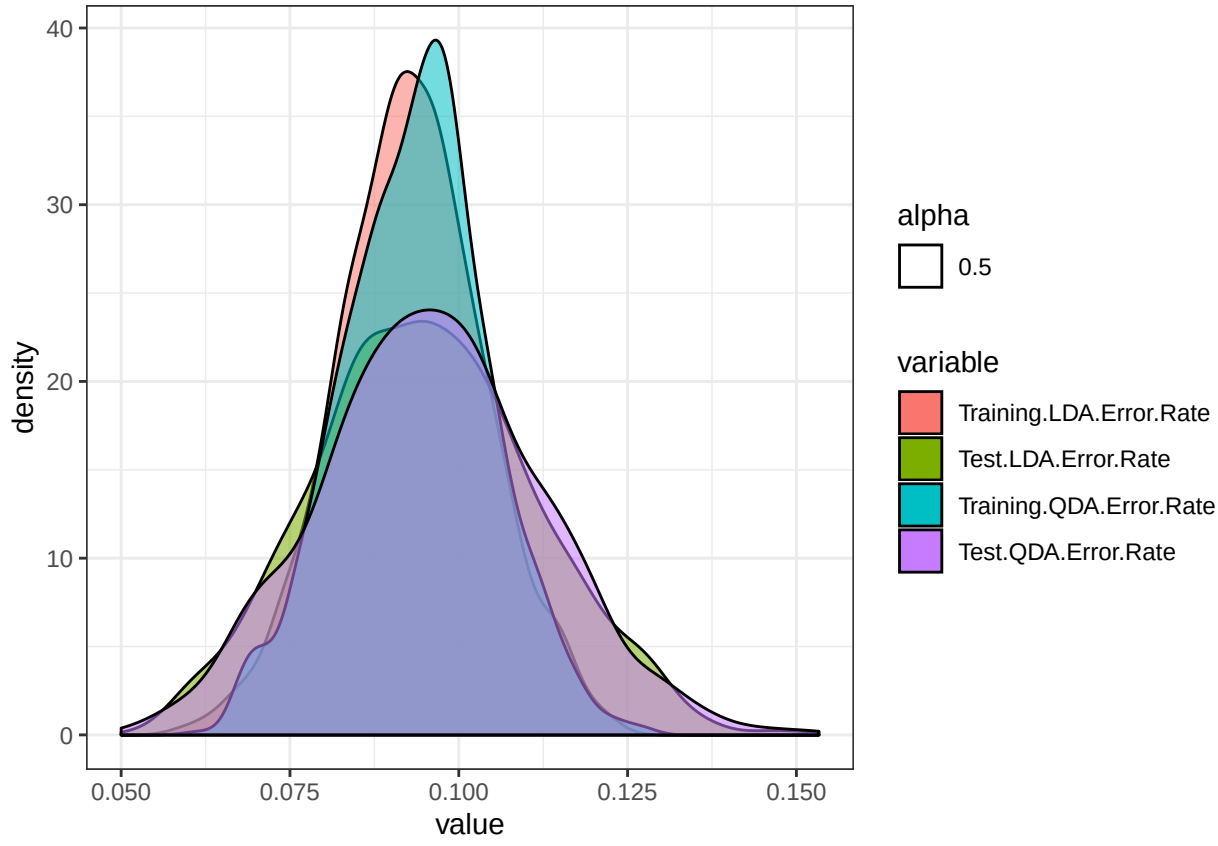
```
table_1 <- tidy(m) %>%
  kable(digits = 3,
```

Table 1: Error Rates Summary: Linear

column	n	mean	sd	median	trimmed	mad	min	max	range	skew
Training.LDA.Error.Rate	1000	0.093	0.011	0.093	0.093	0.007	0.059	0.123	0.064	-0.026
Test.LDA.Error.Rate	1000	0.095	0.016	0.093	0.095	0.010	0.053	0.153	0.100	0.173
Training.QDA.Error.Rate	1000	0.094	0.011	0.094	0.094	0.007	0.061	0.127	0.066	-0.019
Test.QDA.Error.Rate	1000	0.096	0.017	0.097	0.096	0.010	0.050	0.153	0.103	0.132

```
        caption = "Error Rates Summary: Linear")
table_1
```

```
m2 <- melt(m)

m2 %>%
ggplot(aes(x = value, fill = variable, alpha = .5)) +
  geom_density() +
  theme_bw()
```



If the Bayes decision boundary is linear, LDA is expected to perform better on the test set, as QDA is more likely to result in overfitting the linearity with high variance and capturing noises of data. On the other hand, QDA is a more flexible form of fitting than LDA, thus allowing a

better performance on the training set. This density plot reports 4 error rates of LDA and QDA performed on training and test sets. Both LDA and QDA tests perform about the same; it is difficult to intuitively identify huge differences in the density plot. When looking at the table of error rates though, there are small differences found across the error rates. I would like to explain some of these differences to assess the performances of LDA and QDA.

The mean error rates of QDA are slightly higher by .01 for both training and test sets. This means that, overall, QDA had a higher error rate than LDA for both sets. This is a somewhat unexpected result, as QDA is supposed to perform better on the training set. I suspect LDA's outperformance is due to the high performance similarity between two non-parametric methods, or because the Bayes decision boundary is linear. The error rates table, on the other hand, confirms that LDA performs better than QDA on the test set by accurately parameterizing the decision boundary forms.

(20 points) If the Bayes decision boundary is non-linear, do we expect LDA or QDA to perform better on the training set? On the test set?

a. Repeat the following process 1000 times.

i. Simulate a dataset of 1000 observations with $X_1, X_2 \sim \text{Uniform}(-1, +1)$. Y is a binary response variable defined by a Bayes decision boundary of $f(X) = X_1 + X_1^2 + X_2 + X_2^2$, where values 0 or greater are coded TRUE and values less than 0 or coded FALSE. Whereas your simulated Y is a function of $X_1 + X_1^2 + X_2 + X_2^2 + \epsilon$ where $\epsilon \sim N(0, 1)$. That is, your simulated Y is a function of the Bayes decision boundary plus some irreducible error.

ii. Randomly split your dataset into 70/30% training/test sets.

iii. Use the training dataset to estimate LDA and QDA models.

iv. Calculate each model's training and test error rate.

```
# Same codes with the linear one; but different formula in the mutate function.
model_error_2 <- function(n) {
  # creating a data frame of two random uniform variable columns and error columns
  df <- data.frame(X1 = runif(n, -1, 1), X2 = runif(n, -1, 1), Error = runif(n, 0, 1)) %>%
    mutate(Y = ifelse(X1 + X1^2 + X2 + X2^2 + Error >= 0, TRUE, FALSE))
  split <- initial_split(df, prop = .7)
  train <- training(split)
  test <- testing(split)
  # spliting the original dataset into training and test sets by the proportion of 0.7
  lda_1 <- lda(Y ~ X1 + X1^2 + X2 + X2^2, data = train)
  qda_1 <- qda(Y ~ X1 + X1^2 + X2 + X2^2, data = train)
  # LDA results
  train_lda <- predict(lda_1, newdata = train)
  test_lda <- predict(lda_1, newdata = test)
  # LDA error rates
  error_train_lda <- mean(train$Y != train_lda$class)
  error_test_lda <- mean(test$Y != test_lda$class)
  # QDA results
  train_qda <- predict(qda_1, newdata = train)
```

Table 2: Error Rates Summary: Non-linear

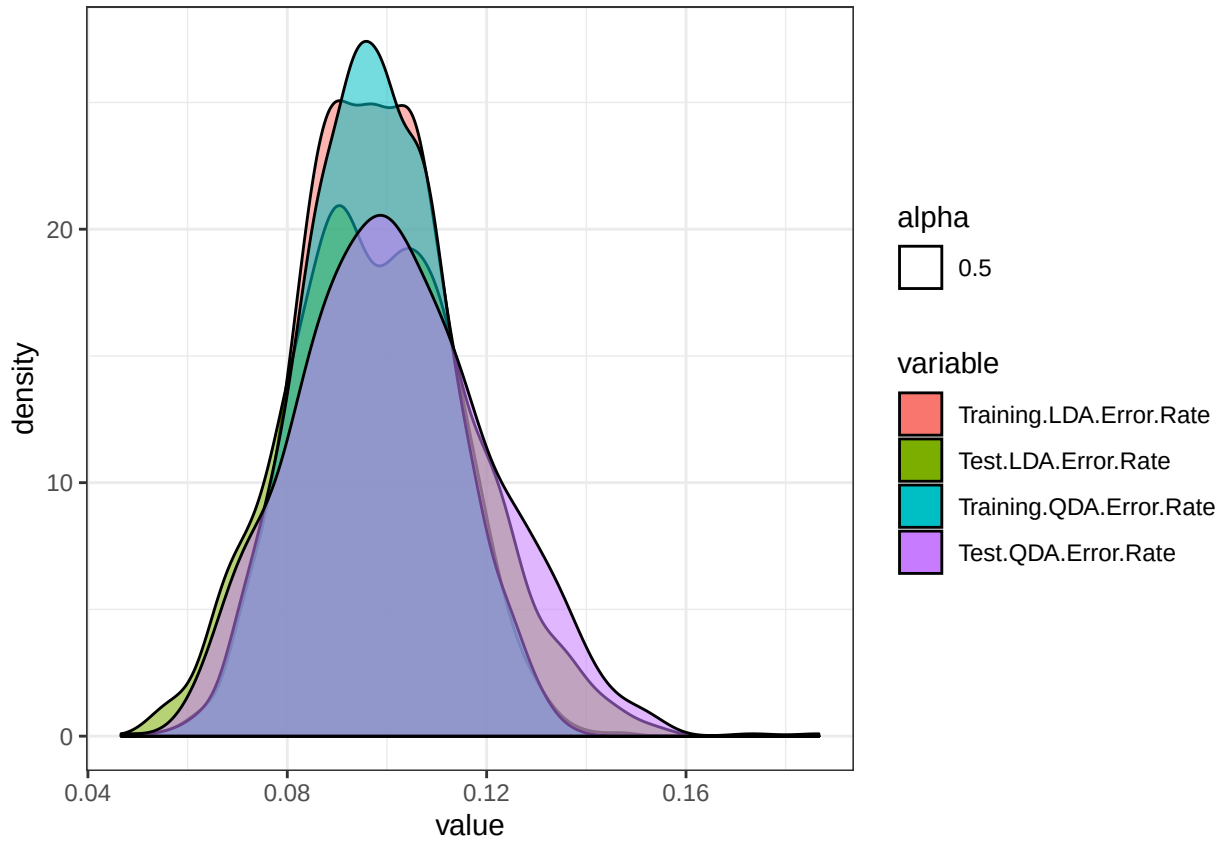
column	n	mean	sd	median	trimmed	mad	min	max	range	skew
Training.LDA.Error.Rate	1000	0.098	0.014	0.097	0.097	0.010	0.056	0.147	0.091	0.100
Test.LDA.Error.Rate	1000	0.099	0.018	0.097	0.098	0.013	0.053	0.153	0.100	0.183
Training.QDA.Error.Rate	1000	0.097	0.014	0.097	0.097	0.010	0.056	0.137	0.081	0.032
Test.QDA.Error.Rate	1000	0.102	0.019	0.100	0.101	0.013	0.047	0.187	0.140	0.317

```
  test_qda <- predict(qda_1, newdata = test)
  # QDA error rates
  error_train_qda <- mean(train$Y != train_qda$class)
  error_test_qda <- mean(test$Y != test_qda$class)
  # merging all the results into a data frame
  results <- data.frame(
    "Training LDA Error Rate" = error_train_lda,
    "Test LDA Error Rate" = error_test_lda,
    "Training QDA Error Rate" = error_train_qda,
    "Test QDA Error Rate" = error_test_qda)
  print(results)
}

k <- model_error_2(1000)
for(i in 1:999) {
  k <- bind_rows(k, model_error_2(1000))
}
```

b. Summarize all the simulations' error rates and report the results in tabular and graphical form. Use this evidence to support your answer.

```
table_2 <- tidy(k) %>%
  kable(digits = 3,
        caption = "Error Rates Summary: Non-linear")
table_2
```

```
k2 <- reshape2::melt(k)

k2 %>%
ggplot(aes(x = value, fill = variable, alpha = .5)) +
  geom_density() +
  theme_bw()
```



If the Bayes decision boundary is non-linear, QDA decision will perform better for both sets with smaller bias. This is because LDA does not capture as much non-linear boundary as QDA. The table and plot above confirm my answer; the error rates of QDA on both sets (TR: .094 TS: .096) are significantly lower than the ones of LDA (TR: .109 TS: .110) each by .015 and .014. This pattern persists in the standard deviation values as well. QDA has smaller *sd* values compared to the LDA error rates. In addition, the differences in mean are also observed in the density plot; the distribution of QDA error rates for both sets are farther on the left side of the plane. Therefore, the error rates table and density plot confirms my answer above.

(20 points) In general, as sample size n increases, do we expect the test error rate of QDA relative to LDA to improve, decline, or be unchanged? Why?

The test error rate of QDA would decrease as sample size increases. A larger number of the training set helps mitigate the high variance, and it will be able to improve the test prediction accuracy with a better fit; in other words, if the sample size is big, the variance of classifiers become less of a concern.

a. Use the non-linear Bayes decision boundary approach from part (2) and vary n across your simulations (e.g., simulate 1000 times for n = c(1e02, 1e03, 1e04, 1e05).

```
# creating an empty list of 1000 rows to merge the output in
#nonlinear_sim <- vector("list", 1000)

# for loop for iteration
# other approaches did not work, so I will combine each of these 4 dataframes (1000 rows each) n

#nonlinear_sim <- model_error_2(1e02)
#for (i in 1:999) {
#  nonlinear_sim <- bind_rows(nonlinear_sim, model_error_2(1e02))
#}

#nonlinear_sim1 <- model_error_2(1e03)
#for (i in 1:999) {
#  nonlinear_sim <- bind_rows(nonlinear_sim, model_error_2(1e03))
#}

#nonlinear_sim2 <- model_error_2(1e04)
#for (i in 1:999) {
#  nonlinear_sim <- bind_rows(nonlinear_sim, model_error_2(1e04))
#}

#nonlinear_sim3 <- model_error_2(1e05)
#for (i in 1:999) {
 # nonlinear_sim <- bind_rows(nonlinear_sim, model_error_2(1e05))
#}

# combining data frames into one and tidy them up
#data_nonlinear <- nonlinear_sim %>%
 # gather(key = type, value = error_rate, Training.LDA.Error.Rate, Test.LDA.Error.Rate,
  #       Training.QDA.Error.Rate, Test.QDA.Error.Rate)
```

b. Plot the test error rate for the LDA and QDA models as it changes over all of these values of n. Use this graph to support your answer.

```
#table_3 <- tidy(nonlinear_sim) %>%
#  kable(digits = 3,
#        caption = "Error Rates Summary: n = 1e02, 1e03, 1e04, 1e05")
#table_3

#data_nonlinear %>%
#ggplot(aes(x = error_rate, fill = type, alpha = .6)) +
#  geom_density() +
#  theme_bw()
```

Modeling voter turnout

(20 points) Building several classifiers and comparing output.

a. Split the data into a training and test set (70/30).

```
mh <- read_csv("mental_health.csv") %>%
  drop_na()

set.seed(420)
mh_samples <- sample(1:nrow(mh),
                     nrow(mh)*0.7,
                     replace = FALSE)
mh_train <- mh[mh_samples, ]
mh_test <- mh[-mh_samples, ]
```

b. Using the training set and all important predictors, estimate the following models with vote96 as the response variable:

i. Logistic regression model

```
logit_mh <- glm(vote96 ~ mhealth_sum + age + educ + black + female + married + inc10, data = mh_
```

ii. Linear discriminant model

```
lda_mh <- MASS::lda(vote96 ~ mhealth_sum + age + educ + black + female + married + inc10, data =
```

iii. Quadratic discriminant model

```
qda_mh <- MASS::qda(vote96 ~ mhealth_sum + age + educ + black + female + married + inc10, data =
```

iv. Naive Bayes (you can use the default hyperparameter settings)

```
mh_train$vote96 <- as.factor(mh_train$vote96)
nb_mh <- naiveBayes(vote96 ~ mhealth_sum + age + educ + black + female + married + inc10, data =
```

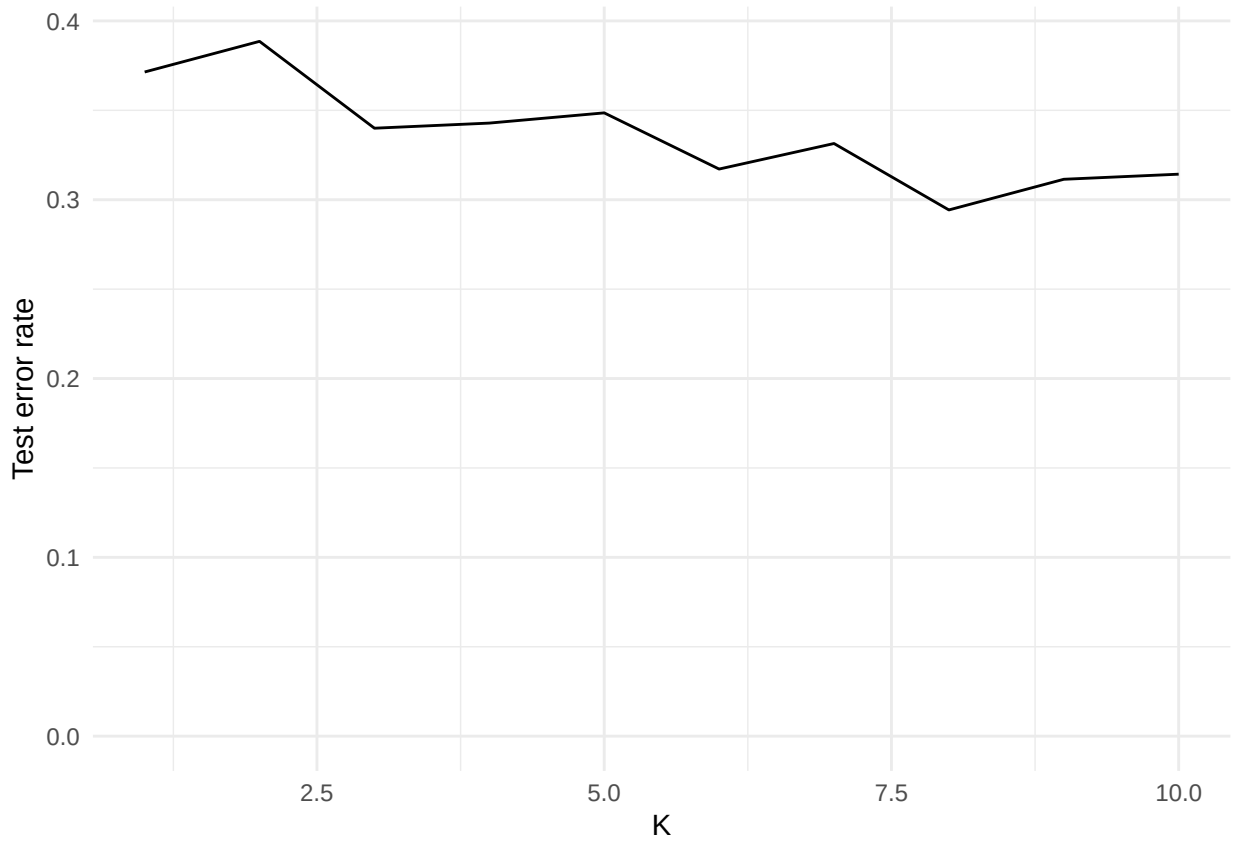
v. K-nearest neighbors with $K = 1, 2, \ldots, 10$ (that is, 10 separate models varying K) and Euclidean distance metrics

```
knn_models <- tibble(k = 1:10,
                     knn_train = map(k, ~ knn(dplyr::select(mh_train, -vote96),
                                              test = dplyr::select(mh_train, -vote96),
                                              cl = mh_train$vote96, k = .)),
                     knn_test = map(k, ~ knn(dplyr::select(mh_train, -vote96),
                                             test = dplyr::select(mh_test, -vote96),
                                             cl = mh_train$vote96, k = .)),
                     err_train = map_dbl(knn_train, ~ mean(mh_test$vote96 != .)),
                     err_test = map_dbl(knn_test, ~ mean(mh_test$vote96 != .)))
```

c. Using the test set, calculate the following model performance metrics:

i. Error rate

```
# logit error rate
prd_logit <- predict(logit_mh, newdata = mh_test)
mh_logit_error <- mean(mh_test$vote96)

# lda error rate
prd_lda <- predict(lda_mh, newdata = mh_test)
mh_lda_error <- mean(mh_test$vote96 != prd_lda$class)

# qda error rate
prd_qda <- predict(qda_mh, newdata = mh_test)
mh_qda_error <- mean(mh_test$vote96 != prd_qda$class)

# naive bayes error rate
bayes_predict <- predict(nb_mh, mh_test)
mh_nb_error <- mean(mh_test$vote96 != bayes_predict)

## Knn
# plotting k values
mh_knn_error <- mean(knn_models$err_test)
ggplot(knn_models, aes(k, err_test)) +
  geom_line() +
  labs(x = "K",
       y = "Test error rate") +
  expand_limits(y = 0) +
  theme_minimal()
```




```
error_table <- bind_rows(data.frame(mh_logit_error, mh_lda_error, mh_qda_error, mh_nb_error, mh_
error_table
```

```
##   mh_logit_error mh_lda_error mh_qda_error mh_nb_error mh_knn_error
## 1      0.6714286    0.2571429          0.3   0.3028571        0.336
```

d. Which model performs the best? Be sure to define what you mean by "best" and identify supporting evidence to support your conclusion(s).

As seen in the table above, the error rate of LDA is the smallest of all, therefore potentially being the best performing model. However, the quality of model is not solely dependent on the error rate; we have to consider the accuracy rate as well. That is, the model should be consistent across different samples of data and explain the data accurately.