Modeling PS2

borui sun

1/28/2020

The Bayes Classifier

1. (20 points) For classification problems, the test error rate is minimized by a simple classifier that assigns each observation to the most likely class given its predictor values:

$$\Pr(Y = j|X = x_0)$$

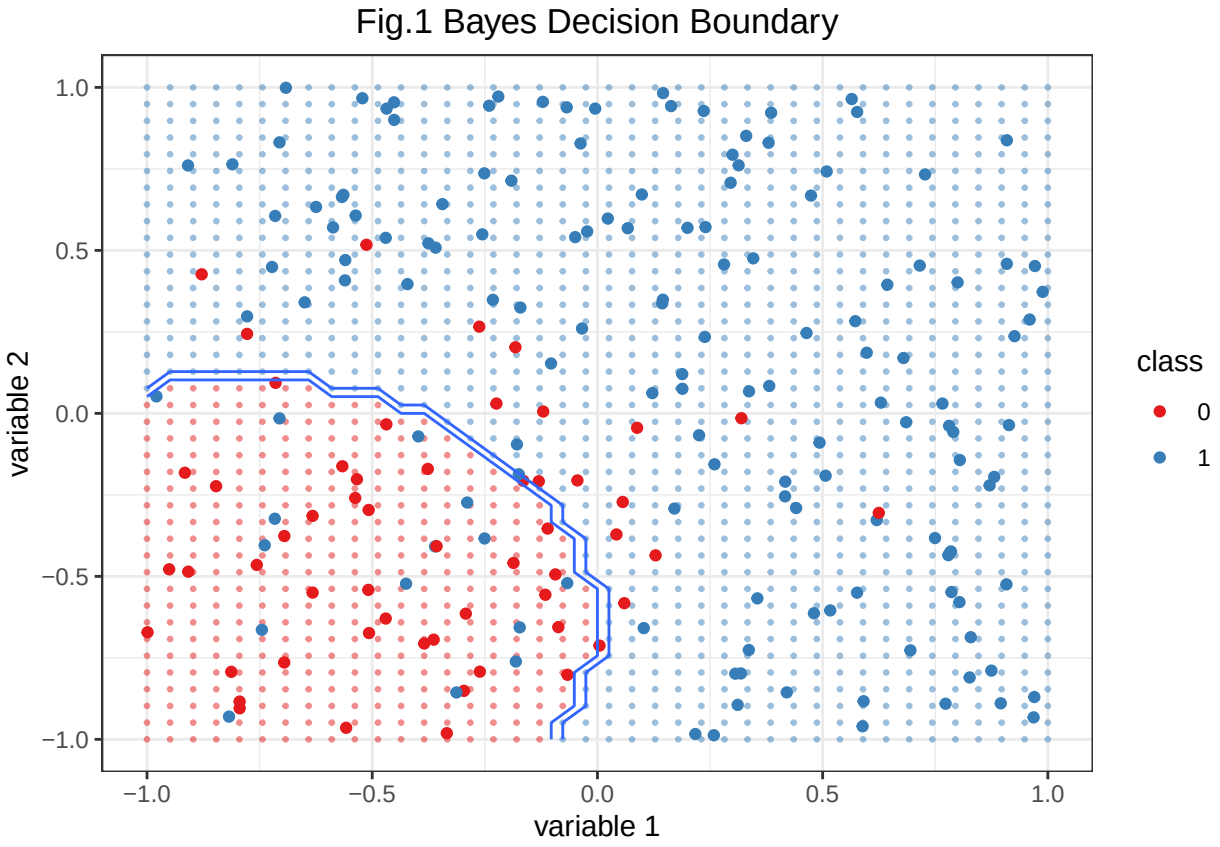
where x_0 is the test observation and each possible class is represented by J. This is a **conditional probability** that $Y = j$, given the observed predictor vector x_0. This classifier is known as the **Bayes classifier**. If the response variable is binary (i.e. two classes), the Bayes classifier corresponds to predicting class one if $\Pr(Y = 1|X = x_0) > 0.5$, and class two otherwise.

Produce a graph illustrating this concept. Specifically, implement the following elements in your program:

   a. Set your random number generator seed.
   b. Simulate a dataset of $N = 200$ with X_1, X_2 where X_1, X_2 are random uniform variables between $[-1, 1]$.
   c. Calculate $Y = X_1 + X_1^2 + X_2 + X_2^2 + \epsilon$, where $\epsilon \sim N(\mu = 0, \sigma^2 = 0.25)$.
   d. Y is defined in terms of the log-odds of success on the domain $[-\infty, +\infty]$. Calculate the probability of success bounded between $[0, 1]$.
   e. Plot each of the data points on a graph and use color to indicate if the observation was a success or a failure.
   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.

```
set.seed(123)

N = 200

simu_sample <- data.frame(
    x1 = runif(N, -1, 1),
    x2 = runif(N, -1, 1),
    epsilon = rnorm(N, mean = 0, sd = sqrt(0.25)) # sd = sigma
) %>%
    mutate(y = x1 + x1^2 + x2 + x2^2 + epsilon,
           pr = exp(y)/(1 + exp(y)), # as y <- log(pr/(1-pr))
           class = as.numeric(pr > 0.5))

bayes <- naiveBayes(as.factor(class) ~ x1 + x2, simu_sample)

sample_grid <- expand.grid(x1 = seq(-1, 1, length.out = 40), x2 = seq(-1, 1, length.out = 40))
sample_grid$class <- predict(bayes, sample_grid, type = "class")

ggplot() +
    geom_point(data = sample_grid, aes(x = x1, y = x2, color = as.factor(class)), alpha = 0.5, size = 0
```

```
  geom_point(data = simu_sample, aes(x = x1, y = x2, color =  as.factor(class))) +
  geom_contour(data = sample_grid, aes(x = x1, y = x2, z = as.numeric(class)),  bins = 2, binwidth = 
  labs(title = "Fig.1 Bayes Decision Boundary",
       x = "variable 1",
       y = "variable 2",
       color = "class")  +
  scale_color_brewer(palette = "Set1")
```



2. (20 points) If the Bayes decision boundary is linear, do we expect LDA or QDA to perform better on the training set? On 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.

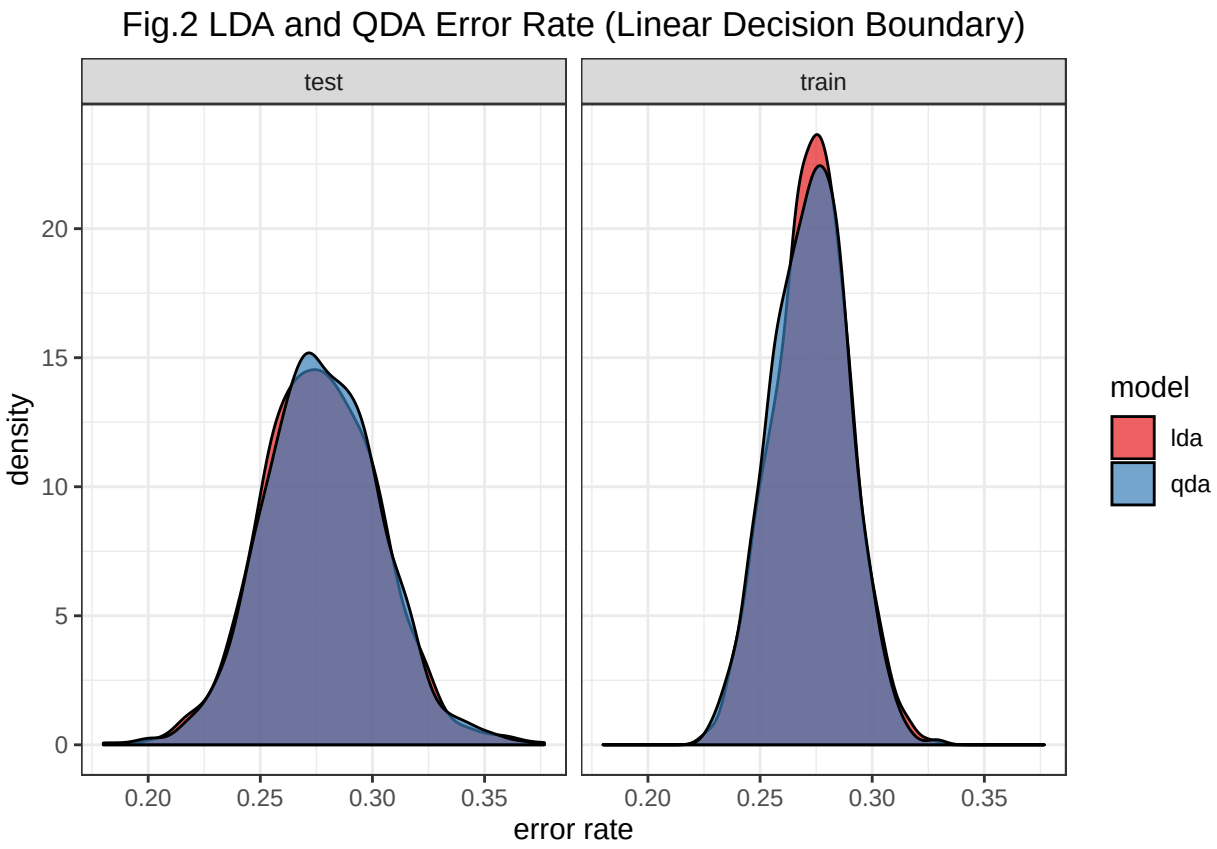
      iv. Calculate each model's training and test error rate.

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

```r
set.seed(234)

error_rate <- function(model, data, outcome){

    accuracy <- predict(model, data) %>%
        {.$class} %>%
        confusionMatrix(as.factor(data[[outcome]])) %>%
        {.$overall[1]}

    return(1 - accuracy)
}

repetition <- 1000

linear <- data.frame(matrix(nrow = 0, ncol = 0))

for (i in 1: repetition){

    N <- 1000

    sample <- data.frame(x1 = runif(N, -1, 1),
                         x2 = runif(N, -1, 1),
                         epsilon = rnorm(N, mean = 0, sd =1)) %>%
        mutate(y = x1 + x2 + epsilon,
               class = y >= 0)

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

    lda <- lda(class ~ x1 + x2, data = train)

    lda_train_error <- error_rate(lda, train, "class")
    lda_test_error <- error_rate(lda, test, "class")

    qda <- qda(class ~ x1 + x2, data = train)

    qda_train_error <- error_rate(qda, train, "class")
    qda_test_error <- error_rate(qda, test, "class")

    id <- i
    result <- data.frame(id, lda_train_error, lda_test_error,
                                    qda_train_error, qda_test_error)
    linear <- bind_rows(linear, result)
}
```

```r
linear %>% pivot_longer(
    cols = ends_with("error"),
    names_to = c("model", "data"),
    names_pattern = "(.*)_(.*)_error$",
    values_to = "error rate"
) %>%
    ggplot(aes(x = `error rate`, fill = model)) +
```

```
    geom_density(alpha = 0.7) +
    facet_wrap(~data) +
    scale_fill_brewer(palette = "Set1") +
    labs(title = "Fig.2 LDA and QDA Error Rate (Linear Decision Boundary)")
```




```
summary(linear[,2:5]) %>% kable()
```

lda_train_error	lda_test_error	qda_train_error	qda_test_error
Min. :0.2243	Min. :0.1800	Min. :0.2257	Min. :0.1833
1st Qu.:0.2625	1st Qu.:0.2600	1st Qu.:0.2600	1st Qu.:0.2600
Median :0.2743	Median :0.2767	Median :0.2729	Median :0.2767
Mean :0.2730	Mean :0.2774	Mean :0.2723	Mean :0.2780
3rd Qu.:0.2843	3rd Qu.:0.2967	3rd Qu.:0.2843	3rd Qu.:0.2967
Max. :0.3271	Max. :0.3767	Max. :0.3300	Max. :0.3700

```
cat(paste0("LDA Train Variance: ", var(linear$lda_train_error), "\n",
           "QDA Train Variance: ", var(linear$qda_train_error), "\n",
           "LDA Test Variance: ", var(linear$lda_test_error), "\n",
           "QDA Test Variance: ", var(linear$qda_test_error)))
```

```
## LDA Train Variance: 0.000280863261220404
## QDA Train Variance: 0.000283524169067026
```

```
## LDA Test Variance: 0.000677885841396952
## QDA Test Variance: 0.000662214203091981
```

We expects that LDA would perform better than QDA when the decision boundary is linear.

Based on Fig.2 and the descriptive summary above, we can see that LDA and QDA perform roughly the same. However, QDA has a slightly smaller error rate (mean: 0.2722, median: 0.2714) than LDA (mean: 0.2731, median: 0.2729) on when fitting to the training data set, but when fitting to the test data set, the average error rate of QDA (mean: 0.277) is slightly larger than the average error rate of LDA. This supports our expectation. While QDA is more flexible than LDA, it also captures more noises and tends to overfitting. This explains why QDA has lower error rate in the training but high error rate in the test data set. LDA, while less flexible, is more consistant and thus has lower variance in the test data set.

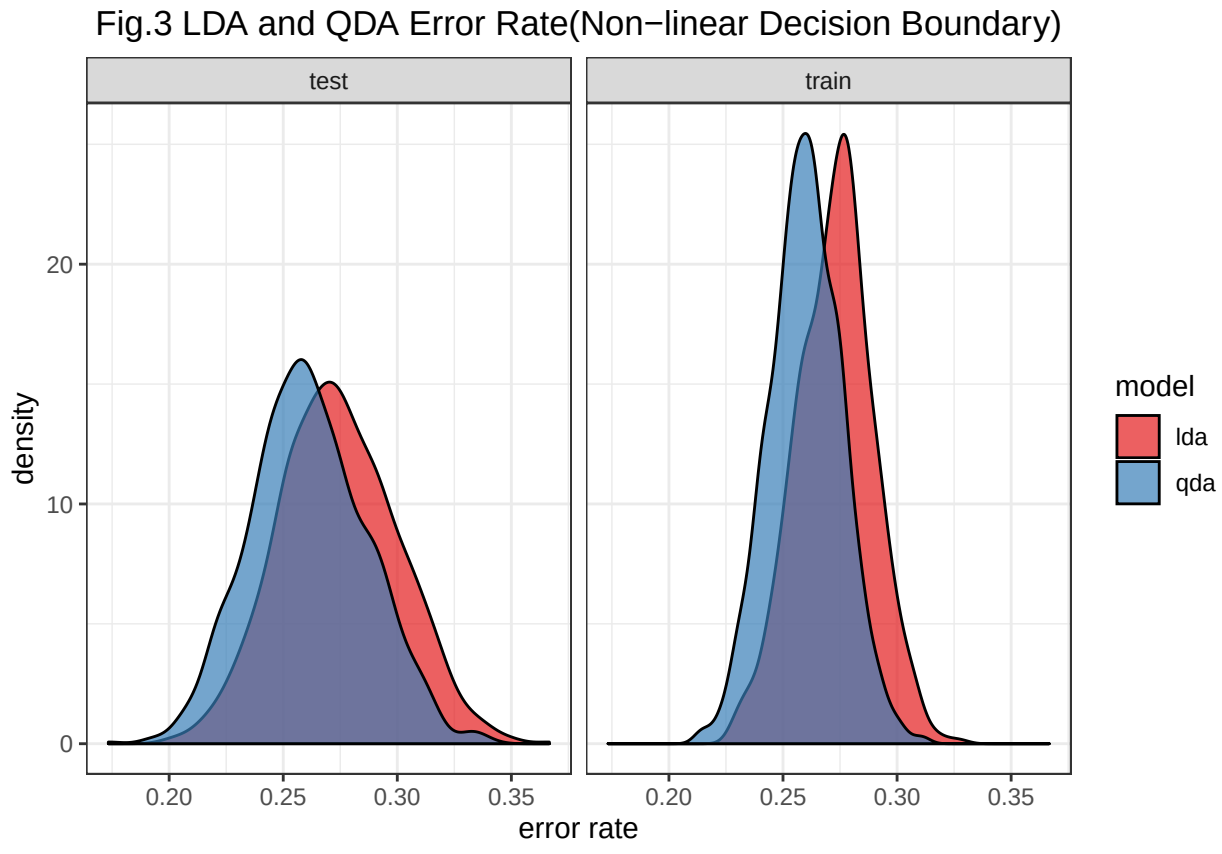
3. (20 points) If the Bayes decision boundary is non-linear, do we expect LDA or QDA to perform better on the training set? On the test set?

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

```
set.seed(345)

non_linear <- data.frame(matrix(nrow = 0, ncol = 0))

for (i in 1: repetition){

    N <- 1000

    sample <- data.frame(x1 = runif(N, -1, 1),
                         x2 = runif(N, -1, 1),
                         epsilon = rnorm(N, mean = 0, sd =1)) %>%
        mutate(y = x1 + x1^2 + x2 + x2^2 + epsilon,
               class = y >= 0)

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

    lda <- lda(class ~ x1 + x1^2 + x2 + x2^2, data = train)

    lda_train_error <- error_rate(lda, train, "class")
    lda_test_error <- error_rate(lda, test, "class")

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

```
    qda_train_error <- error_rate(qda, train, "class")
    qda_test_error <- error_rate(qda, test, "class")

    id <- i
    result <- data.frame(id, lda_train_error, lda_test_error,
                                    qda_train_error, qda_test_error)

    non_linear <- bind_rows(non_linear, result)
}
```

```
non_linear %>% pivot_longer(
    cols = ends_with("error"),
    names_to = c("model", "data"),
    names_pattern = "(.*)_(.*)_error$",
    values_to = "error rate"
) %>%
    ggplot(aes(x = `error rate`, fill = model)) +
    geom_density(alpha = 0.7) +
    facet_wrap(~data) +
    scale_fill_brewer(palette = "Set1") +
    labs(title = "Fig.3 LDA and QDA Error Rate(Non-linear Decision Boundary)")
```




```
summary(non_linear[, 2:5]) %>% kable()
```

lda_train_error	lda_test_error	qda_train_error	qda_test_error
Min. :0.2271	Min. :0.2000	Min. :0.2129	Min. :0.1733
1st Qu.:0.2614	1st Qu.:0.2567	1st Qu.:0.2500	1st Qu.:0.2433
Median :0.2743	Median :0.2733	Median :0.2600	Median :0.2600
Mean :0.2727	Mean :0.2743	Mean :0.2598	Mean :0.2612
3rd Qu.:0.2843	3rd Qu.:0.2933	3rd Qu.:0.2700	3rd Qu.:0.2767
Max. :0.3286	Max. :0.3667	Max. :0.3114	Max. :0.3400

```
cat(paste0("LDA Train Variance: ", var(non_linear$lda_train_error), "\n",
           "QDA Train Variance: ", var(non_linear$qda_train_error), "\n",
           "LDA Test Variance: ", var(non_linear$lda_test_error), "\n",
           "QDA Test Variance: ", var(non_linear$qda_test_error)))
```

```
## LDA Train Variance: 0.000274705203162346
## QDA Train Variance: 0.000250317907703622
## LDA Test Variance: 0.00067313036369703
## QDA Test Variance: 0.000640908897786676
```

We expect that QDA would perform better when the decision boundary is not linear.

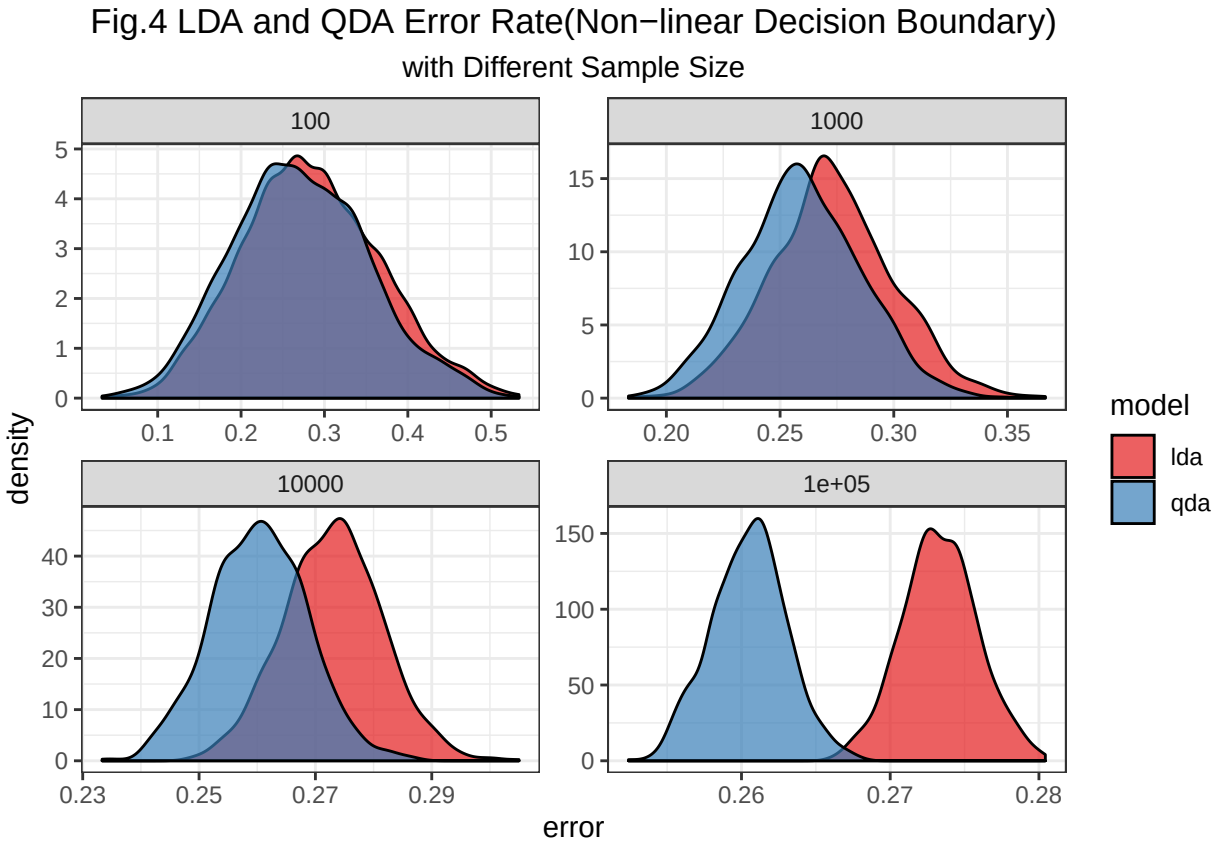
Fig. 3 and the descriptive summary above well support our expectation. When the decision boundary is quadratic, the less flexible LDA model fails to capture the non-linear relationship. In both training and test data set, the error rate of QDA model is substantially less than the error rate of LDA model in every aspect (e.g. minimum value, maximum value, mean, median, variance).

4. (20 points) 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?
    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)`.
    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.*

```
set.seed(456)

sample_size <- c(100, 1000, 10000, 100000)

non_linear_diff_N <- data.frame(matrix(nrow = 0, ncol = 0))

for (i in seq_along(sample_size)){

    non_linear_simu <- data.frame(matrix(nrow = 0, ncol = 0))
    for (r in 1: repetition){

        N <- sample_size[i]
        sample <- data.frame(x1 = runif(N, -1, 1),
                             x2 = runif(N, -1, 1),
                             epsilon = rnorm(N, mean = 0, sd =1)) %>%
            mutate(y = x1 + x1^2 + x2 + x2^2 + epsilon,
                   class = y >= 0)

        split <- initial_split(sample, prop = .7)
```

```
        train <- training(split)
        test <- testing(split)

        lda <- lda(class ~ x1 + x1^2 + x2 + x2^2, data = train)
        lda_test_error <- error_rate(lda, test, "class")

        qda <- qda(class ~ x1 + x1^2 + x2 + x2^2, data = train)
        qda_test_error <- error_rate(qda, test, "class")

        id <- i
        result <- data.frame(id, lda_test_error, qda_test_error, N)
        non_linear_simu <- bind_rows(non_linear_simu, result)
    }
    non_linear_diff_N <- bind_rows(non_linear_diff_N, non_linear_simu)
}
```

```
non_linear_diff_N %>%
    pivot_longer(cols = ends_with("error"),
                 names_to = "model",
                 names_pattern = "(.*)_test_error",
                 values_to = "error") %>%
    ggplot(aes(x = error, fill = model)) +
    geom_density(alpha = 0.7) +
    facet_wrap(~N, scales = "free") +
    scale_fill_brewer(palette = "Set1") +
    labs(title = "Fig.4 LDA and QDA Error Rate(Non-linear Decision Boundary)",
         subtitle = "with Different Sample Size")
```



As the sample size increases, we expect that the test error rate of QDA relative to LDA would improve because the larger number of observations we have, the more flexible model we can afford to fit the data.

As shown in Fig.4, overall QDA has a lower error rate than LDA when decision boundary is non-linear. However, when we only have 100 number of observations, the distribution of error rate for both LDA and QDA overlapps substantially, but as sample size increases, the gap between the two distributions also increases. As sample size increases to 10,000, there is only a small area overlapps between them. The QDA overfitting problem becomes less severe and QDA fits better when the relationship is non-linear as our sample size increases.

Modeling voter turnout

5. (20 points) Building several classifiers and comparing output.
    a. Split the data into a training and test set (70/30).
    b. Using the training set and all important predictors, estimate the following models with `vote96` as the response variable:
        i. Logistic regression model
        ii. Linear discriminant model
        iii. Quadratic discriminant model
        iv. Naive Bayes (you can use the default hyperparameter settings)
        v. K-nearest neighbors with $K = 1, 2, \ldots, 10$ (that is, 10 separate models varying K) and *Euclidean* distance metrics
    c. Using the test set, calculate the following model performance metrics:
        i. Error rate
        ii. ROC curve(s) / Area under the curve (AUC)

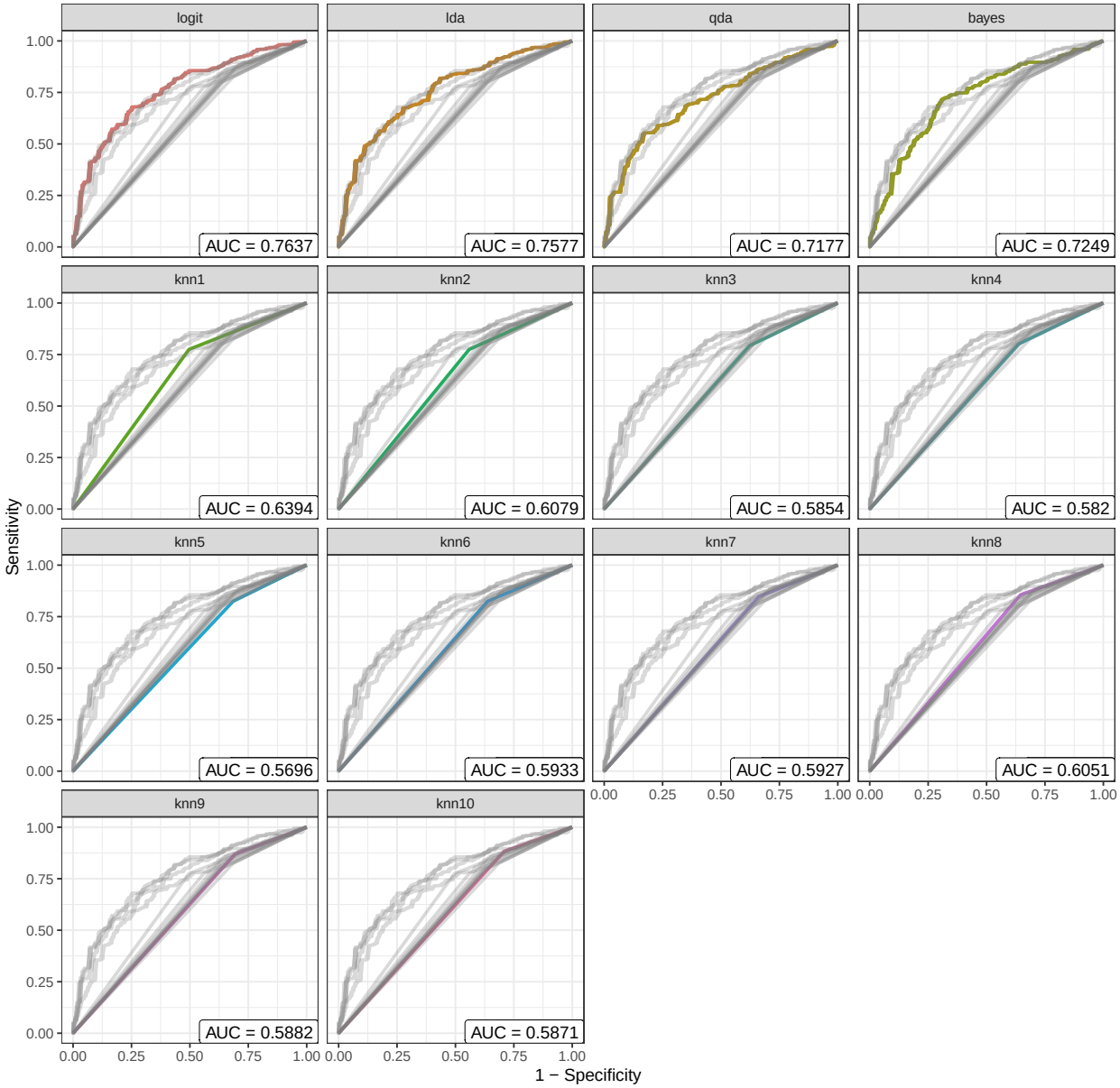
d. Which model performs the best? Be sure to define what you mean by "best" and identify supporting evidence to support your conclusion(s).

```
set.seed(102)

mental_health <- read_csv("mental_health.csv") %>%
    drop_na()
split <- initial_split(mental_health, prop = .7)
train <- training(split)
test <- testing(split)

# logistics regression
logit <- glm(vote96~., train, family = "binomial")
logit_predict <- predict(logit, test)

#lda
lda <- lda(vote96~., train)
lda_predict <- predict(lda, test)[["posterior"]][,2]

#qda
qda <- qda(vote96~., train)
qda_predict <-predict(qda, test)[["posterior"]][,2]

#naive bayes
bayes <- naiveBayes(as.factor(vote96)~., train)
bayes_predict <- predict(bayes, test, type = "raw")[,2]


#knn 1- 10
knn_mse <- tibble(k = 1:10,
                  knn_predict = map(k, ~ knn(dplyr::select(train, -vote96),
                                             test = dplyr::select(test, -vote96),
                                             cl = train$vote96, k = .)))
model_mse <- knn_mse %>%
    mutate(k = paste0("knn", k)) %>%
    `colnames<-`(c("model", "predict")) %>%
    bind_rows(tibble(model = "logit", predict = list(logit_predict))) %>%
    bind_rows(tibble(model = "lda", predict = list(lda_predict))) %>%
    bind_rows(tibble(model = "qda", predict = list(qda_predict))) %>%
    bind_rows(tibble(model = "bayes", predict = list(bayes_predict)))  %>%
    mutate(predict = map(predict, ~ as.numeric(as.character(.))))
```

```
roc_plot_prep <- data.frame(matrix(nrow = 0, ncol = 0))

for (i in 1:dim(model_mse)[1]){
    model <- model_mse$model[i]
    roc_info <- roc(test$vote96, model_mse$predict[[i]], legacy.axes = TRUE)
    roc_df <- tibble(model = model,
                     tp = roc_info$sensitivities,
                     fp = 1 - roc_info$specificities,
                     auc = as.numeric(roc_info$auc))
    roc_plot_prep <- bind_rows(roc_plot_prep, roc_df)

    model_mse$auc[i] <- as.numeric(roc_info$auc)
```

```
}

roc_plot_prep %>%
    mutate(auc = ifelse(tp ==0, auc, NA),
           model = factor(model, levels = c("logit", "lda", "qda", "bayes", paste0("knn", 1:10)))) %>%
    ggplot(aes(x = fp, y = tp, color = model)) +
    geom_line(size = 1) +
    facet_wrap(~model) +
    geom_line(data = rename(roc_plot_prep, model2 = model),
              aes(x = fp, y = tp, group  = model2), alpha = .3, size = 1, color = "grey50") +
    geom_label(aes(label = paste("AUC =", round(auc, 4))),
            x = Inf, y = -Inf, hjust = 1, vjust = 0,
            inherit.aes = FALSE) +
    theme(legend.position = "none") +
    labs(title = "Fig. 5 ROC curve and AUC of Diffteret Models",
         x = "1 - Specificity", y ="Sensitivity")
```

Fig. 5 ROC curve and AUC of Diffteret Models




```
# using 0.5 threshold
model_mse %>% mutate(predict = map(predict, ~.>0.5),
                     mse = map_dbl(predict, ~mean(test$vote96 != .))) %>%
    dplyr::select(-predict) %>% kable()
```

model	auc	mse
knn1	0.6393559	0.3237822
knn2	0.6078598	0.3467049
knn3	0.5853728	0.3581662
knn4	0.5820033	0.3581662
knn5	0.5696425	0.3610315
knn6	0.5932645	0.3438395
knn7	0.5927148	0.3381089

model	auc	mse
knn8	0.6050933	0.3266476
knn9	0.5882280	0.3352436
knn10	0.5871107	0.3323782
logit	0.7636731	0.2865330
lda	0.7577144	0.3065903
qda	0.7177059	0.3352436
bayes	0.7249060	0.3008596

The author defines that a "best" model is the one with a consistent, relatively high accuracy rate when fitting to a new dataset. That is, the "best" model does not necessarily need to have the lowest error rate but it must consistenly have a relatively good accuracy rate in a different train-test random split, or when fitting to a whole new data set. Based on Fig.5 and the MSE and AUc table reported above, logistic model performs best among the 14 models for the following reasons:

- 1. Logistics has the lowest MSE under a threshold of 0.5 as well as the highest AUC. Since we did not adopt any resampling mechanism, a low error rate under a 0.5 threshold in this particular train-test split data set is NOT reliable even though in this case, logistics model has the highest accuracy. However, it could be possible that the classes in our training sample is highly imbalanced and hence using a 0.5 threshold is not appropriate. The highest AUC scores of logistics model indicates that it has an overall better performance than any other models we have for all values of a threshold.

- 2. Regarding model consistency, since our result is solely based one training sample without any resampling method, the author is less inclined to select a non-parametric model because it is more flexible and hence more likely to overfit. For the same reason, QDA is also ruled out.

- 3. Between Logistic, LDA and Bayes, the author choose Logistic model becasue 1) we have a combination of continuous and categorical independent variable (LDA and QDA operate on normally-distributed, continuous-valued features), 2) it relies on less assumptions (i.i.d assumption in Naive Bayes).