HW02 - Classification [MACS 30100]

Adarsh Mathew

02/01/2020

Q1 - Naive Bayes Decision Boundary

```
# Generating the table

set.seed(01302020)
n_obs <- 200

x_1 <- runif(n_obs, -1, 1)
x_2 <- runif(n_obs, -1, 1)
err <-  rnorm(n_obs, mean = 0, sd = 0.5)

nb_df <- cbind(x_1, x_2, err) %>%
  as_tibble() %>%
  mutate(f_x = x_1 + x_1^2 +x_2 +x_2^2 + err,
         y = exp(f_x)/(1+exp(f_x)),
         y_class = as.factor(y > 0.5))

head(nb_df)
```

```
## # A tibble: 6 x 6
##        x_1     x_2      err    f_x     y y_class
##      <dbl>   <dbl>    <dbl>  <dbl> <dbl> <fct>
## 1 -0.985  -0.0760  0.221    0.137 0.534 TRUE
## 2 -0.277   0.755   0.0937   1.22  0.772 TRUE
## 3  0.730   0.464  -0.0844   1.86  0.865 TRUE
## 4 -0.477   0.0954 -0.397   -0.542 0.368 FALSE
## 5  0.0770  0.466  -0.871   -0.106 0.474 FALSE
## 6 -0.0893 -0.447   0.00252 -0.326 0.419 FALSE
```

```
# Training the Naive Bayes model

nb_mod <- train(x = nb_df %>% select(x_1, x_2), y = nb_df$y_class, method = "nb")

confusionMatrix(nb_mod)
```

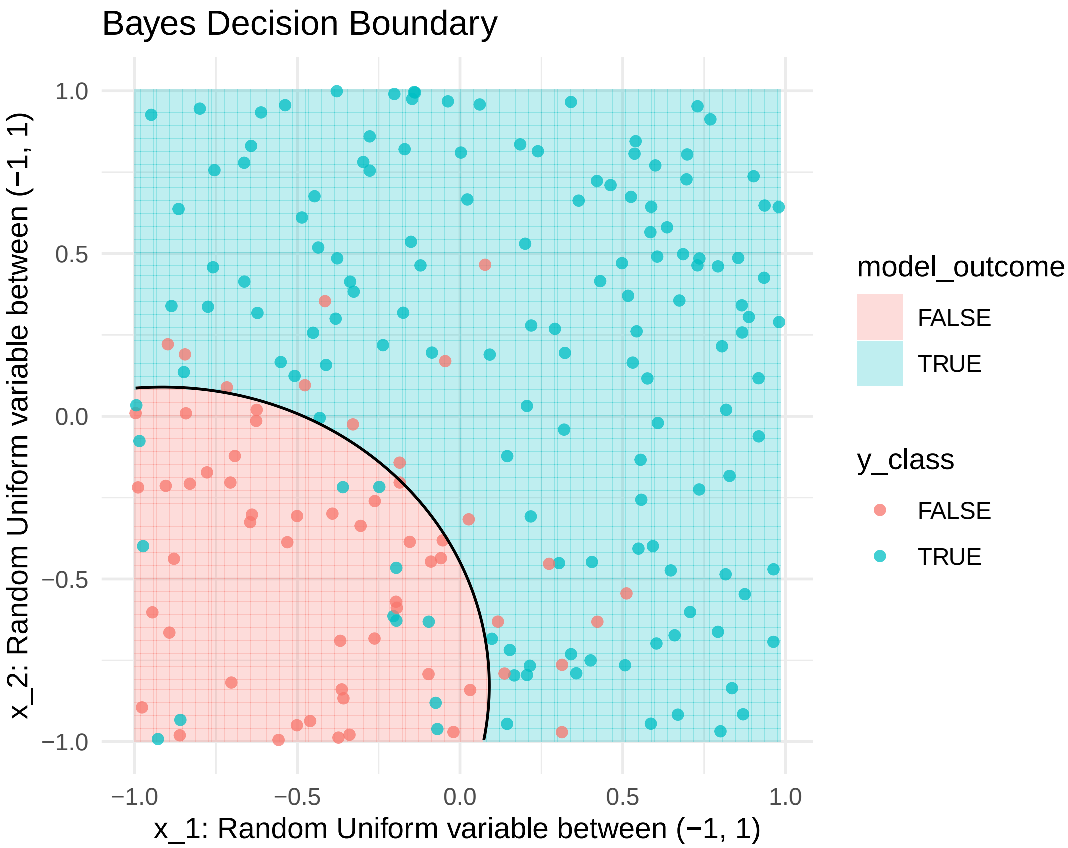
```
## Bootstrapped (25 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  19.6  7.8
```

```
##      TRUE    8.6 64.0
##
##  Accuracy (average) : 0.836
```

```
# Setting up the grid-search to plot the decision boundary

nb_grid <- expand.grid(
  x_1 = seq(min(nb_df$x_1), max(nb_df$x_1), length = n_obs),
  x_2 = seq(min(nb_df$x_2), max(nb_df$x_2), length = n_obs)
)

nb_grid  <- nb_grid %>%
  bind_cols('model_outcome' = predict(nb_mod, newdata = nb_grid),
            'grid_prob' = predict(nb_mod, newdata = nb_grid, type = 'prob')) %>%
  rename(prob_true = 'TRUE', prob_false = 'FALSE')

head(nb_grid)
```

```
##          x_1        x_2 model_outcome prob_false prob_true
## 1 -0.9968940 -0.9945238         FALSE  0.8320680 0.1679320
## 2 -0.9869575 -0.9945238         FALSE  0.8324237 0.1675763
## 3 -0.9770210 -0.9945238         FALSE  0.8327334 0.1672666
## 4 -0.9670845 -0.9945238         FALSE  0.8329972 0.1670028
## 5 -0.9571480 -0.9945238         FALSE  0.8332153 0.1667847
## 6 -0.9472115 -0.9945238         FALSE  0.8333879 0.1666121
```

```
# Plotting the decision boundary

nb_df %>% ggplot(aes(x = x_1, y = x_2)) +
  geom_tile(data = nb_grid, aes(fill = model_outcome), alpha = 0.25) +
  geom_point(aes(color = y_class), alpha = .75) +
  geom_contour(data = nb_grid, aes(z = prob_true), colour = "black", breaks = .5) +
  theme_minimal() +
  #theme(legend.position = "top") +
  coord_equal() +
  ggtitle("Bayes Decision Boundary") +
  labs(x = "x_1: Random Uniform variable between (-1, 1)",
       y = "x_2: Random Uniform variable between (-1, 1)")
```



Bayes Decision Boundary
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
x_1: Random Uniform variable between (−1, 1)
x_2: Random Uniform variable between (−1, 1)
model_outcome
FALSE
TRUE
y_class
FALSE
TRUE

LDA-vs-QDA

Adarsh Mathew

2/1/2020

Q2: LDA vs QDA with a linear Decision Boundary

```
model_df_gen <- function(seed_val, n_obs, sec_order_terms, split_prop, return_type){

  set.seed(seed_val)

  x_1 <- runif(n_obs, -1, 1)
  x_2 <- runif(n_obs, -1, 1)
  err <-  rnorm(n_obs, mean = 0, sd = 1)

  df <- cbind(x_1, x_2, err) %>%
    as_tibble(rownames = NULL) %>%
    mutate(y = ifelse(sec_order_terms == TRUE, x_1 + x_2 + x_1^2 + x_2^2, x_1 + x_2),
           y_class = factor(y >= 0, levels = c(TRUE, FALSE)),
           f_x = y + err,
           y_sim_class = factor(f_x >= 0, levels = c(TRUE, FALSE)))

  df_split <- rsample::initial_split(df, prop = split_prop)
  df_train <- rsample::training(df_split) %>% select(x_1, x_2, y_sim_class)
  df_test <- rsample::testing(df_split) %>% select(x_1, x_2, y_sim_class)

  if(return_type == "train") return(df_train) else return(df_test)

}

model_runner <- function(df_train, model_type){
  df_train_ed <- droplevels(df_train)

  mod <- train(
    df_train_ed %>% select("x_1", "x_2") %>% na.omit(),
    (df_train_ed %>% na.omit())$y_sim_class,
    metric = 'Accuracy',
    method = model_type,
    allowParallel = FALSE
  )

  return(mod)
}

model_acc_gen <- function(model_obj, pred_df){

  pred_df_aug <- pred_df %>%
```

```
    bind_cols('model_outcome' = predict(model_obj, newdata = pred_df),
              'model_prob' = predict(model_obj, newdata = pred_df, type = 'prob')['TRUE']) %>%
    rename(prob_true = 'TRUE')

  #print("Training Accuracy:")
  pred_acc <- postResample(pred_df_aug$model_outcome, pred_df$y_sim_class)[1]

  return(as.numeric(pred_acc))
}
```

```
n_iter <- 1000
#n_cores <- availableCores() - 2
#plan(multicore, workers = n_cores)

discrim_df <- sample(1:1000, n_iter, replace=T) %>%
  as_tibble(rownames = NULL) %>%
  rename(seed_value = value) %>%
  mutate(train_df = map(seed_value, ~model_df_gen(.x, 1000, FALSE, 0.7, "train")),
         test_df = map(seed_value, ~model_df_gen(.x, 1000, FALSE, 0.7, "test")),
         lda_mod = map(train_df, ~model_runner(.x, "lda")),
         lda_err_train = 1 - map2_dbl(lda_mod, train_df, model_acc_gen),
         lda_err_test = 1 - map2_dbl(lda_mod, test_df, model_acc_gen),
         qda_mod = map(train_df, ~model_runner(.x, "qda")),
         qda_err_train = 1 - map2_dbl(qda_mod, train_df, model_acc_gen),
         qda_err_test = 1 - map2_dbl(qda_mod, test_df, model_acc_gen))

#future::plan(future::sequential)

head(discrim_df)
```

```
## # A tibble: 6 x 9
##   seed_value train_df test_df lda_mod lda_err_train lda_err_test qda_mod
##        <int> <list>   <list>  <list>          <dbl>        <dbl> <list>
## 1        595 <tibble~ <tibbl~ <train>         0.389        0.433 <train>
## 2        992 <tibble~ <tibbl~ <train>         0.429        0.437 <train>
## 3         39 <tibble~ <tibbl~ <train>         0.166        0.137 <train>
## 4          9 <tibble~ <tibbl~ <train>         0.109        0.103 <train>
## 5         36 <tibble~ <tibbl~ <train>         0.309        0.363 <train>
## 6        570 <tibble~ <tibbl~ <train>         0.314        0.29  <train>
## # ... with 2 more variables: qda_err_train <dbl>, qda_err_test <dbl>
```

```
discrim_df_plot <- discrim_df %>%
  pivot_longer(c('lda_err_train', 'lda_err_test','qda_err_train', 'qda_err_test'), names_to = "mod_dfnam
  separate(`mod_dfname`, sep = "_", into = c("model_type", "metric", "df_type"))

head(discrim_df_plot)
```

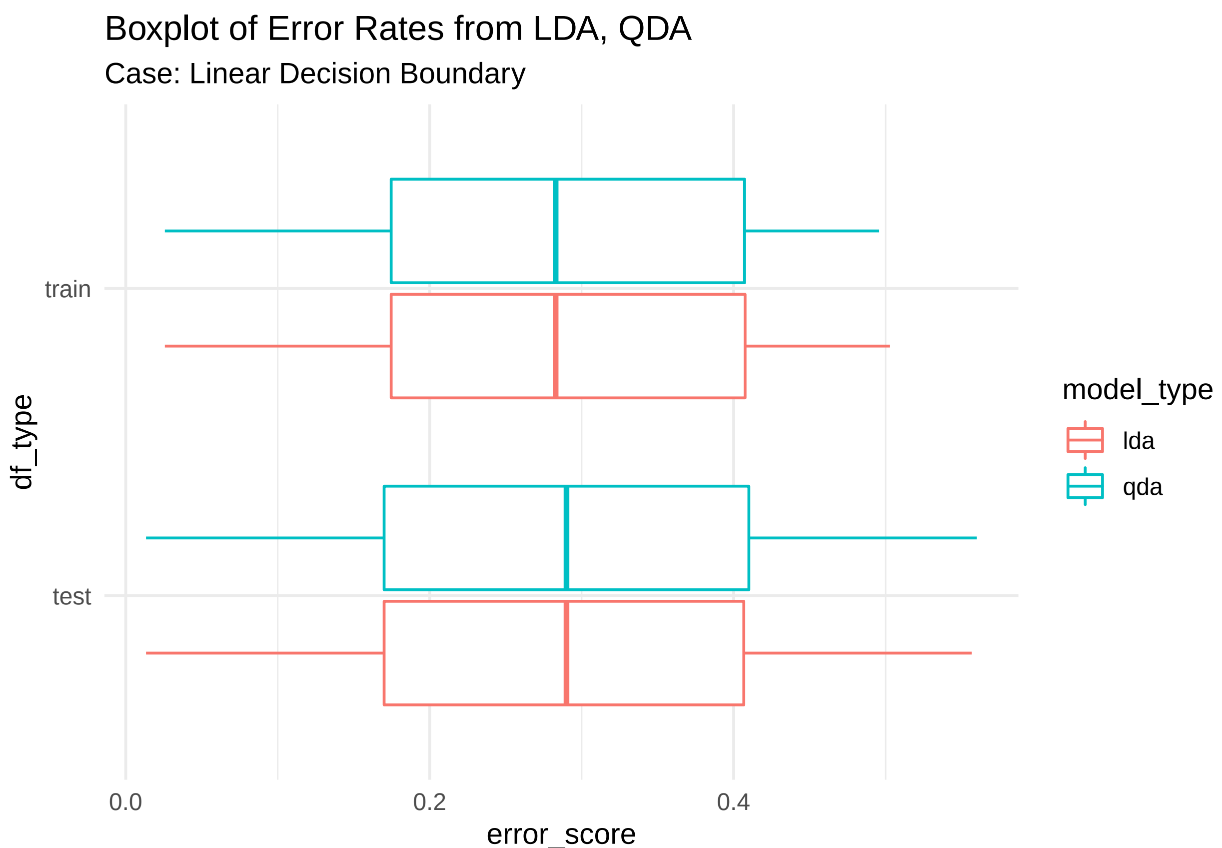
```
## # A tibble: 6 x 9
##   seed_value train_df test_df lda_mod qda_mod model_type metric df_type
##        <int> <list>   <list>  <list>  <list>  <chr>      <chr>  <chr>
## 1        595 <tibble~ <tibbl~ <train> <train> lda        err    train
## 2        595 <tibble~ <tibbl~ <train> <train> lda        err    test
## 3        595 <tibble~ <tibbl~ <train> <train> qda        err    train
## 4        595 <tibble~ <tibbl~ <train> <train> qda        err    test
## 5        992 <tibble~ <tibbl~ <train> <train> lda        err    train
```

```
## 6        992 <tibble~ <tibbl~ <train> <train> lda        err     test
## # ... with 1 more variable: error_score <dbl>
```

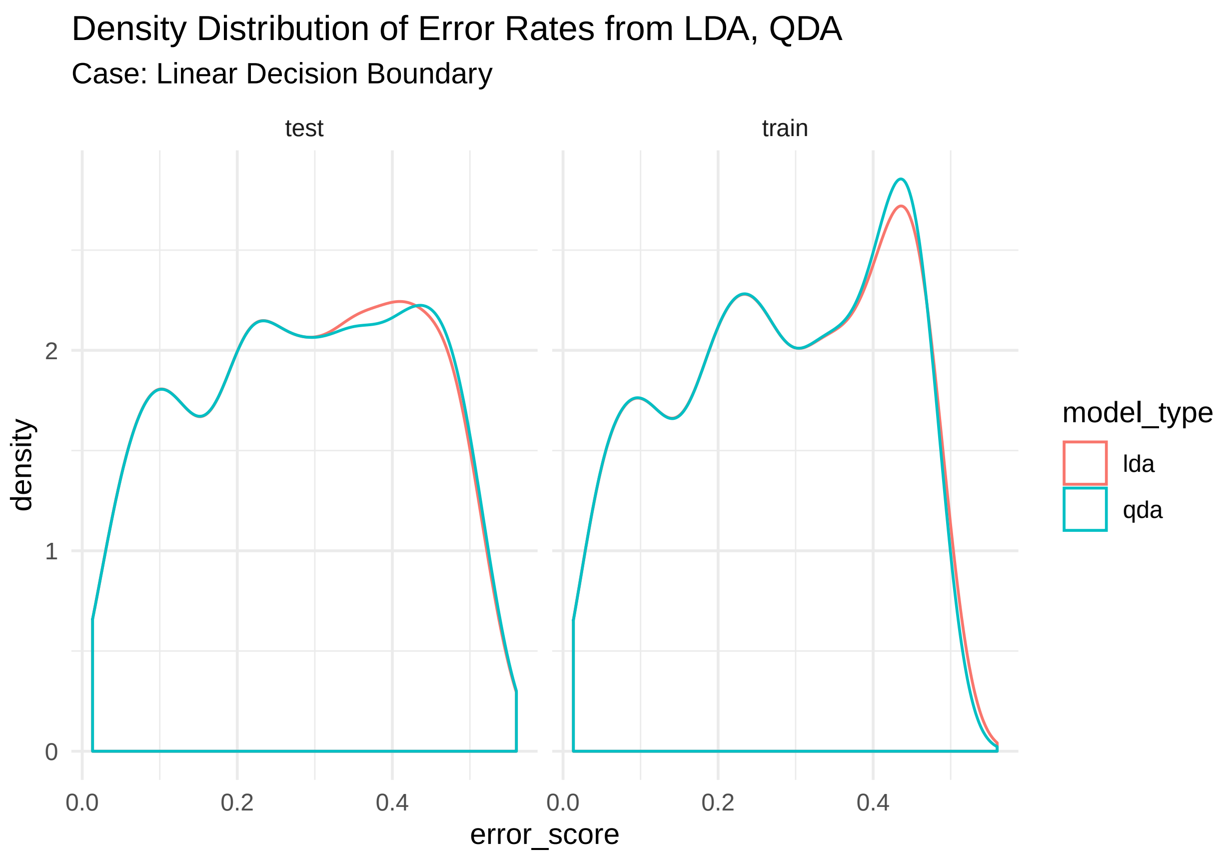
```r
discrim_df_plot %>% group_by(model_type, df_type) %>%
  summarise(mean_error_rate = mean(error_score)) %>%
  pivot_wider(names_from = model_type, values_from = mean_error_rate)
```

```
## # A tibble: 2 x 3
##   df_type   lda   qda
##   <chr>   <dbl> <dbl>
## 1 test    0.284 0.285
## 2 train   0.279 0.279
```

```r
discrim_df_plot %>%
  ggplot() +
  geom_boxplot(aes(x = df_type, y = error_score, colour = model_type)) +
  theme_minimal() + coord_flip() +
  ggtitle("Boxplot of Error Rates from LDA, QDA", subtitle = "Case: Linear Decision Boundary")
```




```r
discrim_df_plot %>%
  ggplot() +
  geom_density(aes(error_score, colour = model_type)) + facet_wrap(~df_type, ncol = 2) +
  theme_minimal() +
  ggtitle("Density Distribution of Error Rates from LDA, QDA", subtitle = "Case: Linear Decision Bounda
```



Conclusion: Both LDA and QDA perform about the same.

Q3: LDA vs QDA with a non-linear Decision Boundary

```
#n_cores <- availableCores() - 2
#plan(multicore, workers = n_cores)

# discrim_df_sec <- seq(1, 1000, by = 1) %>%
#   as_tibble(rownames = NULL) %>%
#   rename(seed_value = value) %>%
#   mutate(train_df = map(seed_value, ~model_df_gen(.x, 1000, TRUE, 0.7, "train")),
#          test_df = map(seed_value, ~model_df_gen(.x, 1000, TRUE, 0.7, "test")),
#          lda_mod = map(train_df, ~model_runner(.x, "lda")),
#          lda_err_train = 1 - map2_dbl(lda_mod, train_df, model_acc_gen),
#          lda_err_test = 1 - map2_dbl(lda_mod, test_df, model_acc_gen),
#          qda_mod = map(train_df, ~model_runner(.x, "qda")),
#          qda_err_train = 1 - map2_dbl(qda_mod, train_df, model_acc_gen),
#          qda_err_test = 1 - map2_dbl(qda_mod, test_df, model_acc_gen))

#head(discrim_df_sec)
```

Since the functional programming approach is giving me inscrutable errors, let's brute-force this with a for-loop.

```
model_err_gen2 <- function(n_obs, split_prop){
```

```r
#set.seed(seed_val)

#print(iter_no)

#x_1 <- runif(n_obs, -1, 1)
#x_2 <- runif(n_obs, -1, 1)
#err <-  rnorm(n_obs, mean = 0, sd = 1)

df <- tibble(x_1 = runif(n_obs, -1, 1),
             x_2 = runif(n_obs, -1, 1),
             err = rnorm(n_obs, mean = 0, sd = 1))%>%
  mutate(y = x_1 + x_2 + x_1^2 + x_2^2,
         y_class = factor(y >= 0, levels = c(TRUE, FALSE)),
         f_x = y + err,
         y_sim_class = factor(f_x >= 0, levels = c(TRUE, FALSE)))



df_split <- rsample::initial_split(df, prop = split_prop)
df_train <- rsample::training(df_split) %>% select(x_1, x_2, y_sim_class) %>% na.omit()
df_test <- rsample::testing(df_split) %>% select(x_1, x_2, y_sim_class) %>% na.omit()

#df_train_ed <- droplevels(df_train)
#print("Here")

#print(head(df_train))
#print(head(df_train$y_sim_class))

# mod <- train(
#   df_train,
#   df_train$y_sim_class,
#   metric = 'Accuracy',
#   method = model_type
# )

lda_mod <- MASS::lda(y_sim_class ~ x_1 + x_2, data = df_train)
qda_mod <- MASS::qda(y_sim_class ~ x_1 + x_2, data = df_train)


train_df_aug <- df_train %>%
  bind_cols('lda_model_outcome' = predict(lda_mod, newdata = df_train)$class,
            'qda_model_outcome' = predict(qda_mod, newdata = df_train)$class)


#print("Training Accuracy:")
lda_train_err <- 1 - postResample(train_df_aug$lda_model_outcome, df_train$y_sim_class)[1]
qda_train_err <- 1 - postResample(train_df_aug$qda_model_outcome, df_train$y_sim_class)[1]

pred_df_aug <- df_test %>%
  bind_cols('lda_model_outcome' = predict(lda_mod, newdata = df_test)$class,
            'qda_model_outcome' = predict(qda_mod, newdata = df_test)$class)

#print("Training Accuracy:")
```

```r
  lda_test_err <- 1 - postResample(pred_df_aug$lda_model_outcome, df_test$y_sim_class)[1]
  qda_test_err <- 1 - postResample(pred_df_aug$qda_model_outcome, df_test$y_sim_class)[1]

  endval <- tibble("lda_train_err" = lda_train_err,
                   "lda_test_err" = lda_test_err,
                   "qda_train_err" = qda_train_err,
                   "qda_test_err" = qda_test_err)

  return(endval)
}
```

```r
discrim_df_sec2 <- model_err_gen2(1000, 0.7)
for(i in 1:999){
  discrim_df_sec2 <- bind_rows(discrim_df_sec2, model_err_gen2(1000, 0.7))
}
head(discrim_df_sec2)
```

```
## # A tibble: 6 x 4
##   lda_train_err lda_test_err qda_train_err qda_test_err
##           <dbl>        <dbl>         <dbl>        <dbl>
## 1         0.263        0.297         0.249        0.283
## 2         0.263        0.28          0.253        0.26
## 3         0.269        0.263         0.259        0.237
## 4         0.254        0.323         0.234        0.293
## 5         0.254        0.253         0.244        0.223
## 6         0.234        0.283         0.220        0.267
```

```r
sec_discrim_df_plot <- discrim_df_sec2 %>%
  pivot_longer(c('lda_train_err', 'lda_test_err','qda_train_err', 'qda_test_err'), names_to = "mod_dfna
  separate(`mod_dfname`, sep = "_", into = c("model_type", "df_type", "metric"))

head(sec_discrim_df_plot)
```

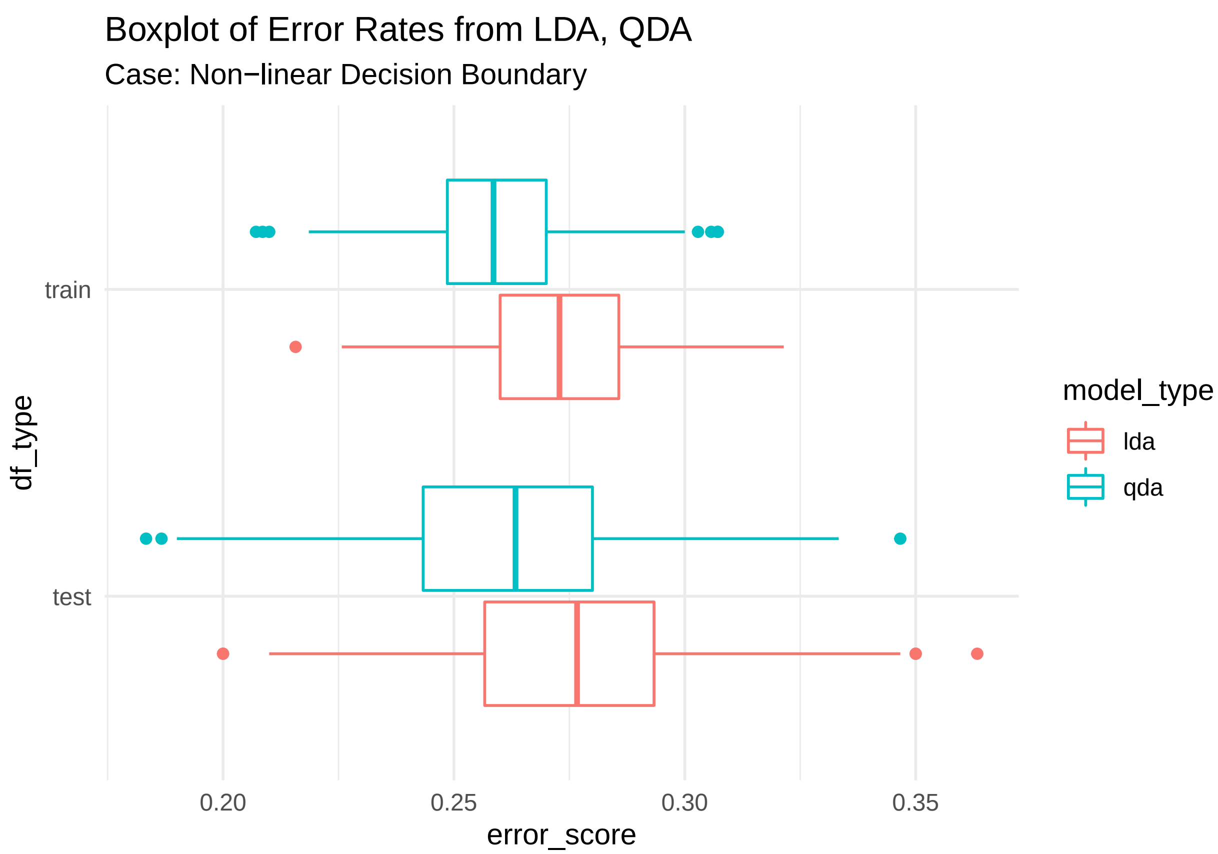
```
## # A tibble: 6 x 4
##   model_type df_type metric error_score
##   <chr>      <chr>   <chr>        <dbl>
## 1 lda        train   err          0.263
## 2 lda        test    err          0.297
## 3 qda        train   err          0.249
## 4 qda        test    err          0.283
## 5 lda        train   err          0.263
## 6 lda        test    err          0.28
```

```r
sec_discrim_df_plot %>% group_by(model_type, df_type) %>%
  summarise(mean_error_rate = mean(error_score)) %>%
  pivot_wider(names_from = model_type, values_from = mean_error_rate)
```

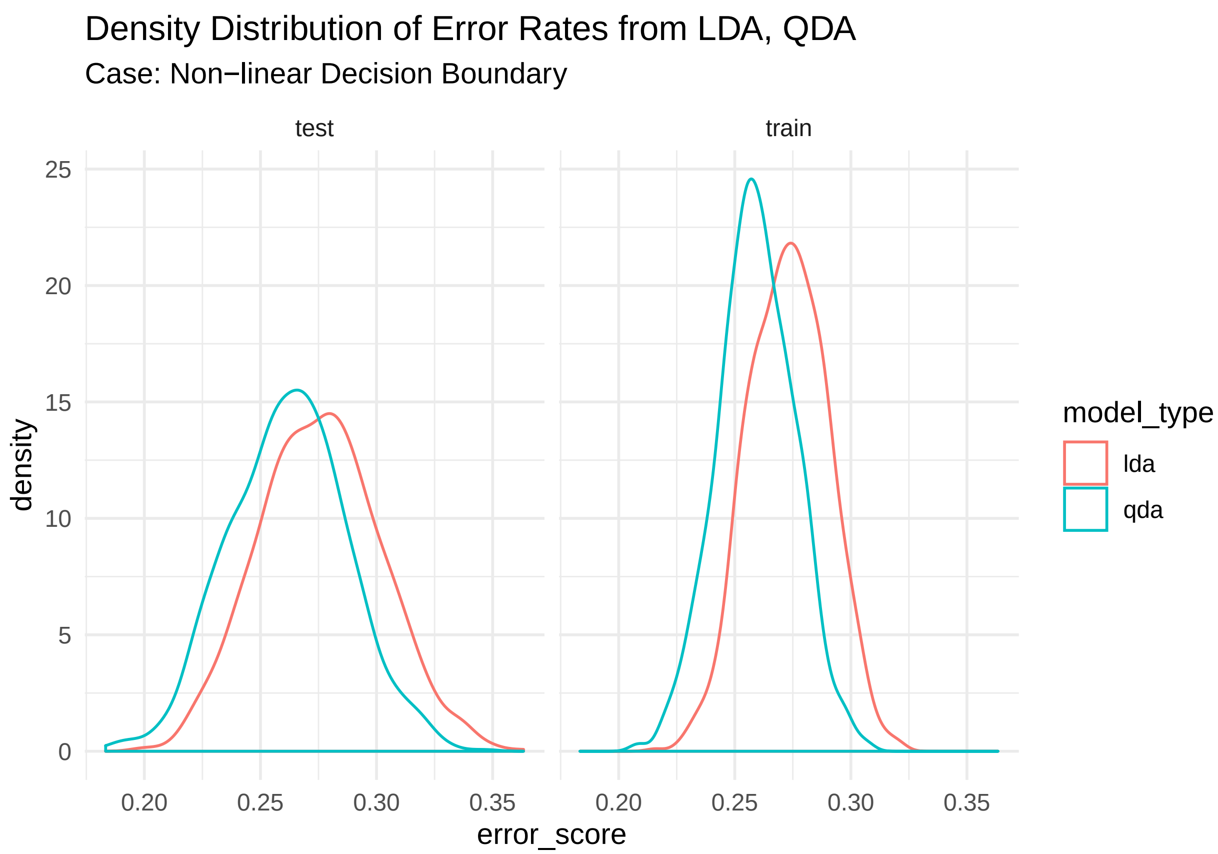
```
## # A tibble: 2 x 3
##   df_type   lda   qda
##   <chr>   <dbl> <dbl>
## 1 test    0.276 0.262
## 2 train   0.273 0.259
```

```r
sec_discrim_df_plot %>%
  ggplot() +
  geom_boxplot(aes(x = df_type, y = error_score, colour = model_type)) +
```

```
  theme_minimal() + coord_flip() +
  ggtitle("Boxplot of Error Rates from LDA, QDA", subtitle = "Case: Non-linear Decision Boundary")
```




```
sec_discrim_df_plot %>%
  ggplot() +
  geom_density(aes(error_score, colour = model_type)) + facet_wrap(~df_type, ncol = 2) +
  theme_minimal() +
  ggtitle("Density Distribution of Error Rates from LDA, QDA", subtitle = "Case: Non-linear Decision Bou
```



Conclusion: We can see that QDA performs better in the case of the non-linear decision boundary.

Q5: Effect of sample size on error rates for the non-linear decision boundary

Re-using the code from Q4:

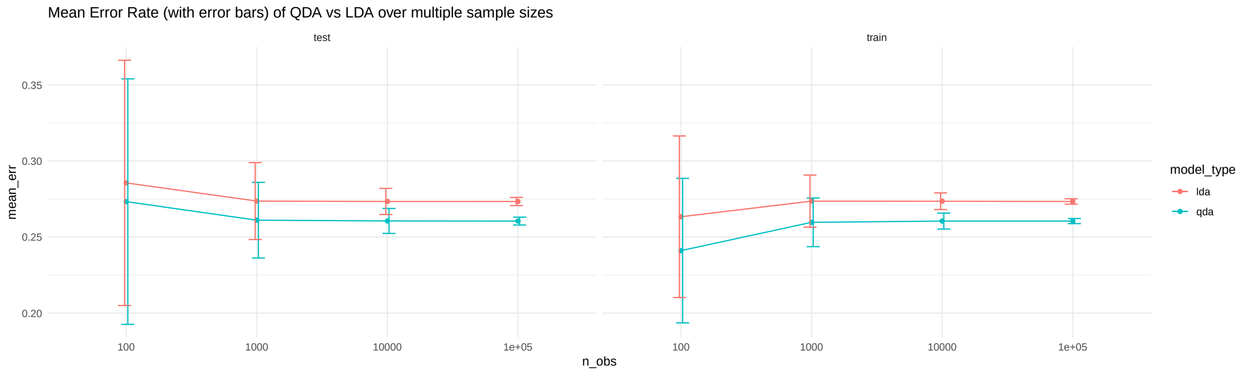
```
n_obs_arr <- c(100,1000, 10000, 100000)

discrim_df_sec_100 <- model_err_gen2(n_obs_arr[1], 0.7)
discrim_df_sec_100 <- discrim_df_sec_100 %>% bind_cols("n_obs" = n_obs_arr[1])
for(i in 1:999){
  discrim_df_sec_100 <- bind_rows(discrim_df_sec_100, model_err_gen2(n_obs_arr[1], 0.7)%>% bind_cols("n_
}

discrim_df_sec_1000 <- model_err_gen2(n_obs_arr[2], 0.7)
discrim_df_sec_1000 <- discrim_df_sec_1000 %>% bind_cols("n_obs" = n_obs_arr[2])
for(i in 1:999){
  discrim_df_sec_1000 <- bind_rows(discrim_df_sec_1000, model_err_gen2(n_obs_arr[2], 0.7)%>% bind_cols(
}

discrim_df_sec_10000 <- model_err_gen2(n_obs_arr[3], 0.7)
discrim_df_sec_10000 <- discrim_df_sec_10000 %>% bind_cols("n_obs" = n_obs_arr[3])
for(i in 1:999){
  discrim_df_sec_10000 <- bind_rows(discrim_df_sec_10000, model_err_gen2(n_obs_arr[3], 0.7)%>% bind_col
}

discrim_df_sec_100000 <- model_err_gen2(n_obs_arr[4], 0.7)
```

```r
discrim_df_sec_100000 <- discrim_df_sec_100000 %>% bind_cols("n_obs" = n_obs_arr[4])
for(i in 1:999){
  discrim_df_sec_100000 <- bind_rows(discrim_df_sec_100000, model_err_gen2(n_obs_arr[4], 0.7)%>% bind_c
}
```

```r
sim_discrim_df_sec <- bind_rows(discrim_df_sec_100, discrim_df_sec_1000, discrim_df_sec_10000, discrim_
  pivot_longer(c('lda_train_err', 'lda_test_err','qda_train_err', 'qda_test_err'), names_to = "mod_dfnam
  separate(`mod_dfname`, sep = "_", into = c("model_type", "df_type", "metric")) %>%
  mutate(n_obs = as.factor(n_obs))

#sim_discrim_df_sec


sim_discrim_df_sec_summ <- sim_discrim_df_sec %>%
  group_by(model_type, df_type, n_obs) %>%
  summarize(mean_err = mean(error_score),
            sd_err = sd(error_score))

(sim_discrim_df_sec_summ)
```

```
## # A tibble: 16 x 5
## # Groups:   model_type, df_type [4]
##    model_type df_type n_obs mean_err  sd_err
##    <chr>      <chr>   <fct>    <dbl>   <dbl>
##  1 lda        test    100      0.286 0.0806
##  2 lda        test    1000     0.274 0.0253
##  3 lda        test    10000    0.273 0.00860
##  4 lda        test    1e+05    0.273 0.00269
##  5 lda        train   100      0.263 0.0531
##  6 lda        train   1000     0.274 0.0171
##  7 lda        train   10000    0.274 0.00548
##  8 lda        train   1e+05    0.273 0.00176
##  9 qda        test    100      0.273 0.0807
## 10 qda        test    1000     0.261 0.0249
## 11 qda        test    10000    0.261 0.00821
## 12 qda        test    1e+05    0.261 0.00258
## 13 qda        train   100      0.241 0.0475
## 14 qda        train   1000     0.260 0.0160
## 15 qda        train   10000    0.261 0.00525
## 16 qda        train   1e+05    0.261 0.00168
```

```r
sim_discrim_df_sec_summ %>%
  ggplot(aes(x = n_obs, y = mean_err, group = model_type, colour = model_type)) +
  geom_line() +
  geom_point()+
  geom_errorbar(aes(ymin=mean_err-sd_err, ymax=mean_err+sd_err), width=.2,
                position=position_dodge(0.05)) +
  facet_wrap(vars(df_type)) + theme_minimal() +
  ggtitle("Mean Error Rate (with error bars) of QDA vs LDA over multiple sample sizes")
```



Conclusion: We notice that QDA has lower error scores relative to LDA consistently, but the effect of increasing sample size is the lower variance in mean error. We get much more stable estimates, with almmost no overlap in the ranges at $n = 10000$.

Q5 - Mental Health Data

```r
mh_df <- read_csv("../problem-set-2/mental_health.csv") %>%
  mutate(age = as.integer(age),
         black = as_factor(black),
         educ = as.integer(educ),
         female = as_factor(female),
         married = as_factor(married),
         mhealth_sum = as.integer(mhealth_sum),
         vote96 = as_factor(vote96))
#head(mh_df)
summary(mh_df)
```

```
##   vote96      mhealth_sum          age             educ        black    female
##  0   : 830   Min.   : 0.000   Min.   :18.00   Min.   : 0.00   0:2432   0:1232
##  1   :1783   1st Qu.: 1.000   1st Qu.:32.00   1st Qu.:12.00   1: 400   1:1600
##  NA's: 219   Median : 2.000   Median :42.00   Median :13.00
##              Mean   : 2.869   Mean   :45.56   Mean   :13.25
##              3rd Qu.: 4.000   3rd Qu.:57.00   3rd Qu.:16.00
##              Max.   :16.000   Max.   :89.00   Max.   :20.00
##              NA's   :1418     NA's   :4       NA's   :12
##  married         inc10
##  0   :1485   Min.   : 0.0535
##  1   :1346   1st Qu.: 2.0062
##  NA's:   1   Median : 3.4774
##              Mean   : 4.5761
##              3rd Qu.: 5.8849
##              Max.   :14.8796
##              NA's   :329
```

We have a large number of `NA`s, and I don't have a good theory for imputation, so I'll go ahead and drop them all for this analysis.

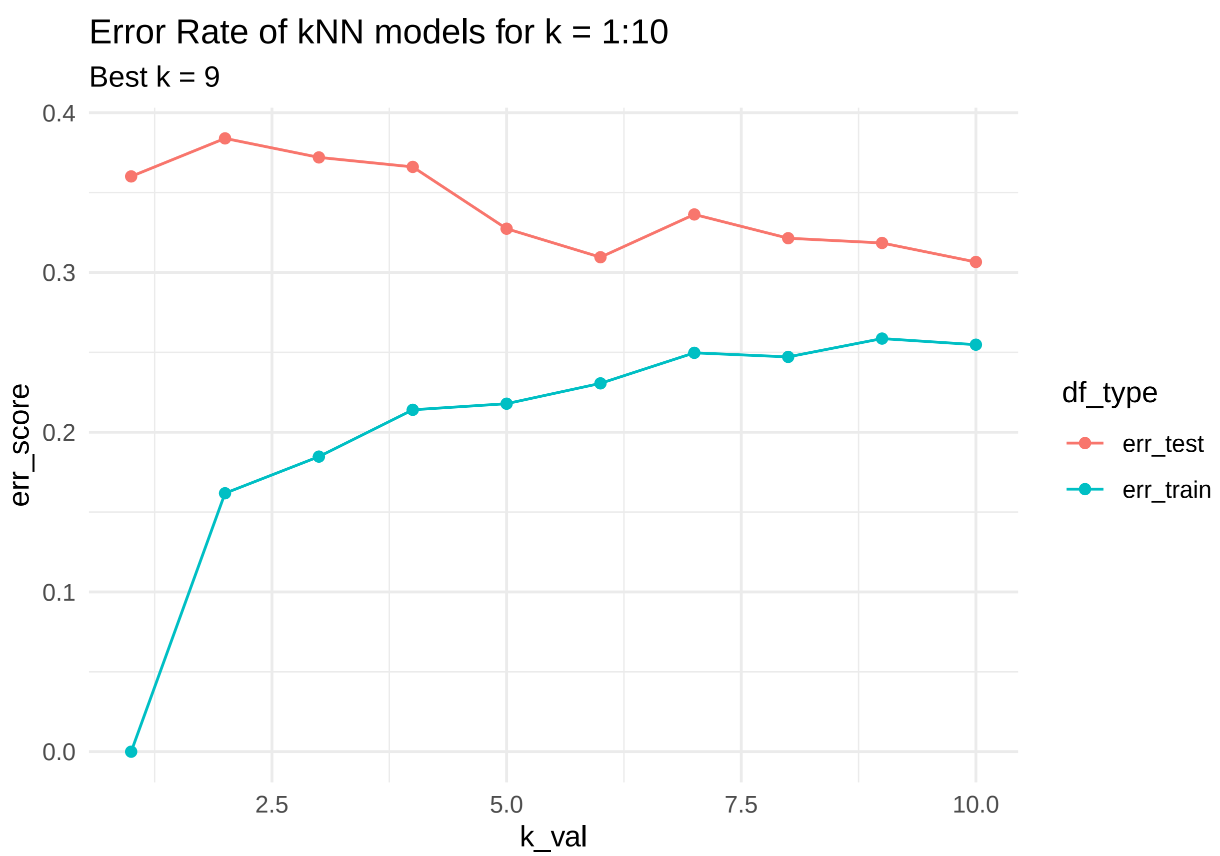
Additionally, the HW prompt says that `mhealth_sum` has values from 0 to 9. So I'll drop anything greater than 9.

```r
mh_df_clean <- mh_df %>% na.omit() %>% filter(mhealth_sum <= 9)

mh_df_split <- rsample::initial_split(mh_df_clean, prop = 0.7)
mh_df_train <- rsample::training(mh_df_split)
mh_df_test <- rsample::testing(mh_df_split)
```

```r
logit_mh_model <- glm(vote96 ~ ., family = "binomial", data = mh_df_train)
lda_mh_model <- MASS::lda(vote96 ~ ., data = mh_df_train)
qda_mh_model <- MASS::qda(vote96 ~ ., data = mh_df_train)
nb_mh_model <- train(x = mh_df_train %>% select(-vote96), y = mh_df_train$vote96, method = "nb")
```

```r
knn_mh_model <- tibble(k_val = 1:10,
                       knn_mod = map(k_val, ~knn3( formula = vote96 ~ .,
                                                   data=mh_df_train, k=.x))) %>%
```

```r
  mutate(train_classes = map(knn_mod, ~predict(.x, newdata = mh_df_train, type = "class")),
         test_classes = map(knn_mod, ~predict(.x, newdata = mh_df_test, type = "class")),
         err_train = 1 - map_dbl(train_classes, ~postResample(.x, mh_df_train$vote96)[1]),
         err_test = 1 - map_dbl(test_classes, ~postResample(.x, mh_df_test$vote96)[1]))

knn_mh_model %>% pivot_longer(c("err_test", "err_train"), names_to = "df_type", values_to = "err_score"
  ggplot(aes(x = k_val, y = err_score, colour = df_type) ) +
  geom_line() + geom_point() + theme_minimal() +
  ggtitle("Error Rate of kNN models for k = 1:10", subtitle = "Best k = 9")
```




```r
knn_mh_model_best <- knn_mh_model[9,'knn_mod']$knn_mod[[1]]
```

```r
logit_res <- predict(logit_mh_model, mh_df_test)
test_prob <- tibble(
  logit = exp(logit_res)/(1+exp(logit_res)),
  lda = predict(lda_mh_model, mh_df_test)$posterior[,'1'],
  qda = predict(qda_mh_model, mh_df_test)$posterior[,'1'],
  nbayes = predict(nb_mh_model, mh_df_test, type = 'prob')$'1',
  knn_9 = predict(knn_mh_model_best, newdata = mh_df_test, type = "prob")[,'1'])
```

```r
mh_df_test_pred <- mh_df_test %>% bind_cols(test_prob %>% mutate_all(~if_else(.x > 0.5, 1, 0)))

class_err_test <- tibble(
  logit = 1 - postResample(mh_df_test_pred$logit, mh_df_test_pred$vote96)[1],
  lda = 1 - postResample(mh_df_test_pred$lda, mh_df_test_pred$vote96)[1],
  qda = 1 - postResample(mh_df_test_pred$qda, mh_df_test_pred$vote96)[1],
```

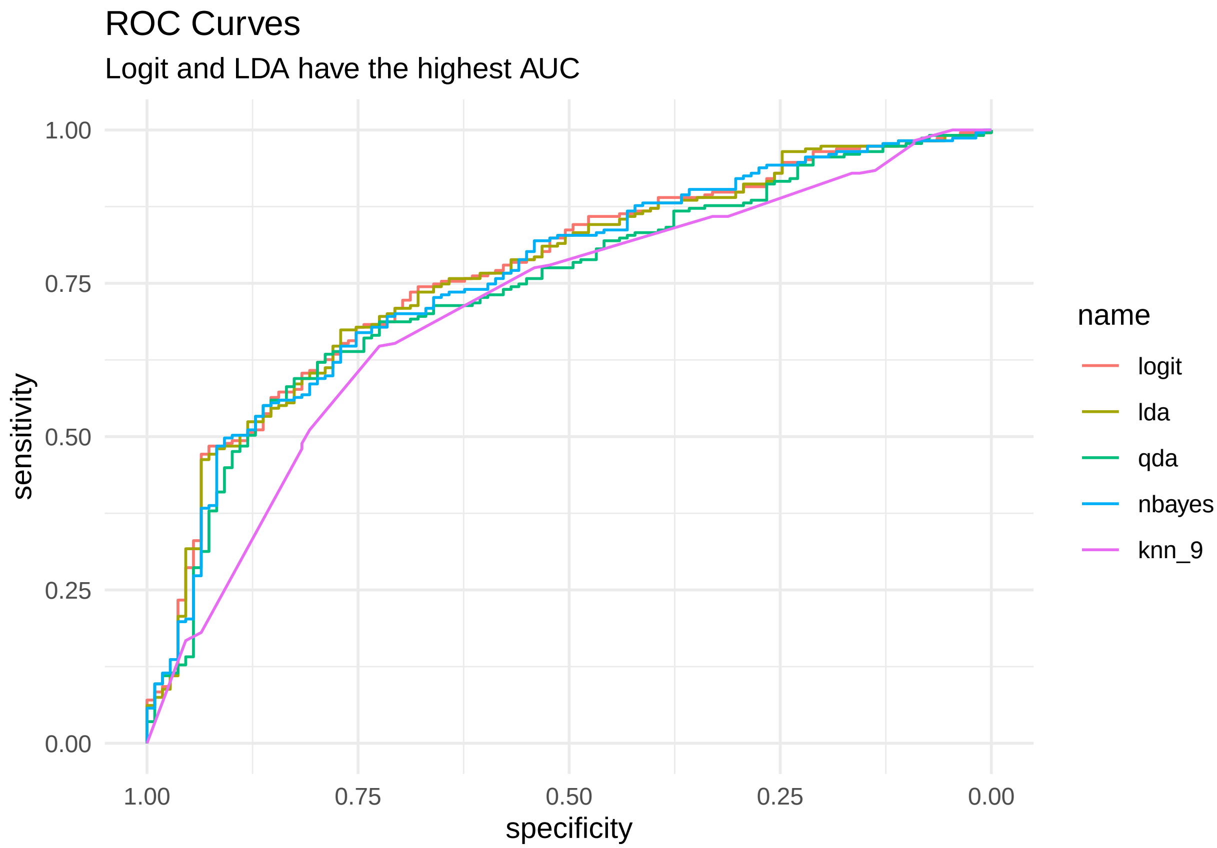
```r
  nbayes = 1 - postResample(mh_df_test_pred$nbayes, mh_df_test_pred$vote96)[1],
  knn_9 = 1 -  postResample(mh_df_test_pred$knn_9, mh_df_test_pred$vote96)[1]
)

class_err_test
```

```
## # A tibble: 1 x 5
##   logit   lda   qda nbayes knn_9
##   <dbl> <dbl> <dbl>  <dbl> <dbl>
## 1 0.277 0.280 0.310  0.286 0.318
```

```r
test_prob_roc <- lapply(test_prob, pROC::roc, response = mh_df_test$vote96)
test_auc <- lapply(test_prob_roc, pROC::auc)
test_auc
```

```
## $logit
## Area under the curve: 0.7674
##
## $lda
## Area under the curve: 0.7658
##
## $qda
## Area under the curve: 0.7425
##
## $nbayes
## Area under the curve: 0.7604
##
## $knn_9
## Area under the curve: 0.7086
```

```r
ggroc(test_prob_roc) +
  theme_minimal() +
  ggtitle("ROC Curves", subtitle = "Logit and LDA have the highest AUC")
```



Conclusion: We have two metrics of identifying the 'best' model here: `AUC` and `test_error_rate`.

- Naive Bayes has the lowest test error rate (0.2857143), while its AUC is 0.7337.
- Logit has a higher test error rate (0.2767857), but it has the highest AUC of 0.745.

The two models are largely comparable without much relative loss in performance. Since this is a binary classification problem, our choice of threshold (0.5 here) plays a key role in class assignment. Our accuracy measures might waver as we change the threshold. AUC is a more stable metric for binary classification. So, if forced to make a choice, I would choose the Logit model.