Takuyo_Ozaki_HW2

Takuyo Ozaki

2/2/2020

```
knitr::opts_chunk$set(message = FALSE, warning = FALSE)
knitr::opts_chunk$set(fig.width = 7, fig.height = 3, fig.align = 'center')

# load necessary package
library(tidyverse)
library(broom)
library(rsample)
library(caret)
library(MASS)
library(tibble)
library(rcfss)
library(knitr)
library(yardstick)
library(e1071)
library(class)
library(ROCR)

# Set the theme as minimal
theme_set(theme_minimal())
```

The Bayes Classifier

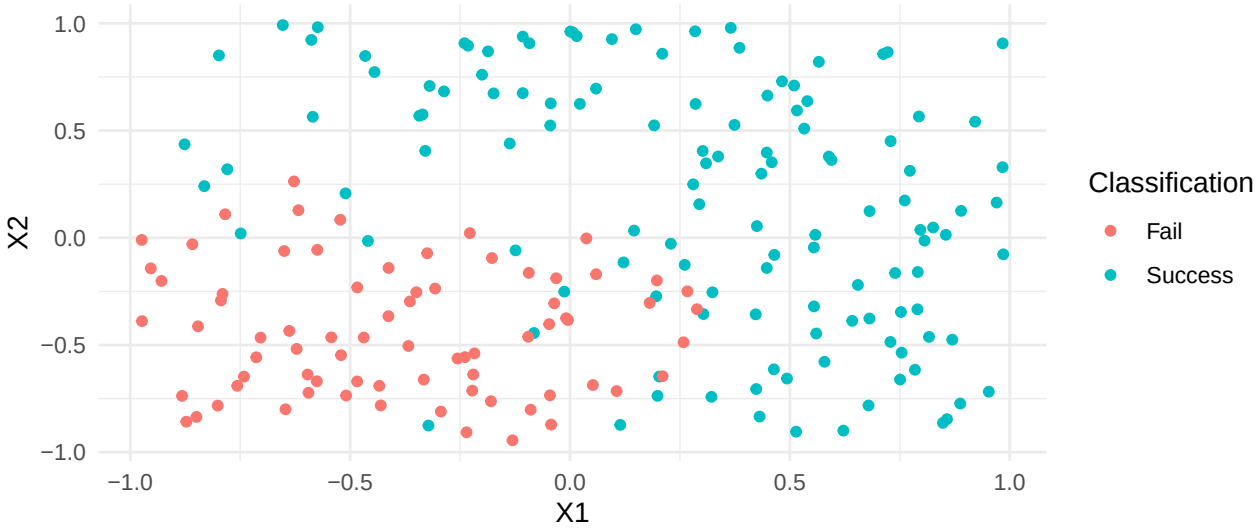
Question 1

```
# a. Set the randome number generator seed
set.seed(0)

# b. Simulate a dataset of N=200 with X1, X2: randome uniform variables [-1, 1]
n1 <- 200
X1 <- runif(n1, min = -1, max = 1)
X2 <- runif(n1, min = -1, max = 1)
predictor <- data.frame(X1, X2)

# c. Calculate Y = X1 + X1^2 + X2 + X2^2 + epsilon ~ N(mu=0, sigma^2=0.25)
epsilon <- rnorm(n1, mean = 0, sd = 0.25)
Y <- X1 + X1^2 + X2 + X2^2 + epsilon
data1 <- cbind(predictor, Y)

# d. Convert Y from the log-odds to the probability
data1$prob <- exp(Y) / (1 + exp(Y))
head(data1) #Check the probability
```

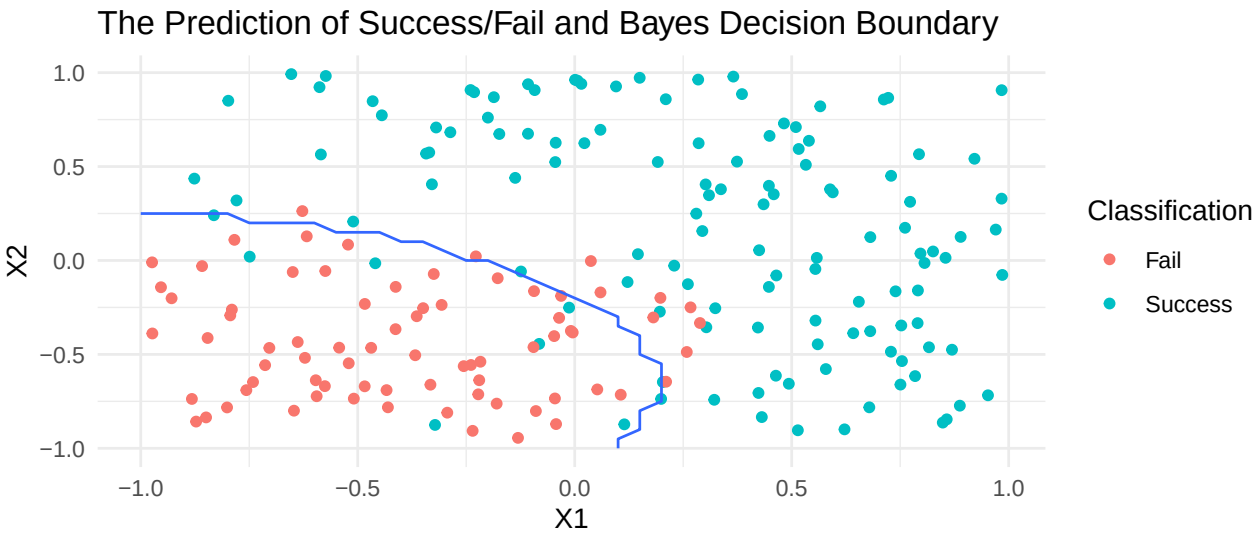
```
##           X1          X2           Y      prob
## 1  0.7933944  0.56570267  2.04716185 0.8856605
```

```
## 2 -0.4689827 -0.46498359 -0.72186459 0.3269825
## 3 -0.2557522 -0.56270943 -0.11906375 0.4702692
## 4  0.1457067  0.03359367  0.35011962 0.5866466
## 5  0.8164156 -0.46209882  1.42829506 0.8066355
## 6 -0.5966361 -0.63766335 -0.08236767 0.4794197
```

```r
# e. Plot the data point (success = blue, failure = red)
data1 <- data1 %>%
  mutate(success = ifelse(prob > 0.5, 1, 0)) %>%
  mutate(Classification = ifelse(prob > 0.5, "Success", "Fail"))
data1 %>%
  ggplot(aes(x = X1, y = X2, color = Classification)) +
  geom_point()
```




```r
# Generate naive bayes model
nb.m1 <- train(x = predictor, y = as.factor(data1$success), method = "nb")

# Create the grid of X1 and X2
d1 <- expand.grid(X1 = seq(-1, 1, 0.05), X2 = seq(-1, 1, 0.05))

# Predict success or fail using the model and the grid
d1$success <- predict(nb.m1, newdata = d1)

# f. Overlay the plot with Bayes decision boundary
# g. Give your plot a meaningful title and axis labels
# h. Colored background grid is optional -> I will not draw it
ggplot() +
  geom_point(data = data1, aes(x = X1, y = X2, color = Classification)) +
  geom_contour(data = d1, aes(x = X1, y = X2, z = as.numeric(success)), bins = 1) +
  labs(title = "The Prediction of Success/Fail and Bayes Decision Boundary")
```



Exploring Simulated Differences between LDA and QDA

Question 2

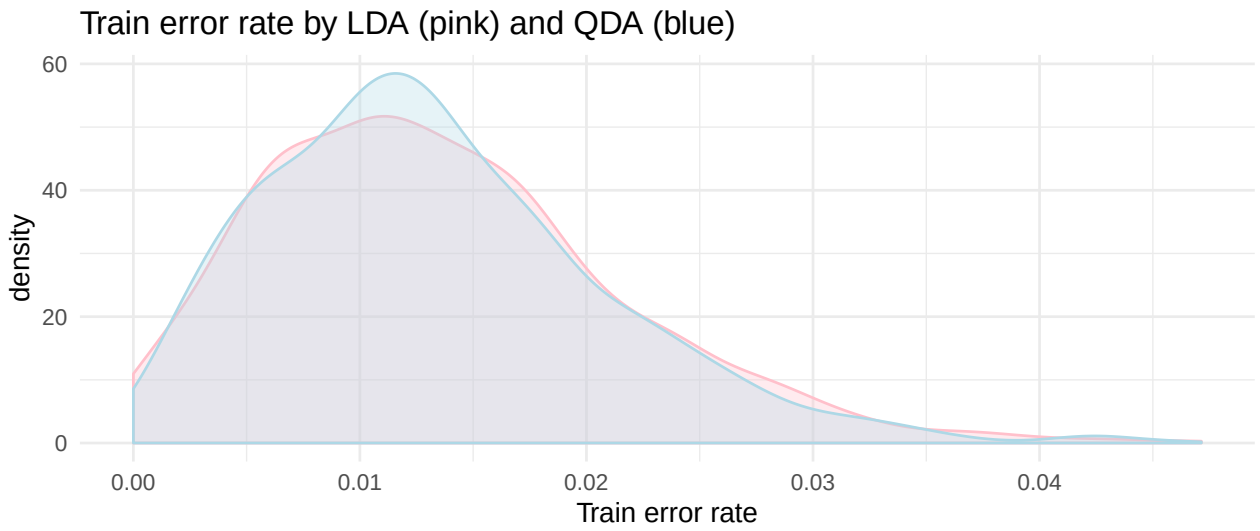
```r
set.seed(1234) #Ensure reproducibility

# Prepare the vector for loop
lda_train_error1 <- rep(NA, 1000)
lda_test_error1 <- rep(NA, 1000)
qda_train_error1 <- rep(NA, 1000)
qda_test_error1 <- rep(NA, 1000)

# Sample size
n2 <- 1000

# Simulate 1000 times to calcurate train/test error with LDA/QDA
for (i in 1:1000) {
  x1 <- runif(n2, min = -1, max = 1)
  x2 <- runif(n2, min = -1, max = 1)
  positive <- (x1 + x2) > 0
  y <- x1 + x2 + rnorm(n2, 0, 1) #Linear model
  df2 <- data.frame(x1, x2, positive, y)
  split <- initial_split(df2, prop = .7)
  train <- training(split) #70% train
  test <- testing(split) #30% test
  lda_m1 <- lda(as.factor(positive) ~ x1 + x2, data = train) #Use train
  lda_m1_train_accuracy <- train %>%
    mutate(.pred = predict(lda_m1, train)$class) %>%
    accuracy(truth = as.factor(positive), estimate = .pred)
  lda_train_error1[i] <- 1 - lda_m1_train_accuracy$.estimate
  lda_m1_test_accuracy <- test %>%
    mutate(.pred = predict(lda_m1, test)$class) %>%
    accuracy(truth = as.factor(positive), estimate = .pred)
  lda_test_error1[i] <- 1 - lda_m1_test_accuracy$.estimate
```

```r
  qda_m1 <- qda(as.factor(positive) ~ x1 + x2, data = train)
  qda_m1_train_accuracy <- train %>%
    mutate(.pred = predict(qda_m1, train)$class) %>%
    accuracy(truth = as.factor(positive), estimate = .pred)
  qda_train_error1[i] <- 1 - qda_m1_train_accuracy$.estimate
  qda_m1_test_accuracy <- test %>%
    mutate(.pred = predict(qda_m1, test)$class) %>%
    accuracy(truth = as.factor(positive), estimate = .pred)
  qda_test_error1[i] <- 1 - qda_m1_test_accuracy$.estimate
}

# Calcurate the average train/test LDA/QDA error
df_q2 <- data.frame(lda_train_error1, qda_train_error1, lda_test_error1, qda_test_error1)
kable(summarise_each(df_q2, mean), digits = 4,
      caption = "Average train error and test error with LDA and QDA")
```

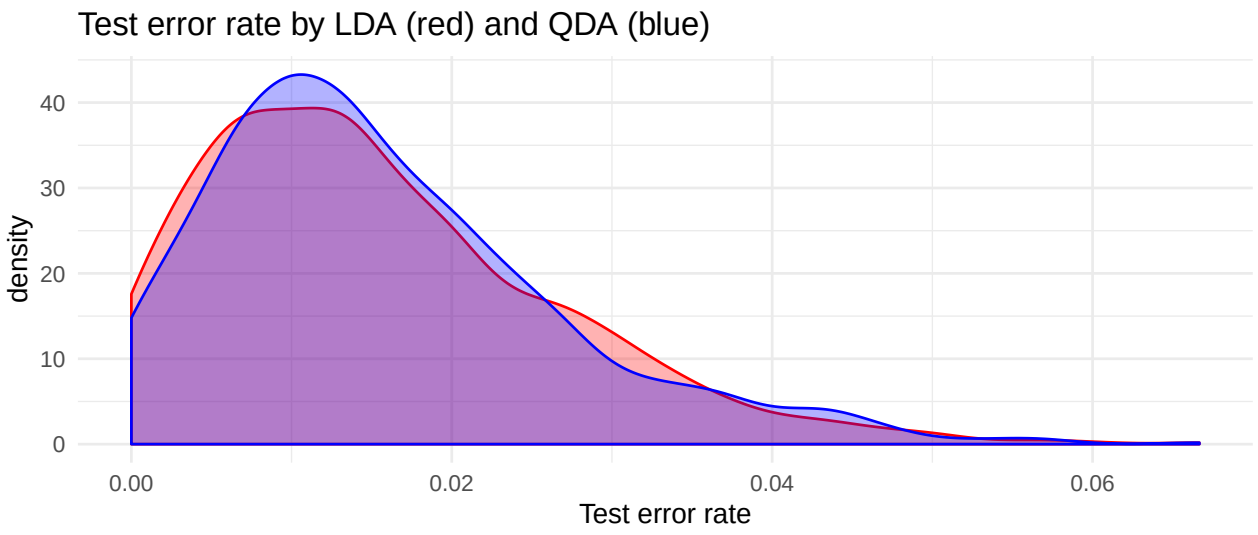
Table 1: Average train error and test error with LDA and QDA

lda_train_error1	qda_train_error1	lda_test_error1	qda_test_error1
0.0134	0.0132	0.0154	0.0157

```r
# Plot the train error with LDA/QDA
ggplot(df_q2) +
  geom_density(aes(x = lda_train_error1), color = "pink", fill = "pink", alpha = 0.3) +
  geom_density(aes(x = qda_train_error1), color = "light blue", fill = "light blue", alpha = 0.3) +
  labs(title = "Train error rate by LDA (pink) and QDA (blue)", x = "Train error rate")
```




```r
# Plot the test error with LDA/QDA
ggplot(df_q2) +
  geom_density(aes(x = lda_test_error1), color = "red", fill = "red", alpha = 0.3) +
  geom_density(aes(x = qda_test_error1), color = "blue", fill = "blue", alpha = 0.3) +
  labs(title = "Test error rate by LDA (red) and QDA (blue)", x = "Test error rate")
```



If the Bayes decision boundary is linear, - on the training set, we expect QDA to perform better than LDA because the higher flexiblity could fit the model. The table above shows that the average training error rate of QDA (0.132) is smaller than the LDA (0.134). Also, the graph above shows that LDA test error distribution has a right fatter tail. These evidences support QDA is expected to perform better on the training set. - on the test set, we expect LDA to perform better than QDA because QDA could overfit the linear Bayes decision boundary. The table above shows that the average test error rate of LDA (0.154) is smaller than the QDA(0.157). Also, the graph above shows that QDA test error distribution has a right fatter tail. These evidences support LDA is expected to perform better on the test set.

Question 3

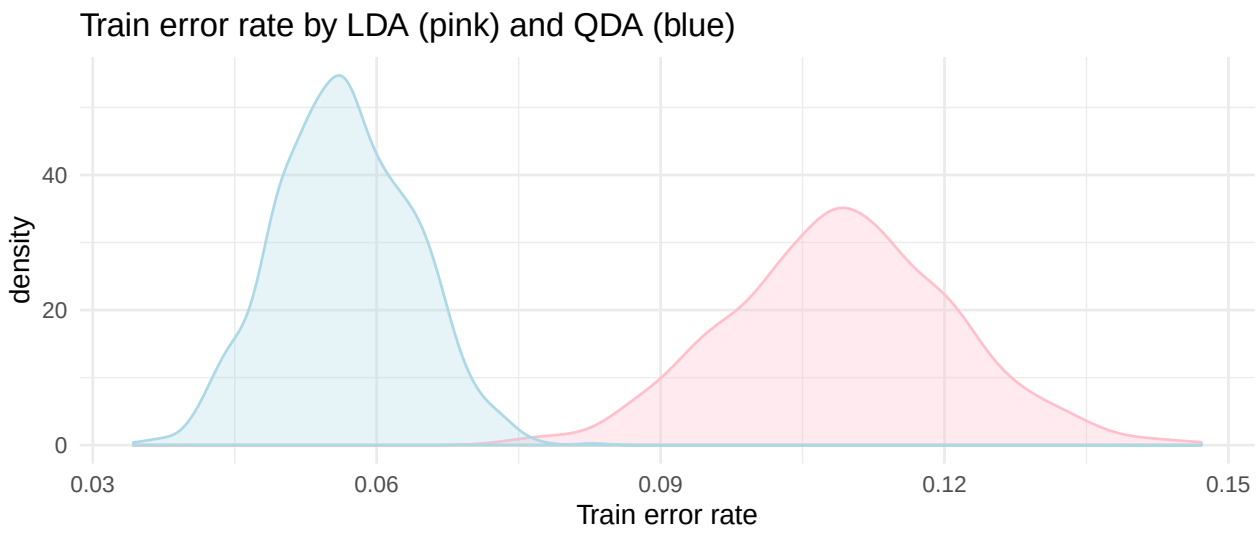
```r
set.seed(1234) #Ensure reproducibility

# Prepare the vector fo loop result
lda_train_error2 <- rep(NA, 1000)
lda_test_error2 <- rep(NA, 1000)
qda_train_error2 <- rep(NA, 1000)
qda_test_error2 <- rep(NA, 1000)

# Simulate 1000 times to calcurate train/test error with LDA/QDA
for (i in 1:1000) {
  x1 <- runif(n2, min = -1, max = 1)
  x2 <- runif(n2, min = -1, max = 1)
  positive <- (x1 + x1^2 + x2 + x2^2) > 0
  y <- x1 + x1^2 + x2 + x2^2 + rnorm(n2, 0, 1) #Non-Linear
  df2 <- data.frame(x1, x2, positive, y)
  split <- initial_split(df2, prop = .7)
  train <- training(split) #70% training
  test <- testing(split) #30% test
  lda_m2 <- lda(as.factor(positive) ~ x1 + x1^2 + x2 + x2^2, data = train)
  lda_m2_train_accuracy <- train %>%
    mutate(.pred = predict(lda_m2, train)$class) %>%
    accuracy(truth = as.factor(positive), estimate = .pred)
```

```
  lda_train_error2[i] <- 1 - lda_m2_train_accuracy$.estimate
  lda_m2_test_accuracy <- test %>%
    mutate(.pred = predict(lda_m2, test)$class) %>%
    accuracy(truth = as.factor(positive), estimate = .pred)
  lda_test_error2[i] <- 1 - lda_m2_test_accuracy$.estimate
  qda_m2 <- qda(as.factor(positive) ~ x1 + x1^2 + x2 + x2^2, data = train)
  qda_m2_train_accuracy <- train %>%
    mutate(.pred = predict(qda_m2, train)$class) %>%
    accuracy(truth = as.factor(positive), estimate = .pred)
  qda_train_error2[i] <- 1 - qda_m2_train_accuracy$.estimate
  qda_m2_test_accuracy <- test %>%
    mutate(.pred = predict(qda_m2, test)$class) %>%
    accuracy(truth = as.factor(positive), estimate = .pred)
  qda_test_error2[i] <- 1 - qda_m2_test_accuracy$.estimate
}

# Calcurate the average train/test LDA/QDA error
df_q3 <- data.frame(lda_train_error2, qda_train_error2, lda_test_error2, qda_test_error2)
kable(summarise_each(df_q3, mean), digits = 4,
      caption = "Average train error and test error with LDA and QDA")
```

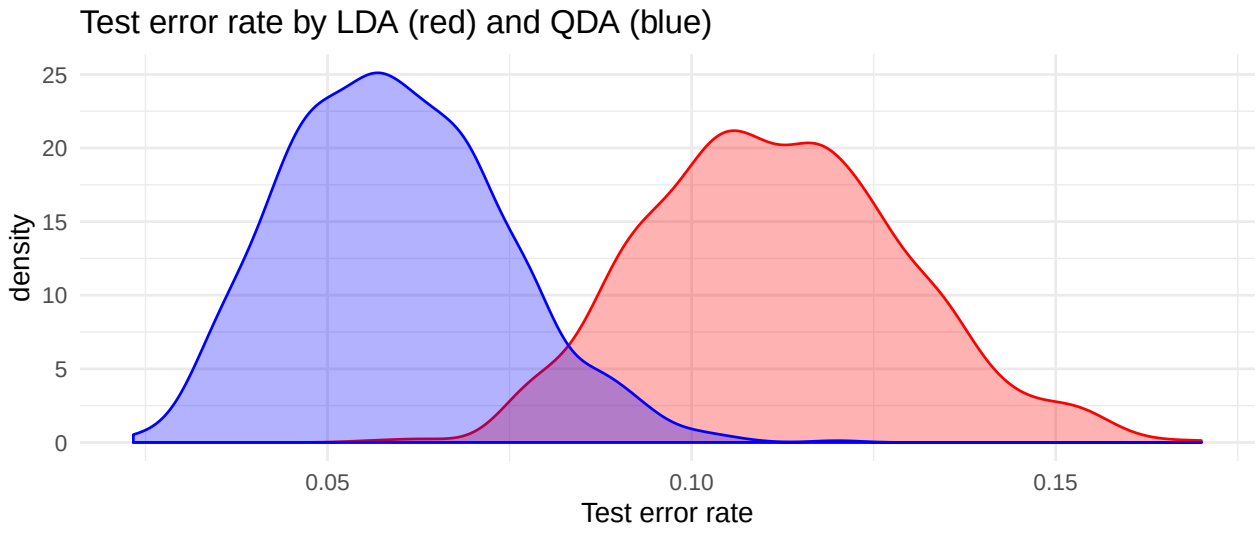
Table 2: Average train error and test error with LDA and QDA

lda_train_error2	qda_train_error2	lda_test_error2	qda_test_error2
0.1092	0.0566	0.1115	0.0591

```
# Plot the train error with LDA/QDA
ggplot(df_q3) +
  geom_density(aes(x = lda_train_error2), color = "pink", fill = "pink", alpha = 0.3) +
  geom_density(aes(x = qda_train_error2), color = "light blue", fill = "light blue", alpha = 0.3) +
  labs(title = "Train error rate by LDA (pink) and QDA (blue)", x = "Train error rate")
```



```r
# Plot the test error with LDA/QDA
ggplot(df_q3) +
  geom_density(aes(x = lda_test_error2), color = "red", fill = "red", alpha = 0.3) +
  geom_density(aes(x = qda_test_error2), color = "blue", fill = "blue", alpha = 0.3) +
  labs(title = "Test error rate by LDA (red) and QDA (blue)", x = "Test error rate")
```

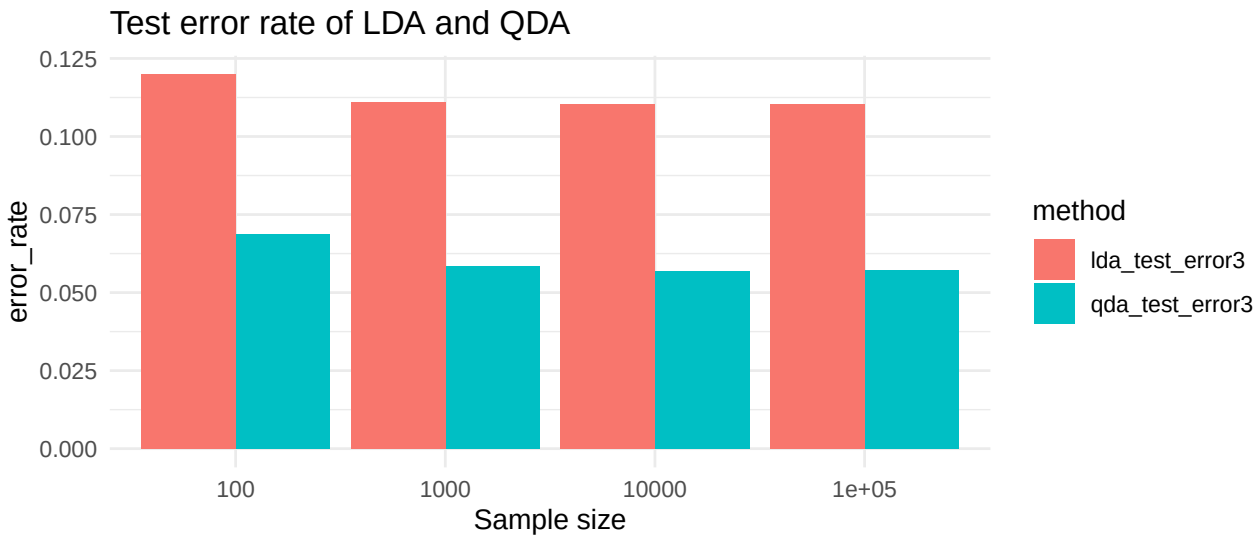


If the Bayes decision boundary is non-linear, we expect QDA to perform better than LDA both on the the training set and the test set because we do not need to care about the overfitting the model's linearity like the question 2. The table above shows that the average error rates of QDA (train:0.0566, test:0.0591) are smaller than the LDA (train:0.1092, test:0.1115) on both the training and the test set. Also, the graph above shows that QDA's error distributions are in the left position compared to LDA's. These evidences support QDA is expected to perform better both on the training and test set.

Question 4

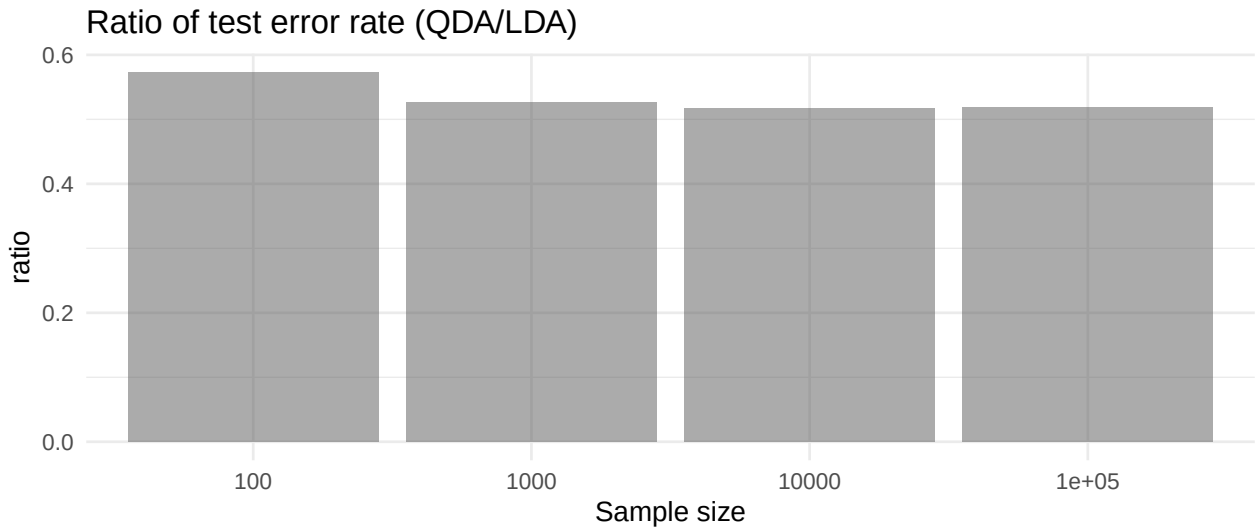
```r
set.seed(1234)

# Generate the function of the simulation by each sample size
sample_size <- function(n){
  lda_test_error3 <- rep(NA, 1000)
  qda_test_error3 <- rep(NA, 1000)
for (i in 1:1000) {
  x1 <- runif(n, min = -1, max = 1)
  x2 <- runif(n, min = -1, max = 1)
  positive <- (x1 + x1^2 + x2 + x2^2) > 0
  y <- x1 + x1^2 + x2 + x2^2 + rnorm(n, 0, 1) #Non-linear
  df2 <- data.frame(x1, x2, positive, y)
  split <- initial_split(df2, prop = .7)
  train <- training(split) #70% training
  test <- testing(split) #30% test
  lda_m3 <- lda(as.factor(positive) ~ x1 + x1^2 + x2 + x2^2, data = train)
  lda_m3_test_accuracy <- test %>%
```

```
    mutate(.pred = predict(lda_m3, test)$class) %>%
    accuracy(truth = as.factor(positive), estimate = .pred)
  lda_test_error3[i] <- 1 - lda_m3_test_accuracy$.estimate
  qda_m3 <- qda(as.factor(positive) ~ x1 + x1^2 + x2 + x2^2, data = train)
  qda_m3_test_accuracy <- test %>%
    mutate(.pred = predict(qda_m3, test)$class) %>%
    accuracy(truth = as.factor(positive), estimate = .pred)
  qda_test_error3[i] <- 1 - qda_m3_test_accuracy$.estimate
}
  df_q4 <- data.frame(lda_test_error3, qda_test_error3)
  df_q4 <- df_q4 %>%
    summarise_each(mean) %>%
    mutate(sample_size = n)
  return(df_q4)
}

# Create the dataframe to summarize the result
df1e02 <- sample_size(100)
df1e03 <- sample_size(1000)
df1e04 <- sample_size(10000)
df1e05 <- sample_size(100000)
df_q4 <- rbind(df1e02, df1e03, df1e04, df1e05)

# Plot the test error rate of LDA and QDA by each sample size
df_q4 %>%
  gather(key = method, value = error_rate, -sample_size) %>%
  ggplot(aes(x = as.factor(sample_size), y = error_rate, fill = method)) +
  geom_col(position = "dodge") +
  labs(title = "Test error rate of LDA and QDA", x = "Sample size")
```




```
# Plot the ratio of test error rate of QDA/LDA by each sample size
df_q4 %>%
  mutate(ratio = qda_test_error3/lda_test_error3) %>%
  ggplot(aes(x = as.factor(sample_size), y = ratio)) +
```

```
  geom_bar(stat = "identity", alpha = 0.5) +
  labs(title = "Ratio of test error rate (QDA/LDA) ", x = "Sample size")
```



In general, as sample size n increases, we can expect the test error rate of QDA relative to LDA could decrease. Usually, a more flexible method (QDA) can have smaller bias but larger variance. However, larger sample size can have an effect on the decrease in variance (or increase not so much), which can decrease the test error rate of QDA relative to LDA. In fact, as the sample size increases, while the test error rate decreases both on the LDA and QDA, the test error rate of QDA relative to LDA (= QDA/LDA) also decreases (i.e., QDA's test error rate basically decreases more than LDA as the sample size increases).

Modeling voter turnout

Question 5

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

```
# Imporet the data
data <- read_csv("mental_health.csv")
data$vote96 <- as.factor(data$vote96)

# Drop the missing value as TA suggests in Piazza
data <- data %>% drop_na()

# a. Split the data into a training and test set (70/30)
split <- initial_split(data, prop = .7)
train <- training(split)
test <- testing(split)
```

b. estimate the models with vote96

```
# Create response and feature data
features <- setdiff(names(train), "vote96")
x <- train[, features]
y <- as.factor(train$vote96)

# Estimate the model: Logistic regression
(logit_m5 <- glm(vote96 ~., data = train, family = binomial))
```

```
##
## Call:  glm(formula = vote96 ~ ., family = binomial, data = train)
##
## Coefficients:
## (Intercept)  mhealth_sum          age         educ        black
##    -4.52619     -0.13042      0.04832      0.24393      0.18437
##      female      married        inc10
##     0.04471      0.36054      0.04297
##
## Degrees of Freedom: 815 Total (i.e. Null);  808 Residual
## Null Deviance:       1018
## Residual Deviance: 841.2     AIC: 857.2
```

```
# Estimate the model: LDA
(lda_m5 <- lda(vote96 ~., data = train))
```

```
## Call:
## lda(vote96 ~ ., data = train)
##
## Prior probabilities of groups:
##         0         1
## 0.3161765 0.6838235
##
## Group means:
##   mhealth_sum      age     educ     black    female   married    inc10
## 0    3.841085 39.27519 12.32946 0.1705426 0.5310078 0.3759690 3.473171
## 1    2.225806 48.05018 13.89785 0.1344086 0.5376344 0.5304659 5.069100
##
## Coefficients of linear discriminants:
##                      LD1
## mhealth_sum -0.14398682
## age          0.04352212
## educ         0.21630986
## black        0.17503985
## female       0.08325976
## married      0.36396200
## inc10        0.03144128
```

```
# Estimate the model: QDA
(qda_m5 <- qda(vote96 ~., data = train))
```

```
## Call:
```

```
## qda(vote96 ~ ., data = train)
##
## Prior probabilities of groups:
##         0         1
## 0.3161765 0.6838235
##
## Group means:
##   mhealth_sum      age     educ     black    female   married    inc10
## 0    3.841085 39.27519 12.32946 0.1705426 0.5310078 0.3759690 3.473171
## 1    2.225806 48.05018 13.89785 0.1344086 0.5376344 0.5304659 5.069100
```

```
# Estimate the model: Naive Bayes
(nb_m5 <- naiveBayes(vote96 ~., data = train))
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##         0         1
## 0.3161765 0.6838235
##
## Conditional probabilities:
##    mhealth_sum
## Y       [,1]     [,2]
##   0 3.841085 3.237959
##   1 2.225806 2.524405
##
##    age
## Y       [,1]     [,2]
##   0 39.27519 15.44457
##   1 48.05018 15.81131
##
##    educ
## Y       [,1]     [,2]
##   0 12.32946 2.589260
##   1 13.89785 2.994662
##
##    black
## Y        [,1]      [,2]
##   0 0.1705426 0.3768398
##   1 0.1344086 0.3413968
##
##    female
## Y        [,1]      [,2]
##   0 0.5310078 0.5000075
##   1 0.5376344 0.4990290
##
##    married
## Y        [,1]      [,2]
##   0 0.3759690 0.4853135
```

```
##   1 0.5304659 0.4995188
##
##    inc10
## Y       [,1]     [,2]
##   0 3.473171 2.791448
##   1 5.069100 3.926546
```

```
# Estimate the model: K-NN
(knn_m5 <- train(x = x, y = y, method = "knn",
                 tuneGrid = expand.grid(k = 1:10)))
```

```
## k-Nearest Neighbors
##
## 816 samples
##   7 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 816, 816, 816, 816, 816, 816, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.6445831  0.1850947
##    2  0.6473278  0.1873601
##    3  0.6472576  0.1778777
##    4  0.6486251  0.1739259
##    5  0.6519990  0.1721077
##    6  0.6612423  0.1868204
##    7  0.6688270  0.1967447
##    8  0.6677235  0.1878936
##    9  0.6782385  0.2094497
##   10  0.6792329  0.2078637
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 10.
```

The best model among K-nearest neighbors with K $= 1, 2, \ldots, 10$ is k $= 10$.

b. Error rate

```
# i. Logit
logit_accuracy <- augment(logit_m5, newdata = test) %>%
  as_tibble() %>%
  mutate(.prob = logit2prob(.fitted),
         .pred = factor(round(.prob))) %>%
  accuracy(truth = vote96, estimate = .pred)
(logit_error <- 1 - logit_accuracy$.estimate)
```

```
## [1] 0.2836676
```

```r
# ii. LDA
lda_accuracy <- test %>%
  mutate(.pred = predict(lda_m5, test)$class) %>%
  accuracy(truth = vote96, estimate = .pred)
(lda_error <- 1 - lda_accuracy$.estimate)
```

```
## [1] 0.2951289
```

```r
# iii. QDA
qda_accuracy <- test %>%
  mutate(.pred = predict(qda_m5, test)$class) %>%
  accuracy(truth = vote96, estimate = .pred)
(qda_error <- 1 - qda_accuracy$.estimate)
```

```
## [1] 0.2836676
```

```r
# iv: Naive Bayes
nb_accuracy <- test %>%
  mutate(.pred = predict(nb_m5, test)) %>%
  accuracy(truth = vote96, estimate = .pred)
(nb_error <- 1 - nb_accuracy$.estimate)
```

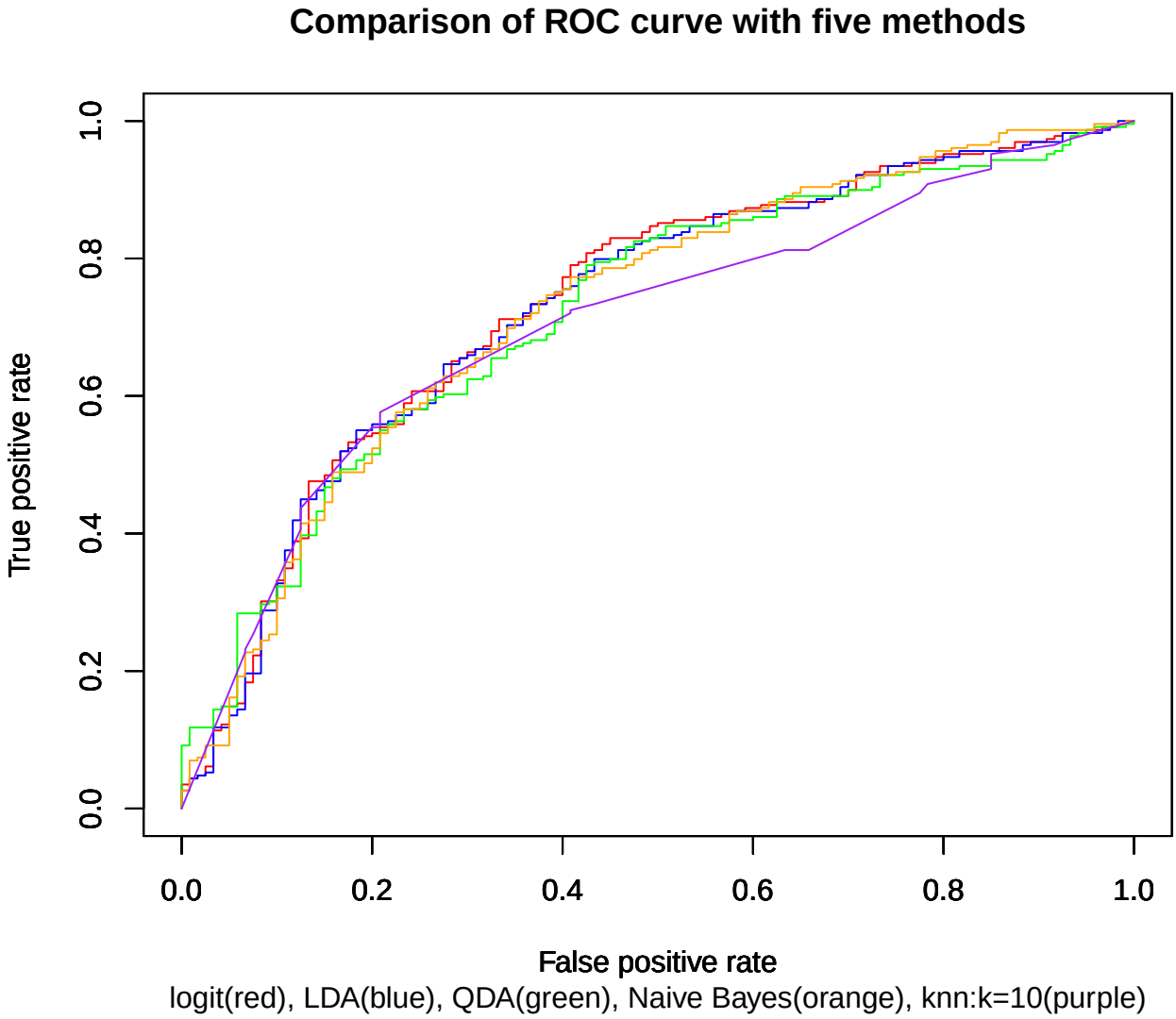
```
## [1] 0.2979943
```

```r
# v: knn
mse_knn <- tibble(k = 1:10,
                  knn_test = map(k, ~ knn(train = dplyr::select(train, -vote96),
                                          test  = dplyr::select(test, -vote96),
                                          cl = train$vote96, k = .)),
                  err_test = map_dbl(knn_test, ~ mean(test$vote96 != .)))
kable(dplyr::select(mse_knn, -knn_test))
```

k	err_test
1	0.3323782
2	0.3352436
3	0.3237822
4	0.3180516
5	0.3295129
6	0.3008596
7	0.3323782
8	0.3495702
9	0.3467049
10	0.3524355

ii. ROC

```
set.seed(0)

# Plot logit ROC
logit <- augment(logit_m5, newdata = test) %>% as_tibble() %>%
  mutate(.prob = logit2prob(.fitted))
logit_prob <- logit$.prob
logit_pred <- prediction(logit$.prob, test$vote96)
logit_perf <- performance(logit_pred, "tpr", "fpr")
plot(logit_perf, col = "red")
par(new = T)

# Plot LDA ROC
lda_prob <- predict(lda_m5, newdata = test, type = "prob")
lda_pred <- prediction(lda_prob$posterior[,2], test$vote96)
lda_perf <- performance(lda_pred, "tpr", "fpr")
plot(lda_perf, col = "blue")
par(new = T)

# Plot QDA ROC
qda_prob <- predict(qda_m5, newdata = test, type = "prob")
qda_pred <- prediction(qda_prob$posterior[,2], test$vote96)
qda_perf <- performance(qda_pred, "tpr", "fpr")
plot(qda_perf, col = "green")
par(new = T)

# Plot Naive Bayes ROC
nb_m5_prob <- train(x = x, y = y, method = "nb")
nb_prob <- predict(nb_m5_prob, newdata = test, type = "prob")
nb_pred <- prediction(nb_prob['1'], test$vote96)
nb_perf <- performance(nb_pred, "tpr", "fpr")
plot(nb_perf, col = "orange")
par(new = T)

# Plot knn (k = 10) ROC
knn_prob <- predict(knn_m5, newdata = test, type = "prob")
knn_pred <- prediction(knn_prob['1'], test$vote96)
kn_perf <- performance(knn_pred, "tpr", "fpr")
plot(kn_perf, col = "purple", main = "Comparison of ROC curve with five methods",
     sub = "logit(red), LDA(blue), QDA(green), Naive Bayes(orange), knn:k=10(purple)")
```



Note: I plot knn ROC only in case of K = 10, because this case is the best among k =1,2,..10 and also it could be messy if I plot all the other worse case.

ii. AUC

```
# Caluculate logit AUC
logit_auc <- performance(logit_pred, measure = "auc")
logit_auc@y.values
```

```
## [[1]]
## [1] 0.734607
```

```
# Caluculate LDA AUC
lda_auc <- performance(lda_pred, measure = "auc")
lda_auc@y.values
```

```
## [[1]]
## [1] 0.7299854
```

```
# Caluculate QDA AUC
qda_auc <- performance(qda_pred, measure = "auc")
qda_auc@y.values
```

```
## [[1]]
## [1] 0.7222707
```

```
# Caluculate Naive Bayes AUC
nb_auc <- performance(nb_pred, measure = "auc")
nb_auc@y.values
```

```
## [[1]]
## [1] 0.7276201
```

```
# Caluculate knn:k=10 AUC
knn_auc <- performance(knn_pred, measure = "auc")
knn_auc@y.values
```

```
## [[1]]
## [1] 0.7049309
```

d "best" model

I think the best model is the logistic regression model for the following two reasons.

- First, in terms of prediction, the logistic regression model is the best because it has the smallest error rate and the largest AUC among the five models (although the difference is slight).
- Second, in terms of interpretation, the logit model is more intuitive than the other models: the coefficient is the effect of the independent variable on the "odds ratio".