Homework 2: Classification Methods

Wanitchaya Poonpatanapricha

1) The Bayes Classifier

a)

```
# set random number generator seed
set.seed(171)
```

b)

```
# a dataset with random uniform variables between [-1, 1]
n <- 200
x1 <- runif(n, min = -1, max = 1)
x2 <- runif(n, min = -1, max = 1)
```

c)

```
# create error term
err <- rnorm(n, mean = 0, sd = 0.25^0.5)

# create table of x1, x2, and error
x_df <- tibble(x1,x2,err)

# calculate Y
x_df <- x_df %>%
  mutate(y = x1 + x1^2 + x2 + x2^2 + err)
```

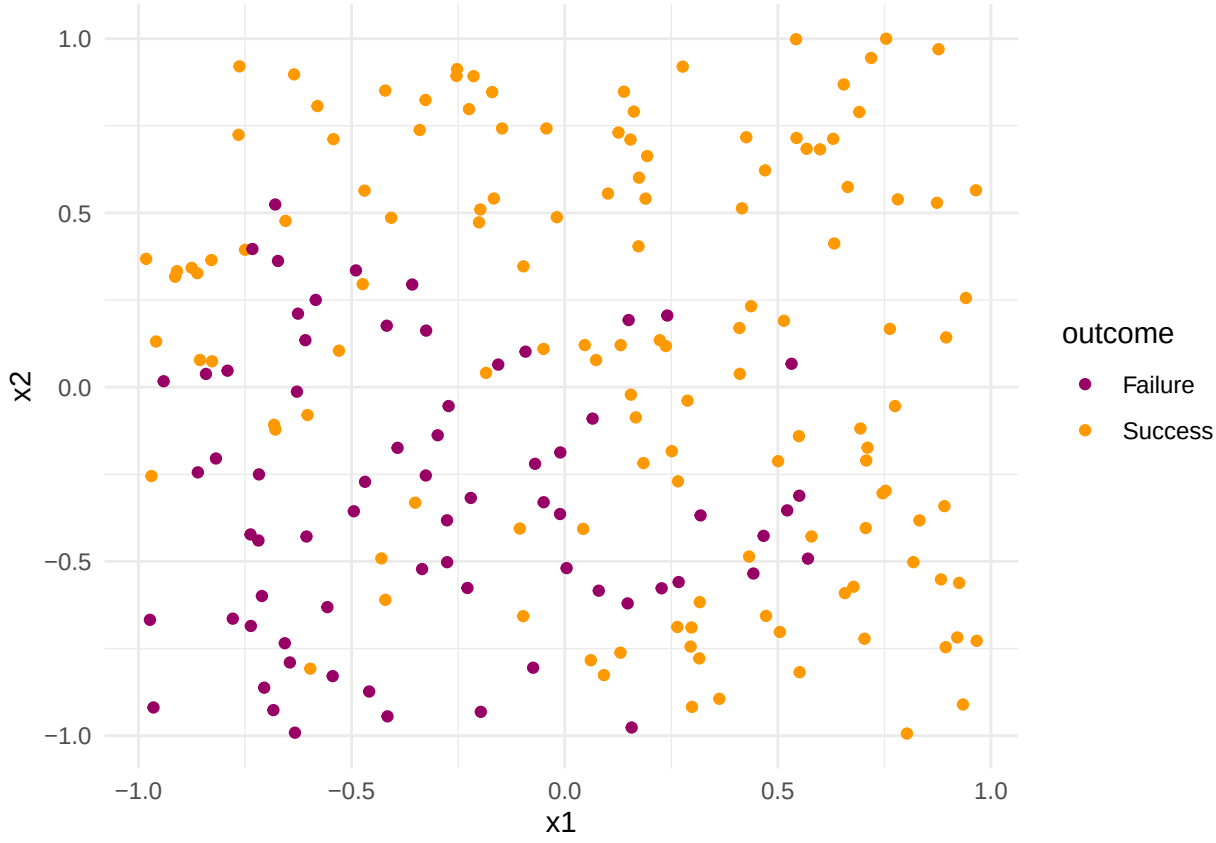
d)

```
# turn y from log odd into probability
e <- exp(1)
x_df <- x_df %>%
  mutate(prob = e^y/(1+e^y))
```

e)

```
# cast probability into binary outcome
x_df <- x_df %>%
  mutate(outcome = ifelse(prob > 0.5, "Success","Failure"))

# plot scatter plot of observations with each outcome
ggplot(x_df, aes(x1, x2, color = outcome)) +
```

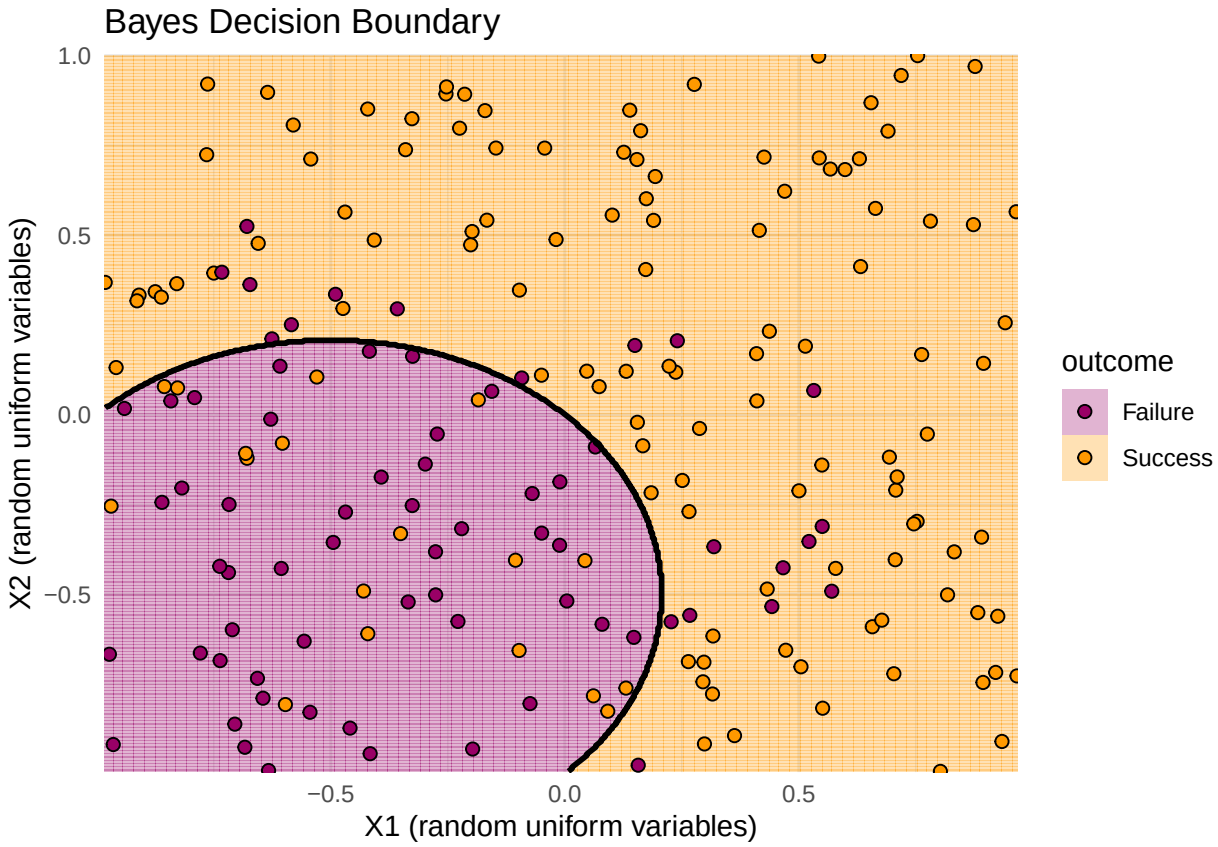
```
  geom_point() +
  scale_color_manual(values = c("#990066","#FF9900"))
```



f), g), h)

```
# prepare data for training bayes decision boundary
#    I do not train/test split the data since we are not asked to
#    calculate the test error. Hence, I think it is a waste of data
#    if we only going to plot training data.
df <- x_df %>%
  select(x1, x2, outcome) %>%
  mutate(outcome = as.factor(outcome))
df_x <- df[, 1:2] # input (x1, x2)
df_y <- df$outcome # output (binary outcome)

# create the bayes boundary line
nbp <- 500
x1_pred <- seq(min(df_x$x1), max(df_x$x1), length = nbp)
x2_pred <- seq(min(df_x$x2), max(df_x$x2), length = nbp)
Grid <- expand.grid(x1 = x1_pred, x2 = x2_pred) %>%
  mutate(y = x1 + x1^2 + x2 + x2^2,
         prob = e^y/(1+e^y),
         outcome = as.factor(ifelse(prob > 0.5, "Success", "Failure")))

# plot the nb boundary line on top of the scatter plot of actual outcomes
```

```r
ggplot(data = df, aes(x1, x2)) +
  geom_tile(data = Grid, aes(fill = outcome), alpha = 0.3) + # background color
  geom_point(aes(fill = outcome), colour="black",pch=21, size = 2) + # actual outcomes
  geom_contour(data = Grid, aes(z = as.numeric(outcome)),
                  color = "black", breaks = c(1.5), size = 1) + # boundary line
  scale_x_continuous(expand = c(0,0)) +
  scale_y_continuous(expand = c(0,0)) +
  labs(title = "Bayes Decision Boundary",
       y = "X2 (random uniform variables)",
       x = "X1 (random uniform variables)") +
  scale_color_manual(values=c("#990066","#FF9900")) +
  scale_fill_manual(values=c("#990066","#FF9900"))
```



2) Exploring Simulated Differences between LDA and QDA (Linear)

On training set, QDA will perform better in general because it is more flexible than LDA and hence can fit better than LDA. However, on testing set, when the Bayes decision boundary is linear, LDA will perform better because QDA may overfit the training set.

a)

```r
# function that generate simulated data of n observations
# given its nature type (linear or non-linear)
generate_data <- function(is_linear, n){
```

```r
  x1 <- runif(n, min = -1, max = 1) # random, uniform x1 [-1, 1]
  x2 <- runif(n, min = -1, max = 1) # random, uniform x2 [-1, 1]
  err <- rnorm(n, mean = 0, sd = 1) # random, normal error [0, 1]
  # calculate y as binary outcome base on data structure type
  if (is_linear){
    df_2 <- tibble(x1,x2,err) %>%
      mutate(y = ifelse(x1 + x2 + err >= 0, TRUE, FALSE)) # when data is linear
  }
  else{
    df_2 <- tibble(x1,x2,err) %>%
      mutate(y = ifelse(x1 + x1^2 + x2 + x2^2 + err >= 0, TRUE, FALSE)) # non-linear
  }
  return(df_2) # table with x1, x2, error, and y
}

# function that simulate the lda/qda once and calculate all error rates
sim_once <- function(formula_str, is_linear, n){

  # i) generate simulated data
  df_2 <- generate_data(is_linear, n)

  # ii) split train/test 70/30
  split <- initial_split(df_2, prop = .7)
  train <- training(split)
  test <- testing(split)

  # iii) training lda and qda based on given formular
  lda_lin <- MASS::lda(as.formula(formula_str), data = train)
  qda_lin <- MASS::qda(as.formula(formula_str), data = train)

  # iv)
  # calculate lda error rate in training
  train$lda_pred <- predict(lda_lin, train)$class # get predicted outcome
  train <- train %>%
    mutate(y = as.factor(y)) %>%
    mutate(lda_loss = ifelse(y == lda_pred, 0, 1)) # if wrong prediction, loss is 1
  train_lda_loss <- sum(train$lda_loss)/nrow(train) # calculate error rate
  # calculate qda error rate in training
  train$qda_pred <- predict(qda_lin, train)$class
  train <- train %>%
    mutate(y = as.factor(y)) %>%
    mutate(qda_loss = ifelse(y == qda_pred, 0, 1))
  train_qda_loss <- sum(train$qda_loss)/nrow(train)
  # calculate lda error rate in testing
  test$lda_pred <- predict(lda_lin, test)$class
  test <- test %>%
    mutate(y = as.factor(y)) %>%
    mutate(lda_loss = ifelse(y == lda_pred, 0, 1))
  test_lda_loss <- sum(test$lda_loss)/nrow(test)
  # calculate qda error rate in testing
  test$qda_pred <- predict(qda_lin, test)$class
  test <- test %>%
    mutate(y = as.factor(y)) %>%
```

```
    mutate(qda_loss = ifelse(y == qda_pred, 0, 1))
  test_qda_loss <- sum(test$qda_loss)/nrow(test)

  # return  4 types of error rates
  return (c(train_lda_loss, test_lda_loss, train_qda_loss, test_qda_loss))
}

# function that simulates 1000 lda/qda and stores all error rates
sim_1000 <- function(formula_str, is_linear, n){
  # initialize empty table to hold the error rate
  loss_df <- data.frame(matrix(NA, nrow=4000, ncol=3))
  names(loss_df) <- c("Method", "Session", "Error")

  # run the simulation 1000 times
  for (i in seq(1000)) {
    # get 4 error rates from one simulation
    loss <- sim_once(formula_str, is_linear, n)
    # training lda error rate
    loss_df[4*(i-1)+1,1] <- "LDA"
    loss_df[4*(i-1)+1,2] <- "Train"
    loss_df[4*(i-1)+1,3] <- loss[1]
    # testing lda error rate
    loss_df[4*(i-1)+2,1] <- "LDA"
    loss_df[4*(i-1)+2,2] <- "Test"
    loss_df[4*(i-1)+2,3] <- loss[2]
    # training qda error rate
    loss_df[4*(i-1)+3,1] <- "QDA"
    loss_df[4*(i-1)+3,2] <- "Train"
    loss_df[4*(i-1)+3,3] <- loss[3]
    # testing qda error rate
    loss_df[4*(i-1)+4,1] <- "QDA"
    loss_df[4*(i-1)+4,2] <- "Test"
    loss_df[4*(i-1)+4,3] <- loss[4]
  }

  return(loss_df) # return all error rates for 1000 simulations
}

# calculate loss for 1000 iteration of lda/qda linear simulated data of size 1000
lin_loss <- sim_1000("y ~ x1 + x2", TRUE, 1000)
```

b)

```
lin_loss %>%
  group_by(Method, Session) %>%
  summarise(Mean_Error_Rate = mean(Error),
            SD_Error_Rate = sd(Error)) %>%
  kable(digits = 4,
        caption = "Means and Standard Deviations across 1,000 simulations (Linear)")

ggplot(lin_loss, aes(x = Error, fill = Method)) +
  geom_density(size = 0.5, alpha = 0.3) +
```

```
facet_wrap(~Session, ncol = 1) +
labs(title = "Error rate distributions across 1,000 simulations (Linear)") +
scale_fill_manual(values = c("#990066","#FF9900"))
```

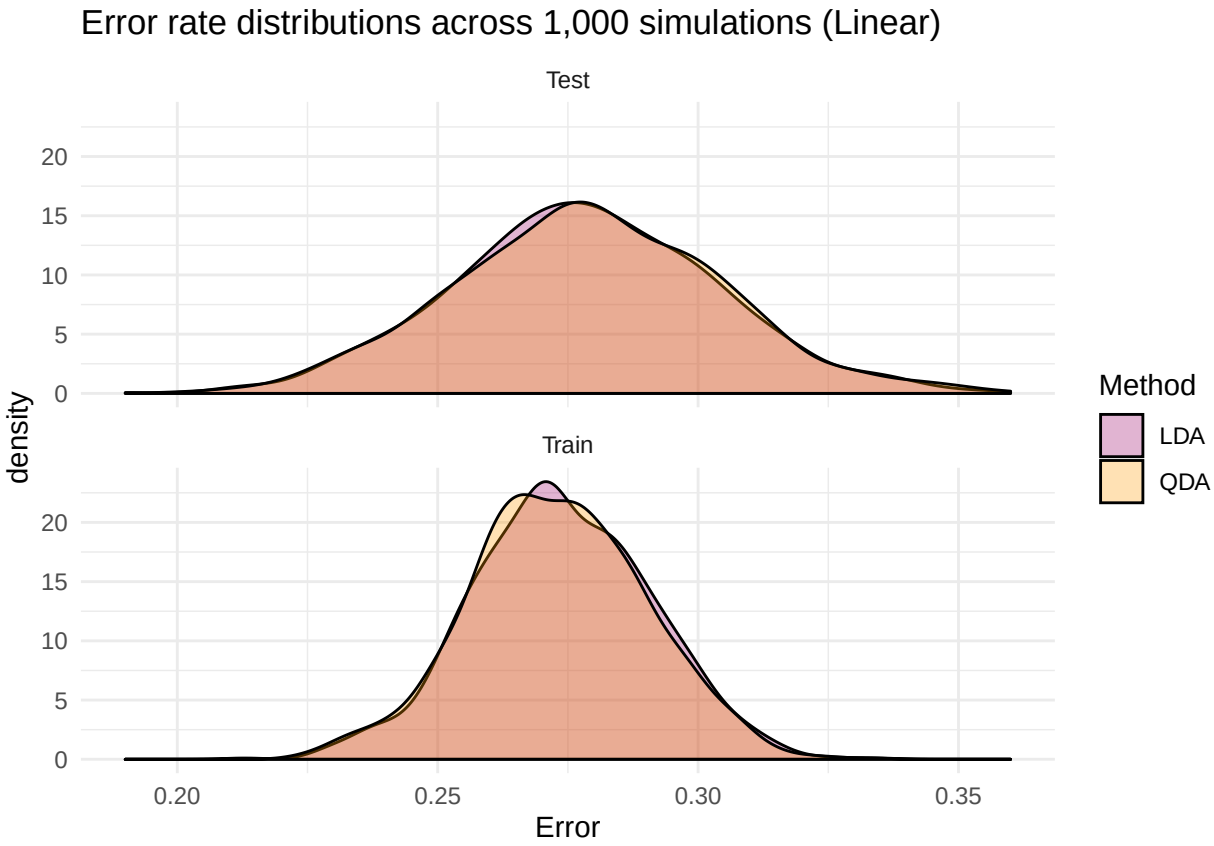


Table 1: Means and Standard Deviations across 1,000 simulations (Linear)

Method	Session	Mean_Error_Rate	SD_Error_Rate
LDA	Test	0.2782	0.0253
LDA	Train	0.2738	0.0172
QDA	Test	0.2788	0.0256
QDA	Train	0.2730	0.0171

From the simulation, the performance differences between LDA and QDA in both training and testing sets are in fact very small and likely are not significant. However, we still can see that in the training set, QDA has slightly lower mean error rate than LDA. On the other hand, in the testing set, LDA has slightly lower mean error rate than QDA. The direction of differences supports my answer prior to the simulation. In terms of the variance of error rates distribution, QDA has lower standard deviation than LDA in training set but has higher standard deviation than LDA in testing set, which again, support my answer.

3) Exploring Simulated Differences between LDA and QDA (Non-Linear)

As in 2), we expect QDA to perform better in training set because it is more flexible and hence fit better. For the Bayes decision boundary that is non-linear, we also expect QDA to perform better than LDA in testing

set because LDA won't be able to capture the general non-linear trend of the Bayes decision boundary while QDA can.

a)

```
# calculate loss for 1000 iteration of lda/qda non-linear simulated data of size 1000
non_lin_loss <- sim_1000("y ~ x1 + x1^2 + x2 + x2^2", FALSE, 1000)
```

b)

```
non_lin_loss %>%
  group_by(Method, Session) %>%
  summarise(Mean_Error_Rate = mean(Error),
            SD_Error_Rate = sd(Error)) %>%
  kable(digits = 4,
        caption = "Means and Standard Deviations across 1,000 simulations (Non-Linear)")

ggplot(non_lin_loss, aes(x = Error, fill = Method)) +
  geom_density(size = 0.5, alpha = 0.3) +
  facet_wrap(~Session, ncol = 1) +
  labs(title = "Error rate distributions across 1,000 simulations (Non-Linear)") +
  scale_fill_manual(values = c("#990066","#FF9900"))
```

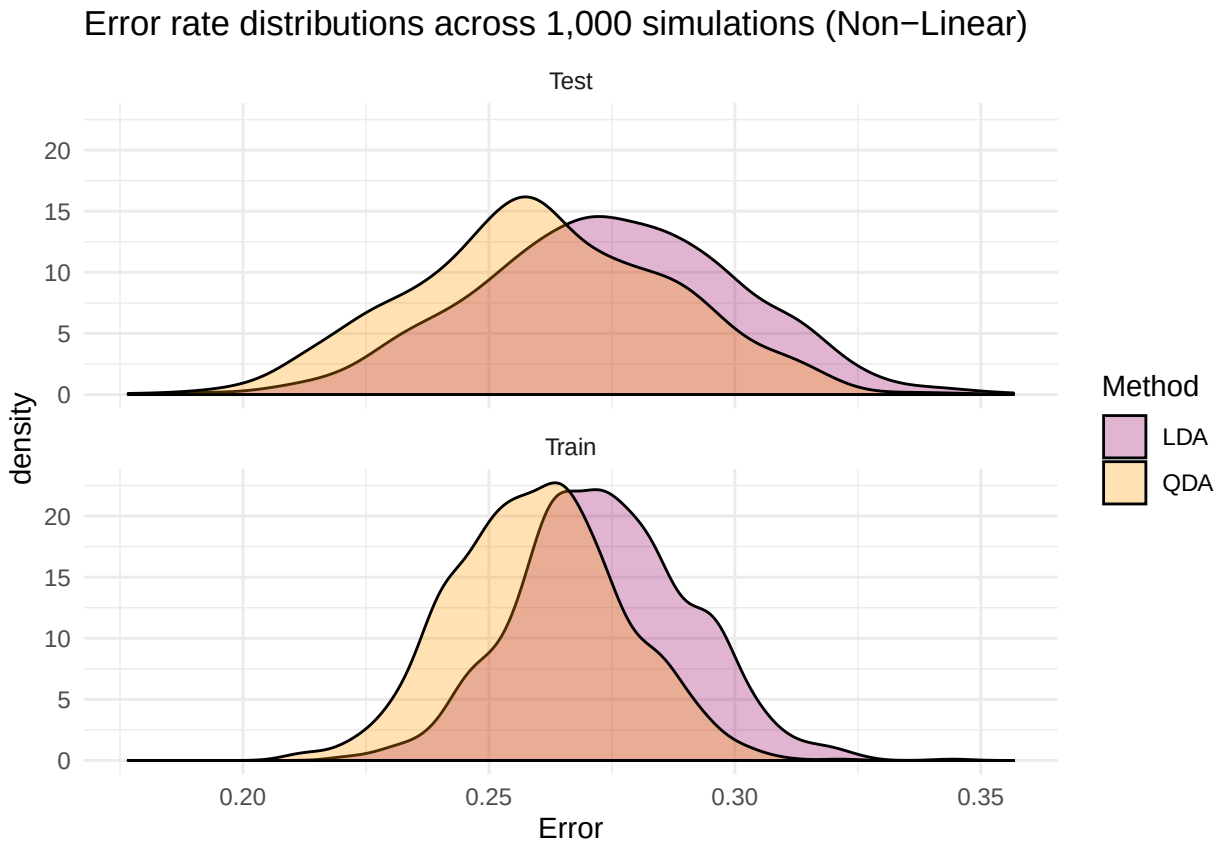


Table 2: Means and Standard Deviations across 1,000 simulations (Non-Linear)

Method	Session	Mean_Error_Rate	SD_Error_Rate
LDA	Test	0.2741	0.0266
LDA	Train	0.2736	0.0176
QDA	Test	0.2607	0.0262
QDA	Train	0.2599	0.0170

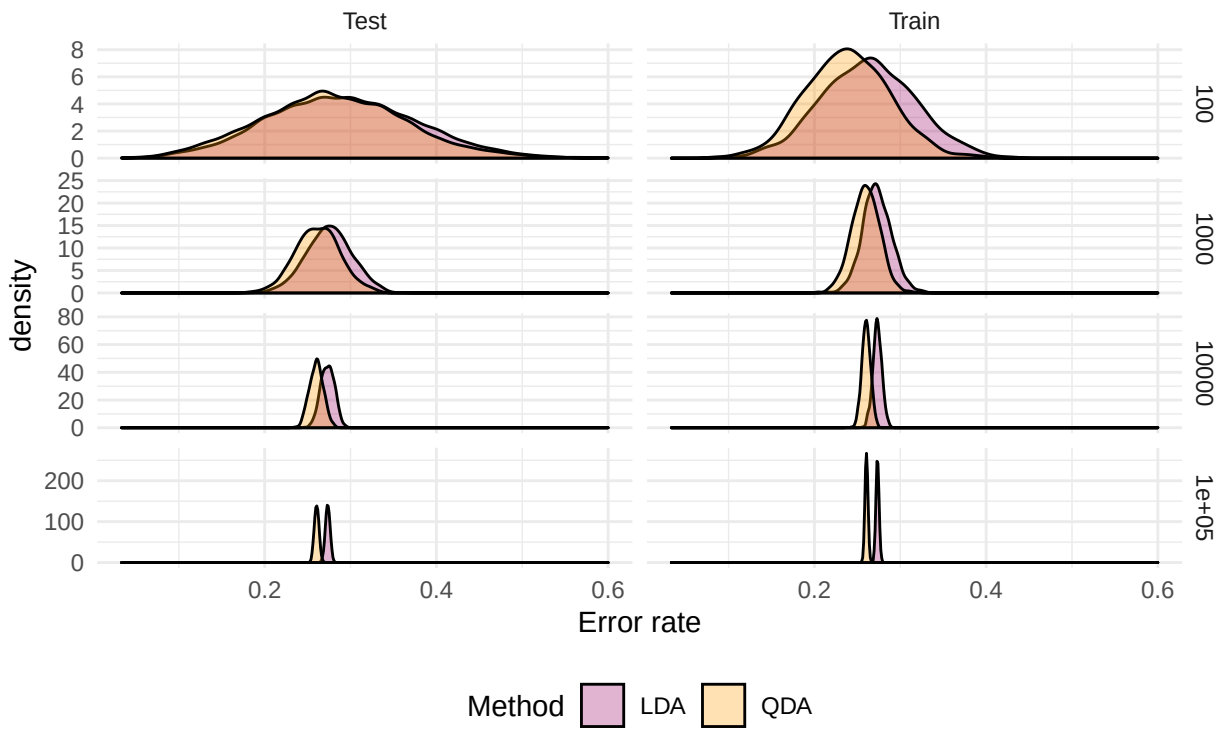
From the simulation, we can see QDA has lower mean error rates as well as standard deviations in both training and testing sets, supporting my answer prior to simulation. The performance differences between LDA and QDA in the non-linear case are larger than in the linear case. This is likely because in linear case, both LDA and QDA can capture the trend well. However, in the non-linear case, LDA is too inflexible that it cannot capture the non-linear trend.

4) Exploring Simulated Differences between LDA and QDA (Sample size)

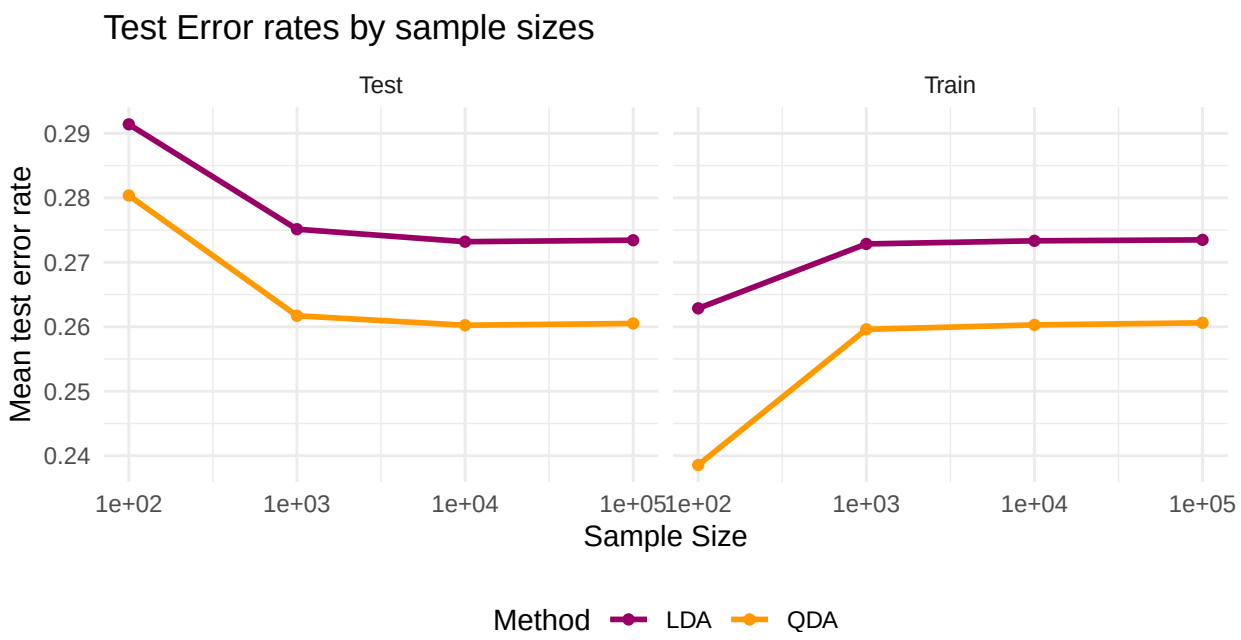
Across different sample sizes and training/testing sets, QDA would perform better than LDA since the data is non-linear. Nevertheless, as sample size increases, we should see less gap in performances between LDA and QDA because larger sample sizes can, to certain extent, halp with LDA inflexibility problem.

```
n <- c(1e02, 1e03, 1e04, 1e05) # sample sizes to vary
loss_n <- vector("list", length(n)) # initializing data structure
for (i in seq(length(n))){
  # getting error rates table for each sample size
  loss_n[[i]] <- sim_1000("y ~ x1 + x1^2 + x2 + x2^2", FALSE, n[i])
}
```

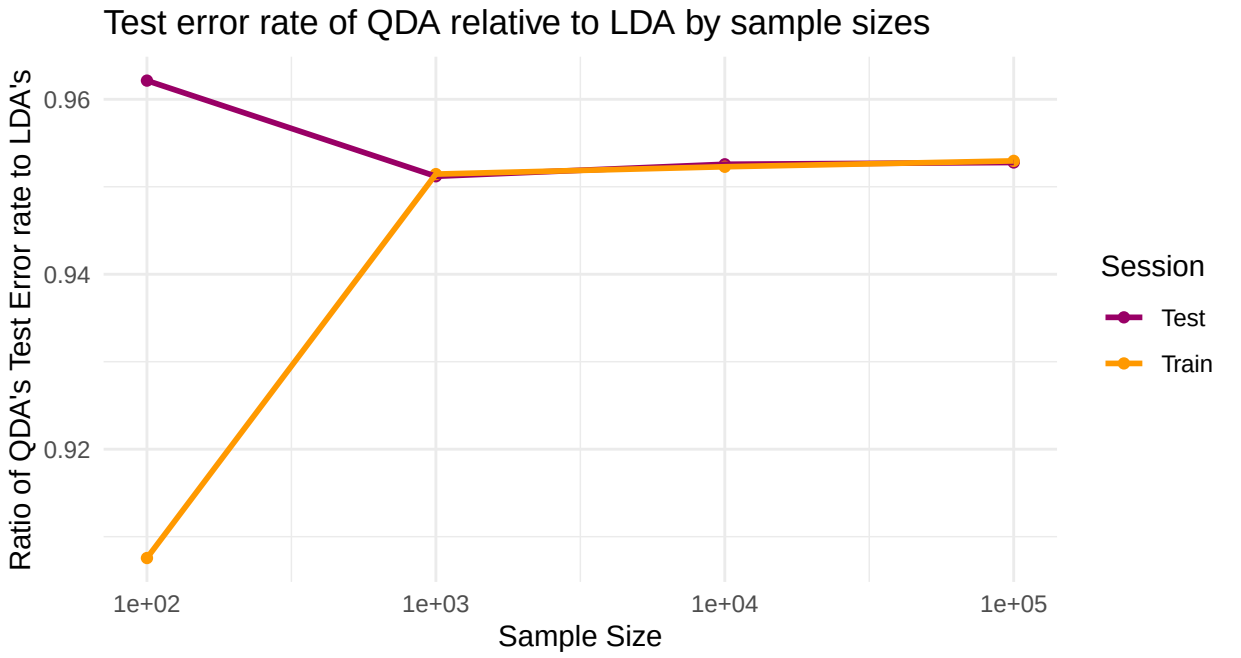
```
loss_n_df <- as.data.frame(cbind(n, loss_n)) %>% # attach sample sizes
  unnest() # turn into one long dataframe

ggplot(loss_n_df, aes(x = Error, fill = Method)) +
  geom_density(size = 0.5, alpha = 0.3) +
  facet_grid(n~Session, scales="free_y") +
  labs(title = "Error rate distributions across 1,000 simulations",
       subtitle = "Varied sample size; Non-Linear",
       x = "Error rate") +
  theme(legend.position = "bottom") +
  scale_fill_manual(values = c("#990066","#FF9900"))
```




```
loss_n_df %>%
  group_by(n, Method, Session) %>%
  summarise(Mean_Error_Rate = mean(Error)) %>%
  ggplot(aes(n, Mean_Error_Rate, color = Method)) +
  geom_point() +
  geom_line(size = 1) +
  scale_x_log10() +
  facet_wrap(~Session) +
  labs(y = "Mean test error rate",
       x = "Sample Size",
       title = "Test Error rates by sample sizes") +
  scale_color_manual(values = c("#990066","#FF9900")) +
  theme(legend.position = "bottom")
```




```
loss_n_df %>%
  group_by(n, Method, Session) %>%
  summarise(Mean_Error_Rate = mean(Error)) %>%
  spread(Method, Mean_Error_Rate) %>%
  mutate(QDA_LDA_ratio = QDA/LDA) %>%
  ggplot(aes(n, QDA_LDA_ratio, color = Session)) +
  geom_point() +
  geom_line(size = 1) +
  scale_x_log10() +
  labs(y = "Ratio of QDA's Test Error rate to LDA's",
       x = "Sample Size",
       title = "Test error rate of QDA relative to LDA by sample sizes") +
  scale_color_manual(values = c("#990066","#FF9900"))
```



From the simulation, we actually see some interesting patterns of changes as sample size increases. For training set, as sample size increases, the error rates for both QDA and LDA increase. On the other hand, for testing set, as sample size increases, the error rates for both QDA and LDA decrease. In terms of relative error rates, for training set, as sample size increases, the QDA's and LDA's error rates become more similar to each other. This part of the result reflects my answer prior to the simulation. On the other hand, for testing set, as sample size increases, the QDA's and LDA's error rates become less similar to each other (but with lower magnitude of change than in the training set). In addition, we can see that for both LDA and QDA, as sample size increases, the test error rates converge to the train error rates. This probably happens because as sample size increases, both LDA and QDA are less likely to overfit. Hence, the train error rates actually increase while the test error rates decrease.

5) Modeling voter turnout

```
# read and preprocess the data
mh <- read_csv("mental_health.csv") %>%
  mutate(vote96 = as.factor(vote96)) %>% # turn outcome to binary classification
  select(mhealth_sum, age, educ, black,
         female, married, inc10, vote96) %>% # select the key variables
  drop_na() # drop data with NA
```

a)

```
# training/testing split 70/30
mh_split <- initial_split(data = mh, prop = 0.7)
mh_train <- training(mh_split)
mh_test <- testing(mh_split)
```

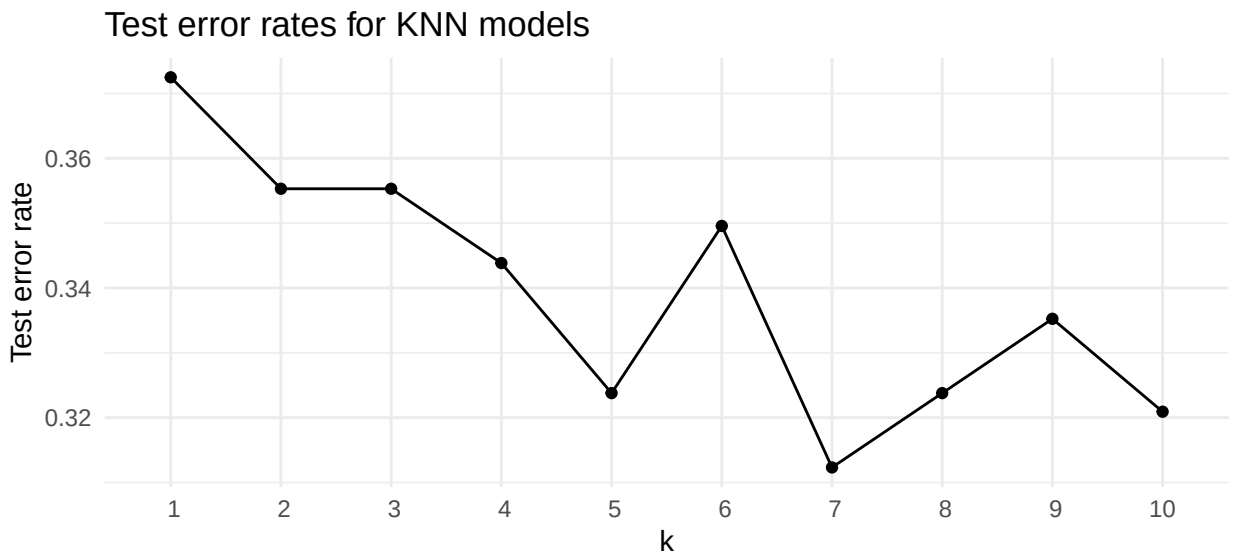
b)

```
form <- "vote96 ~ mhealth_sum + age + educ + black + female + married + inc10"
# i) Logistic regression
logit_mh <- glm(as.formula(form), data = mh_train, family = "binomial")
# ii) LDA
lda_mh <- MASS::lda(as.formula(form), data = mh_train)
# iii) QDA
qda_mh <- MASS::qda(as.formula(form), data = mh_train)
# iv) Naive Bayes
x_nb <- mh_train %>%
  select(-vote96)
y_nb <- mh_train$vote96
nb_mh <- naiveBayes(vote96~., data = mh_train)
# v) knn (k from 1 to 10)
err_knn <- tibble(k = 1:10,
                  knn_test = map(k, ~ class::knn(select(mh_train, -vote96),
                                                 test = select(mh_test, -vote96),
                                                 cl = mh_train$vote96, k = .)),
                  err_test = map_dbl(knn_test, ~ mean(mh_test$vote96 != .))) %>%
  mutate(knn = paste0("KNN_",k)) # name each model
```

c)

i)

```
# logistic error rate
pred_logit_cont <- predict(logit_mh, newdata = mh_test) # probability
pred_logit <- ifelse(predict(logit_mh, newdata = mh_test) >= 0.5, 1, 0) # binary
err_logit <- mean(mh_test$vote96 != pred_logit)

# lda error rate
pred_lda <- predict(lda_mh, newdata = mh_test)
err_lda <- mean(mh_test$vote96 != pred_lda$class)

# qda error rate
pred_qda <- predict(qda_mh, newdata = mh_test)
err_qda <- mean(mh_test$vote96 != pred_qda$class)

# naive bayes error rate
pred_nb <- predict(nb_mh, newdata = mh_test) # binary
pred_nb_cont <- predict(nb_mh, newdata = mh_test, type = "raw")[,2] # probability
err_nb <- mean(mh_test$vote96 != pred_nb)

# knn error rates
err_knn %>%
  ggplot(aes(k, err_test)) +
  geom_point() +
  geom_line() +
  scale_x_discrete(name ="k",
                   limits=c(1:10)) +
  labs(title = "Test error rates for KNN models",
       y = "Test error rate")
```




```
# tabulate test error rates across models
Method <- c("Logistic Regression", "LDA", "QDA", "Naive Bayes", err_knn$knn)
Error_rate <- c(err_logit, err_lda, err_qda, err_nb, err_knn$err_test)
as.data.frame(cbind(Method, Error_rate)) %>%
  arrange(Error_rate) %>%
```

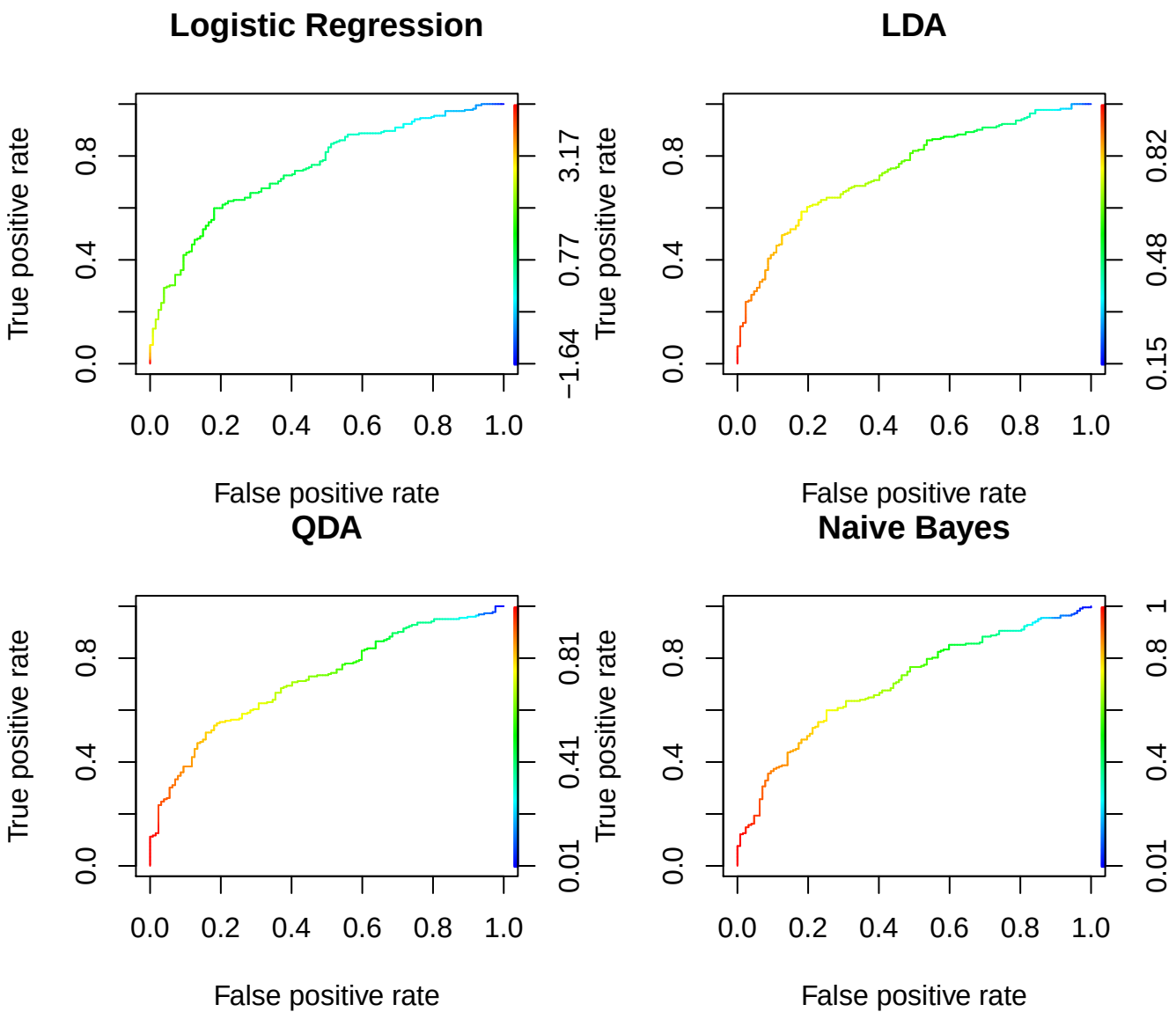
```
  mutate(Error_rate = as.numeric(as.character(Error_rate))) %>%
  kable(digits = 4, caption = "Test error rates across Models")
```

Table 3: Test error rates across Models

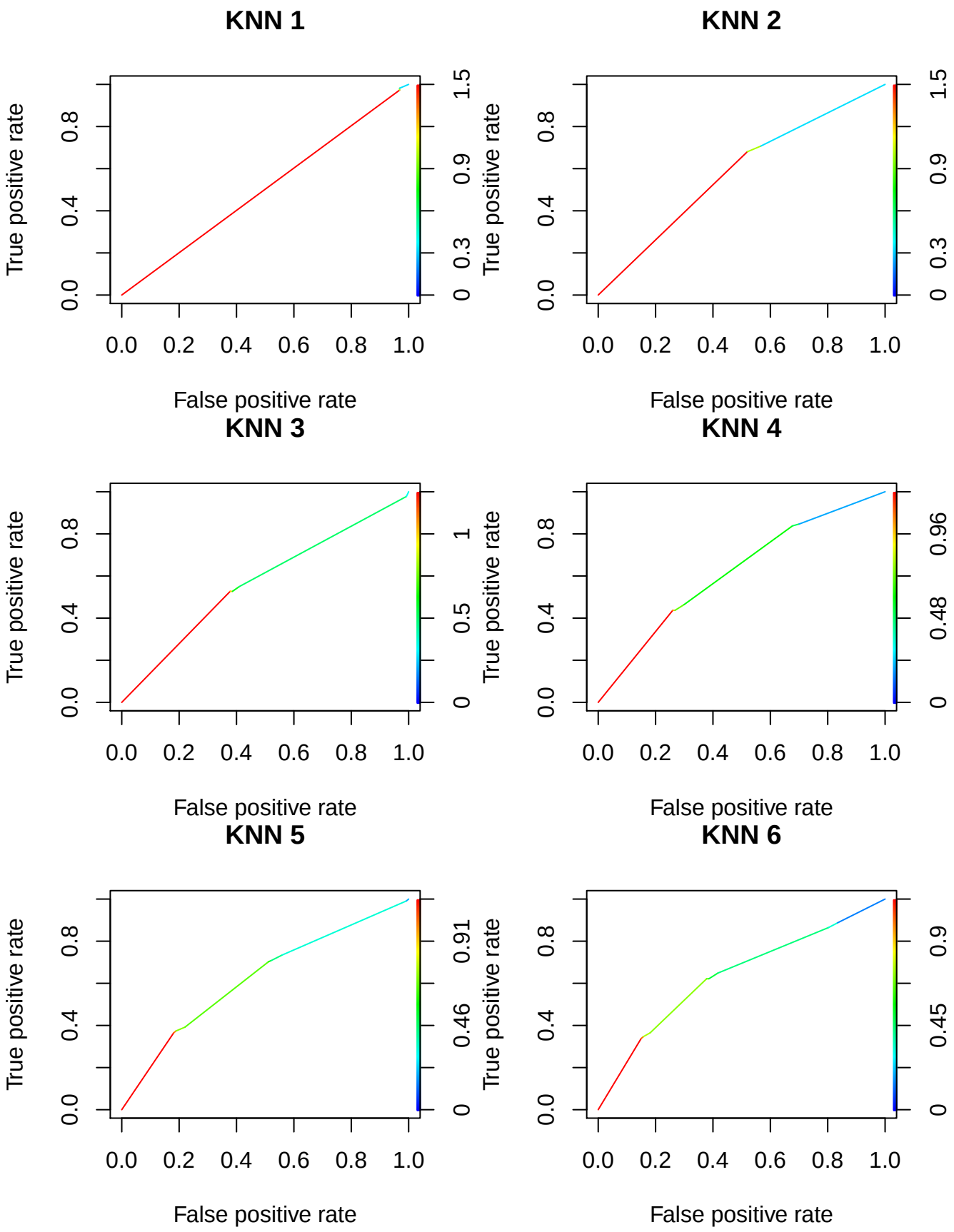
Method	Error_rate
Logistic Regression	0.3095
LDA	0.3123
KNN_7	0.3123
Naive Bayes	0.3181
KNN_10	0.3209
KNN_5	0.3238
KNN_8	0.3238
QDA	0.3295
KNN_9	0.3352
KNN_4	0.3438
KNN_6	0.3496
KNN_2	0.3553
KNN_3	0.3553
KNN_1	0.3725

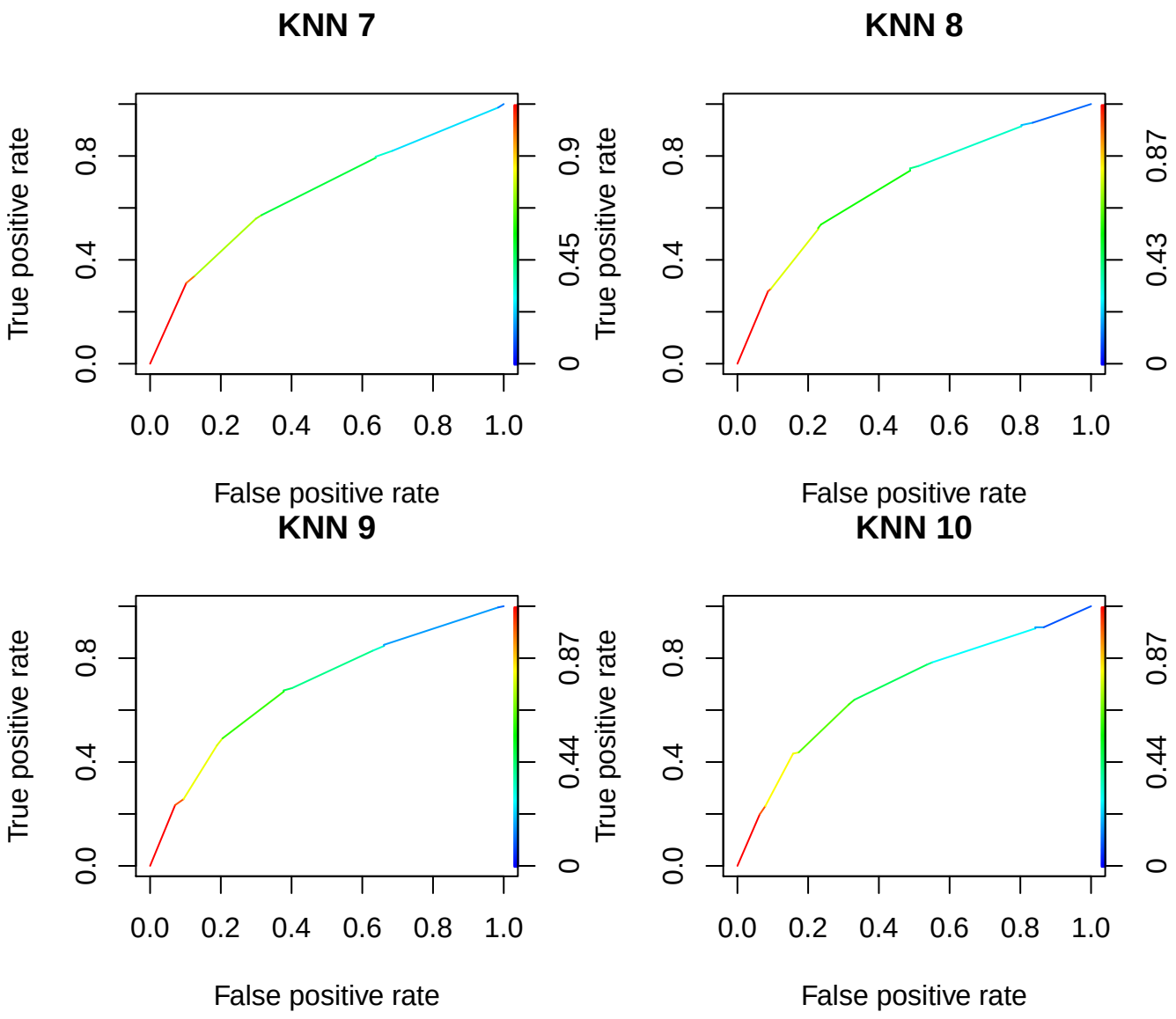
ii) ROC

```
# Logistic regression
pred1 <- prediction(pred_logit_cont, mh_test$vote96)
perf1 <- performance(pred1,"tpr","fpr")
# LDA
pred2 <- prediction(pred_lda$posterior[,2], mh_test$vote96)
perf2 <- performance(pred2,"tpr","fpr")
# QDA
pred3 <- prediction(pred_qda$posterior[,2], mh_test$vote96)
perf3 <- performance(pred3,"tpr","fpr")
# Naive Bayes
pred4 <- prediction(pred_nb_cont, mh_test$vote96)
perf4 <- performance(pred4,"tpr","fpr")

perf <- c(perf1, perf2, perf3, perf4)
main <- c("Logistic Regression", "LDA", "QDA", "Naive Bayes")

# plot ROC for the first 4 models
i <- 1
for (p in perf){
  plot(p,colorize=TRUE, main=main[i])
  i <- i+1
}
```




```
# plot ROC for KNN models
# https://stackoverflow.com/questions/11741599/how-to-plot-a-roc-curve-for-a-knn-model
knn_roc <- function(k){
  knn <- class::knn(select(mh_train, -vote96),
          test = select(mh_test, -vote96),
          cl = mh_train$vote96, k = k, prob=TRUE)
  prob <- attr(knn, "prob")
  prob <- 2*ifelse(knn == "-1", 1-prob, prob) - 1
  pred <- prediction(prob, mh_test$vote96)
  perf <- performance(pred,"tpr","fpr")
  plot(perf,colorize=TRUE, main=paste("KNN", k))
}
# plot all 10 knn models' ROCs
for (i in seq(10)){
  knn_roc(i)
}
```


KNN 1
True positive rate
False positive rate
KNN 2
True positive rate
False positive rate
KNN 3
True positive rate
False positive rate
KNN 4
True positive rate
False positive rate
KNN 5
True positive rate
False positive rate
KNN 6
True positive rate
False positive rate



ii) AUC

```
# empty dataframe to hold AUCs
auc_df <- as.data.frame(matrix(NA, nrow = 14, ncol = 2))
names(auc_df) <- c("Method", "AUC")
# Logistic
auc_df[1,1] <- "Logistic Regression"
auc_df[1,2] <- performance(pred1, measure = "auc")@y.values[[1]]
# LDA
auc_df[2,1] <- "LDA"
auc_df[2,2] <- performance(pred2, measure = "auc")@y.values[[1]]
# QDA
auc_df[3,1] <- "QDA"
auc_df[3,2] <- performance(pred3, measure = "auc")@y.values[[1]]
# Naive Bayes
auc_df[4,1] <- "Naive Bayes"
auc_df[4,2] <- performance(pred4, measure = "auc")@y.values[[1]]
```

```r
# calculate AUCs for all knn models
knn_auc <- function(k){
  knn <- class::knn(select(mh_train, -vote96),
          test = select(mh_test, -vote96),
          cl = mh_train$vote96, k = k, prob=TRUE)
  prob <- attr(knn, "prob")
  prob <- 2*ifelse(knn == "-1", 1-prob, prob) - 1
  pred <- prediction(prob, mh_test$vote96)
  return(performance(pred, measure = "auc")@y.values[[1]])
}
for (i in seq(5,14)){
  auc_df[i,1] <- paste("KNN", i-4)
  auc_df[i,2] <- knn_auc(i)
}

auc_df %>%
  arrange(desc(AUC)) %>%
  kable(digits = 4, caption = "AUC across different models")
```

Table 4: AUC across different models

Method	AUC
Logistic Regression	0.7510
LDA	0.7477
KNN 10	0.7194
QDA	0.7162
KNN 9	0.7105
Naive Bayes	0.7025
KNN 8	0.6979
KNN 7	0.6876
KNN 5	0.6854
KNN 4	0.6841
KNN 6	0.6820
KNN 3	0.6545
KNN 2	0.6340
KNN 1	0.6228

d)

From the test error rates and ROCs/AUCs across models, we can see that logistic regression has the lowest error rates as well as highest ROC/AUC. Since the lower the test error rate, the more accurate the model, and the higher the ROC/AUC, the better the model in distinguishing between classes, the logistic regression is the best model for this particular data.
Note that from the simulation, low test error rate relative to other models does not gaurantee high ROC/AUC relative to other models, and vice versa. Hence, if there is no convergence in lowest test error rate and highest ROC/AUC like we have logistive regression in this particular case, we will need to choose the *best* model based on our goal and underlying sample distribution. If the sample distribution is very skewed or our goal is to not overfit to a single class, we should use ROC/AUC as our criterion. However, if our goal is to build a model that is representative but we don't care about the model being discriminative, test error rates would be sufficient.
In addition, Corinna Cortes and Mehryar Mohri (2004) also suggested that if we want to use AUC as the model selection criterion, we should train the model using an algorithm specifically designed to globally

optimize the AUC rather than other algorithms. Hence, if we want to use ROC/AUC as the main criterion, we may need to check what is the objective function optimized in our algorithms used to train our models.