Problem Set 2

Akira Masuda

2020/2/2

```
Course: MACS30100 Perspectives on Computational Modeling (Winter 2020)
Author: Akira Masuda (ID: alakira)
```

The Bayes Classifier

1.

a. Set seed

```r
# Set seed
set.seed(110)
```

b. Simulate a dataset

```r
x_1 = runif(200, -1, 1)
x_2 = runif(200, -1, 1)
```

c. Calculate Y

```r
ep = rnorm(200, 0, 0.25)
y = x_1 + x_1^2 + x_2 + x_2^2 + ep
```

d. Calculate the probability of success

The log-odds is calculated by the following equation.

$$\log\left(\frac{Pr(success)}{1 - Pr(success)}\right) = X_1 + X_1^2 + X_2 + X_2^2 + \epsilon$$

By transforming the equation, the probability of success is:

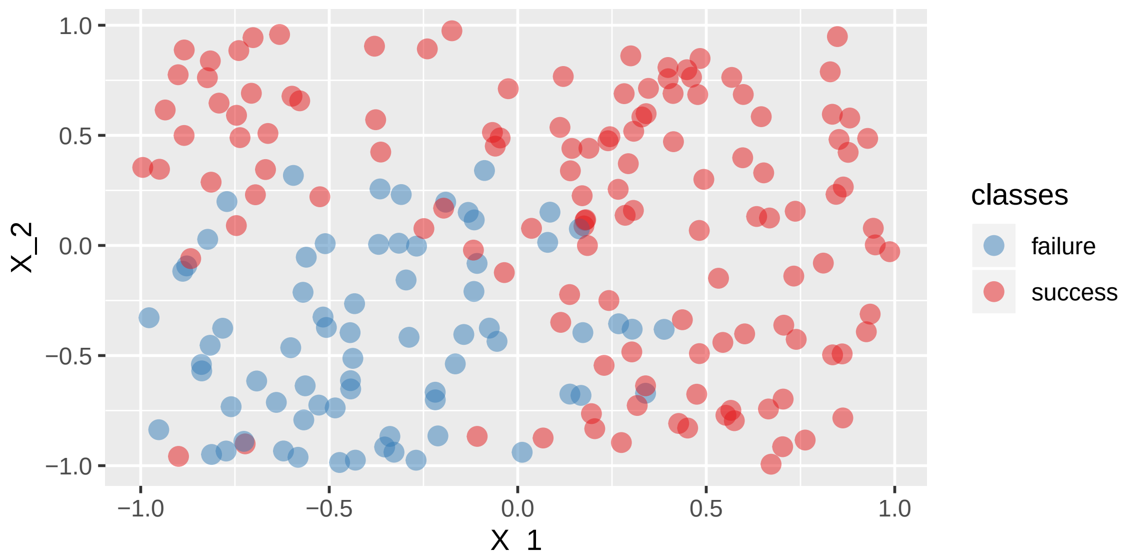
$$Pr(success) = \frac{e^{X_1 + X_1^2 + X_2 + X_2^2 + \epsilon}}{1 + e^{X_1 + X_1^2 + X_2 + X_2^2 + \epsilon}} = \frac{e^Y}{1 + e^Y}$$

```r
pr <- exp(y) / (1 + exp(y))
```

e. Plot each point from the dataset

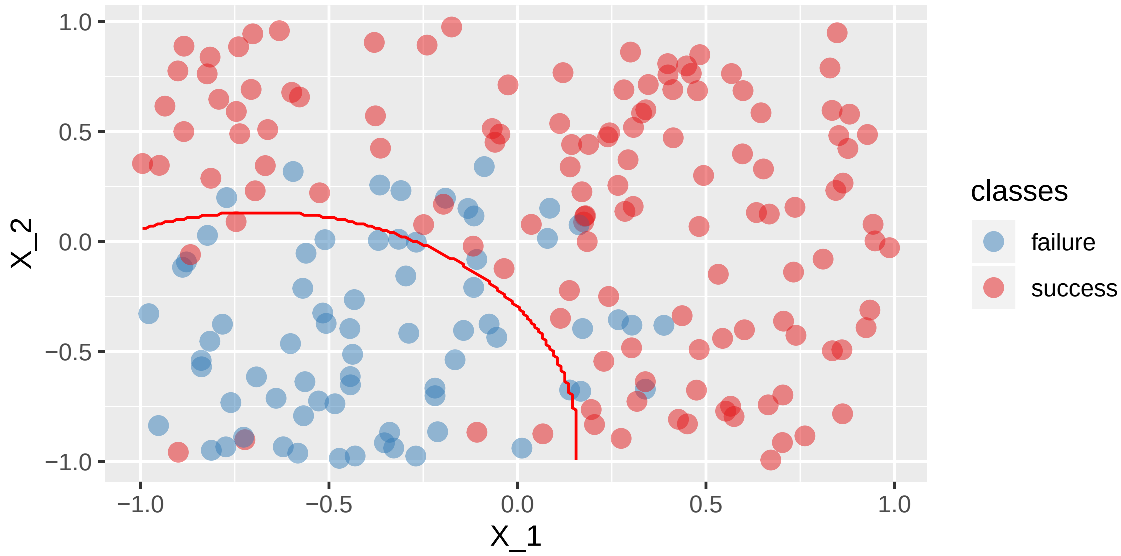
```r
# Dataframe
df <- data.frame(X_1 = x_1, X_2 = x_2, Pr = pr)
df <- df %>%
  mutate(cl = case_when(pr > 0.5 ~ 'success',
                        pr <=  0.5 ~ 'failure'),
         cl_n = case_when(pr > 0.5 ~ 1,
```

```
                          pr <= 0.5 ~0))
twoClassColor <- brewer.pal(3,'Set1')[1:2]
names(twoClassColor) <- c('success','failure')

df %>%
  ggplot(aes(x=X_1, y=X_2)) +
  geom_point(aes(color = cl), size = 3, alpha = .5) +
  scale_colour_manual(name = 'classes', values = twoClassColor)
```



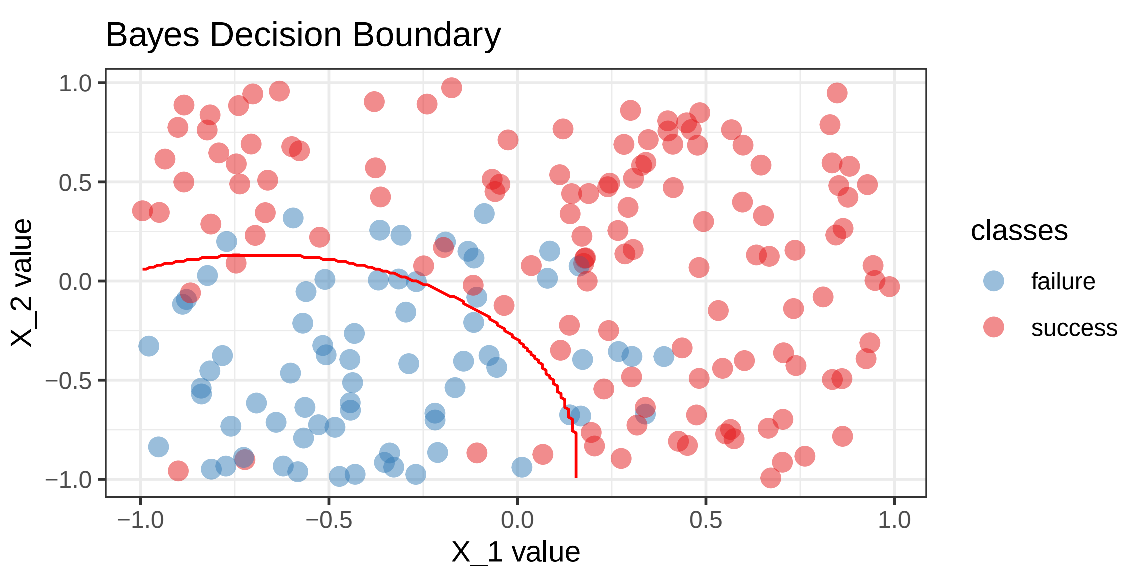
f. Overlay with Bayes decision boundary

```
# Refering to  http://www.cmap.polytechnique.fr/~lepennec/R/Learning/Learning.html
V <- 10
T <- 4
TrControl <- trainControl(method = "repeatedcv",
                          number = V,
                          repeats = T)
df_model <- df %>%
  dplyr::select(X_1, X_2, cl)
nbp = 200
Pred1 <- seq(min(df_model$X_1), max(df_model$X_1), length = nbp)
Pred2 <- seq(min(df_model$X_2), max(df_model$X_2), length = nbp)
Grid <- expand.grid(X_1 = Pred1, X_2 = Pred2)

Model <- train(data=df_model, cl ~ ., method = "nb", trControl = TrControl,
               tuneGrid = data.frame(usekernel = c(FALSE), fL = c(0), adjust = c(1)))
Pred <- predict(Model, newdata = Grid)

df_model %>%
  ggplot(aes(x=X_1, y=X_2)) +
  geom_point(aes(color = cl), size = 3, alpha = .5) +
  geom_contour(data = cbind(Grid, classes = Pred),
               aes(z = as.numeric(classes)),
               color = "red", breaks = c(1.5)) +
scale_colour_manual(name = 'classes', values = twoClassColor)
```



g. Title and axis labels & h. Colored Background

```
df_model %>%
  ggplot(aes(x=X_1, y=X_2)) +
  geom_point(aes(color = cl), size = 3, alpha = .5) +
  geom_contour(data = cbind(Grid, classes = Pred),
               aes(z = as.numeric(classes)),
               color = "red", breaks = c(1.5)) +
  scale_colour_manual(name = 'classes', values = twoClassColor) +
  ggtitle("Bayes Decision Boundary") +
  xlab('X_1 value') +
  ylab('X_2 value') +
  theme_bw()
```



Exploring Simulated Differences between LDA and QDA

2. In case Bayes decision boundary is linear

When the bayes decision boundary is linear, the performances of LDA and QDA will be similar on both training and test set. They would both perform well, but in general the performance for the training set is better than that of the test set.

a. Repeat simulation 1,000 times

```r
train_lda2 = c()
test_lda2 = c()
train_qda2 = c()
test_qda2 = c()

for (i in 1:1000) {
  # i data generation
  x_1 = runif(1000, -1, 1)
  x_2 = runif(1000, -1, 1)
  ep = rnorm(1000, 0, 0.25)
  y = case_when(x_1 + x_2 + ep >= 0 ~ TRUE,
                x_1 + x_2 + ep < 0 ~ FALSE)
  df2 <- data.frame(x_1, x_2, y)

  # ii split
  split <- initial_split(df2, prop = .7)
  train <- training(split)
  test <- testing(split)

  # iii training
  lda2 <- MASS::lda(y ~ x_1 + x_2, data = train)
  qda2 <- MASS::qda(y ~ x_1 + x_2, data = train)

  # iv error
  train_lda2 = c(train_lda2, sum(predict(lda2, train)$class != train$y) / 700)
  test_lda2 = c(test_lda2, sum(predict(lda2, test)$class != test$y) / 300)
  train_qda2 = c(train_qda2, sum(predict(qda2, train)$class != train$y) / 700)
  test_qda2 = c(test_qda2, sum(predict(qda2, test)$class != test$y) / 300)
}
```

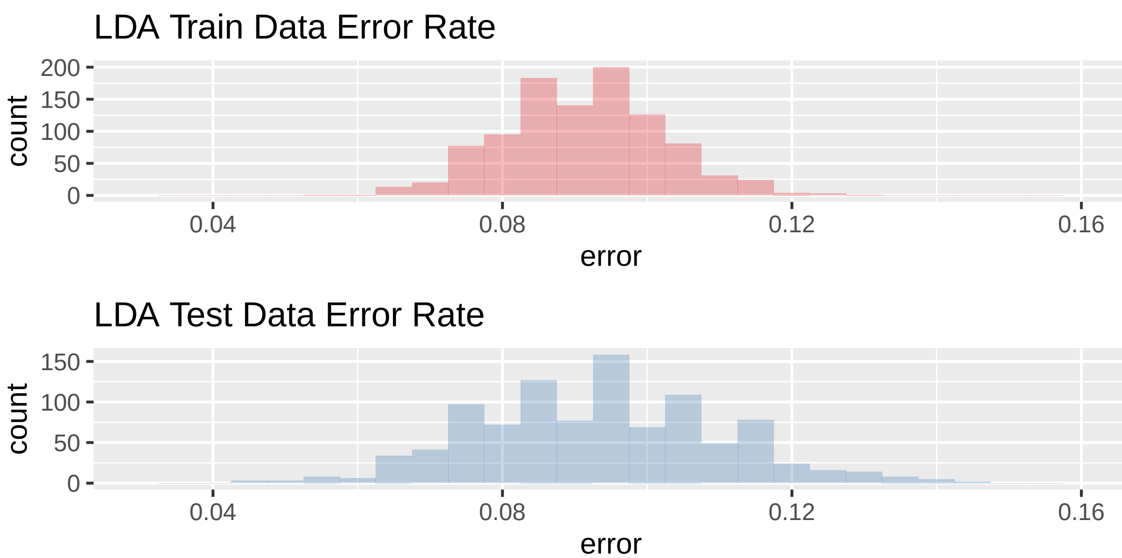
b. Summarize the results

```r
df2_result <- data.frame(train_lda2, test_lda2, train_qda2, test_qda2)
df2_result <- df2_result %>%
  gather(dset, error)
df2_result %>%
  group_by(dset) %>%
  summarise(mean = mean(error), sd = sd(error))
```

dset	mean	sd
test_lda2	0.0934167	0.0168609
test_qda2	0.0938367	0.0169069
train_lda2	0.0911400	0.0109330
train_qda2	0.0909057	0.0110357

```
lda_p1 <- df2_result %>%
  ggplot(aes(x=error)) +
  scale_colour_manual(name = 'classes', values = twoClassColor) +
  geom_histogram(data=subset(df2_result, dset == 'train_lda2'),
                 fill = brewer.pal(3,'Set1')[1],
                 alpha = 0.3, binwidth = 0.005) +
  scale_x_continuous(limits = c(0.03,0.16)) +
  ggtitle("LDA Train Data Error Rate")

lda_p2 <- df2_result %>%
  ggplot(aes(x=error)) +
  scale_colour_manual(name = 'classes', values = twoClassColor) +
  geom_histogram(data=subset(df2_result, dset == 'test_lda2'),
                 fill = brewer.pal(3,'Set1')[2],
                 alpha = 0.3, binwidth = 0.005) +
  scale_x_continuous(limits = c(0.03,0.16)) +
  ggtitle("LDA Test Data Error Rate")

grid.arrange(lda_p1, lda_p2, nrow = 2)
```

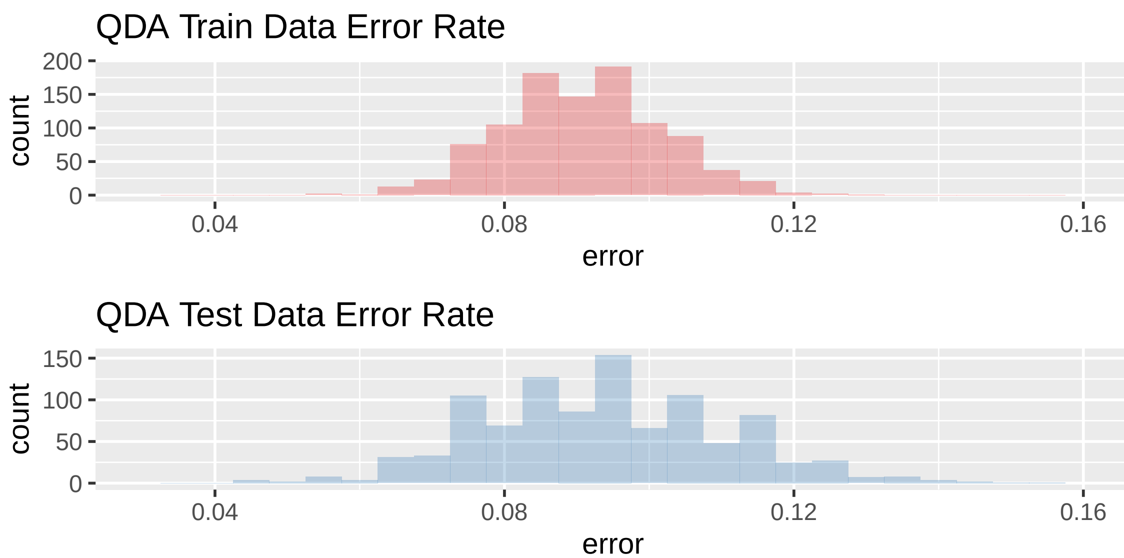
```
## Warning: Removed 2 rows containing missing values (geom_bar).

## Warning: Removed 2 rows containing missing values (geom_bar).
```




```
qda_p1 <- df2_result %>%
  ggplot(aes(x=error)) +
  scale_colour_manual(name = 'classes', values = twoClassColor) +
  geom_histogram(data=subset(df2_result, dset == 'train_qda2'),
                 fill = brewer.pal(3,'Set1')[1],
                 alpha = 0.3, binwidth = 0.005) +
  scale_x_continuous(limits = c(0.03,0.16)) +
  ggtitle("QDA Train Data Error Rate")

qda_p2 <- df2_result %>%
  ggplot(aes(x=error)) +
```

```
  scale_colour_manual(name = 'classes', values = twoClassColor) +
  geom_histogram(data=subset(df2_result, dset == 'test_qda2'),
                 fill = brewer.pal(3,'Set1')[2],
                 alpha = 0.3, binwidth = 0.005) +
  scale_x_continuous(limits = c(0.03,0.16)) +
  ggtitle("QDA Test Data Error Rate")

grid.arrange(qda_p1, qda_p2, nrow = 2)
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).

## Warning: Removed 2 rows containing missing values (geom_bar).
```



3. In case Bayes decision boundary is non-linear

When the bayes decision boundary is non-linear, the performances of QDA is higher than that of LDA on both training and test set. This is because QDA could fit the observation with its flexibility.

```
train_lda2 = c()
test_lda2 = c()
train_qda2 = c()
test_qda2 = c()

for (i in 1:1000) {
  # i data generation
  x_1 = runif(1000, -1, 1)
  x_2 = runif(1000, -1, 1)
  ep = rnorm(1000, 0, 0.25)
  y = case_when(x_1 + x_1^2 + x_2 + x_2^2 + ep >= 0 ~ TRUE,
                x_1 + x_1^2 + x_2 + x_2^2 + ep < 0 ~ FALSE)
  df2 <- data.frame(x_1, x_2, y)

  # ii split
  split <- initial_split(df2, prop = .7)
  train <- training(split)
  test <- testing(split)
```

```
  # iii training
  lda2 <- MASS::lda(y ~ x_1 + x_2, data = train)
  qda2 <- MASS::qda(y ~ x_1 + x_2, data = train)

    # iv error
  train_lda2 = c(train_lda2, sum(predict(lda2, train)$class != train$y) / 700)
  test_lda2 = c(test_lda2, sum(predict(lda2, test)$class != test$y) / 300)
  train_qda2 = c(train_qda2, sum(predict(qda2, train)$class != train$y) / 700)
  test_qda2 = c(test_qda2, sum(predict(qda2, test)$class != test$y) / 300)
}
```

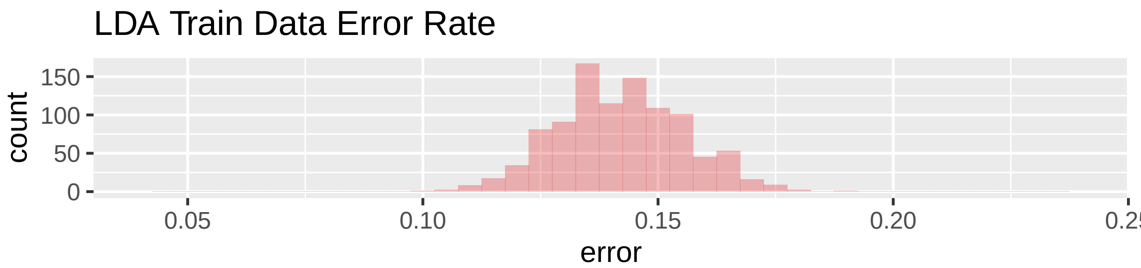
b. Summarize the results

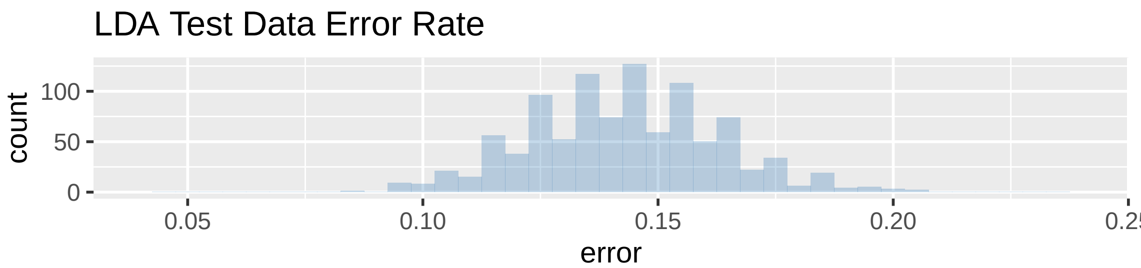
```
df2_result <- data.frame(train_lda2, test_lda2, train_qda2, test_qda2)
df2_result <- df2_result %>%
  gather(dset, error)
df2_result %>%
  group_by(dset) %>%
  summarise(mean = mean(error), sd = sd(error))
```

dset	mean	sd
test_lda2	0.1427233	0.0199135
test_qda2	0.1067600	0.0171807
train_lda2	0.1420243	0.0133372
train_qda2	0.1051986	0.0110349

```
lda_p1 <- df2_result %>%
  ggplot(aes(x=error)) +
  scale_colour_manual(name = 'classes', values = twoClassColor) +
  geom_histogram(data=subset(df2_result, dset == 'train_lda2'),
                 fill = brewer.pal(3,'Set1')[1],
                 alpha = 0.3, binwidth = 0.005) +
  scale_x_continuous(limits = c(0.04,0.24)) +
  ggtitle("LDA Train Data Error Rate")

lda_p2 <- df2_result %>%
  ggplot(aes(x=error)) +
  scale_colour_manual(name = 'classes', values = twoClassColor) +
  geom_histogram(data=subset(df2_result, dset == 'test_lda2'),
                 fill = brewer.pal(3,'Set1')[2],
                 alpha = 0.3, binwidth = 0.005) +
  scale_x_continuous(limits = c(0.04,0.24)) +
  ggtitle("LDA Test Data Error Rate")

grid.arrange(lda_p1, lda_p2, nrow = 2)
```

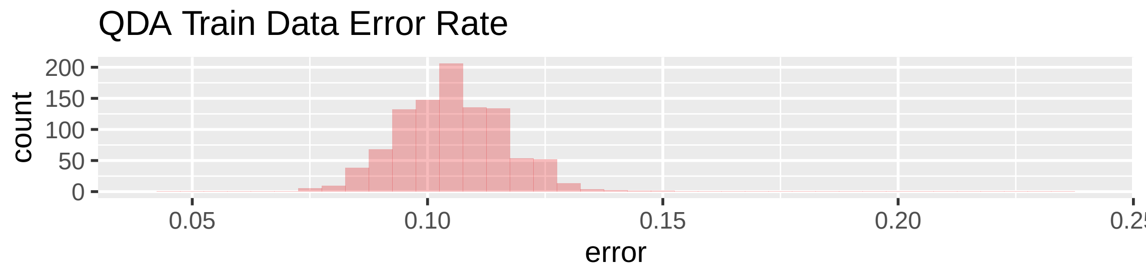
```
## Warning: Removed 2 rows containing missing values (geom_bar).

## Warning: Removed 2 rows containing missing values (geom_bar).
```

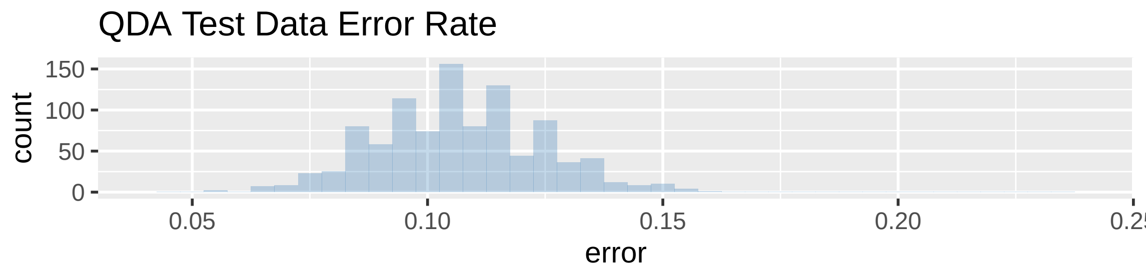





```
qda_p1 <- df2_result %>%
  ggplot(aes(x=error)) +
  scale_colour_manual(name = 'classes', values = twoClassColor) +
  geom_histogram(data=subset(df2_result, dset == 'train_qda2'),
                 fill = brewer.pal(3,'Set1')[1],
                 alpha = 0.3, binwidth = 0.005) +
  scale_x_continuous(limits = c(0.04,0.24)) +
  ggtitle("QDA Train Data Error Rate")

qda_p2 <- df2_result %>%
  ggplot(aes(x=error)) +
  scale_colour_manual(name = 'classes', values = twoClassColor) +
  geom_histogram(data=subset(df2_result, dset == 'test_qda2'),
                 fill = brewer.pal(3,'Set1')[2],
                 alpha = 0.3, binwidth = 0.005) +
  scale_x_continuous(limits = c(0.04,0.24)) +
  ggtitle("QDA Test Data Error Rate")

grid.arrange(qda_p1, qda_p2, nrow = 2)
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).

## Warning: Removed 2 rows containing missing values (geom_bar).
```





4. Increasing the sample size

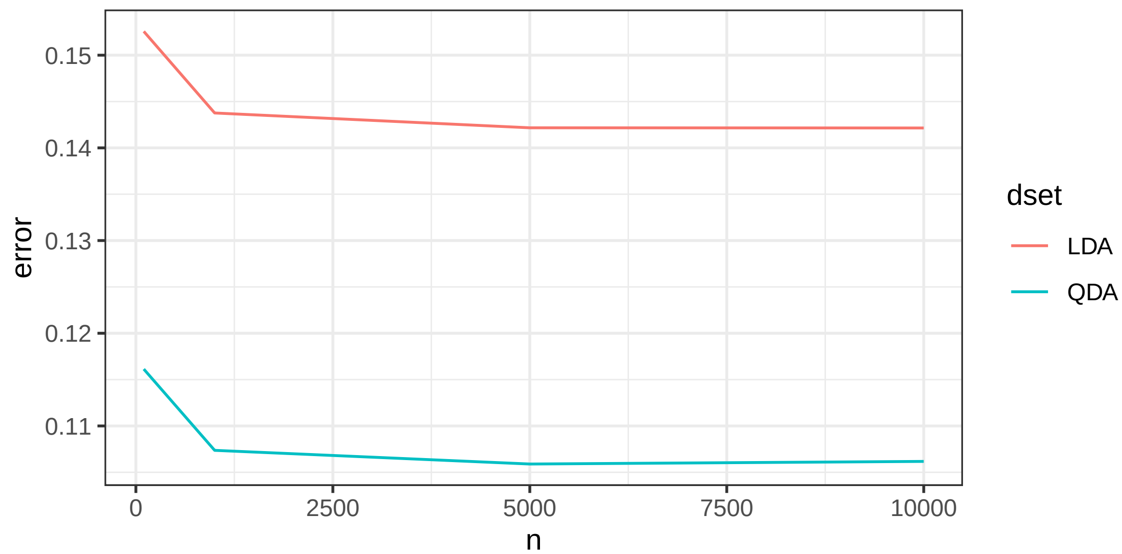
When the number of observation increases, the test error rate decreases in both LDA and QDA. However, QDA will still perform better than LDA since LDA cannot adapt the observations under its inflexibility model restriction. This is shown in the following plot.

```
train_lda4_mean = c()
test_lda4_mean = c()
train_qda4_mean = c()
test_qda4_mean = c()
for (n in c(100, 1000, 5000, 10000)) {
  train_lda4 = c()
  test_lda4 = c()
  train_qda4 = c()
  test_qda4 = c()
  for (i in 1:1000) {
    # i data generation
    x_1 = runif(n, -1, 1)
    x_2 = runif(n, -1, 1)
    ep = rnorm(n, 0, 0.25)
    y = case_when(x_1 + x_1^2 + x_2 + x_2^2 + ep >= 0 ~ TRUE,
                  x_1 + x_1^2 + x_2 + x_2^2 + ep < 0 ~ FALSE)
    df <- data.frame(x_1, x_2, y)

    # ii split
    split <- initial_split(df, prop = .7)
    train <- training(split)
    test <- testing(split)

    # iii training
    lda4 <- MASS::lda(y ~ x_1 + x_2, data = train)
    qda4 <- MASS::qda(y ~ x_1 + x_2, data = train)

    # iv error
    train_lda4 = c(train_lda4, sum(predict(lda4, train)$class != train$y)
```

```r
                    / (0.7*n))
    test_lda4 = c(test_lda4, sum(predict(lda4, test)$class != test$y)
                   / (0.3*n))
    train_qda4 = c(train_qda4, sum(predict(qda4, train)$class != train$y)
                    / (0.7*n))
    test_qda4 = c(test_qda4, sum(predict(qda4, test)$class != test$y)
                   / (0.3*n))
  }
  train_lda4_mean = c(train_lda4_mean, mean(train_lda4))
  test_lda4_mean = c(test_lda4_mean, mean(test_lda4))
  train_qda4_mean = c(train_qda4_mean, mean(train_qda4))
  test_qda4_mean = c(test_qda4_mean, mean(test_qda4))
}
```

b. Summarize the results

```r
df4 <- data.frame(n=c(100, 1000, 5000, 10000),
                  LDA=test_lda4_mean, QDA=test_qda4_mean)
df4 <- df4 %>%
  gather(dset, error, -n)
df4 %>%
  ggplot(aes(x=n, y=error, color=dset)) +
  geom_line() +
  theme_bw()
```



Modeling voter turnout

5. Building classifiers

a. Split data

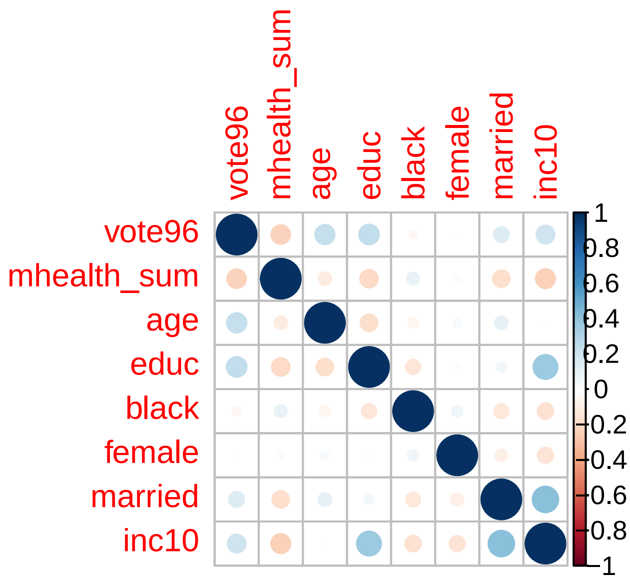
Since there are NAs in the data, I first omit those observations that contains NAs, then split them into training set and test set.

```r
df5 <- read.csv('mental_health.csv')
# Drop NAs
```

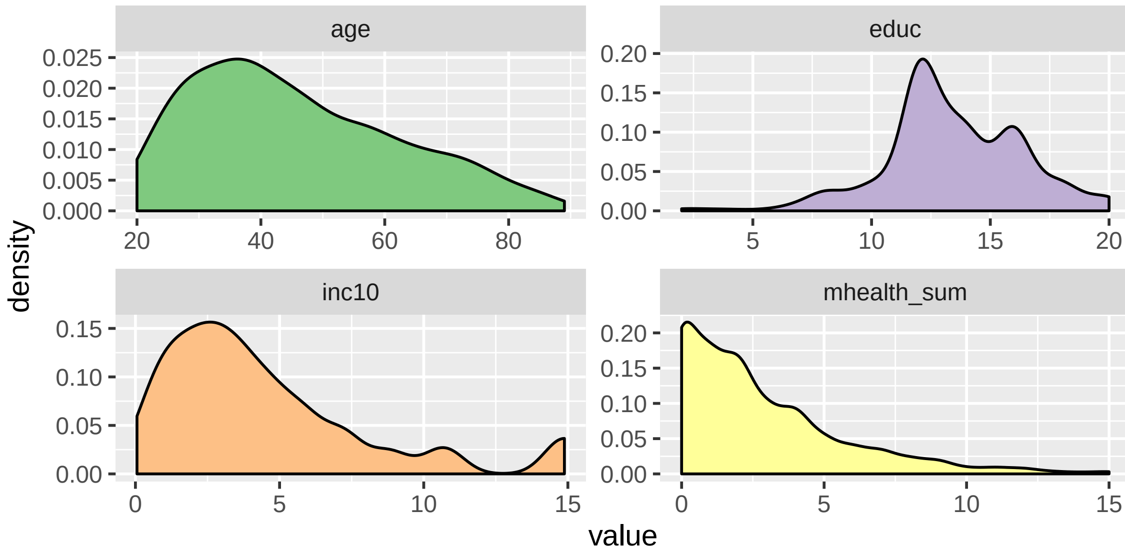
```
df5 <- df5 %>%
  drop_na()
split <- initial_split(df5, prop = .7)
train <- training(split)
test <- testing(split)
```

From the Lab notebook, below are some simple plots for EDA.

```
# checking correlation across several features
train %>%
  select_if(is.numeric) %>%
  cor() %>%
  corrplot::corrplot()
```




```
# checking normality assumption for continuous features
train %>%
  dplyr::select(age, educ, inc10, mhealth_sum) %>%
  gather(metric, value) %>%
  ggplot(aes(value, fill = metric)) +
  geom_density(show.legend = FALSE) +
  scale_fill_brewer(type = "qual") +
  facet_wrap(~ metric, scales = "free")
```



b. Estimation

For the following training and predicting, I use the *caret* package.
The arguments of the function can be found in https://topepo.github.io/caret/available-models.html

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

# Logistic regression
m5.lr <- train(x = x, y = y, method = "glm")

# LDA
m5.lda <- train(x = x, y = y, method = "lda")

# QDA
m5.qda <- train(x = x, y = y, method = "qda")

# Naive Bayes
m5.nb <- train(x = x, y = y, method = "nb")

# K-NN with Euclidean distance metrics
m5.knn <- train(x = x, y = y, method = "knn",
                tuneGrid = expand.grid(k = 1:10))
# Results for each k
m5.knn$results
```

k	Accuracy	Kappa	AccuracySD	KappaSD
1	0.6618557	0.1949534	0.0228630	0.0506534
2	0.6545882	0.1723908	0.0222806	0.0463840
3	0.6573793	0.1700734	0.0271060	0.0576105
4	0.6654442	0.1778314	0.0295223	0.0665285
5	0.6736892	0.1850777	0.0240061	0.0565450
6	0.6831175	0.2007244	0.0230336	0.0548166

k	Accuracy	Kappa	AccuracySD	KappaSD
7	0.6943582	0.2201405	0.0227228	0.0507625
8	0.6913362	0.2074770	0.0232375	0.0510845
9	0.6972801	0.2157595	0.0239008	0.0572396
10	0.7007214	0.2193240	0.0209960	0.0516002

From the results of kNN above, the best k is 10. So I will use $k = 10$ in the following calculations.

c. Performance metrics

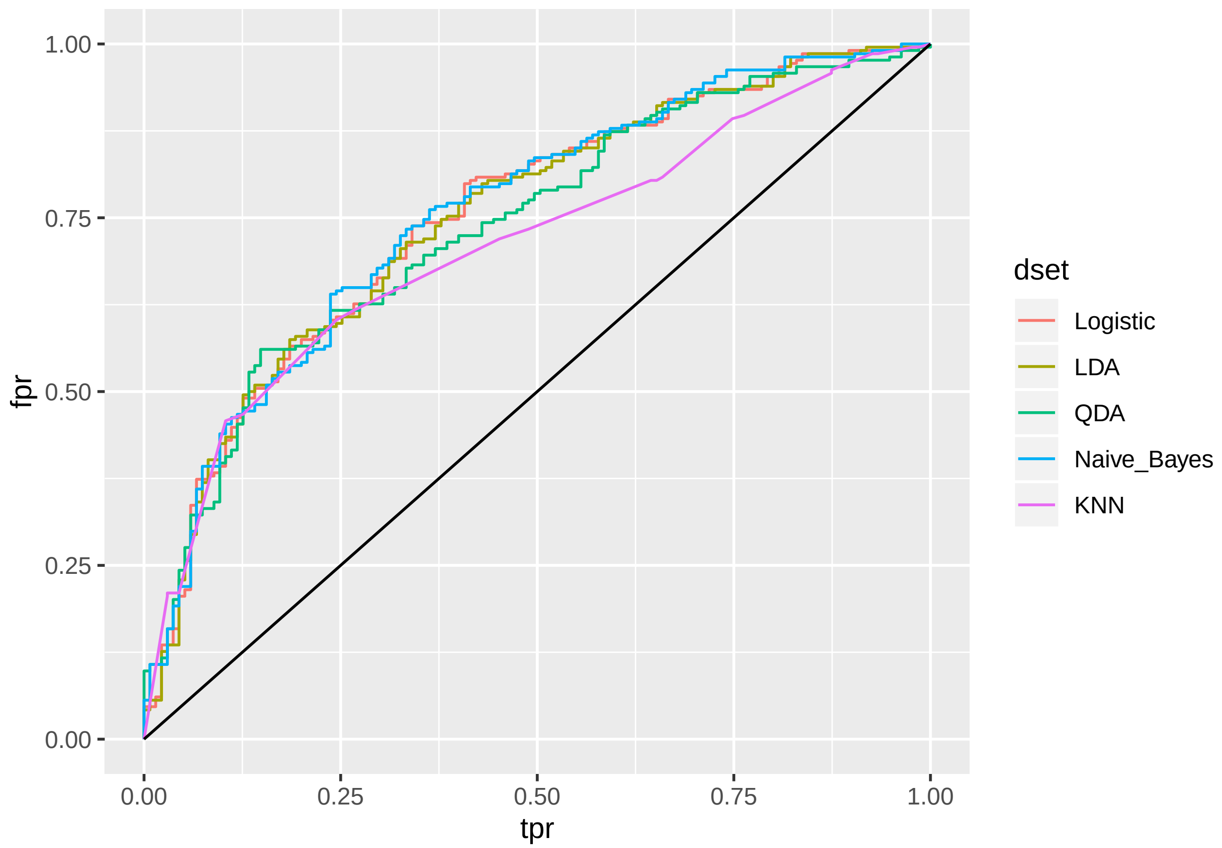
```
# Logistic regression
pred.c <- predict(m5.lr, newdata = test)
pred.p <- predict(m5.lr, newdata = test, type = 'prob')
pred.roc <- prediction(pred.p['1'], as.factor(test$vote96))
auc.tmp <- performance(pred.roc, 'auc')
perf <- performance(pred.roc, 'tpr', 'fpr')
roc.lr <- data.frame(x = slot(perf, 'x.values'),
                     y = slot(perf, 'y.values'),
                     dset = 'Logistic')
colnames(roc.lr) <- c('tpr', 'fpr', 'dset')

lr.perf <- data.frame(Logistic = c(
  confusionMatrix(pred.c, as.factor(test$vote96))[3]$overall['Accuracy'],
  confusionMatrix(pred.c, as.factor(test$vote96))[4]$byClass,
  AUC = as.numeric(auc.tmp@y.values)))

# LDA
pred.c <- predict(m5.lda, newdata = test)
pred.p <- predict(m5.lda, newdata = test, type = 'prob')
pred.roc <- prediction(pred.p['1'], as.factor(test$vote96))
auc.tmp <- performance(pred.roc, 'auc')
perf <- performance(pred.roc, 'tpr', 'fpr')
roc.lda <- data.frame(x = slot(perf, 'x.values'),
                     y = slot(perf, 'y.values'),
                     dset = 'LDA')
colnames(roc.lda) <- c('tpr', 'fpr', 'dset')

lda.perf <- data.frame(LDA = c(
  confusionMatrix(pred.c, as.factor(test$vote96))[3]$overall['Accuracy'],
  confusionMatrix(pred.c, as.factor(test$vote96))[4]$byClass,
  AUC = as.numeric(auc.tmp@y.values)))

# QDA
pred.c <- predict(m5.qda, newdata = test)
pred.p <- predict(m5.qda, newdata = test, type = 'prob')
pred.roc <- prediction(pred.p['1'], as.factor(test$vote96))
auc.tmp <- performance(pred.roc, 'auc')
perf <- performance(pred.roc, 'tpr', 'fpr')
roc.qda <- data.frame(x = slot(perf, 'x.values'),
                      y = slot(perf, 'y.values'),
                      dset = 'QDA')
colnames(roc.qda) <- c('tpr', 'fpr', 'dset')
```

```r
qda.perf <- data.frame(QDA = c(
  confusionMatrix(pred.c, as.factor(test$vote96))[3]$overall['Accuracy'],
  confusionMatrix(pred.c, as.factor(test$vote96))[4]$byClass,
  AUC = as.numeric(auc.tmp@y.values)))

# Naive Bayes
pred.c <- predict(m5.nb, newdata = test)
pred.p <- predict(m5.nb, newdata = test, type = 'prob')
pred.roc <- prediction(pred.p['1'], as.factor(test$vote96))
auc.tmp <- performance(pred.roc, 'auc')
perf <- performance(pred.roc, 'tpr', 'fpr')
roc.nb <- data.frame(x = slot(perf, 'x.values'),
                     y = slot(perf, 'y.values'),
                     dset = 'Naive_Bayes')
colnames(roc.nb) <- c('tpr', 'fpr', 'dset')

nb.perf <- data.frame(Naive_Bayes = c(
  confusionMatrix(pred.c, as.factor(test$vote96))[3]$overall['Accuracy'],
  confusionMatrix(pred.c, as.factor(test$vote96))[4]$byClass,
  AUC = as.numeric(auc.tmp@y.values)))

# K-NN with Euclidean distance metrics
pred.c <- predict(m5.knn, newdata = test)
pred.p <- predict(m5.knn, newdata = test, type = 'prob')
pred.roc <- prediction(pred.p['1'], as.factor(test$vote96))
auc.tmp <- performance(pred.roc, 'auc')
perf <- performance(pred.roc, 'tpr', 'fpr')
roc.knn <- data.frame(x = slot(perf, 'x.values'),
                      y = slot(perf, 'y.values'),
                      dset = 'KNN')
colnames(roc.knn) <- c('tpr', 'fpr', 'dset')

knn.perf <- data.frame(KNN = c(
  confusionMatrix(pred.c, as.factor(test$vote96))[3]$overall['Accuracy'],
  confusionMatrix(pred.c, as.factor(test$vote96))[4]$byClass,
  AUC = as.numeric(auc.tmp@y.values)))

d <- data.frame(x= c(0, 1), y= c(0, 1))

cbind(lr.perf, lda.perf, qda.perf, nb.perf, knn.perf)
```

	Logistic	LDA	QDA	Naive_Bayes	KNN
Accuracy	0.6762178	0.6876791	0.6704871	0.6876791	0.6332378
Sensitivity	0.3481481	0.3555556	0.4222222	0.2740741	0.2814815
Specificity	0.8831776	0.8971963	0.8271028	0.9485981	0.8551402
Pos Pred Value	0.6527778	0.6857143	0.6063830	0.7708333	0.5507246
Neg Pred Value	0.6823105	0.6881720	0.6941176	0.6744186	0.6535714
Precision	0.6527778	0.6857143	0.6063830	0.7708333	0.5507246
Recall	0.3481481	0.3555556	0.4222222	0.2740741	0.2814815
F1	0.4541063	0.4682927	0.4978166	0.4043716	0.3725490
Prevalence	0.3868195	0.3868195	0.3868195	0.3868195	0.3868195
Detection Rate	0.1346705	0.1375358	0.1633238	0.1060172	0.1088825
Detection Prevalence	0.2063037	0.2005731	0.2693410	0.1375358	0.1977077

	Logistic	LDA	QDA	Naive_Bayes	KNN
Balanced Accuracy	0.6156629	0.6263759	0.6246625	0.6113361	0.5683108
AUC	0.7515749	0.7503634	0.7374178	0.7566978	0.7113361

```r
rbind(roc.lr, roc.lda, roc.qda, roc.nb, roc.knn) %>%
  ggplot(aes(x=tpr, y=fpr)) +
  geom_line(aes(color=dset)) +
  geom_line(data=d, aes(x=x, y=y))
```



d. Results

For this question, I would choose logistic regression despite that its performance is not the best among the models. Since our main concern is whether the respondent's mental health affects its participation on politics or not, we care much about the model's interpretations. Logistic regression is relatively easier to interpret its coefficients. In this case, we can calculate the estimated probability of 1996 presidential voting by using the coefficiences below.

```r
coef(m5.lr$finalModel) %>%
  round(digits = 3)
```

```
## (Intercept) mhealth_sum         age        educ       black      female     married       inc10
##      -4.477      -0.087       0.045       0.245       0.301      -0.004       0.293       0.064
```

Moreover, the results in the table above show that the accuracies of logistic regression are not so different from these of the others.