Problem Set 2

MACS 30100 Winter 2020

Nak Won Rim

The Bayes Classifier

1.

a. Set your random number generator seed.

```r
set.seed(24)
```

b. Simulate a dataset of $N = 200$ with X_1, X_2 where X_1, X_2 are random uniform variables between $[-1, 1]$.

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

c. Calculate $Y = X_1 + X_1^2 + X_2 + X_2^2 + \epsilon$, where $\epsilon \sim N(\mu = 0, \sigma^2 = 0.25)$.

```r
epsilon <- rnorm(200, mean=0, sd=0.5)
y = x_1 + x_1 ^ 2 + x_2 + x_2 ^ 2 + epsilon
```

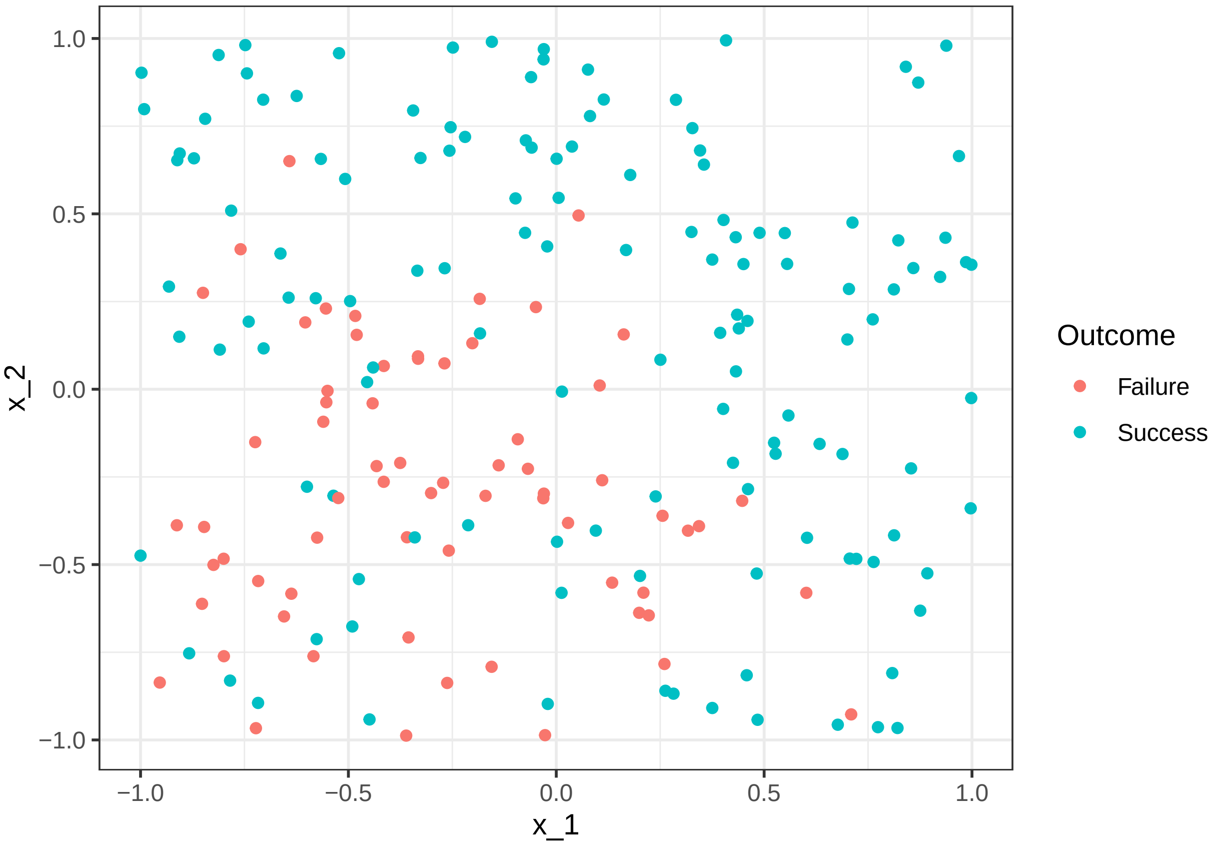
d. Y is defined in terms of the log-odds of success on the domain $[-\infty, +\infty]$. Calculate the probability of success bounded between $[0, 1]$

$$Y = log(\frac{Pr(Success)}{1 - Pr(Success)}) \Leftrightarrow Pr(Success) = \frac{\exp(Y)}{1 - \exp(Y)}$$

```r
# creating a dataframe for easiler manipulation
df1 <- data.frame(x_1, x_2, y)
df1 <- df1 %>%
  mutate(y = exp(y) / (1 + exp(y)))
```

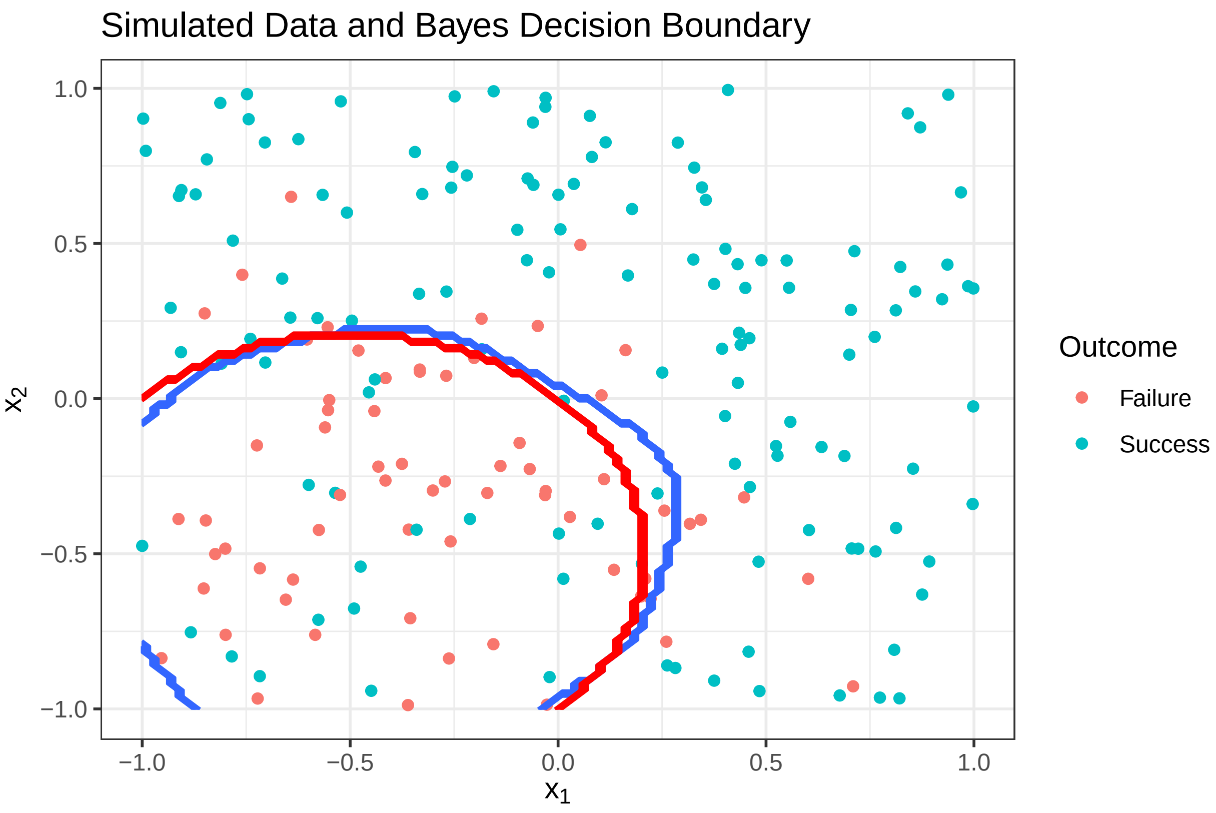
e. Plot each of the data points on a graph and use color to indicate if the observation was a success or a failure.

```r
df1 <- df1 %>%
  mutate(Outcome = as.factor(if_else(y > 0.5, "Success", "Failure")))
ggplot(data=df1, mapping=aes(x_1, x_2)) +
  geom_point(aes(colour=Outcome)) +
  theme_bw()
```



f. Overlay the plot with Bayes decision boundary, calculated using X_1, X_2.
g. Give your plot a meaningful title and axis labels.
h. The colored background grid is optional.

```
grid <- expand.grid(x_1=seq(-1, 1, length.out=100), x_2=seq(-1, 1, length.out=100))
nb.m1 <- train(
  x = df1[, c("x_1","x_2")],
  y = df1$Outcome,
  method = "nb"
  )
nb_grid <- data.frame(x_1 = grid$x_1,
                      x_2 = grid$x_2,
                      Outcome = predict(nb.m1, grid))
x_1 <- grid$x_1
x_2 <- grid$x_2
y <- x_1 + x_1 ^ 2 + x_2 + x_2 ^ 2
cp_grid <- data.frame(x_1, x_2, y)
cp_grid <- cp_grid %>%
  mutate(y = exp(y) / (1 + exp(y)))
cp_grid <- cp_grid %>%
  mutate(Outcome = as.factor(if_else(y > 0.5, "Success", "Failure")))
ggplot(data=df1, mapping=aes(x_1, x_2)) +
  geom_point(aes(colour=Outcome)) +
  geom_contour(data = nb_grid, aes(x_1, x_2, z=as.numeric(Outcome))) +
  geom_contour(data = cp_grid, aes(x_1, x_2, z=as.numeric(Outcome)), color="red") +
  labs(title="Simulated Data and Bayes Decision Boundary",
       caption = "The red line is the true boundary while the blue line is boundary predicted by Naive
  xlab(expression(x[1])) +
  ylab(expression(x[2])) +
  theme_bw()
```



The red line is the true boundary while the blue line is boundary predicted by Naive Bayes

Exploring Simulated Differences between LDA and QDA

2. If the Bayes decision boundary is linear, do we expect LDA or QDA to perform better on the training set? On the test set?

I expect that QDA will perform better on the training set since QDA is more flexible than LDA. This means that QDA will be able to fit more closely to the data points in the training set, showing less training set errors. However, in the test set, a reverse pattern is expected. QDA will be more vulnerable to overfitting if the Bayes decision boundary is linear since it is flexible enough to account for the variance caused by errors in the training set. Therefore, LDA, which is less flexible and less vulnerable to overfitting, is expected to perform better in the test set.

a. Repeat the following process 1000 times.

   i. Simulate a dataset of 1000 observations with $X_1, X_2 \sim \text{Uniform}(-1, +1)$. Y is a binary response variable defined by a Bayes decision boundary of $f(X) = X_1 + X_2$, where values 0 or greater are coded TRUE and values less than 0 or coded FALSE. Whereas your simulated Y is a function of $X_1 + X_2 + \epsilon$ where $\epsilon \sim N(0, 1)$. That is, your simulated Y is a function of the Bayes decision boundary plus some irreducible error.

ii. Randomly split your dataset into 70/30% training/test sets.
iii. Use the training dataset to estimate LDA and QDA models.
iv. Calculate each model's training and test error rate.

```r
set.seed(95)
errors_df1 <- data.frame(matrix(NA, nrow = 1000, ncol = 4))
colnames(errors_df1) <- c("lda_tr", "lda_te", "qda_tr", "qda_te")
for(i in 1:1000){
# part i
x_1 = runif(1000, min=-1, max=1)
x_2 = runif(1000, min=-1, max=1)
epsilon <- rnorm(1000, 0, 1)
simulated_y = x_1 + x_2 + epsilon
df2 <- data.frame(x_1, x_2, simulated_y)
df2 <- df2 %>%
  mutate(outcome = as.factor(if_else(df2$simulated_y >= 0, TRUE, FALSE)))
# part ii
split1 <- initial_split(df2, prop = .7)
train1 <- training(split1)
test1 <- testing(split1)
# part iii
lda_mod <- MASS::lda(outcome ~ x_1 + x_2, data = train1)
qda_mod <- MASS::qda(outcome ~ x_1 + x_2, data = train1)
# part iv
errors_df1[i, 1] <- (sum(predict(lda_mod, train1)$class != train1$outcome) / length(train1$outcome))
errors_df1[i, 2] <- (sum(predict(lda_mod, test1)$class != test1$outcome) / length(test1$outcome))
errors_df1[i, 3] <- (sum(predict(qda_mod, train1)$class != train1$outcome) / length(train1$outcome))
errors_df1[i, 4] <- (sum(predict(qda_mod, test1)$class != test1$outcome) / length(test1$outcome))
}
```

b. Summarize all the simulations' error rates and report the results in tabular and graphical form.

Alongside a table, I will draw a density plot and a boxplot to report the results.

Table:

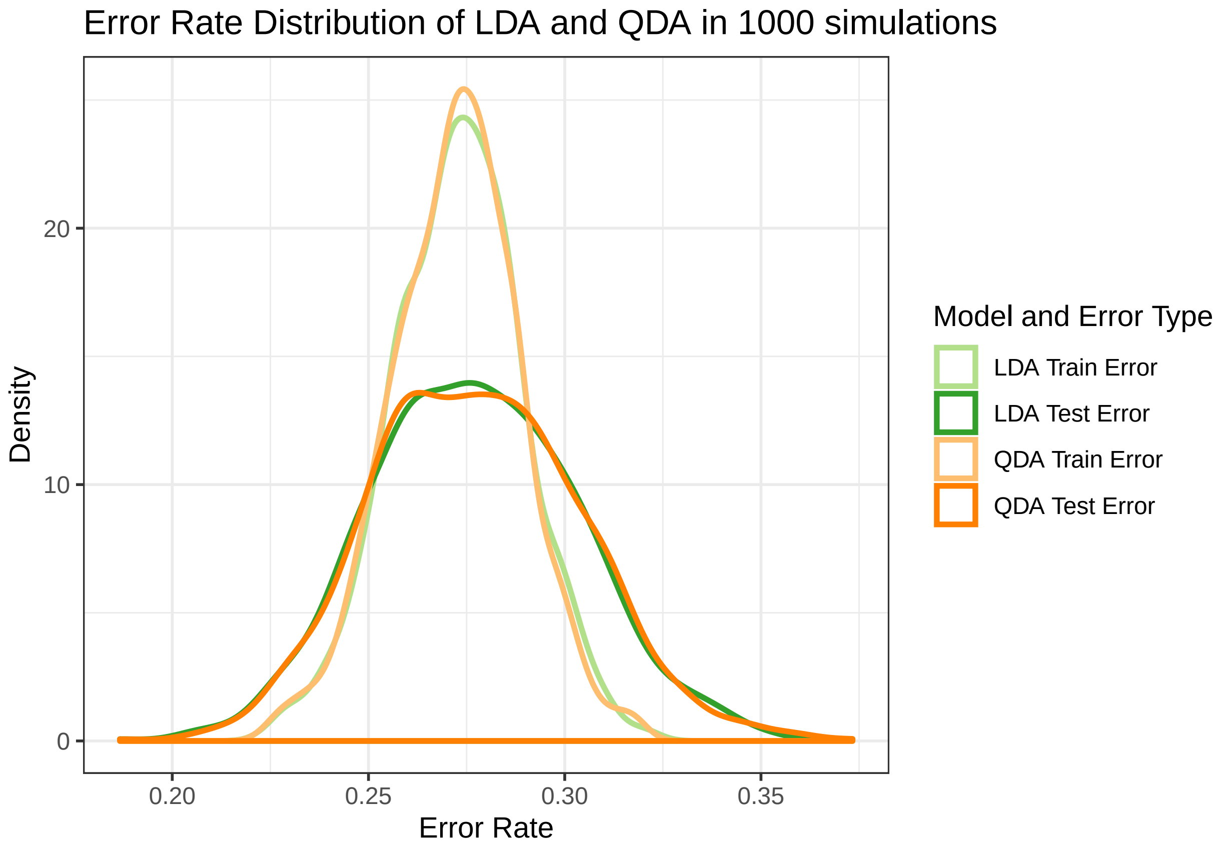
```r
# melt the df for easier visulalization
melted_df1 <- melt(errors_df1, id.vars=NULL)
summary_table <-  melted_df1 %>%
  group_by(variable) %>%
  summarise(mean = mean(value), sd = sd(value))
colnames(summary_table) <- c("Model and Error Type", "Mean", "Standarad Deviation")
summary_table$`Model and Error Type` <- c("LDA Train Error","LDA Test Error",
                                          "QDA Train Error","QDA Test Error")
kable(summary_table, align="ccc",  digits = 6,
      caption = "Error Rate of LDA and QDA in 1000 simulations")
```

Table 1: Error Rate of LDA and QDA in 1000 simulations

Model and Error Type	Mean	Standarad Deviation
LDA Train Error	0.272797	0.016504
LDA Test Error	0.277067	0.026554
QDA Train Error	0.272306	0.016321
QDA Test Error	0.277457	0.026632

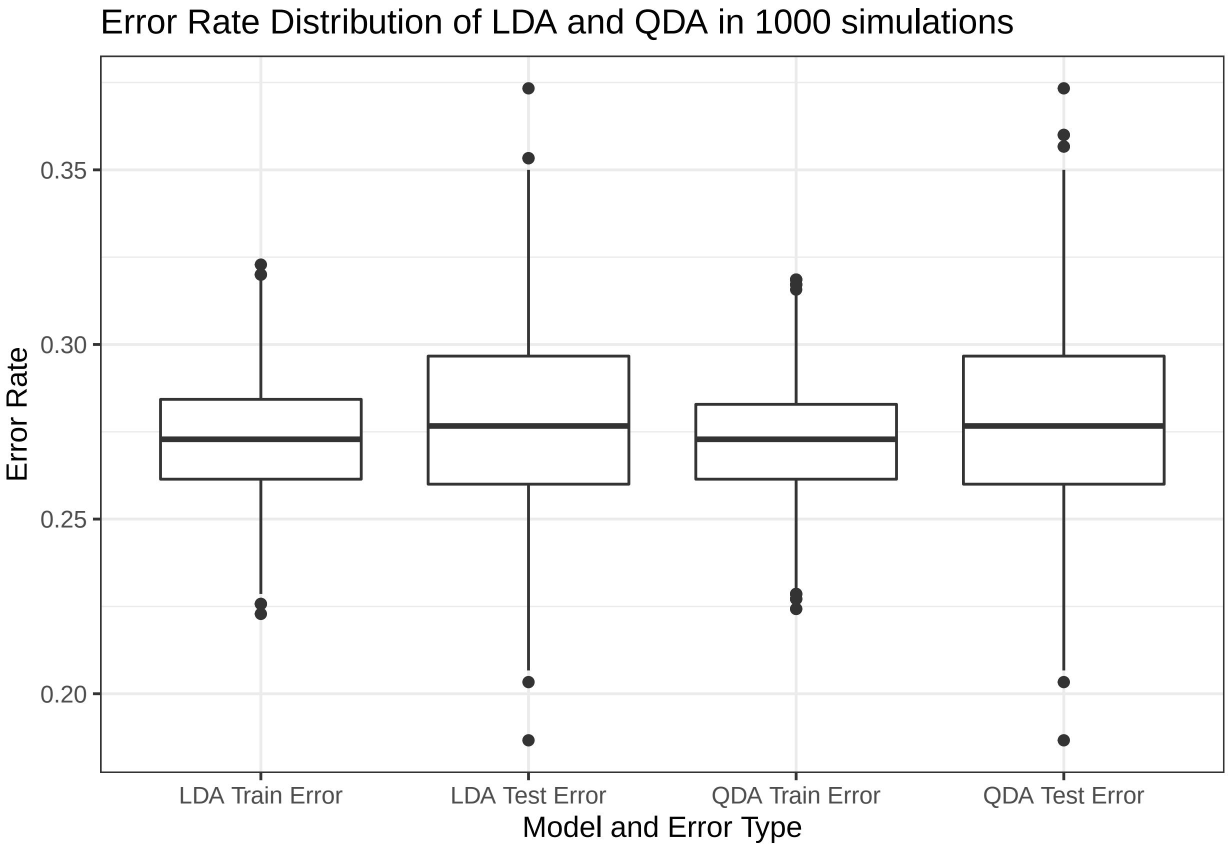
Density plot:

```
ggplot(data=melted_df1, aes(value, color = variable)) +
  geom_density(size = 1)+
  scale_color_manual(values=c("#b2df8a", "#33a02c", "#fdbf6f", "#ff7f00"),
                     labels = c("LDA Train Error","LDA Test Error",
                              "QDA Train Error","QDA Test Error"))+
  labs(title="Error Rate Distribution of LDA and QDA in 1000 simulations",
       color = "Model and Error Type") +
  xlab("Error Rate") +
  ylab("Density") +
  theme_bw()
```



Boxplot:

```
ggplot(data=melted_df1, aes(variable, value))+
  geom_boxplot()+
  scale_x_discrete(labels = c("LDA Train Error","LDA Test Error",
                              "QDA Train Error","QDA Test Error"))+
  labs(title="Error Rate Distribution of LDA and QDA in 1000 simulations") +
  xlab("Model and Error Type") +
  ylab("Error Rate") +
  theme_bw()
```



We can see from the table and plots that the difference in error rate distribution is very small between LDA and QDA. The density lines shown in the density plot almost overlap with each other, and the difference between the mean values is very small. We do see that QDA performs a little better (mean error rate difference = 0.000491) in the training set and LDA performs a little better in the test set (mean error rate difference = 0.00466), which is consistent with the predictions I made before the simulation. However, it seems very risky to draw any conclusion from the simulation results.

3. If the Bayes decision boundary is non-linear, do we expect LDA or QDA to perform better on the training set? On the test set?

I expect that QDA will perform better at both the training set and the test set in this case. Following the logic from question 2, QDA will perform better in the training set because it's flexibility allows a closer fit to the data points. QDA will also perform better in the test set since LDA would not be able to account for the non-linearity of the Bayes decision boundary and will show a relatively poor fit.

a. Repeat the following process 1000 times.

   i. Simulate a dataset of 1000 observations with $X_1, X_2 \sim \text{Uniform}(-1, +1)$. Y is a binary response variable defined by a Bayes decision boundary of $f(X) = X_1 + X_1^2 + X_2 + X_2^2$, where values 0 or greater are coded TRUE and values less than 0 or coded FALSE. Whereas your simulated Y is a function of $X_1 + X_1^2 + X_2 + X_2^2 + \epsilon$ where $\epsilon \sim N(0, 1)$. That is, your simulated Y is a function of the Bayes decision boundary plus some irreducible error.
   ii. Randomly split your dataset into 70/30% training/test sets.
   iii. Use the training dataset to estimate LDA and QDA models.
   iv. Calculate each model's training and test error rate.

Since the process of 3.a has to be repeated in problem 4 too, I will define a function `sim_100` to do the simulation:

```r
sim_1000 <- function(n){
  errors_df <- data.frame(matrix(NA, nrow = 1000, ncol = 4))
  colnames(errors_df) <- c("lda_tr", "lda_te", "qda_tr", "qda_te")
  for(i in 1:1000){
  # part i
  x_1 = runif(n, min=-1, max=1)
  x_2 = runif(n, min=-1, max=1)
  epsilon <- rnorm(n, 0, 1)
  simulated_y = x_1 + x_1 ^ 2 + x_2 + x_2 ^ 2 + epsilon
  df <- data.frame(x_1, x_2, simulated_y)
  df <- df %>%
    mutate(outcome = as.factor(if_else(df$simulated_y >= 0, TRUE, FALSE)))
  ## part ii
  split <- initial_split(df, prop = .7)
  train <- training(split)
  test <- testing(split)
  ## part iii
  lda_mod <- MASS::lda(outcome ~ x_1 + x_2, data = train)
  qda_mod <- MASS::qda(outcome ~ x_1 + x_2, data = train)
  errors_df[i, 1] <- (sum(predict(lda_mod, train)$class != train$outcome) / length(train$outcome))
  errors_df[i, 2] <- (sum(predict(lda_mod, test)$class != test$outcome) / length(test$outcome))
  errors_df[i, 3] <- (sum(predict(qda_mod, train)$class != train$outcome) / length(train$outcome))
  errors_df[i, 4] <- (sum(predict(qda_mod, test)$class != test$outcome) / length(test$outcome))
  }
  return(errors_df)
}
set.seed(133)
errors_df2 <- sim_1000(1000)
```

b. Summarize all the simulations' error rates and report the results in tabular and graphical form. Use this evidence to support your answer.

Analogous to problem 2, I will draw a density plot and a boxplot to report the results alongside a table.

Table:

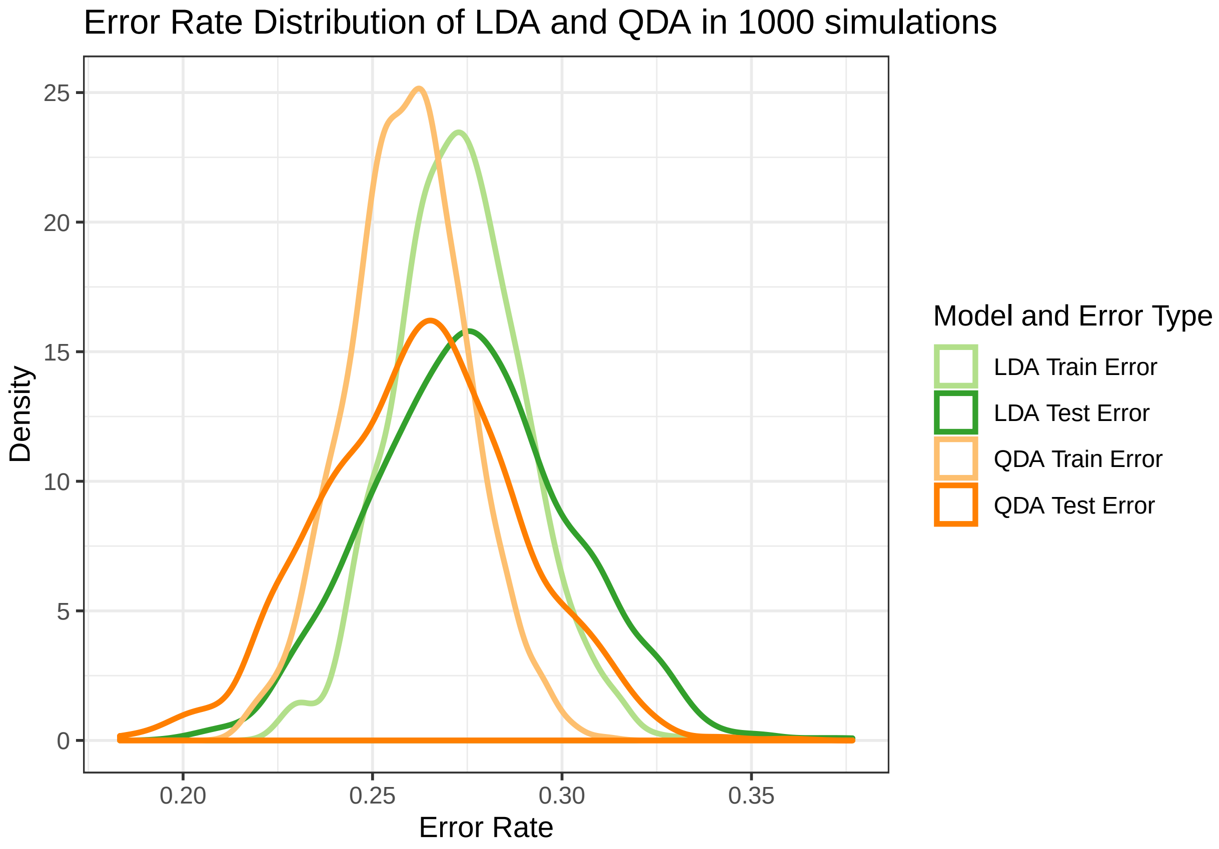
```r
# melt the df for easier visulalization
melted_df2 <- melt(errors_df2, id.vars=NULL)
summary_table2 <-  melted_df2 %>%
  group_by(variable) %>%
  summarise(mean = mean(value), sd = sd(value))
colnames(summary_table2) <- c("Model and Error Type", "Mean", "Standarad Deviation")
summary_table2$`Model and Error Type` <- c("LDA Train Error","LDA Test Error",
                                          "QDA Train Error","QDA Test Error")
kable(summary_table2, align="ccc",  digits = 6,
      caption = "Error Rate of LDA and QDA in 1000 simulations")
```

Table 2: Error Rate of LDA and QDA in 1000 simulations

Model and Error Type	Mean	Standarad Deviation
LDA Train Error	0.272969	0.017039
LDA Test Error	0.275940	0.025872
QDA Train Error	0.259251	0.015741
QDA Test Error	0.262837	0.025741

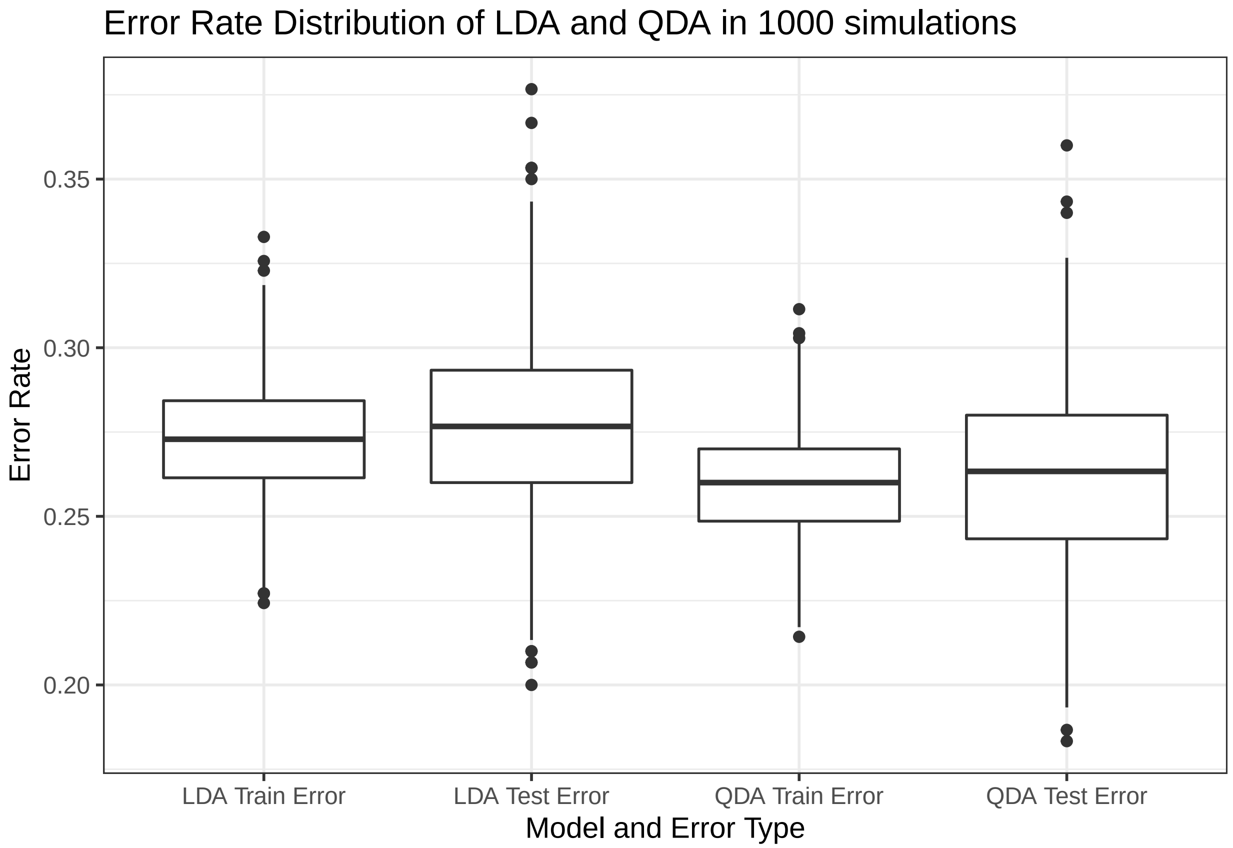
Density plot:

```r
ggplot(data=melted_df2, aes(value, color = variable)) +
  geom_density(size = 1)+
  scale_color_manual(values=c("#b2df8a", "#33a02c", "#fdbf6f", "#ff7f00"),
                     labels = c("LDA Train Error","LDA Test Error",
                                "QDA Train Error","QDA Test Error"))+
  labs(title="Error Rate Distribution of LDA and QDA in 1000 simulations",
       color = "Model and Error Type") +
  xlab("Error Rate") +
  ylab("Density") +
  theme_bw()
```



Boxplot:

```
ggplot(data=melted_df2, aes(variable, value)) +
  geom_boxplot()+
  scale_x_discrete(labels = c("LDA Train Error","LDA Test Error",
                              "QDA Train Error","QDA Test Error"))+
  labs(title="Error Rate Distribution of LDA and QDA in 1000 simulations") +
  xlab("Model and Error Type") +
  ylab("Error Rate") +
  theme_bw()
```



We can see from the simulations that there is a difference between LDA and QDA in the error rates distribution. QDA shows lower error rates in both the training sets (mean error rate difference = 0.013718) and the test sets (mean error rate difference = 0.013103) as expected. Again, I suspect that this difference comes from LDA's inability to capture the non-linear Bayes decision boundary both in the training set and the test set.

4. In general, as sample size n increases, do we expect the test error rate of QDA relative to LDA to improve, decline, or be unchanged? Why?

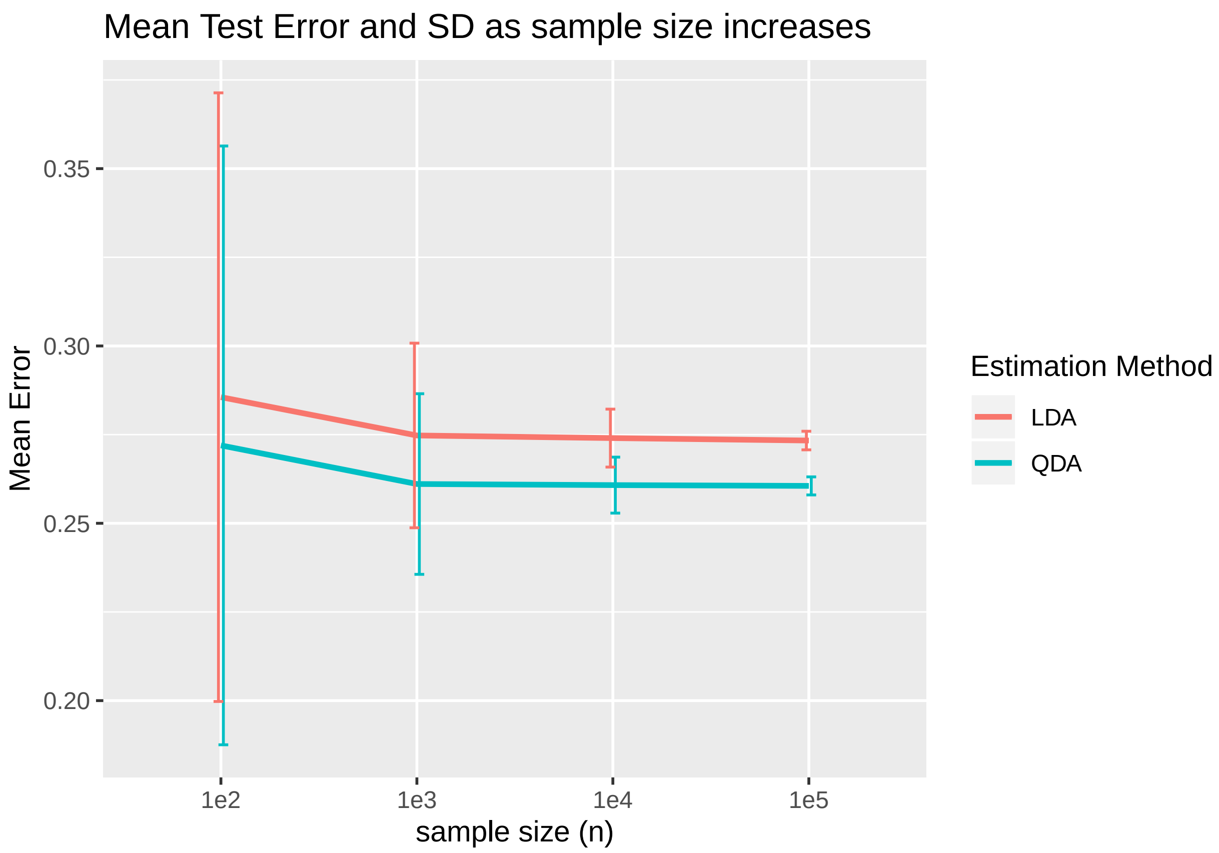
As the sample size increases, I expect that the test error rate of QDA relative to LDA will improve. This is because as the sample size increases, more data points will be included in the training set, and this will be able to decrease the probability of overfitting in QDA. Therefore, QDA will show less error rate in the test set as the sample size increases. In addition, an increase in the sample size will decrease the variance in general. Since variance is a major concern in flexible models, this could be another reason why QDA, which is more flexible than LDA, will show better performance as the sample size increase.

‘ a. Use the non-linear Bayes decision boundary approach from part (2) and vary n across your simulations (e.g., simulate 1000 times for n = c(1e02, 1e03, 1e04, 1e05).

```
set.seed(151)
errors_df3 <- sim_1000(1e2)
errors_df4 <- sim_1000(1e3)
errors_df5 <- sim_1000(1e4)
errors_df6 <- sim_1000(1e5)
n_size_err <- rbind(data.frame(melt(errors_df3$lda_te), type="LDA", n="1e2"),
                    data.frame(melt(errors_df4$lda_te), type="LDA", n="1e3"),
                    data.frame(melt(errors_df5$lda_te), type="LDA", n="1e4"),
                    data.frame(melt(errors_df6$lda_te), type="LDA", n="1e5"),
                    data.frame(melt(errors_df3$qda_te), type="QDA", n="1e2"),
                    data.frame(melt(errors_df4$qda_te), type="QDA", n="1e3"),
                    data.frame(melt(errors_df5$qda_te), type="QDA", n="1e4"),
                    data.frame(melt(errors_df6$qda_te), type="QDA", n="1e5"))
```

‘ b. Plot the test error rate for the LDA and QDA models as it changes over all of these values of n. *Use this graph to support your answer.*

```
summarise_df <- n_size_err %>%
        group_by(type, n) %>%
        dplyr::summarise(mean = mean(value), sd = sd(value))
ggplot(summarise_df, aes(x=n, y=mean, color=type)) +
  geom_line(aes(group=type), size=1) +
  geom_errorbar(aes(ymin=mean-sd, ymax=mean+sd), width=.1,
    position=position_dodge(0.05))+
  labs(title="Mean Test Error and SD as sample size increases", color="Estimation Method") +
  xlab("sample size (n)") +
  ylab("Mean Error")
```



We see a very interesting pattern in the graph. One thing to note is that QDA performs consistently better in all sample sizes. This probably reflects the non-linearity of the Bayes decision boundary. Another thing to notice is that the variation of error rates, seen in the graph through standard deviation, consistently decreases as the sample size increases. This is consistent with the increase in sample size decreasing variation, as discussed before the simulation. However, the mean test error rate does not seem to decrease consistently as the sample size increases. There is a decrease in mean error rate in both LDA and QDA when the sample size increases from 1e2 to 1e3, but the mean test error rate seems to remain quite constant after that. This could be seen that the fit of the models reached some sort of ceiling as the sample size becomes larger than 1e3.

Importantly, the difference in the mean test error rate seems to remain constant as the sample size increase. This is different from the original expectation before the simulation, where I expected that the difference will increase as the sample size increases. However, another way to look at it will be to think that as the variance of the test error rates decreases, QDA will outperform LDA more consistently. We can see a large overlap of the error bars (SD) in the sample size of 1e2, but this overlap decreases as the sample size increases, ultimately leading to no overlap in the sample size of 1e5. If we pick one random instance from the 1000 simulations, the probability of QDA outperforming LDA in that particular instance will be higher. This could be seen as QDA performing better than LDA as the sample size increases.

To sum up, the test error rate of QDA relative to LDA seems to be unchanged as sample size increases, contrary to the expectation before the simulation. However, we could argue that QDA will perform better than LDA as the sample size increases if we take account of the decreased variance.

5.

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

```
set.seed(381)
mental_health_df <- na.omit(read.csv("mental_health.csv"))
split2 <- initial_split(mental_health_df, prop=.7)
train2 <- training(split2)
test2 <- testing(split2)
```

b. Using the training set and all important predictors, estimate the following models with `vote96` as the response variable:

   i. Logistic regression model

```
turnout_logit <- glm(vote96 ~ ., data = train2, family = binomial)
```

‘ ii. Linear discriminant model

```
turnout_lda <- MASS::lda(vote96 ~ ., data = train2)
```

‘ iii. Quadratic discriminant model

```
turnout_qda <- MASS::qda(vote96 ~ ., data = train2)
```

‘ iv. Naive Bayes

```
turnout_nb <- train(
  x = select(train2, -vote96),
  y = as.factor(train2$vote96),  method = "nb")
```

‘ v. K-nearest neighbors with $K = 1, 2, \ldots, 10$ and Euclidean distance metrics

```
# note that the error rates for knn models are already calculated in the turnout_knn$err_test
turnout_knn <- tibble(k = 1:10,
                      knn_test = map(k, ~ class::knn(train = select(train2, -vote96),
                                                      test = select(test2, -vote96),
                                                      cl = train2$vote96, k = .)),
                      knn_prob = map(k, ~ attr(class::knn(train = select(train2, -vote96),
                                                      test = select(test2, -vote96),
                                                      cl = train2$vote96, k = ., prob = TRUE), "prob"))
                      err_test = map_dbl(knn_test, ~ mean(test2$vote96 != .)))
```

c. Using the test set, calculate the following model performance metrics:

   i. Error rate

```
# First we predict the values in the test set using the models except knn, which is already calculated
logit_predict <- augment(turnout_logit, newdata = test2) %>%
  as_tibble() %>%
  mutate(.prob = rcfss::logit2prob(.fitted),
         .pred = factor(round(.prob)))
lda_predict <- predict(turnout_lda, test2)
qda_predict <- predict(turnout_qda, test2)
nb_predict <- predict(turnout_nb, test2)
```

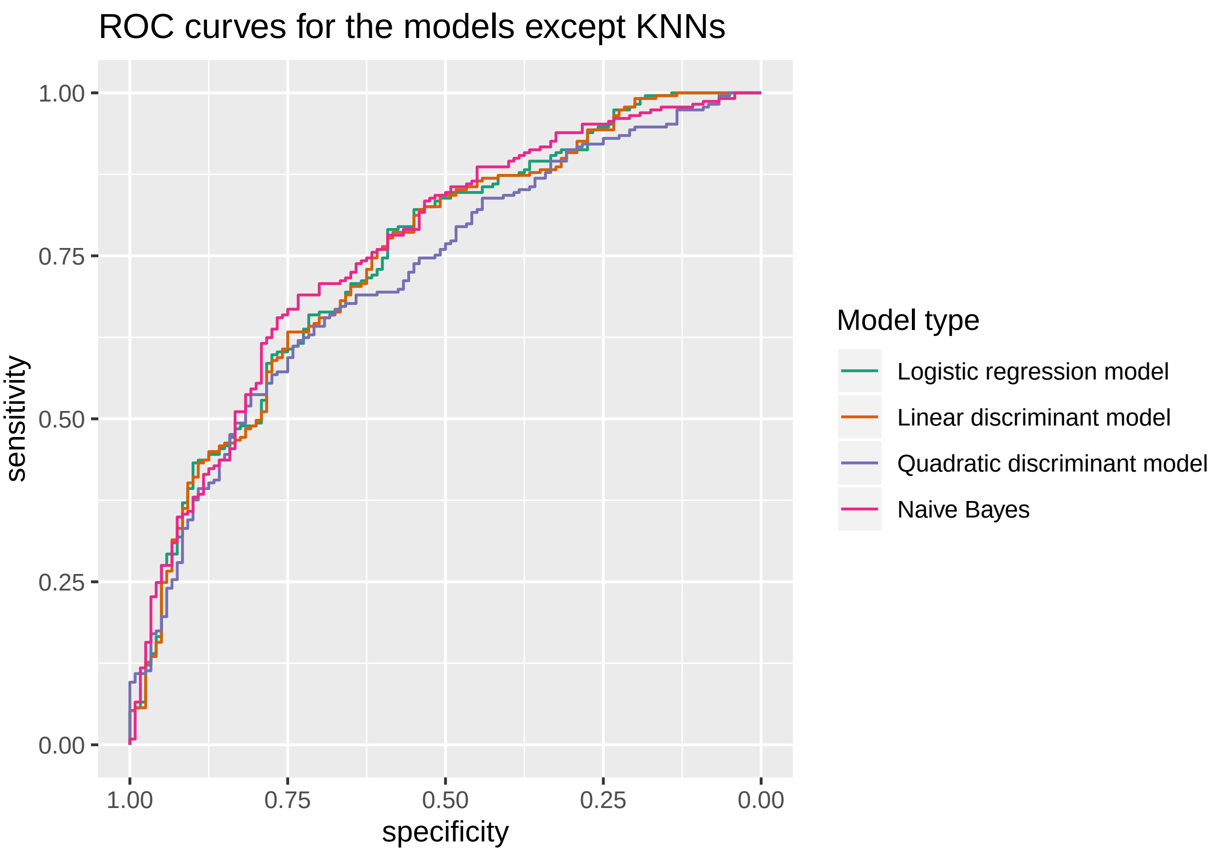
```r
# then we calculate the error rate and store it in the error_table.
error_table <- rbind(data.frame(te = (sum(as.factor(logit_predict$vote96) != logit_predict$.pred) /
                                                     length(logit_predict$.pred))),
                     data.frame(te = (sum(lda_predict$class != test2$vote96) / length(test2$vote96))),
                     data.frame(te = (sum(qda_predict$class != test2$vote96) / length(test2$vote96))),
                     data.frame(te = (sum(nb_predict != test2$vote96) / length(test2$vote96))),
                     data.frame(te = turnout_knn$err_test)
                     )
error_table$type = c("Logistic regression model", "Linear discriminant model",
                     "Quadratic discriminant model", "Naive Bayes",
                     "K-nearest neighbors with K = 1", "K-nearest neighbors with K = 2",
                     "K-nearest neighbors with K = 3", "K-nearest neighbors with K = 4",
                     "K-nearest neighbors with K = 5", "K-nearest neighbors with K = 6",
                     "K-nearest neighbors with K = 7", "K-nearest neighbors with K = 8",
                     "K-nearest neighbors with K = 9", "K-nearest neighbors with K = 10")
error_table <- error_table[, c(2,1)]
kable(error_table, col.names=c("Model Type", "Test Error Rate"), align = "c")
```

Model Type	Test Error Rate
Logistic regression model	0.2922636
Linear discriminant model	0.2951289
Quadratic discriminant model	0.3151862
Naive Bayes	0.2808023
K-nearest neighbors with K = 1	0.3839542
K-nearest neighbors with K = 2	0.3524355
K-nearest neighbors with K = 3	0.3495702
K-nearest neighbors with K = 4	0.3409742
K-nearest neighbors with K = 5	0.3323782
K-nearest neighbors with K = 6	0.3381089
K-nearest neighbors with K = 7	0.3237822
K-nearest neighbors with K = 8	0.3151862
K-nearest neighbors with K = 9	0.3151862
K-nearest neighbors with K = 10	0.3123209

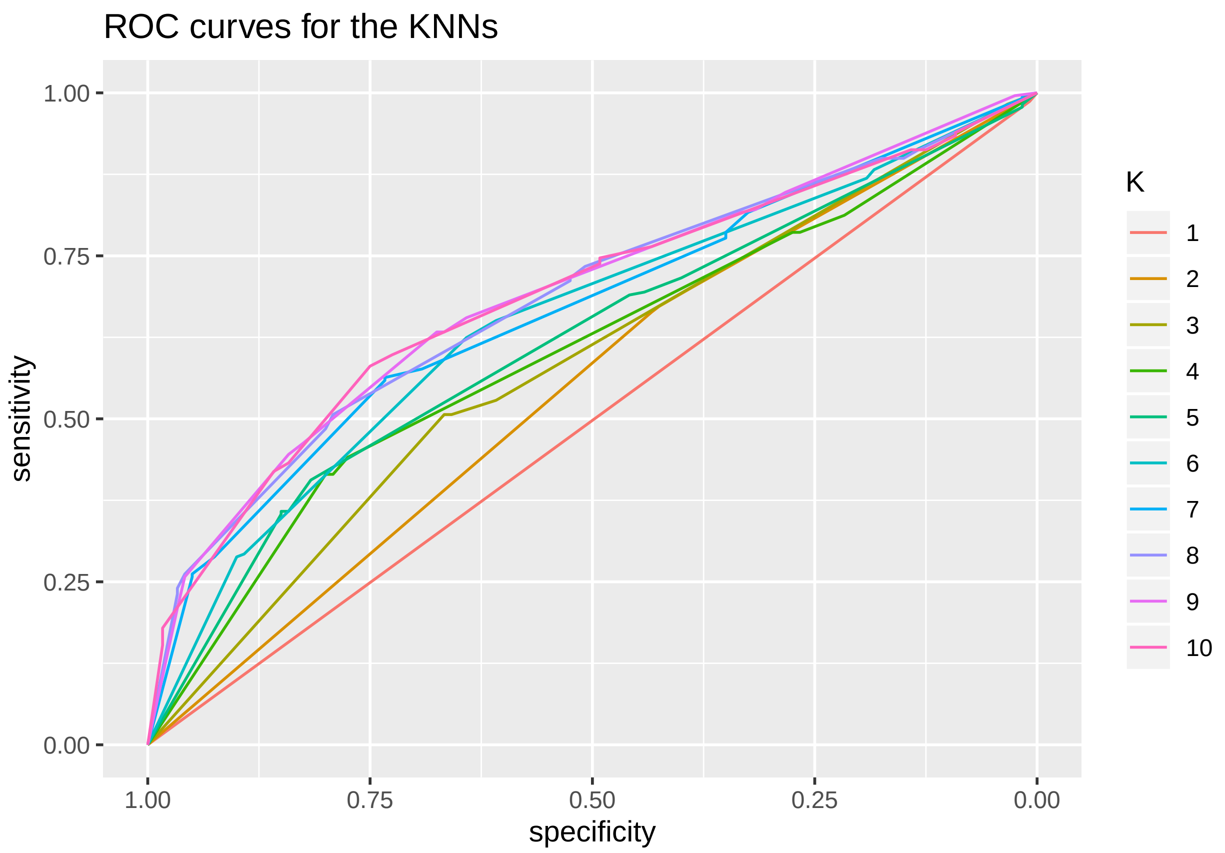
```
ii-1. ROC curve(s)
```

I will plot the ROC curves for the KNN models separately from the other four models to increase clarity.

```r
roc_logit <- pROC::roc(test2$vote96, logit_predict$.prob)
roc_lda <- pROC::roc(test2$vote96, lda_predict$posterior[, 1])
roc_qda <- pROC::roc(test2$vote96, qda_predict$posterior[, 1])
roc_nb <- pROC::roc(test2$vote96, predict(turnout_nb, test2, type="prob")$"1")
pROC::ggroc(list(roc_logit, roc_lda, roc_qda, roc_nb)) +
  labs(title = "ROC curves for the models except KNNs", color = "Model type")+
  scale_color_brewer(palette="Dark2", labels= c("Logistic regression model", "Linear discriminant model
                                               "Quadratic discriminant model", "Naive Bayes"))
```



```
roc_knn <- vector("list", length = 10)
for(i in 1:10){
  roc_knn[[i]] <- pROC::roc(test2$vote96, turnout_knn$knn_prob[[i]])
}
pROC::ggroc(roc_knn) +
  labs(title = "ROC curves for the KNNs", color = "K")
```



‘ ii-2. Area under the curve (AUC)

```
auc_table <- data.frame(matrix(NA, nrow = 14, ncol = 2))
colnames(auc_table) <- c("type", "auc")
auc_table$type = c("Logistic regression model", "Linear discriminant model",
                   "Quadratic discriminant model", "Naive Bayes",
                   "K-nearest neighbors with K = 1", "K-nearest neighbors with K = 2",
                   "K-nearest neighbors with K = 3", "K-nearest neighbors with K = 4",
                   "K-nearest neighbors with K = 5", "K-nearest neighbors with K = 6",
                   "K-nearest neighbors with K = 7", "K-nearest neighbors with K = 8",
                   "K-nearest neighbors with K = 9", "K-nearest neighbors with K = 10")
auc_table$auc[1] <- pROC::auc(test2$vote96, logit_predict$.prob)
auc_table$auc[2] <- pROC::auc(test2$vote96, lda_predict$posterior[, 1])
auc_table$auc[3] <- pROC::auc(test2$vote96, qda_predict$posterior[, 1])
auc_table$auc[4] <- pROC::auc(test2$vote96, predict(turnout_nb, test2, type="prob")$"1")
for(i in 1:10){
  auc_table$auc[i + 4] <- pROC::auc(test2$vote96, turnout_knn$knn_prob[[i]])
}
kable(auc_table, col.names=c("Model Type", "AUC"), align = "c")
```

Model Type	AUC
Logistic regression model	0.7462882
Linear discriminant model	0.7449782
Quadratic discriminant model	0.7200146
Naive Bayes	0.7579694
K-nearest neighbors with K = 1	0.4976164

Model Type	AUC
K-nearest neighbors with K = 2	0.5491448
K-nearest neighbors with K = 3	0.5819505
K-nearest neighbors with K = 4	0.6016739
K-nearest neighbors with K = 5	0.6176310
K-nearest neighbors with K = 6	0.6490539
K-nearest neighbors with K = 7	0.6648836
K-nearest neighbors with K = 8	0.6810226
K-nearest neighbors with K = 9	0.6889010
K-nearest neighbors with K = 10	0.6862991

d. Which model performs the best? Be sure to define what you mean by "best" and identify supporting evidence to support your conclusion(s).

We have discussed error rate, proportional reduction in error and ROC curve / AUC as a evaluation metric for model accuaraccy during class. Of these metrics, we had seen test error rate and ROC / AUC in this problem. Of the two metrics, I think AUC might be better to evaluate the "best" performing models in this case, since about 68% of the data has 1 (or True) in the response variable `vote96`. This means that we will have about 32% error rate if we predict everybody to vote, making the error rate a little bit harder to interpret. Since we have a slightly screwed distribution of the response variable, I think it will be better to evaluate the AUC, since this takes account of both false positive rate and true positive rate. In other words, I think the model with the best AUC could be seen as the "best" performing model. In our simulation, we can see that ROC curve of Naive Bayes model is closest to the top left corner of the plot, and has the highest AUC. Therefore, I think Naive Bayes performs "best". I note that Naive Bayes also has lowest test error rate in the simulation, so even if we choose error rate as the evaluation metric, we would think the Naive Bayes will be the best model. However, even in the hypothetical case that some model showed lower test error rate than Naive Bayes model, I would have claimed that Naive Bayes performs the best if it had the highest AUC because of the reasons discussed above.

This article was written using R Markdown and was knitted to pdf using the tinytex package