Hu_Chun_HW2

Hu Chun

2/2/2020

```
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```
library(tidymodels)
```

```
## -- Attaching packages ---------------------------------------------- tidymodels 0.0.3 --
```

```
## v broom     0.5.2     v purrr     0.3.3
## v dials     0.0.4     v recipes   0.1.9
## v dplyr     0.8.3     v rsample   0.0.5
## v infer     0.5.1     v tibble    2.1.3
## v parsnip   0.0.5     v yardstick 0.0.4
```

```
## -- Conflicts ------------------------------------------------- tidymodels_conflicts() --
## x purrr::discard()       masks scales::discard()
## x dplyr::filter()        masks stats::filter()
## x dplyr::lag()           masks stats::lag()
## x purrr::lift()          masks caret::lift()
## x dials::margin()        masks ggplot2::margin()
## x yardstick::precision() masks caret::precision()
## x yardstick::recall()    masks caret::recall()
## x recipes::step()        masks stats::step()
```

The Bayes Classifier

1.

a.

```
set.seed(1234)
```

b.

```
df <- data.frame(x1 = runif(200, min=-1, max=1), x2 = runif(200, min=-1, max=1))
```

c.

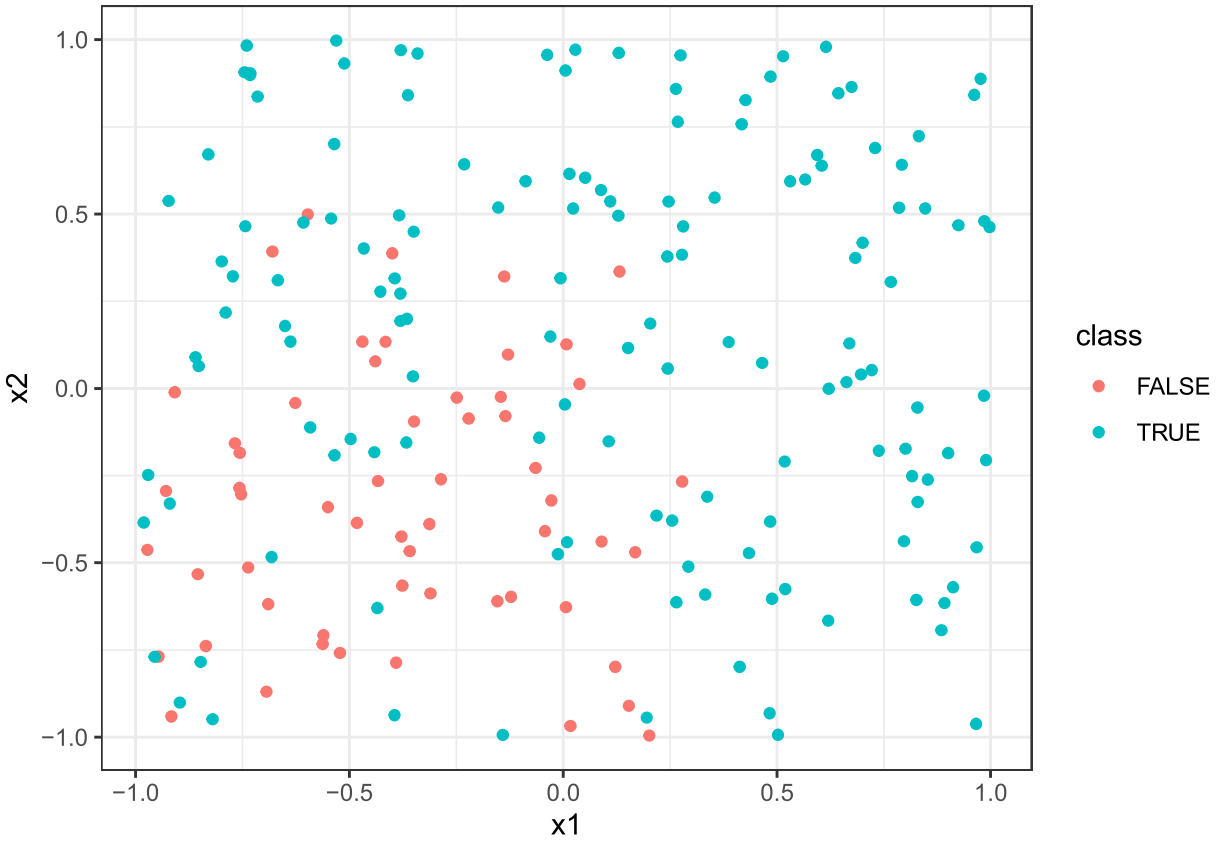
```
df$y <- df$x1 + df$x1^2 + df$x2 + df$x2^2 + rnorm(200, mean=0, sd=0.5)
```

d.

```
df$prob <- exp(df$y) / (1 + exp(df$y))
```

e.

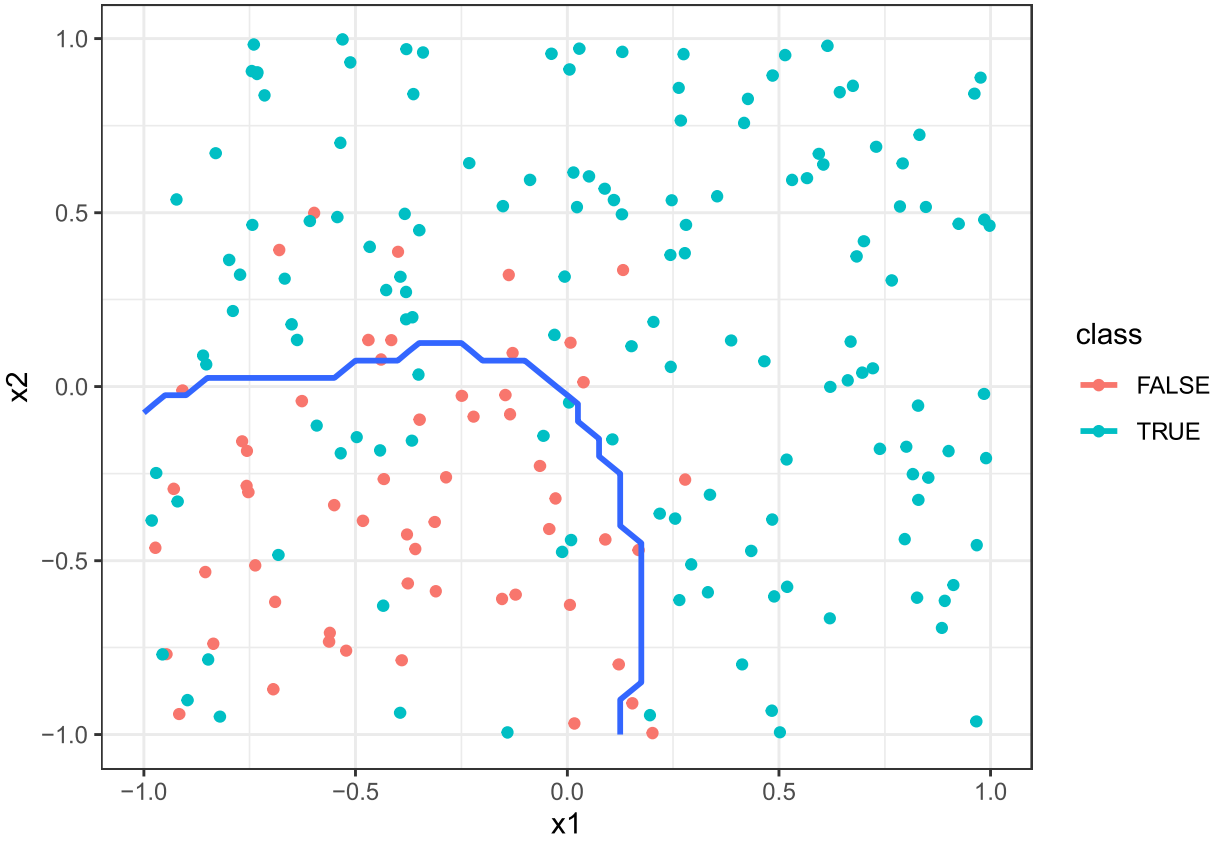
```
df$class <- as.factor(df$prob > 0.5)

ggplot(df, aes(x1, x2, color=class)) +
  geom_point() +
  theme_bw()
```



f.

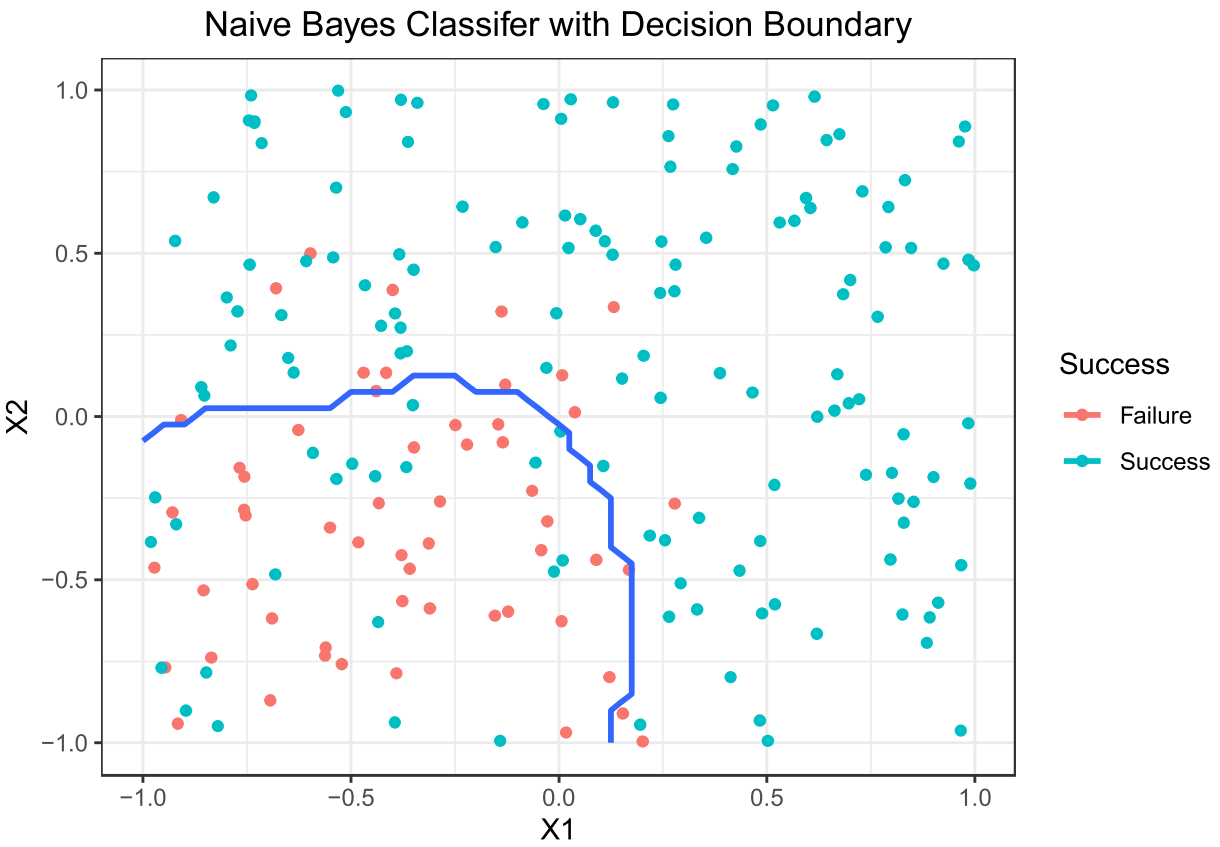
```
train_control <- trainControl(
  method = 'cv',
  number = 10
)

nb.m1 <- train(
  x = df[, c('x1', 'x2')],
  y = df$class,
  method = 'nb',
  trControl = train_control
```

```
)

grid_val <- seq(-1, 1, by = .05)
grid <- expand.grid(x1=grid_val, x2=grid_val)
pred <- predict(nb.m1, newdata=grid)
grid$class <- as.numeric(pred)

ggplot(df, aes(x1, x2, z=class, color=class)) +
  geom_point() +
  geom_contour(data=grid, breaks = c(1.5), size = 1) +
  theme_bw()
```



g.

```
ggplot(df, aes(x1, x2, z=class, col=class)) +
  geom_point() +
  geom_contour(data=grid, breaks = c(1.5), size = 1) +
  theme_bw() +
  scale_color_discrete(name = "Success", labels = c("Failure", "Success")) +
  labs(x = "X1", y = "X2") +
  ggtitle('Naive Bayes Classifer with Decision Boundary') +
  theme(plot.title = element_text(hjust=.5))
```



Exploring Simulated Differences between LDA and QDA

2.

a.

```
simulation <- function(n) {
  # i.
  x1_ <- runif(n, min=-1, max=1)
  x2_ <- runif(n, min=-1, max=1)
  y_ <- x1_ + x2_ + rnorm(n, 0, 1)
  class_ <- y_ >= 0
  df_ = data.frame(x1_, x2_, y_, class_)

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

  # iii.
  lda_model <- MASS::lda(class_ ~ x1_ + x2_, data = train)
  qda_model <- MASS::qda(class_ ~ x1_ + x2_, data = train)

  # iv.
  lda_tr <- mean(train$class != predict(lda_model, train)$class)
```

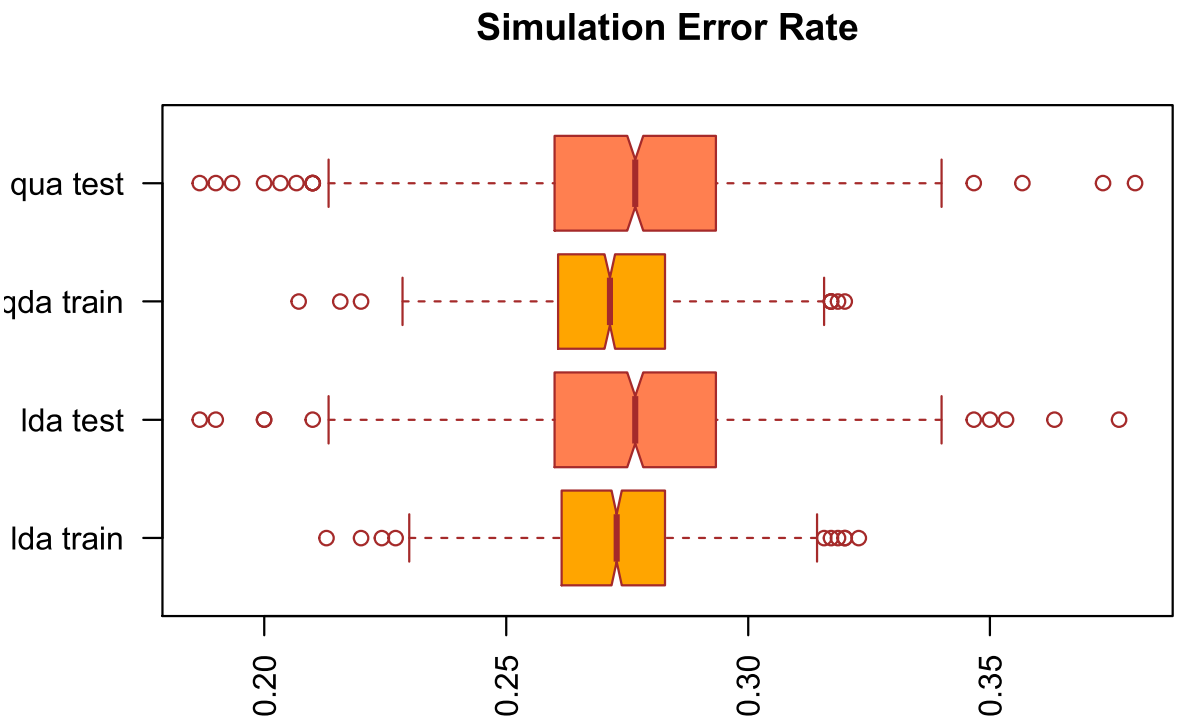
```
  lda_te <- mean(test$class != predict(lda_model, test)$class)
  qda_tr <- mean(train$class != predict(qda_model, train)$class)
  qda_te <- mean(test$class != predict(qda_model, test)$class)

  return(tibble(lda_tr, lda_te, qda_tr, qda_te))
}

sim = data.frame()
for (i in 1:1000){
  sim <- rbind(sim, simulation(1000))
}
```

b.

```
knitr::kable(psych::describe(sim))
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range
lda_tr	1	1000	0.2724800	0.0165259	0.2728571	0.2724518	0.016944	0.2128571	0.3228571	0.1100000
lda_te	2	1000	0.2759667	0.0266646	0.2766667	0.2761375	0.024710	0.1866667	0.3766667	0.1900000
qda_tr	3	1000	0.2717329	0.0163840	0.2714286	0.2717464	0.016944	0.2071429	0.3200000	0.1128571
qda_te	4	1000	0.2763633	0.0265954	0.2766667	0.2765458	0.024710	0.1866667	0.3800000	0.1933333

```
boxplot(sim,
        main = "Simulation Error Rate",
        names = c("lda train", "lda test", "qda train", "qua test"),
        las = 2,
        col = c("orange","coral"),
        border = "brown",
        horizontal = TRUE,
        notch = TRUE
        )
```



###

From the descriptive statistics and the graph above, we can see that LDA and QDA have almost the same performance. Both methods produced the same mean, median, sd, and maximum value. For minimum value, LDA has slightly higher training and testing values. Interestingly, for test set, QDA are more skewed to the right than LDA.

3.

a.

```
simulation2 <- function(n) {
  # i.
  x1_ <- runif(n, min=-1, max=1)
  x2_ <- runif(n, min=-1, max=1)
  y_ = x1_ + x1_^2 + x2_ + x2_^2 + rnorm(n, 0, 1)
  class_ <- y_ >= 0
  df_ = data.frame(x1_, x2_, y_, class_)

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

  # iii.
  lda_model <- MASS::lda(class_ ~ x1_ + x2_, data = train)
  qda_model <- MASS::qda(class_ ~ x1_ + x2_, data = train)

  # iv.
  lda_tr <- mean(train$class!=predict(lda_model, train)$class)
  lda_te <- mean(test$class!=predict(lda_model, test)$class)
  qda_tr <- mean(train$class!=predict(qda_model, train)$class)
```

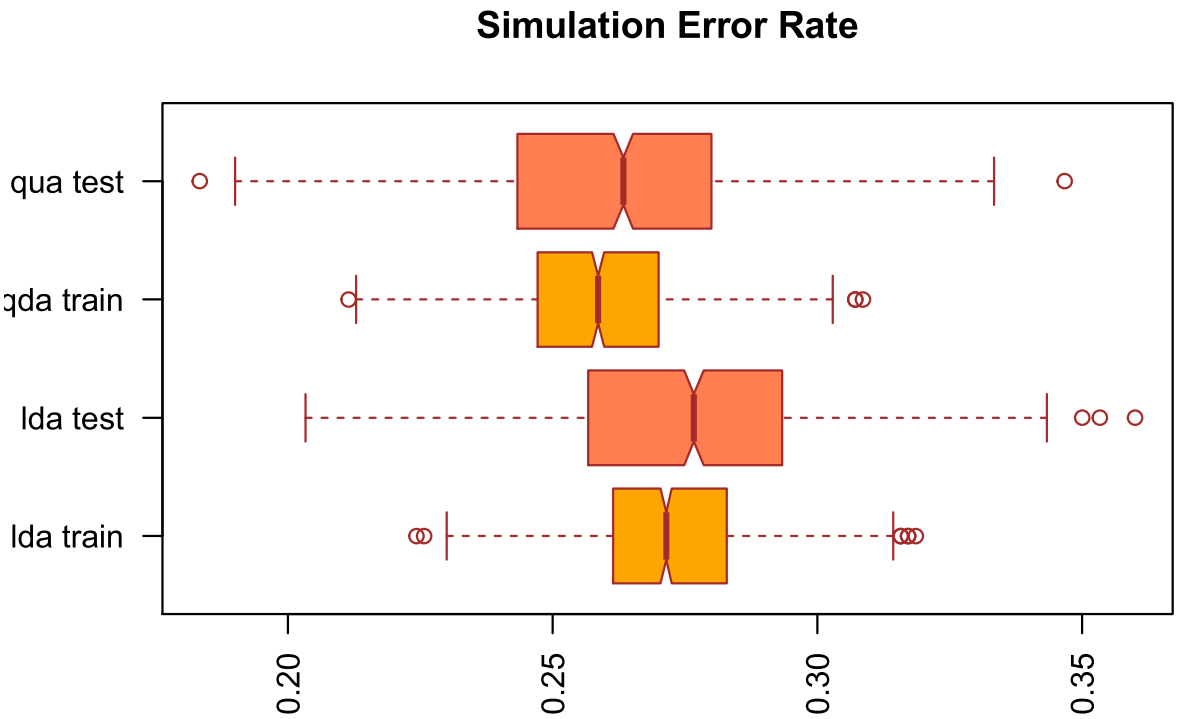
```
  qda_te <- mean(test$class!=predict(qda_model, test)$class)

  return(tibble(lda_tr, lda_te, qda_tr, qda_te))
}

sim2 = data.frame()
for (i in 1:1000){
  sim2 <- rbind(sim2, simulation2(1000))
}
```

b.

```
knitr::kable(psych::describe(sim2))
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range
lda_tr	1	1000	0.2722443	0.0164903	0.2714286	0.2721071	0.016944	0.2242857	0.3185714	0.0942857
lda_te	2	1000	0.2751267	0.0261963	0.2766667	0.2750500	0.029652	0.2033333	0.3600000	0.1566667
qda_tr	3	1000	0.2587243	0.0158138	0.2585714	0.2584804	0.016944	0.2114286	0.3085714	0.0971429
qda_te	4	1000	0.2624067	0.0263530	0.2633333	0.2621583	0.029652	0.1833333	0.3466667	0.1633333

```
boxplot(sim2,
        main = "Simulation Error Rate",
        names = c("lda train", "lda test", "qda train", "qua test"),
        las = 2,
        col = c("orange","coral"),
        border = "brown",
        horizontal = TRUE,
        notch = TRUE
        )
```



###

Overall, LDA consistently produced higher training and testing error than QDA, although the difference is not significant. LDA training error is slightly skewed to the right, and QDA testing error is slightly skewed to the left.

4.

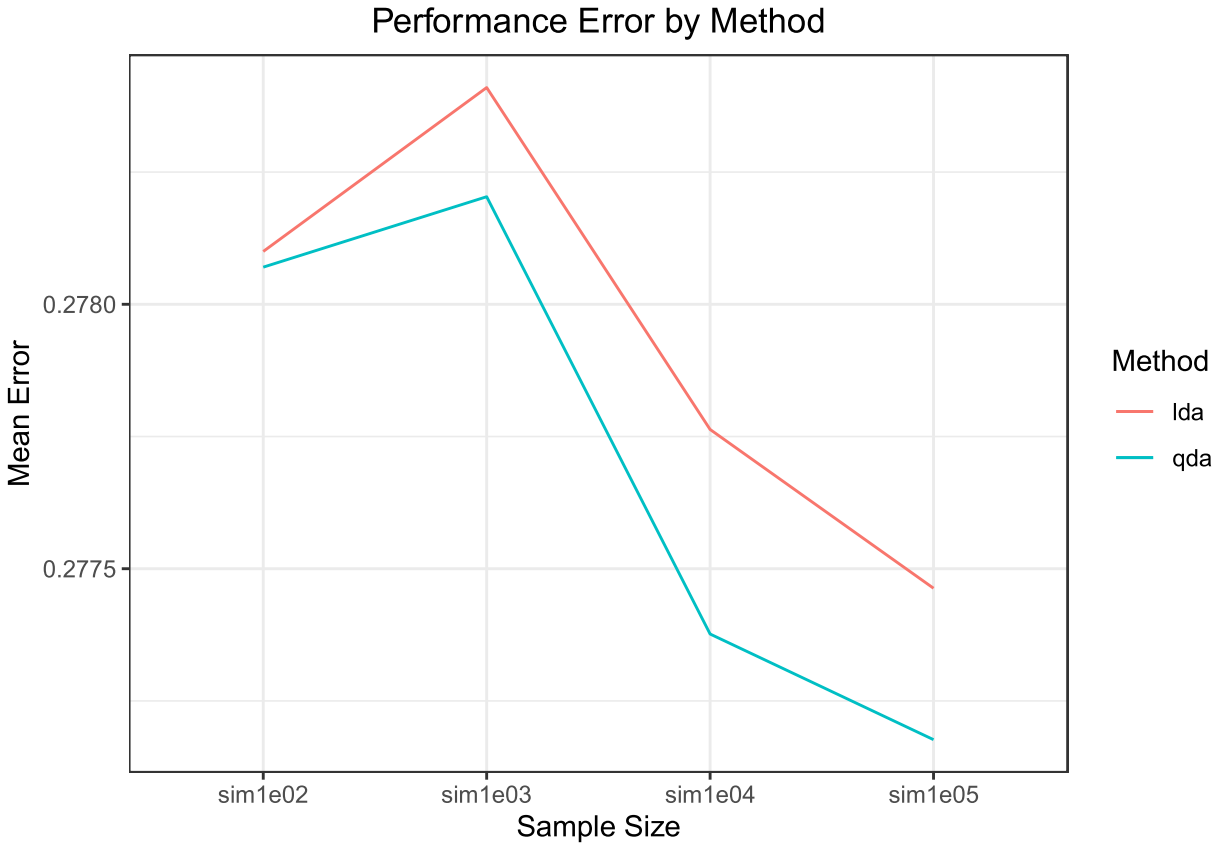
a.

```
simulate_n <- function(n){
  sim = data.frame()
  for (i in 1:1000){
    sim <- rbind(sim, simulation(1000))
  }
  return(sim)
}

sim_1e02 = simulate_n(1e02)
sim_1e03 = simulate_n(1e03)
sim_1e04 = simulate_n(1e04)
sim_1e05 = simulate_n(1e05)
```

b.

```
dat <- data.frame(
  lda_err = c(mean(sim_1e02$lda_te), mean(sim_1e03$lda_te), mean(sim_1e04$lda_te), mean(sim_1e05$lda_t
  qda_err = c(mean(sim_1e02$qda_te), mean(sim_1e03$qda_te), mean(sim_1e04$qda_te), mean(sim_1e05$qda_t
  label = c("sim1e02", "sim1e03", "sim1e04", "sim1e05")
  )
```

```
ggplot(dat, aes(x=label)) +
  geom_line(aes(y=lda_err, color = "steelblue", group=1)) +
  geom_line(aes(y=qda_err, color = "coral", group=1)) +
  theme_bw() +
  scale_color_discrete(name = "Method", labels = c("lda", "qda")) +
  labs(x="Sample Size", y="Mean Error") +
  ggtitle('Performance Error by Method') +
  theme(plot.title = element_text(hjust=.5))
```



Using the non-linear Bayes decision boundary approach, LDA produces a higher test error than QDA at all sample sizes. Both method produce a slightly lower error at 1e02 sample size; mean errors peak at 1e03 sample size and then start to decrease. Overall, QDA performs better in large sample size than LDA.

Modeling Voter Turnout

5.

a.

```
mental_health = read.csv("/Users/reneehu/Desktop/mental_health.csv")
mental_health = na.omit(mental_health)

split_m <- initial_split(mental_health, prop = .7)
train_m <- training(split_m)
test_m <- testing(split_m)
```

b.

```
mh_logit <- glm(vote96 ~ ., data = train_m, family = "binomial")

mh_lda <- MASS::lda(vote96 ~ ., data = train_m)

mh_qda <- MASS::qda(vote96 ~ ., data = train_m)

mh_bayes <- train(
  x = train_m[-1],
  y = as.factor(train_m$vote96),
  method = 'nb'
)
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 247

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 89

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 116

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 192

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 215

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 239

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 247

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 278

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 51

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 85

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 113

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 187

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 202
```

```
mh_knn <- tibble(k = 1:10,
                 knn_train = map(k, ~ class::knn(select(train_m, -vote96),
                                                 test = select(test_m, -vote96),
                                                 cl = train_m$vote96, k = .)),
                 knn_test = map(k, ~ attr(class::knn(select(train_m, -vote96),
                                                 test = select(test_m, -vote96),
                                                 cl = train_m$vote96, k = .,
                                                 prob = TRUE), 'prob')),
                 err_test = map_dbl(knn_test, ~ mean(test_m$vote96 != .)))
```

c.

```
# Error rate
test_pred <- tibble(log = predict(mh_logit, test_m),
                    lda = predict(mh_lda, test_m)$posterior[,1],
                    qda = predict(mh_qda, test_m)$posterior[,1],
                    nb = predict(mh_bayes, test_m, type='prob')[,1])

get_err_rate <- function(pred){
  pred_class <- pred >= 0.5
  err_rate <- mean(test_m$vote96 != pred_class)
  return(err_rate)
}

log_err = get_err_rate(test_pred$log)
lda_err = get_err_rate(test_pred$lda)
qda_err = get_err_rate(test_pred$qda)
nb_err = get_err_rate(test_pred$nb)
knn_err = tibble(mh_knn$err_test)

err_tab <- tibble(log_err, lda_err, qda_err, nb_err)

knitr::kable(err_tab)
```

log_err	lda_err	qda_err	nb_err
0.2320917	0.7535817	0.756447	0.739255

```
knitr::kable(knn_err)
```

mh_knn$err_test
0.3123209
0.5501433
0.6647564
0.7249284
0.7707736
0.7965616
0.8194842
0.8395415
0.8653295
0.8739255

```r
# ROC: log, lda, qda, nb
test_knn <- as_tibble(mh_knn$knn_test, .name_repair = 'unique')
```

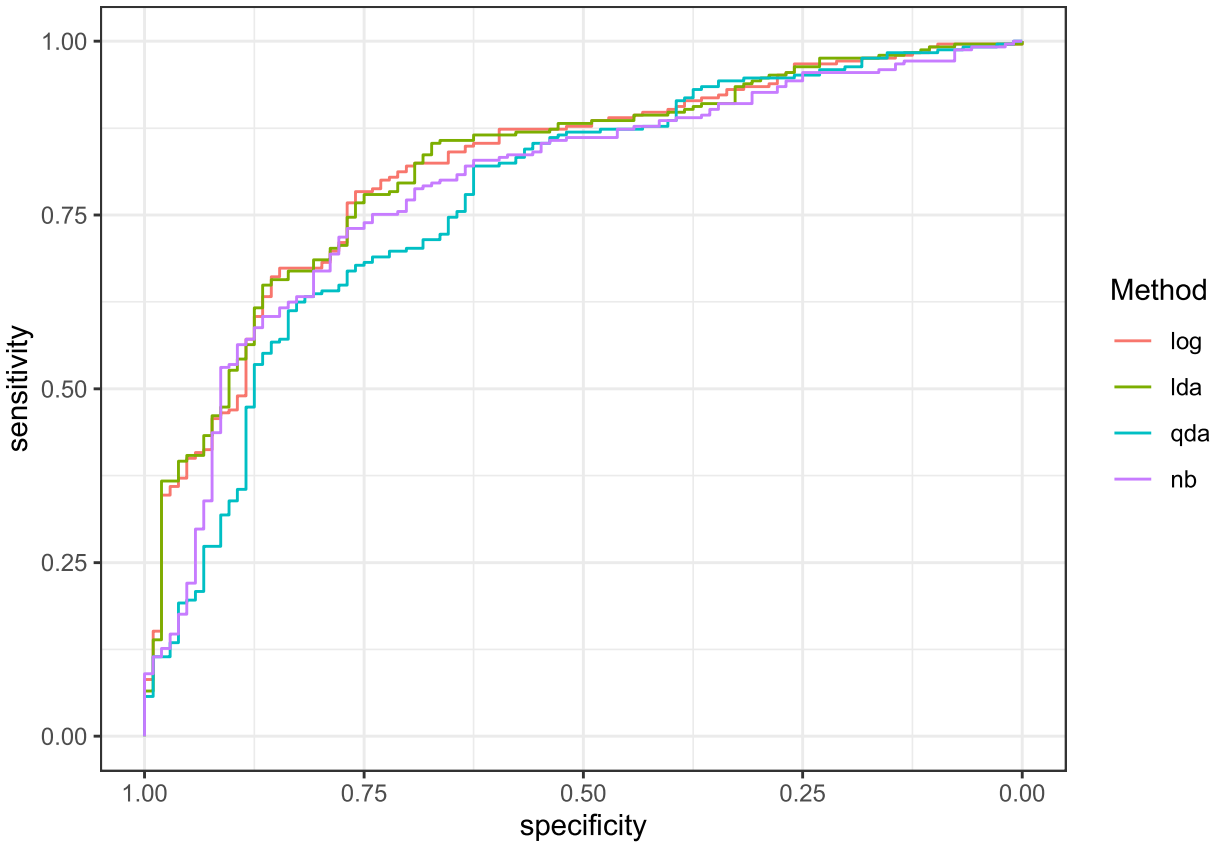
```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
## * ... and 5 more problems
```

```r
tests <- cbind(test_pred, test_knn)
roc_obj <- lapply(tests, FUN=pROC::roc, response=test_m$vote96)
```

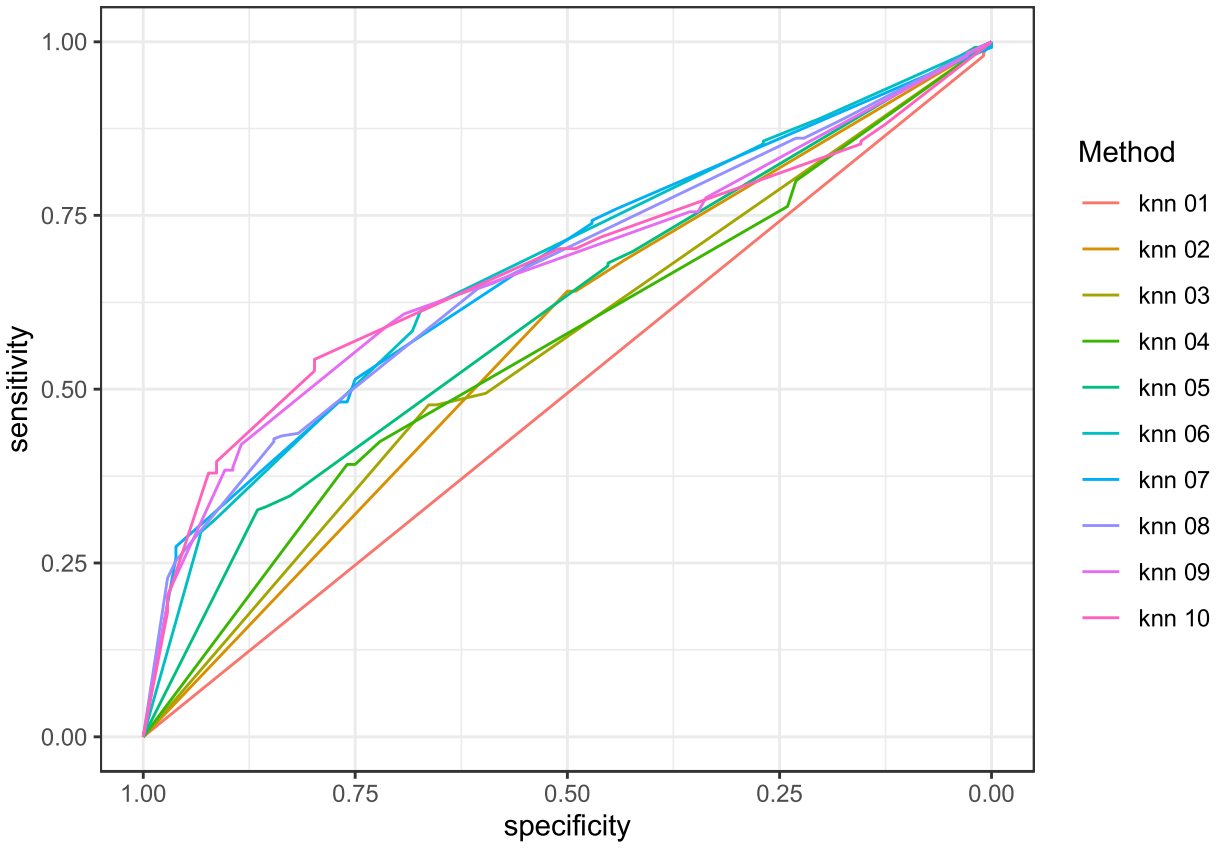
```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls > cases
## Setting levels: control = 0, case = 1
## Setting direction: controls > cases
## Setting levels: control = 0, case = 1
## Setting direction: controls > cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
pROC::ggroc(roc_obj[c('log', 'lda', 'qda', 'nb')]) +
  theme_bw() +
  labs(color = "Method")
```




```
# ROC: knn*10
pROC::ggroc(roc_obj[5:14]) +
  theme_bw() +
  scale_color_discrete(name = "Method", labels = c(sprintf("knn %02d", seq(1,10))))
```




```
# AUC Stats
auc <- c()
for (i in roc_obj){
  auc <- c(auc, i$auc)
}

method <- c("log", "lda", "qda", "nb", "knn 01", "knn 02", "knn 03", "knn 04", "knn 05", "knn 06", "kn

auc_stats <- tibble(method, auc)
auc_stats
```

```
## # A tibble: 14 x 2
##    method   auc
##    <chr>  <dbl>
##  1 log    0.819
##  2 lda    0.820
##  3 qda    0.774
##  4 nb     0.791
##  5 knn 01 0.495
##  6 knn 02 0.569
##  7 knn 03 0.558
##  8 knn 04 0.563
##  9 knn 05 0.607
## 10 knn 06 0.670
## 11 knn 07 0.671
## 12 knn 08 0.667
## 13 knn 09 0.672
```

```
## 14 knn 10 0.671
```

d.

If we use AUC scores to measure the performance of models, both logistic regression and LDA model produce the highest AUC score. It shows that these two models are better at distinguishing between classes. If we further compare their test error, logistic regression produces lower test error rate than LDA. Hence, the best model is logistic regression.