Hu_Anqi_HW2

Anqi Hu

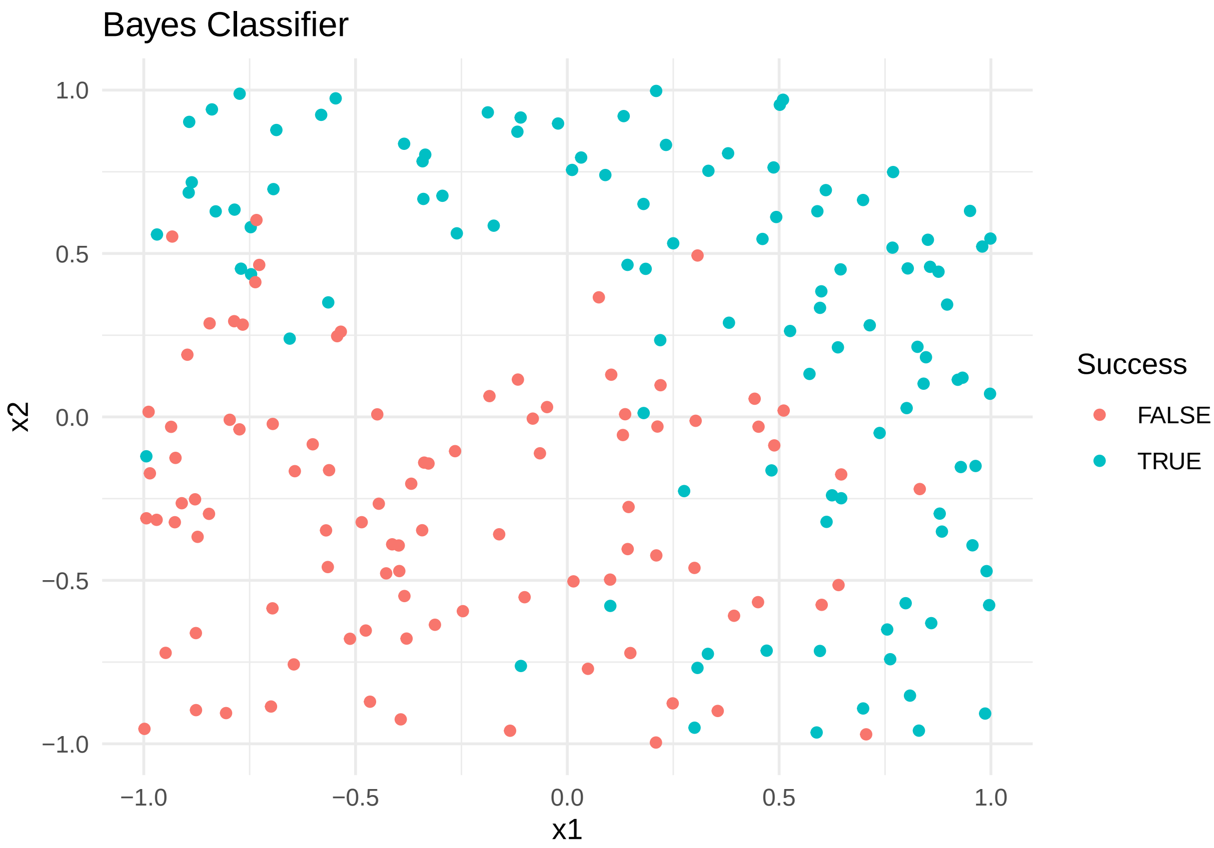
2/2/2020

```r
# set seed
set.seed(90210)
options(digits = 3)
theme_set(theme_minimal())
```

The Bayes Classifier

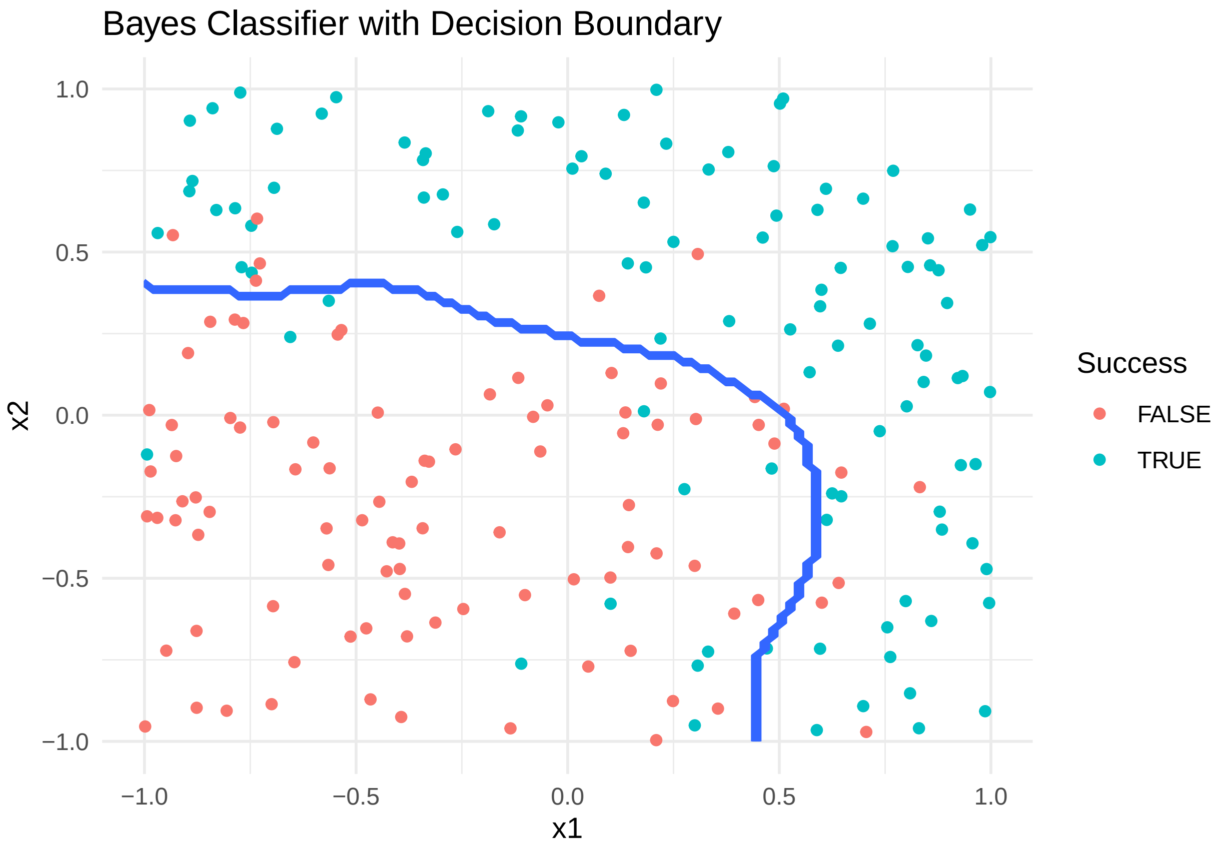
Problem 1

```r
# simulate, calculate y
x1 = runif(200, -1, 1)
x2 = runif(200, -1, 1)
y = x1 + x1^2 + x2 + x2^2 + rnorm(200, 0, 0.5)
sim_logit <- data.frame(x1, x2, y)

# calculate success
logit2prob <- function(x){
  exp(x) / (1 + exp(x))
}

sim_logit <- sim_logit %>%
  mutate(prob = logit2prob(y))

sim_logit$success <- ifelse(sim_logit$y > 0.5, TRUE, FALSE)

# plot
ggplot(sim_logit, aes(x1, x2)) +
  geom_point(aes(col = sim_logit$success)) +
  labs(title = "Bayes Classifier",
       x = "x1",
       y = "x2") +
  scale_color_discrete(name = "Success")
```




```r
new_df <- expand_grid(x1 = seq(-1, 1, length.out = 100),
                      x2 = seq(-1, 1, length.out = 100))

new_df <- as.data.frame(new_df)

new_x <- sim_logit[, 1:2]
new_y <- sim_logit$success
new_train <- cbind(new_x, new_y)

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

nb.m1 <- train(
  x = new_x,
  y = as.factor(new_y),
  method = "nb",
  trControl = train_control
)

new_df$pred <- predict(nb.m1, newdata = new_df)

# plot overlay with boundary
ggplot(new_train) +
  geom_point(aes(x = new_x$x1, y = new_x$x2, col = new_y)) +
  geom_contour(data = new_df,
```

```r
               aes(x1, x2, z=as.numeric(pred))) +
  labs(title = "Bayes Classifier with Decision Boundary",
       x = "x1",
       y = "x2") +
  scale_color_discrete(name = "Success")
```



LDA and QDA

Problem 2

a

```r
linear_error_df <- data.frame(matrix(ncol = 4, nrow = 0))

i = 0
while (i < 1000) {
  x1 = runif(1000, -1, 1)
  x2 = runif(1000, -1, 1)
  simulate <- tibble(x1, x2)
  y <- x1 + x1^2 + x2 + x2^2 + rnorm(1000, 0, 1)
  simulate$y <- y > 0

  split <- initial_split(simulate, prop = .7)
  train <- training(split)
```

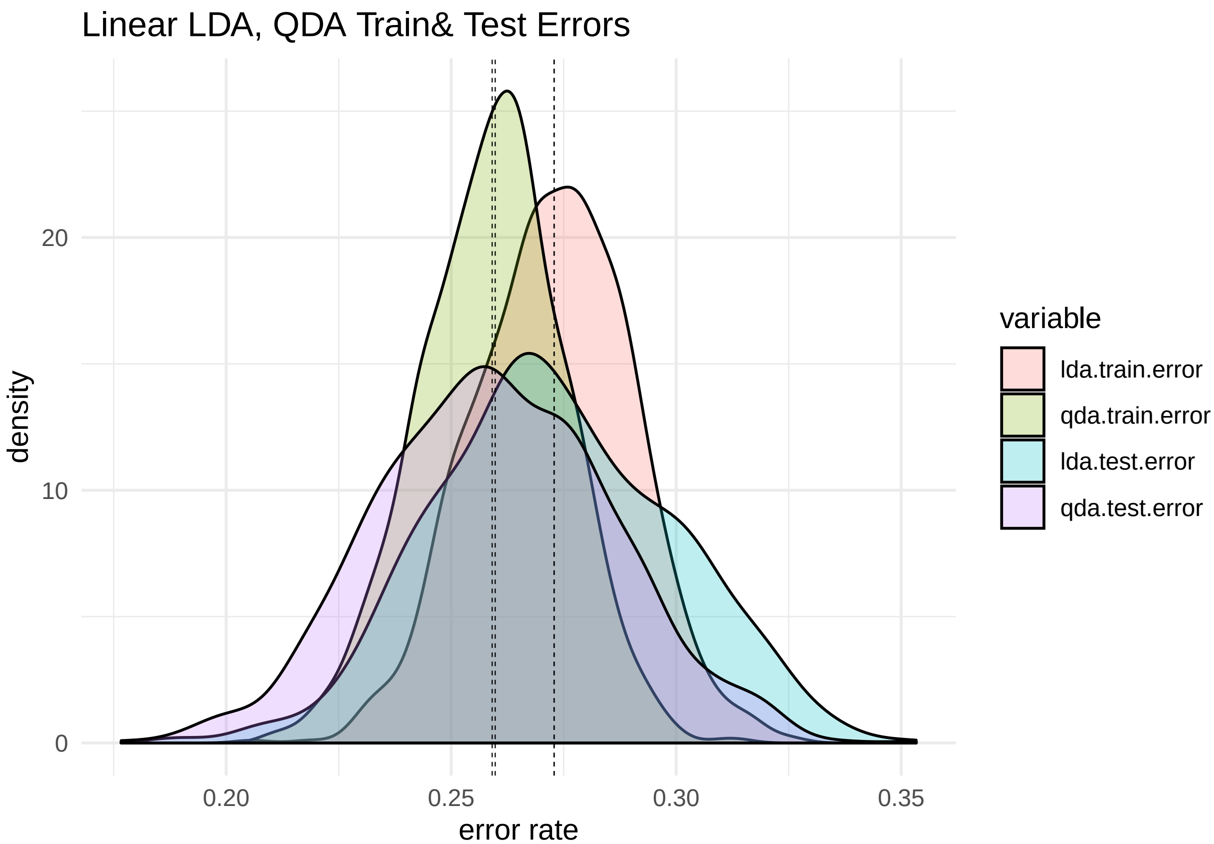
```r
  test <- testing(split)

  lda_m1 <- lda(y ~ x1 + x2, data = train)
  qda_m1 <- qda(y ~ x1 + x2, data = train)

  #LDA model
  predmodel_train_lda <- predict(lda_m1, data=train)
  predmodel_test_lda <- predict(lda_m1, newdata=test)

  #QDA model
  predmodel_train_qda <- predict(qda_m1, data=train)
  predmodel_test_qda <- predict(qda_m1, newdata=test)

  train_error <- train %>%
    summarise(lda.train.error = mean(train$y != predmodel_train_lda$class),
              qda.train.error = mean(train$y != predmodel_train_qda$class))

  test_error <- test %>%
    summarise(lda.test.error = mean(test$y != predmodel_test_lda$class),
              qda.test.error = mean(test$y != predmodel_test_qda$class))

  errors <- cbind(train_error, test_error)
  linear_error_df <- rbind(linear_error_df, errors)
i = i + 1
}
```

b

```r
linear_sum <- do.call(cbind, lapply(linear_error_df, summary))

sum_df <- cbind(as.data.frame(apply(linear_error_df, 2, sd)),
                as.data.frame(t(linear_sum))[, 4])
colnames(sum_df) <- c("St.Dev", "Mean")

forplot <- melt(linear_error_df)

sum_df
```

```
##                 St.Dev  Mean
## lda.train.error 0.0172 0.273
## qda.train.error 0.0158 0.259
## lda.test.error  0.0266 0.273
## qda.test.error  0.0260 0.260
```

```r
ggplot(forplot, aes(x = value, fill = variable)) +
  geom_density(alpha = 0.25) +
  geom_vline(data = sum_df["Mean"], aes(xintercept= Mean),
             color= "black", linetype="dashed", size=0.2) +
  labs(title = "Linear LDA, QDA Train& Test Errors",
       x = "error rate") +
  scale_color_discrete(name = "Type of Error")
```



Compared to LDA, QDA has a lower training error on average, as well as a smaller variance. Similarly, when it comes to testing error, QDA also seems to be performing slightly better (i.e. a lower mean and a smaller variance). These differences in performance only seem to be moderate. In addition, from the graph we can see that the training errors of both models center around 0.26. This is also the case for testing error, as the means are centered around 0.27. Overall, the training errors and the testing errors seem to be equidistant, with QDA performing better.

Problem 3

a

```
nonlinear_error_df = data.frame(matrix(ncol = 4, nrow = 0))

i = 0
while (i < 1000) {
  x1 = runif(1000, -1, 1)
  x2 = runif(1000, -1, 1)
  simulate <- tibble(x1, x2)
  y <- x1 + x1^2 + x2 + x2^2 + rnorm(1000, 0, 1)
  simulate$y <- y > 0

  split <- initial_split(simulate, prop = .7)
  train <- training(split)
  test <- testing(split)

  lda_m2 <- lda(y ~ x1 + x1^2 + x2 + x2^2, data = train)
```

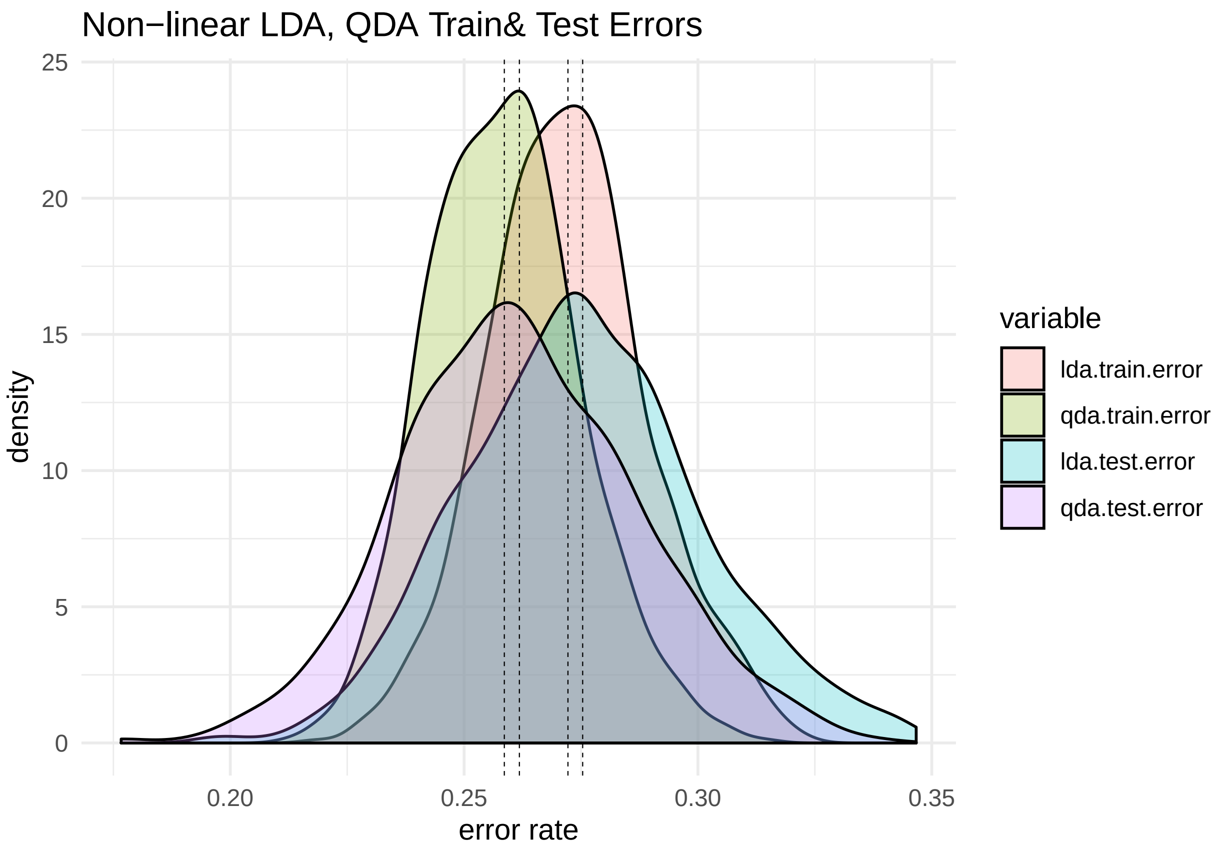
```r
  qda_m2 <- qda(y ~ x1 + x1^2 + x2 + x2^2, data = train)

  #LDA model
  predmodel_train_lda <- predict(lda_m2, data=train, type = "prob")
  predmodel_test_lda <- predict(lda_m2, newdata=test)

  #QDA model
  predmodel_train_qda <- predict(qda_m2, data=train)
  predmodel_test_qda <- predict(qda_m2, newdata=test)

  train_error <- train %>%
    summarise(lda.train.error = mean(train$y != predmodel_train_lda$class),
              qda.train.error = mean(train$y != predmodel_train_qda$class))

  test_error <- test %>%
    summarise(lda.test.error = mean(test$y != predmodel_test_lda$class),
              qda.test.error = mean(test$y != predmodel_test_qda$class))

  errors <- cbind(train_error, test_error)
  nonlinear_error_df <- rbind(nonlinear_error_df, errors)
i = i + 1
}
```

b

```r
nonlinear_sum <- do.call(cbind, lapply(nonlinear_error_df, summary))

sum_df <- cbind(as.data.frame(apply(nonlinear_error_df, 2, sd)),
                as.data.frame(t(nonlinear_sum))[, 4])
colnames(sum_df) <- c("St.Dev", "Mean")

forplot <- melt(nonlinear_error_df)

sum_df
```

```
##                 St.Dev  Mean
## lda.train.error 0.0170 0.272
## qda.train.error 0.0159 0.259
## lda.test.error  0.0253 0.275
## qda.test.error  0.0252 0.262
```

```r
ggplot(forplot, aes(x = value, fill = variable)) +
  geom_density(alpha = 0.25) +
  geom_vline(data = sum_df["Mean"], aes(xintercept= Mean),
             color= "black", linetype="dashed", size=0.2) +
  labs(title = "Non-linear LDA, QDA Train& Test Errors",
       x = "error rate") +
  scale_color_discrete(name = "Type of Error")
```



Compared to LDA, QDA has a lower training error on average, as well as a smaller variance. Similarly, when it comes to testing error, QDA also seems to be performing slightly better (i.e. a lower mean but same variance as LDA). Like the case for the linear model, these differences in performance only seem to be moderate. In addition, from the graph we can see that the training errors of both models center around 0.265. This is also the case for testing error, as the means are centered around 0.27. Overall, the training errors and the testing errors seem to be equidistant, with QDA performing better.

Problem 4

a

```
new_func <- function(n_size) {
  x1 = runif(n_size, -1, 1)
  x2 = runif(n_size, -1, 1)
  simulate <- tibble(x1, x2)
  y <- x1 + x1^2 + x2 + x2^2 + rnorm(n_size, 0, 1)
  simulate$y <- y > 0

  split <- initial_split(simulate, prop = .7)
  train <- training(split)
  test <- testing(split)

  lda_m3 <- lda(y ~ x1 + x1^2 + x2 + x2^2, data = train)
  qda_m3 <- qda(y ~ x1 + x1^2 + x2 + x2^2, data = train)

  #LDA model
```

```
  predmodel_train_lda = predict(lda_m3, data=train)
  predmodel_test_lda = predict(lda_m3, newdata=test)

  #QDA model
  predmodel_train_qda = predict(qda_m3, data=train)
  predmodel_test_qda = predict(qda_m3, newdata=test)

  test_error <- test %>%
    summarise(lda.test.error = mean(test$y != predmodel_test_lda$class),
              qda.test.error = mean(test$y != predmodel_test_qda$class))
}

error_1e02 = data.frame(matrix(ncol = 4, nrow = 0))
error_1e03 = data.frame(matrix(ncol = 4, nrow = 0))
error_1e04 = data.frame(matrix(ncol = 4, nrow = 0))
error_1e05 = data.frame(matrix(ncol = 4, nrow = 0))

i = 0
while (i < 1000) {
  error_1e02 <- rbind(error_1e02, new_func(100))
  error_1e03 <- rbind(error_1e03, new_func(1000))
  error_1e04 <- rbind(error_1e04, new_func(10000))
  error_1e05 <- rbind(error_1e05, new_func(100000))
  i = i + 1
}

all_errors <- cbind(error_1e02, error_1e03, error_1e04, error_1e05)
```

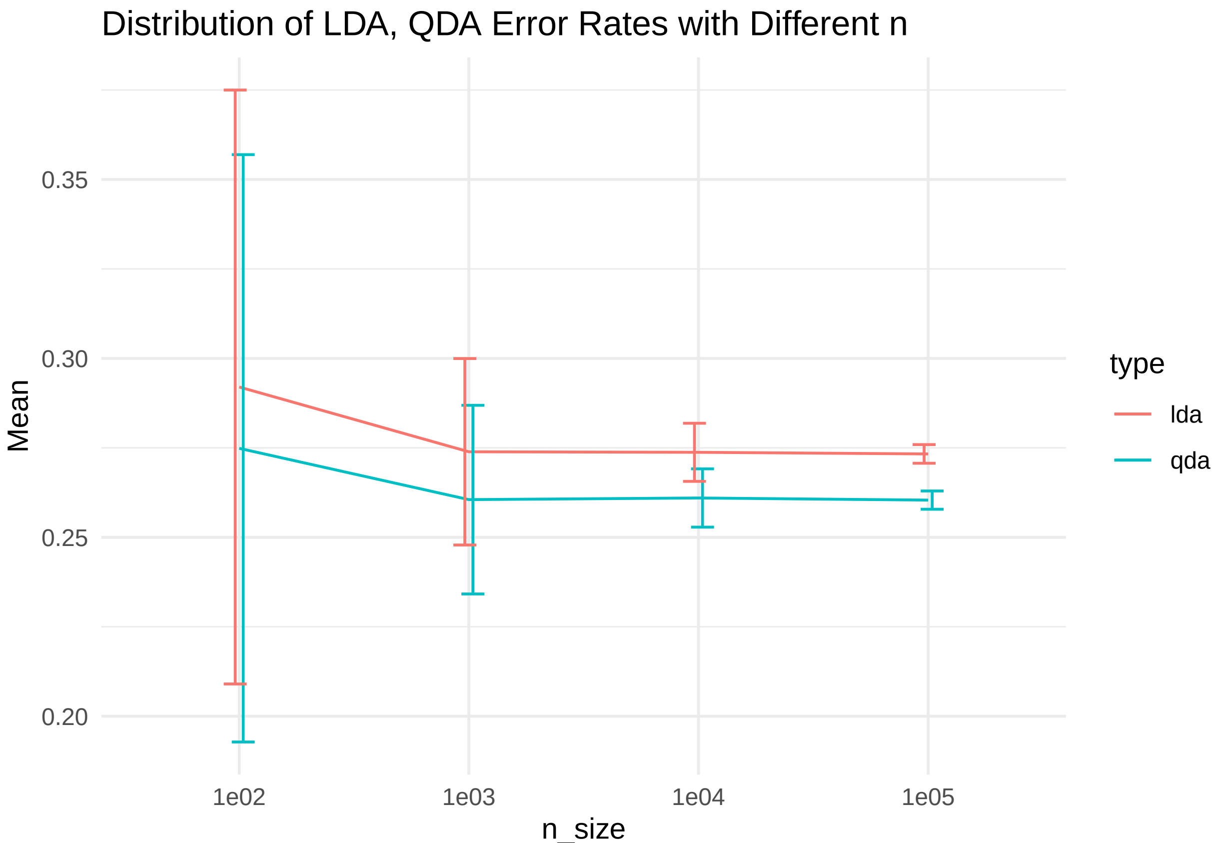
b

```
all_sum <- do.call(cbind, lapply(all_errors, summary))

type <- rep(c("lda", "qda"), 4)
n_size <- c("1e02", "1e02", "1e03", "1e03", "1e04", "1e04", "1e05", "1e05")

all_sum_df <- cbind(type, n_size, as.data.frame(apply(all_errors, 2, sd)), as.data.frame(t(all_sum))[, 4
colnames(all_sum_df) <- c("type", "n_size", "St.Dev", "Mean")

all_sum_df
```

```
##   type n_size  St.Dev  Mean
## 1  lda   1e02 0.08299 0.292
## 2  qda   1e02 0.08207 0.275
## 3  lda   1e03 0.02605 0.274
## 4  qda   1e03 0.02636 0.261
## 5  lda   1e04 0.00812 0.274
## 6  qda   1e04 0.00814 0.261
## 7  lda   1e05 0.00260 0.273
## 8  qda   1e05 0.00255 0.260
```

```
ggplot(all_sum_df, aes(x = n_size, y = Mean, group = type, col = type)) +
  geom_line() +
  geom_errorbar(aes(ymin = Mean-St.Dev, ymax = Mean+St.Dev),
```

```
                width = 0.2,
                position= position_dodge(0.07)) +
  labs(title = "Distribution of LDA, QDA Error Rates with Different n")
```



The mean value of testing error rates decreases as the number of observations increases. This is true for both LDA and QDA. In addition, the variance of the models decreases significantly as n increases in orders of magnitude. The QDA error seems to be approaching 0.26 and LDA error is approaching 0.275.

Modeling Voter Turnout

Problem 5

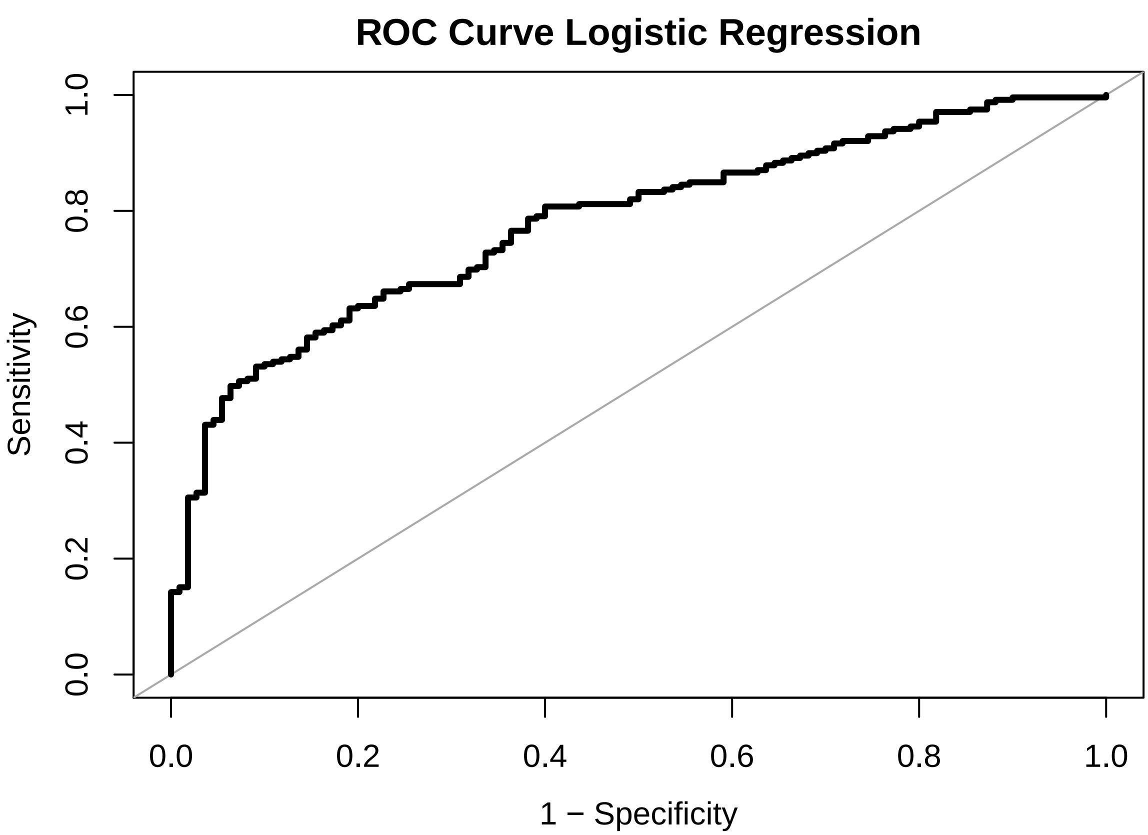
a

```
voter <- read.csv(file = "mental_health.csv")
voter <- as.tibble(na.omit(voter))

split <- initial_split(voter, prop = .7)
train <- training(split)
test <- testing(split)

features <- setdiff(names(train), "vote96")
x1 <- train[, features]
y1 <- train$vote96 > 0
```

```r
x2 <- test[, features]
y2 <- test$vote96 > 0
```

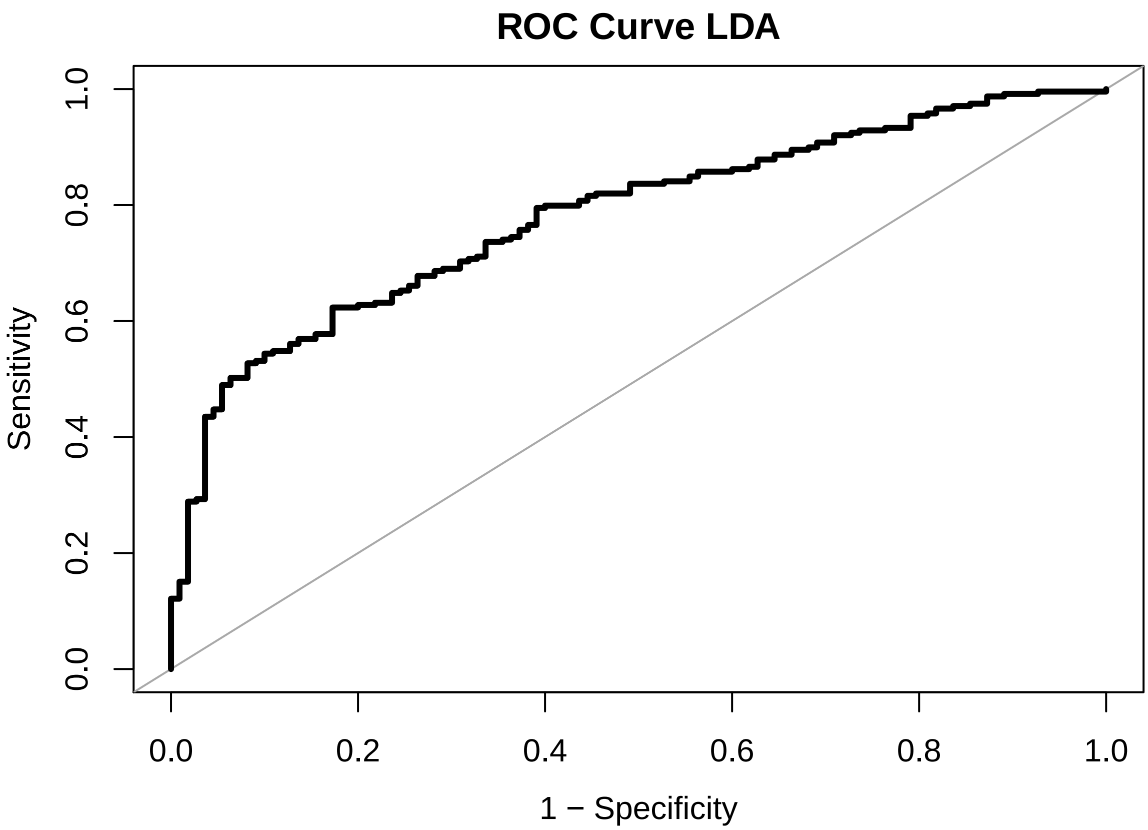
b

```r
# i. Logistic regression
logit_mod <- glm(vote96 ~ mhealth_sum + age + educ + black + female + married + inc10,
                 data = train, family = "binomial")

log_prob <- predict(logit_mod, newdata = test,
                    select = c(2:8), type = "response")

log_pred <- ifelse(log_prob > 0.5, TRUE, FALSE)

#error rate
log_err <- mean(log_pred != test$vote96)

#ROC, AUC
log_roc <- roc(as.numeric(test$vote96), log_prob)
log_auc <- auc(as.numeric(test$vote96), log_prob)

plot(log_roc, legacy.axes = TRUE, asp = NA, lwd = 3, col = "black",
     main = "ROC Curve Logistic Regression")
```

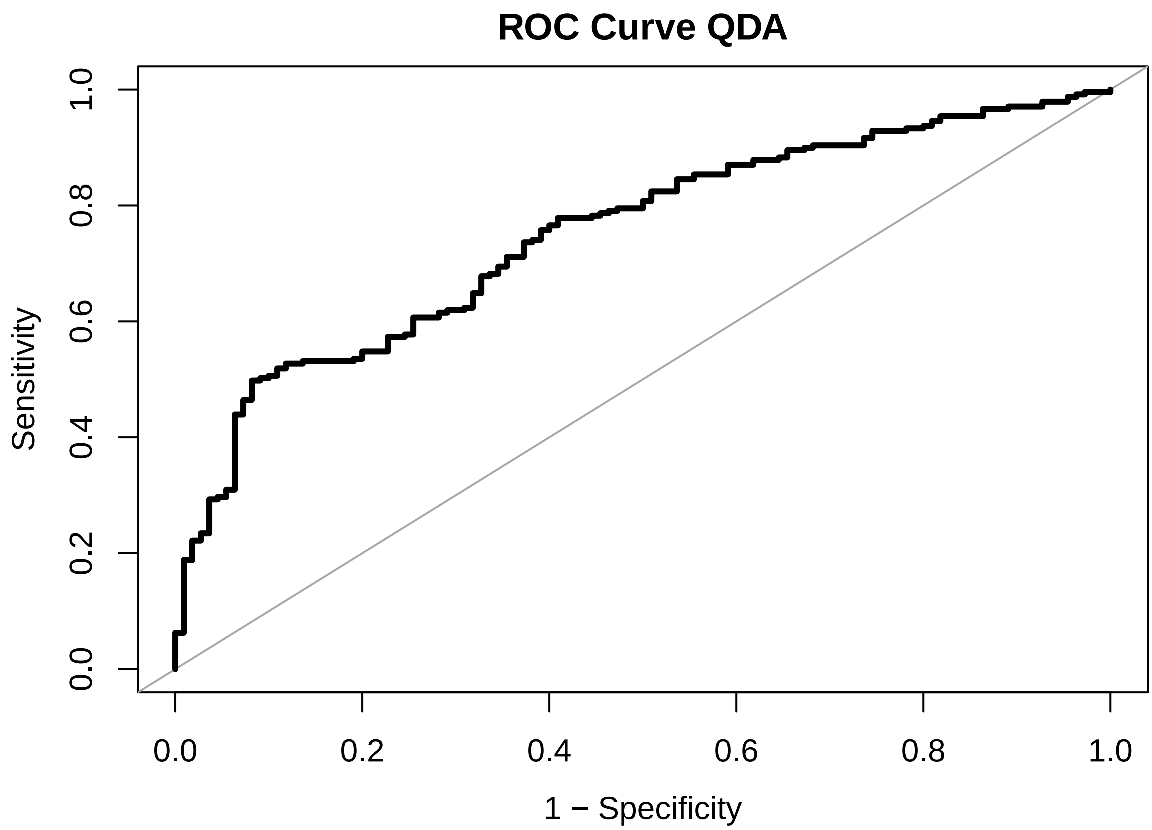



```r
# ii. LDA
lda_m4 <- lda(vote96 ~ mhealth_sum + age + educ + black + female + married + inc10, data = train)
```

```r
lda_pred <- predict(lda_m4, newdata = test, select = c(2:8))

#error rate
lda_err<- mean((as.numeric(lda_pred$class)-1) != test$vote96)

#ROC, AUC
lda_prob <- lda_pred$posterior[, 2]

lda_roc <- roc(as.numeric(test$vote96), lda_prob)
lda_auc <- auc(as.numeric(test$vote96), lda_prob, direction = c("auto"))
plot(lda_roc, legacy.axes = TRUE, asp = NA, lwd = 3, col = "black", main = "ROC Curve LDA")
```

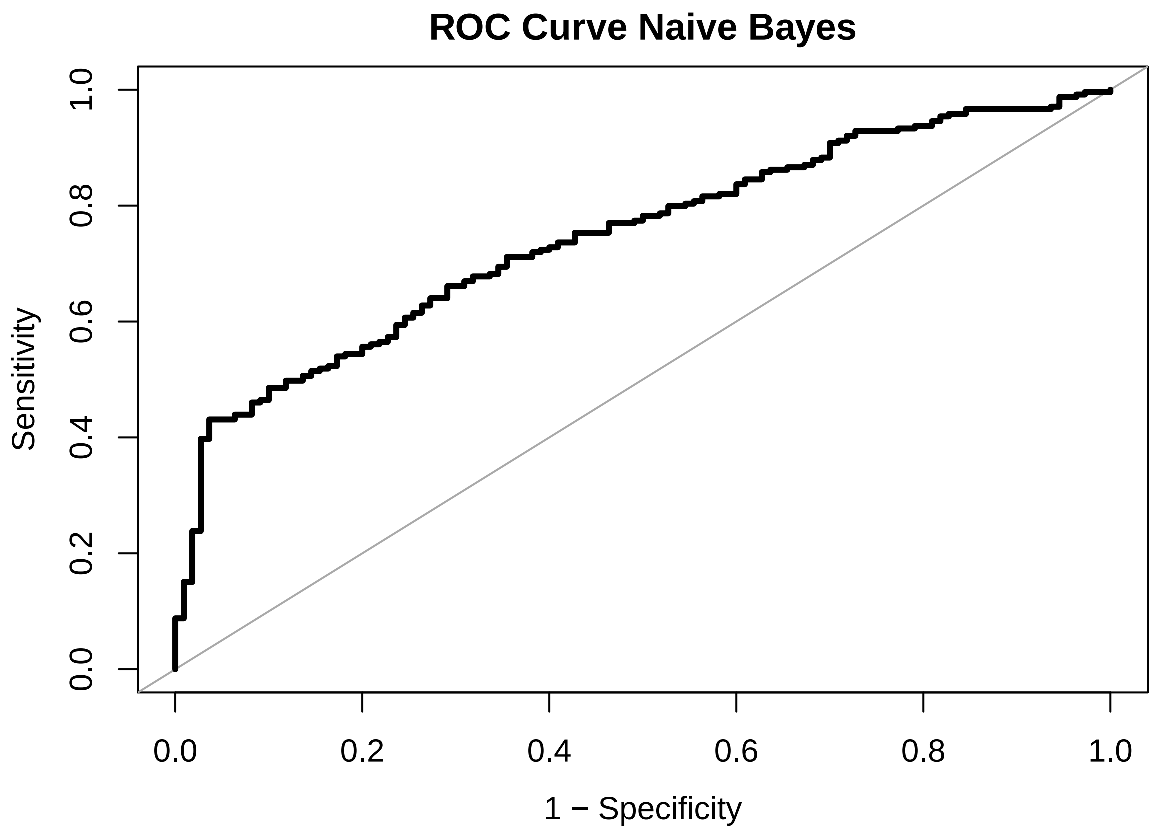



```r
# iii. QDA
qda_m4 <- qda(vote96 ~ mhealth_sum + age + educ + black + female + married + inc10, data = train)

qda_pred <- predict(qda_m4, newdata = test, select = c(2:8))

#error rate
qda_err <- mean((as.numeric(qda_pred$class)-1) != test$vote96)

#ROC, AUC
qda_prob <- qda_pred$posterior[, 2]

qda_roc <- roc(as.numeric(test$vote96), qda_prob)
qda_auc <- auc(as.numeric(test$vote96), qda_prob, direction = c("auto"))
plot(qda_roc, legacy.axes = TRUE, asp = NA, lwd = 3, col = "black", main = "ROC Curve QDA")
```

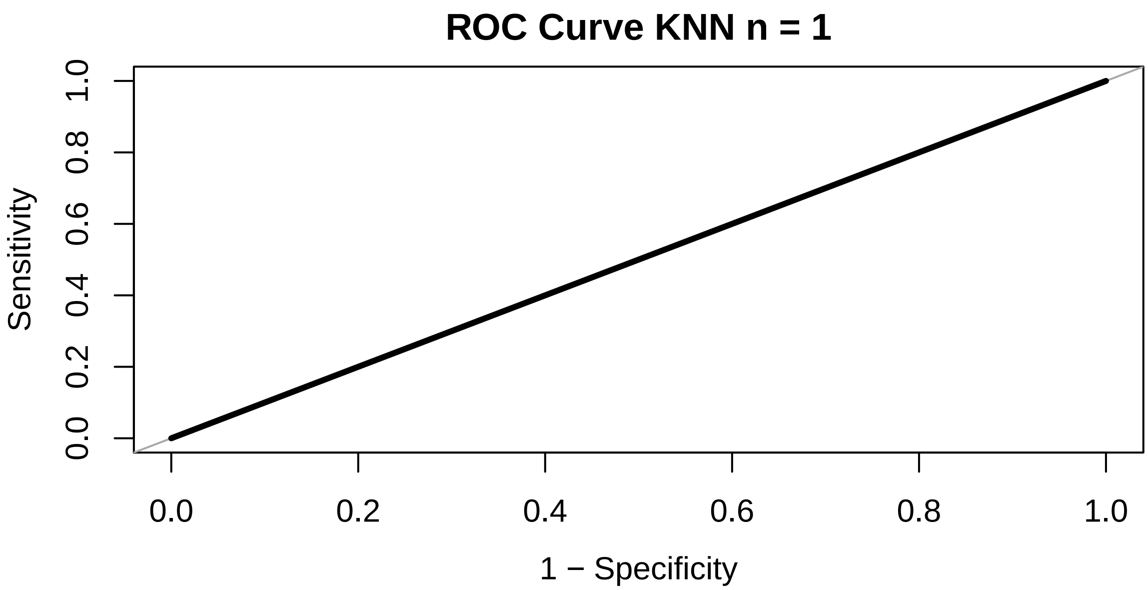


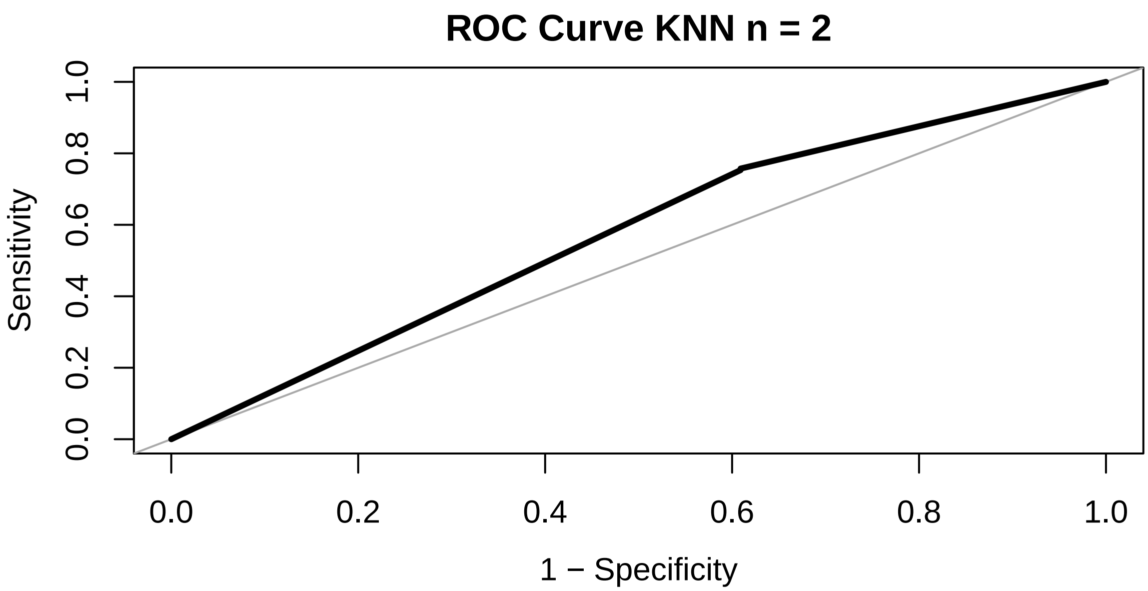

```r
# iv. Naive Bayes
test <- test %>%
  mutate(prob = logit2prob(vote96))

train <- train %>%
  mutate(prob = logit2prob(vote96))

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

nb.m3 <- train(
  x = x1,
  y = as.factor(train$vote96),
  method = "nb",
  trControl = train_control
)

nb_pred <- predict(nb.m3, newdata = test, select = c(2:8), type = "raw")

nb_prob <- predict(nb.m3, newdata = test, select = c(2:8), type = "prob")
nb_prob <- nb_prob[, "1"]

#error rate
nb_err <- mean(nb_pred != test$vote96)
```

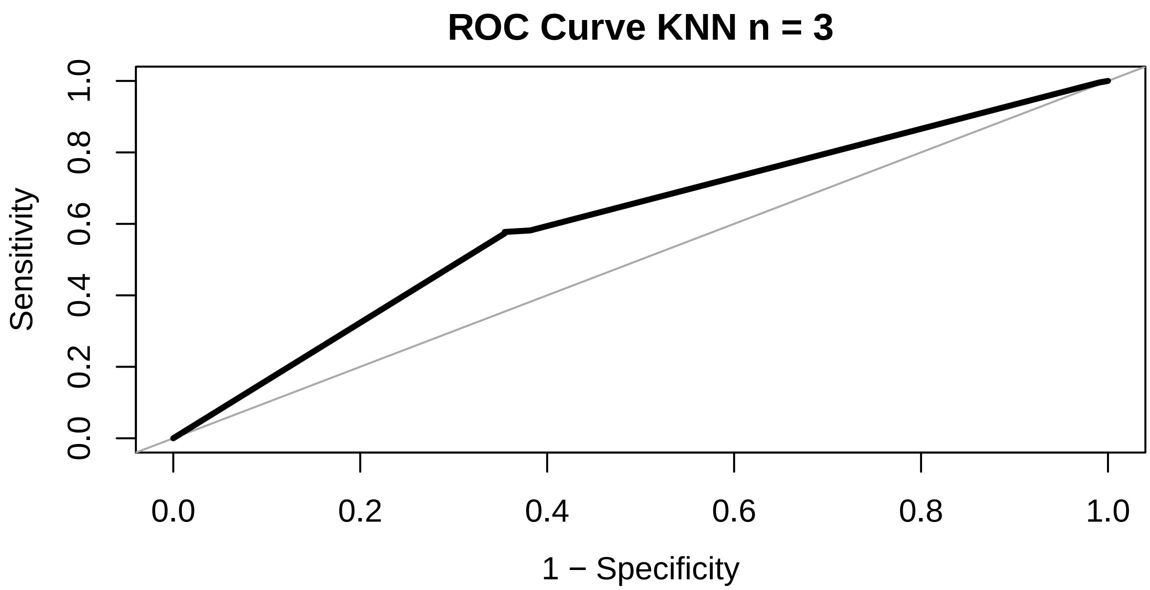
```
#ROC, AUC
nd_roc <- roc(as.numeric(test$vote96), nb_prob)
nb_auc <- auc(as.numeric(test$vote96), nb_prob, direction = c("auto"))
plot(nd_roc, legacy.axes = TRUE, asp = NA, lwd = 3, col = "black", main = "ROC Curve Naive Bayes")
```

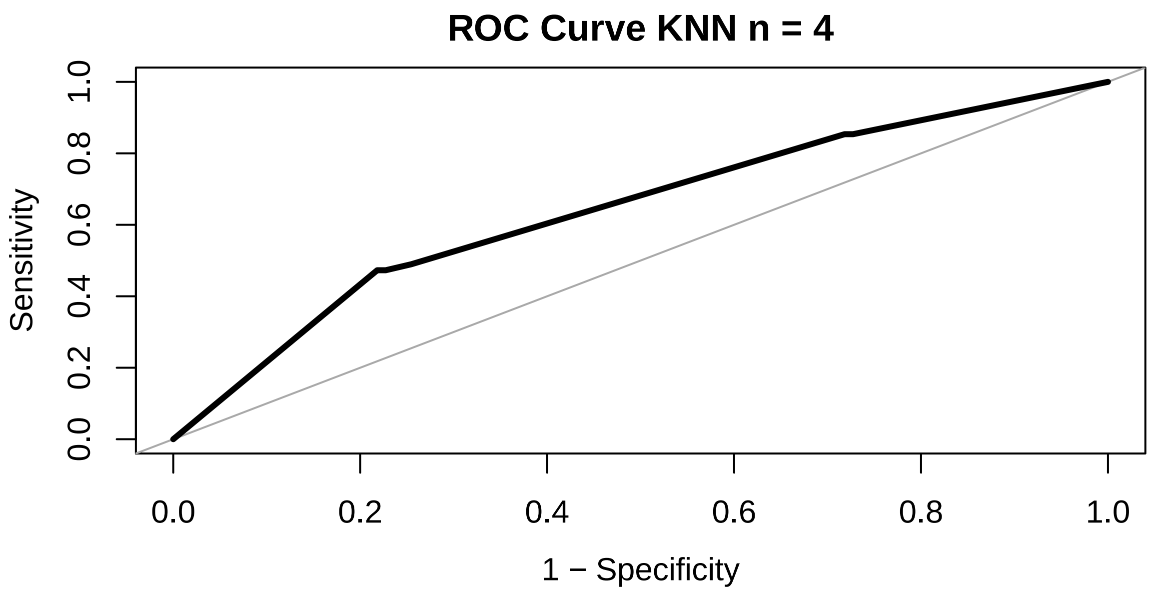


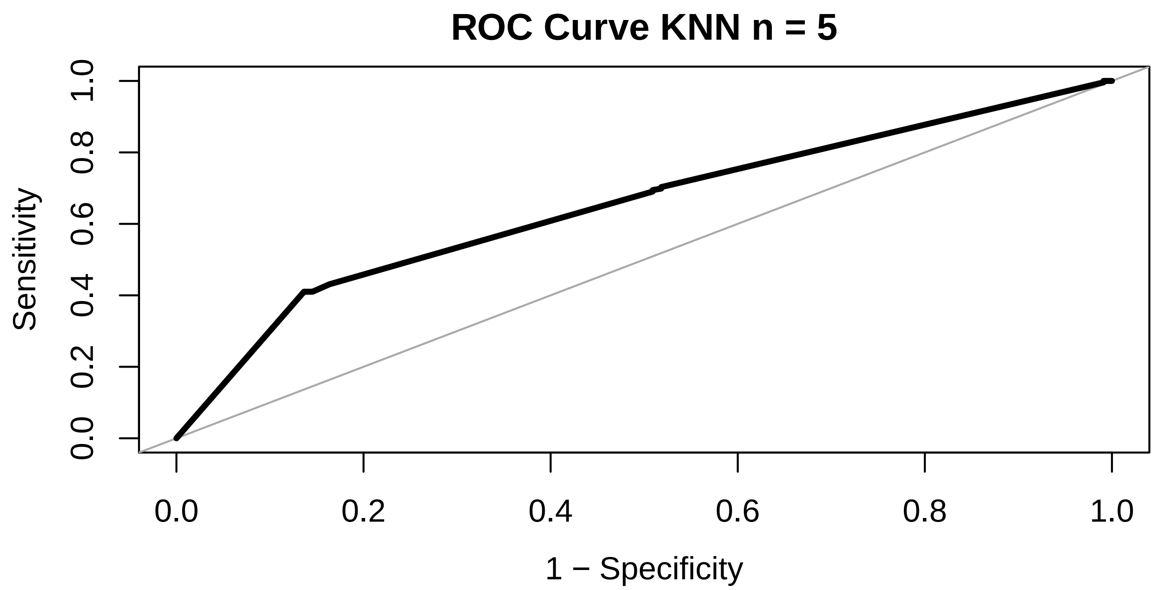

```
# v. KNN
knn_err <- data.frame(matrix(ncol = 0, nrow = 1))
all_knn_prob <- data.frame(matrix(ncol = 0, nrow = 10))
knn_auc <- data.frame(matrix(ncol = 0, nrow = 1))

for (i in 1:10) {
  mod <- knn(train = train, test = test, cl = train$vote96, k = i, prob = TRUE)
  error_rate = mean((as.numeric(mod) - 1) != test$vote96)
  knn_prob <- attr(mod, "prob")
  #ROC
  knn_roc <- roc(as.numeric(test$vote96), knn_prob)
  plot(knn_roc, legacy.axes = TRUE, asp = NA, lwd = 3, col = "black", main = paste("ROC Curve KNN n =",
  auc_1 <- auc(as.numeric(test$vote96), knn_prob)
  knn_err <- cbind(knn_err, error_rate)
  knn_auc <- cbind(knn_auc, auc_1)
}
```


ROC Curve KNN n = 1
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
1 − Specificity

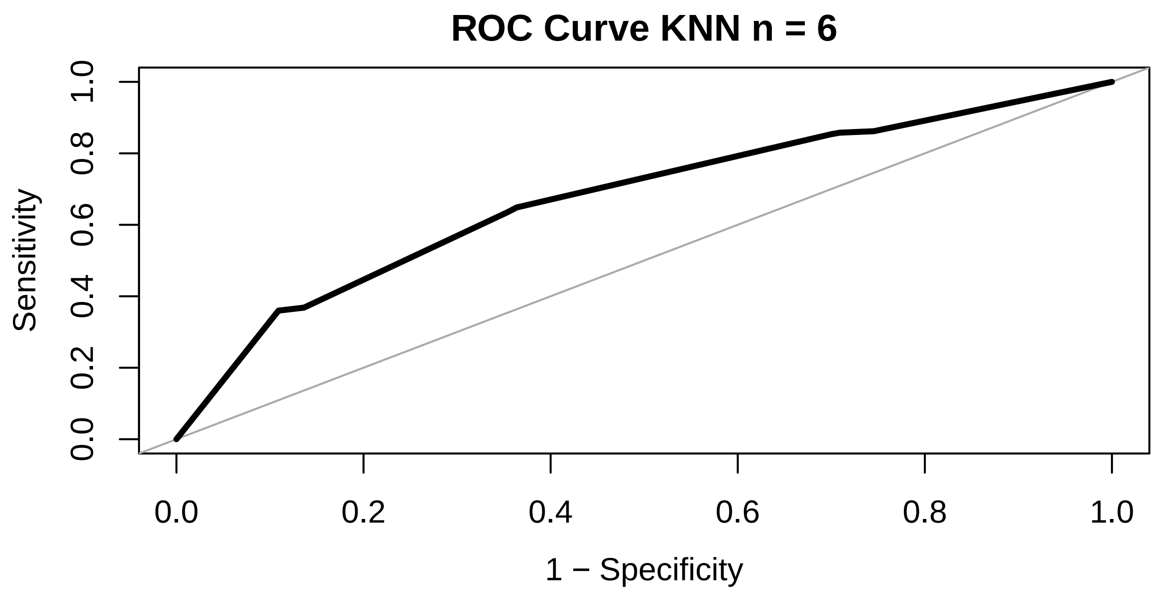

ROC Curve KNN n = 2
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
1 − Specificity


ROC Curve KNN n = 3
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
1 − Specificity

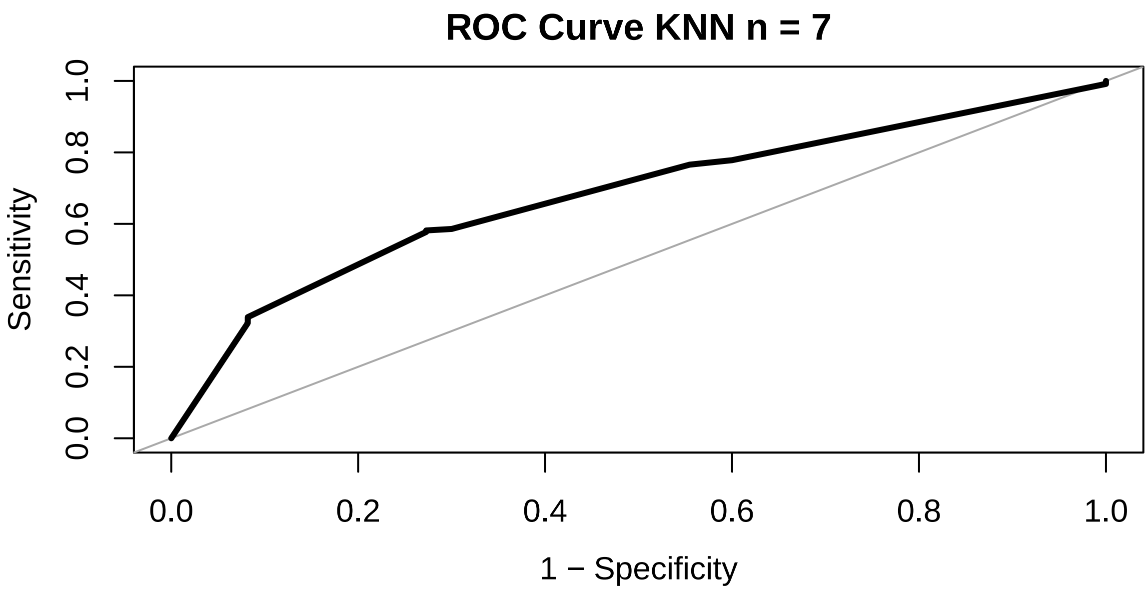

ROC Curve KNN n = 4
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
1 − Specificity

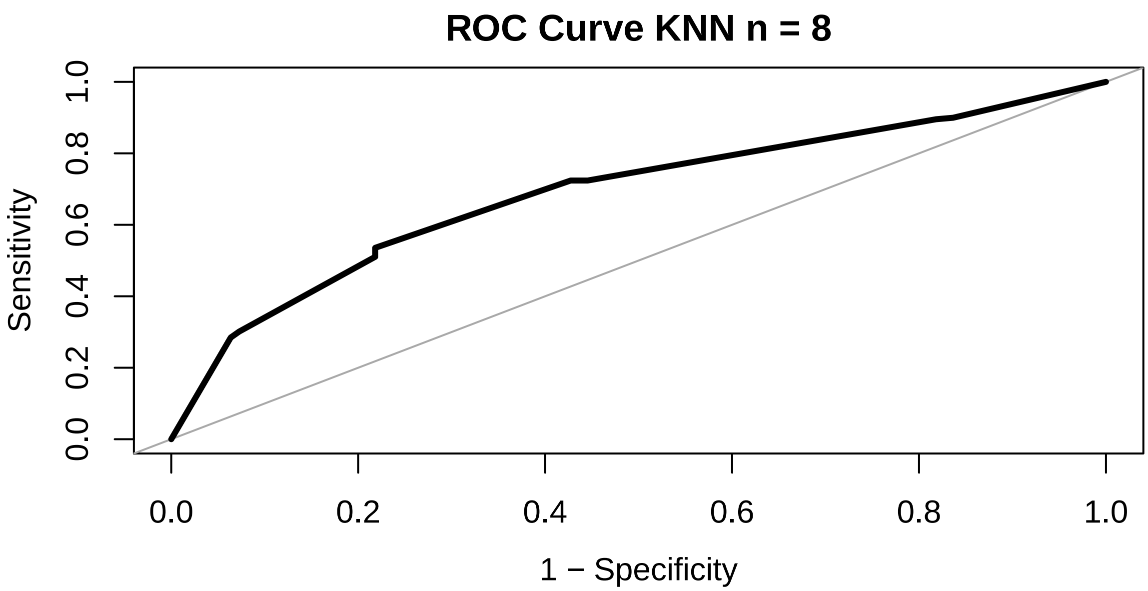


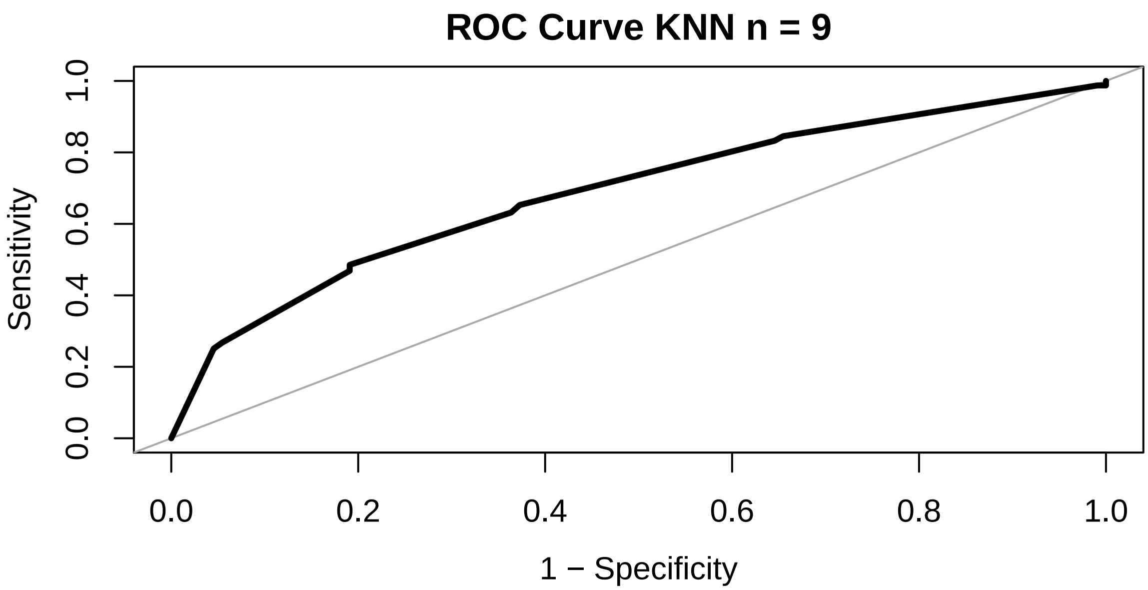
ROC Curve KNN n = 5
Sensitivity
1 − Specificity
0.0
0.2
0.4
0.6
0.8
1.0

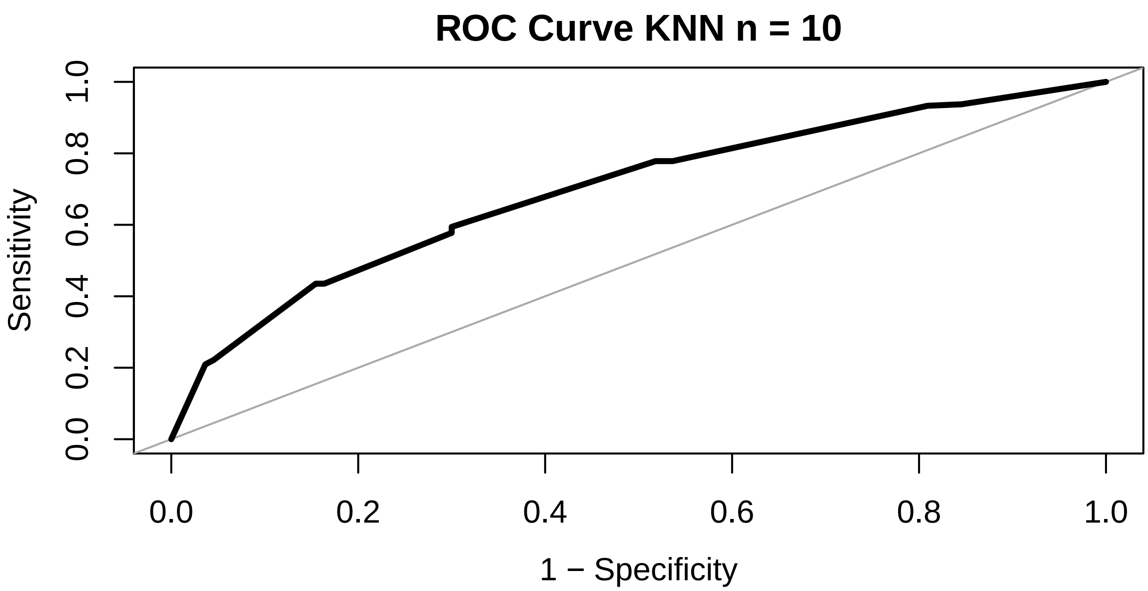


ROC Curve KNN n = 6
Sensitivity
1 − Specificity
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve KNN n = 7
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
1 − Specificity


ROC Curve KNN n = 8
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
1 − Specificity





c

```
colnames(knn_err) <- paste("k", c(1:10), "err")
colnames(knn_auc) <- paste("k", c(1:10), "auc")

model_errors <- cbind(log_err, lda_err, qda_err, nb_err, knn_err)
model_aucs <- cbind(log_auc, lda_auc, qda_auc, nb_auc, knn_auc)

rownames(model_errors) <- "Error Rate"
rownames(model_aucs) <- "AUC"

as.data.frame(apply(model_errors, 1, sort))
```

```
##            Error Rate
## k 9 err         0.218
```

```
## k 1 err       0.221
## k 10 err      0.221
## k 8 err       0.223
## k 7 err       0.229
## k 6 err       0.232
## k 2 err       0.238
## k 3 err       0.244
## k 5 err       0.244
## k 4 err       0.264
## lda_err       0.284
## nb_err        0.287
## log_err       0.289
## qda_err       0.292
```

```r
as.data.frame(apply(model_aucs, 1, sort))
```

```
##             AUC
## k 1 auc  0.500
## k 2 auc  0.573
## k 3 auc  0.607
## k 4 auc  0.641
## k 5 auc  0.650
## k 6 auc  0.674
## k 7 auc  0.678
## k 9 auc  0.688
## k 8 auc  0.689
## k 10 auc 0.696
## nb_auc   0.746
## qda_auc  0.749
## log_auc  0.780
## lda_auc  0.780
```

d

Overall, KNN model with K = 3 has the lowest error rate. In terms of the type of model used, KNN models are performing better than all the other ones. However, the KNN models do not seem to be increasing in performance as K increases. Interpreting with error rates, the Naive Bayes model has the highest error rate and is thus the worst-performing.

By definition, AUC represents the probability for a model to rank a random positive observation more highly than a random negative example. The lowest AUC value occurs for KNN model when K = 1, as this model would not have any class separation capacity. Generally, all KNN models perform worse than the other models, as their AUC values are lower. Unlike with error rates, the KNN models are almost ranked in order. The best-performing model in this regard would be the LDA. It has 76.1% chance of distinguishing between the two classes.