Boseong Yun - PSET 2

Boseong Yun

2/9/2021

Overview

```
# Global code chunk settings ---------------------------------------
knitr::opts_chunk$set(message = FALSE, error = FALSE, warning = FALSE)

# Install and load packages ----------------------------------------
packages <- c(
  "tidyverse",
  "here", # used for reproducibility
  "tidymodels", # tidymodels
  "discrim", # Linear Discriminant Analysis (LDA)
  "MASS" # modeling
)

# Change to install = TRUE to install the required packages---------------------------------------
pacman::p_load(packages, character.only = TRUE, install = FALSE)
```

A Theoretical Problem

1. (25 points) In classification problems, we minimize the generalization ("test") error rate by a simple classifier that assigns each observation to the most likely class given some set of input/predictor features,

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

where x_0 is the test observation and each possible class is represented by $j \in \{1, \dots, J\}$, which in the binary context is $\{0, 1\}$. The formula above is the **Bayes classifier**, which represents the conditional probability that $Y = j$, given the observed predictor value x_0. In the binary context, the Bayes classifier corresponds to predicting $j = 1$ if $\Pr(Y = 1|X = x_0) > 0.5$, else $j = 0$.

If the Bayes decision boundary is non-linear, then, would we expect LDA or QDA (both based on the Bayes classifier) to perform better on the training set? What about on the test set?

Answer this question with a simulation exercise. That is, follow the steps below and be sure to **numerically** and **visually** present error rates for both classifiers. Use this evidence to support your answer.

Repeat (simulate) the following process 1000 times (hint: I'd consider writing a function to make your simulation simpler to run, but of course this is up to you.):

a. Create a dataset with $n = 1000$, with two input features, $X_1, X_2 \sim \text{Uniform}(-1, +1)$. Also, create a response, Y, and let it be binary defined by $f(X) = X_1 + X_1^2 + X_2 + X_2^2$, where values 0 or greater are coded `TRUE` and values less than 0 or coded `FALSE`. Note: your Y is a function of the Bayes decision boundary ($X_1 + X_1^2 + X_2 + X_2^2$, as this non-linear model defines separation between the two classes), plus some error.

```r
# Set seed
set.seed(112)

# Setting the models

# Specifying LDA model
mod_lda <- discrim_linear() %>%
  set_engine("MASS") %>%
  set_mode("classification")

# Specifying QDA Model
mod_qda <- discrim_regularized(frac_common_cov = 0) %>%
  set_engine("klaR") %>%
  set_mode("classification")

# Creating an output path for the 1000 simulations
output <- tibble(
  sim = 1:1000,
  LDA_Train = 1:1000,
  LDA_Test = 1:1000,
  QDA_Train = 1:1000,
  QDA_Test = 1:1000
) %>%
  mutate_all(as.numeric)


# Creating a for loop that repeats the simulation 1000 times
for(i in 1:1000) {

  # Saving the data frame
  df <- tibble(
    x1 = runif(n = 1000, min = -1, max = 1),
    x2 = runif(n = 1000, min = -1, max = 1),
    x1s = x1^2, # x1 squared
    x2s = x2^2, # x2 squared
    y = x1 + x2 + x1s + x2s + rnorm(n = 1000) # rnorm refers to random error
    ) %>%
    mutate(y = as.factor(ifelse(y > 0, 1, 0)))

  # Splitting the data into 80/20% training/test sets
  split_sim <- initial_split(df, prop = 0.8)

  # Assigning the splits
  train_sim <- training(split_sim)
  test_sim <- testing(split_sim)

  # LDA fit
  lda_fit <- mod_lda %>% fit(y ~., data = train_sim)

  # QDA fit
  qda_fit <- mod_qda %>% fit(y ~., data = train_sim)
```

```r
  ## Calculating each model's training and test error rate

  # training error rate for LDA
  output[i, 2] <- lda_fit %>%
    predict(new_data = train_sim) %>%
    bind_cols(train_sim) %>%
    metrics(truth = y,
            estimate = .pred_class) %>%
    mutate(error = 1 - .estimate) %>%
    .[[1, 4]] # selecting the index that matches to the error rate

  # testing error rate for LDA
  output[i, 3] <- lda_fit %>%
    predict(new_data = test_sim) %>%
    bind_cols(test_sim) %>%
    metrics(truth = y,
            estimate = .pred_class) %>%
    mutate(error = 1 - .estimate) %>%
    .[[1, 4]] # selecting the index that matches to the error rate

  # training error rate for QDA
  output[i, 4] <- qda_fit %>%
    predict(new_data = train_sim) %>%
    bind_cols(train_sim) %>%
    metrics(truth = y,
            estimate = .pred_class) %>%
    mutate(error = 1 - .estimate) %>%
    .[[1, 4]] # selecting the index that matches to the error rate

  # testing error rate for QDA
  output[i, 5] <- qda_fit %>%
    predict(new_data = test_sim) %>%
    bind_cols(test_sim) %>%
    metrics(truth = y,
            estimate = .pred_class) %>%
    mutate(error = 1 - .estimate) %>%
    .[[1, 4]] # selecting the index that matches to the error rate

}
```

e. Present results (error rates for both sets of data and both classifiers) **visually** and **numerically**

```r
# Visual representation of the results for the error rates
output %>%
  dplyr::select(-sim) %>%
  pivot_longer(1:4) %>%
  mutate(name = as.factor(name)) %>%
  ggplot(aes(x = name, y = value, fill = name)) +
  geom_boxplot() +
  labs(
    title = "The Mean Error Rates Across Models",
    x = "Model",
    y = "Error Rate",
    fill = "Model"
```

```
  ) +
  theme_minimal()
```


```
# Numeric Summary: mean, median, standard error
output %>%
  dplyr::select(-sim) %>%
  pivot_longer(everything(), names_to = "Model") %>%
  group_by(Model) %>%
  summarise(
    Mean = mean(value),
    Median = median(value),
    SE = sd(value)
  ) %>%
  knitr::kable(
   caption = "The Numerical Summary for Mean Error Rates Across Models"
  )
```

Table 1: The Numerical Summary for Mean Error Rates Across Models

Model	Mean	Median	SE
LDA_Test	0.2569799	0.2562814	0.0308697
LDA_Train	0.2530911	0.2534332	0.0154736
QDA_Test	0.2580503	0.2562814	0.0313251
QDA_Train	0.2527928	0.2534332	0.0150717

f. Offer a *few sentence discussion* after results both answering the question and discussing differences in

LDA and QDA approaches to classification in non-linear contexts like this.

Answer: I am going to answer this question by linking back to the first question that asks whether we epxect LDA or QDA (both based on the Bayes classifier) to perform better on the training set and test set when the Bayes decision boundary is non-linear.

- **First of all, I expect the QDA to perform better on the training set when the Bayes decision boundary is non-linear in most cases. The high flexibility of the QDA allows the QDA to capture non-linear relationship and thus outperform LDA. Since the true relationship is non-linear, I would also expect QDA to perform better on the test set.**
- **In this simulation, we can see that the mean training error rate for QDA is lower than the mean training error rate for LDA This is in accordance with the expectation. However, we can see that the testing error rate for QDA is slightly higher than the testing error rate for LDA. This can be due to random chance or noise. The testing error rate has higher variance than the training error rate. While the finiding is against the expectation, the QDA will perform better in the testing error rate when the sample size is large because the true relationship is non-linear.**

An Applied Problem

For this applied problem, we will return to the 2016 ANES pilot study. Using these data, you will solve the classic political problem: predict party affiliation as a function of feelings toward a variety of things, concepts, and people as well as respondents' self-reported ideologies. The theoretical assumption here is that concepts indirectly related to one's party affiliation actually drive their party affiliation. Thus, we should be able to predict party affiliation as a function of non-partisan concepts. This is certainly debatable, and thus it's a perfect task for classification.

2. Answer the following questions, taking care to *discuss results and output at a technical and substantive level throughout* (e.g., what do ROC curves tell us and why? What are their relationships to AUC? What do the functional forms of different classifiers tell us about the quality of the solutions we get? What do the patterns *substantively* mean for our goal of predicting party affiliation? And so on.)

a. Load the data.
b. Preprocess the data to:
   - keep only four feeling thermometers for major 2016 politicians (2 extreme and 2 moderate from each party: `fttrump`, `ftobama`, `fthrc`, `ftrubio`), ideology on a five point scale (`ideo5`) party id (`pid3`)
   - recode party to a dichotomous feature where `1 = democrat` and `0 = all others`
   - drop NAs
   - make the response (democrat) a factor

```
# Read file ------------------------------------------
anes_2016 <- read_csv(here("data/anes_pilot_2016.csv")) %>%
  drop_na()

# Preprocessing the data
data <- anes_2016 %>%
  dplyr::select(fttrump, ftobama, fthrc, ftrubio, ideo5, pid3) %>%
  mutate(democrat = as.factor(ifelse(pid3 == 1, 1, 0))) %>%
  dplyr::select(-pid3) %>%
  drop_na()
```

c. Set the seed, and split the data into training (0.8) and testing (0.2) sets.

```
# Setting the seed
set.seed(122233)
```

```r
# Split
split <- initial_split(data, prop = 0.8)

# Assigning training and testing data
train <- training(split)
test <- testing(split)
```

d. Fit the following classifiers using 10-fold cross-validation:
   - Logistic regression
   - Linear discriminant analysis
   - Quadratic discriminant analysis
   - K-nearest neighbors with $k = 1, 2, \ldots, 10$ (that is, 10 *separate* models varying k for each) and Euclidean distance metrics

```r
# define mod and engine: logistic
mod_logistic <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

# define mod and engine: LDA
# Source: https://discrim.tidymodels.org/reference/discrim_linear.html
mod_lda <- discrim_linear() %>%
  set_engine("MASS") %>%
  set_mode("classification")

# define mod and engine: QDA
# Source: https://www.gmudatamining.com/lesson-12-r-tutorial.html
mod_qda <- discrim_regularized(frac_common_cov = 0) %>%
  set_engine("klaR") %>%
  set_mode("classification")


# Creating the validation splits
validation_splits <- vfold_cv(v = 10, data = train)

### Fitting the classifiers ###

# Fitting Logistic Regression
logistic_res <- fit_resamples(
  mod_logistic,
  democrat ~.,
  validation_splits,
  control = control_resamples(save_pred = TRUE)
)


# Fitting LDA
lda_res <- fit_resamples(
  mod_lda,
  democrat ~.,
  validation_splits,
  control = control_resamples(save_pred = TRUE)
)
```

```r
# Fitting QDA
qda_res <- fit_resamples(
  mod_qda,
  democrat ~.,
  validation_splits,
  control = control_resamples(save_pred = TRUE)
)

# Fitting KNNs: saving the KNN models into a list

knn_res <- list()

for(i in 1:10) {

  # Setting up the model
  model_knn <- nearest_neighbor(neighbors = i, dist_power = 2) %>%
    # dist_power set to Euclidean distance metrics
    set_engine("kknn") %>%
    set_mode("classification")

  # Saving the results
  knn_res[[i]] <- fit_resamples(
      model_knn,
      democrat ~ .,
      validation_splits,
      control = control_resamples(save_pred = TRUE)
    )
}
```

e. Evaluate each model's performance using the test set. Select the best model based on the test set performance via:
   - Error rate

Answer: In this given classification problem, the error rate represents the proportion of the misclassified cases. It can alternatively be defined as $1 - Accuracy$ or as $1 - \frac{correctly\ classified}{classified\ total}$. Accuracy or the error rate (since error rate is 1 - Accuracy) is an important metric used to decide the best model. While there are other metrics to consider such as such as roc_auc to determine the model, accuracy (or the error rate) is an important metric that shows whether the model make fewer misclassfications. The following results show that the error rates are the lowest for the logstic model and LDA. The error rates are also low for QDA. The error rates for KNN models, however, are big. This can be attributed to the fact that the number of predictors is relatively large to the small sample size of the given data. This raises an important issue about how the functional forms of classfiers can influence the quality of solutions we obtain.

```r
# Creating a customized function that returns the error rate for each model
# The results are identifical to substracting accuracy rates obtained from collect_metrics
# function from 1.
find_error <- function(cv_results) {

  # Saving the confusion matrix from the cv results
  x <- cv_results %>%
    unnest(.predictions) %>%
    conf_mat(democrat, .pred_class) %>%
    .$table
```

```r
  # Summing up TP and TN (the accurately predicted ones)
  diag <- sum(diag(x))

  # Summing up the whole confusion matrix
  all <- sum(x)

  # Returning the error rate (1 - accuracy)
  return(1 - (diag/all))
}

# Saving the dataframe for the error rate
error_df <- tibble(
  Model = c("Logistic", "LDA", "QDA", paste0("KNN", 1:10)),
  Error = c(
    find_error(logistic_res),
    find_error(lda_res),
    find_error(qda_res),
    map(knn_res, find_error) %>%
        set_names(paste0("knn", 1:10)) %>%
      bind_rows()
    )
  ) %>%
  unnest(Error)

# Displaying the error_df
error_df %>%
  knitr::kable(caption = "The Mean Error Rates for Each Model")
```

Table 2: The Mean Error Rates for Each Model

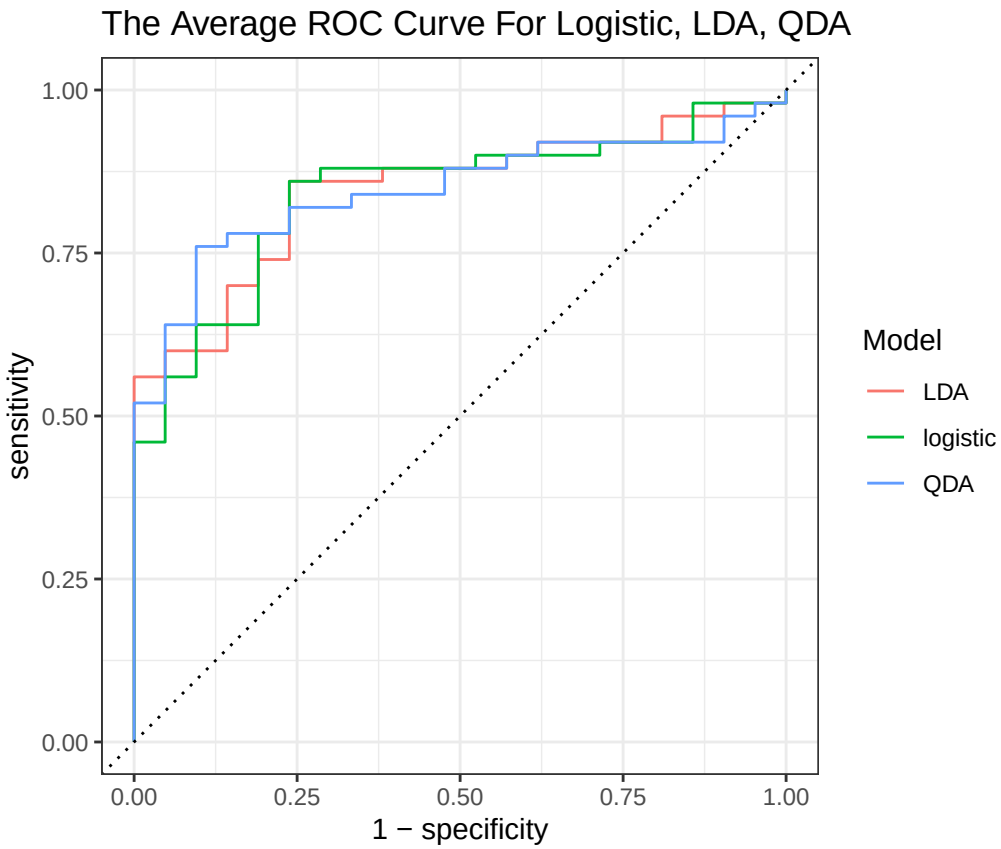
Model	Error
Logistic	0.1830986
LDA	0.1830986
QDA	0.1971831
KNN1	0.3239437
KNN2	0.3239437
KNN3	0.3239437
KNN4	0.3239437
KNN5	0.2676056
KNN6	0.2394366
KNN7	0.2394366
KNN8	0.2394366
KNN9	0.2676056
KNN10	0.2676056

- ROC curve

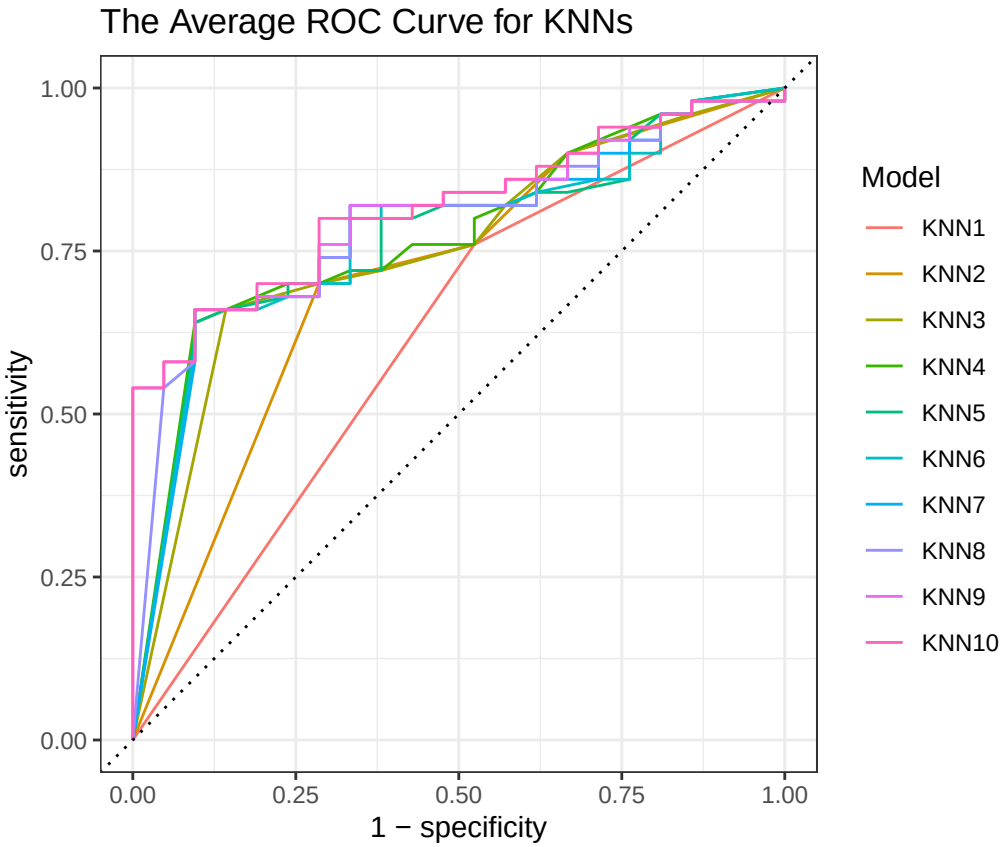
Answer: The ROC Curve (receiver operating characteristic curve) is a graph that shows the the performance of classification models at different classification threshold levels. The y-axis of the graph is the true positive rate (sensitivity) and the x-axis of the graph is the false postivie rate (1 - specificity). That is, the graph provides an easy way to assess the tradeoff between sensitivy and specifiy where the curve close to the top-left indicates better performance. That is, we can choose the most appropriate cutoff level that has high true positive rates and low

false positive rates. In this problem, the curves for both QDA and LDA models are located in the top-left corner relative to other curves and thus indicate good performance. Other models do not seem to be any better than QDA and LDA models. The ROC Curve makes it easy to visualize the performance.

```
# knn roc_curve data frame
knn_curve <- knn_res %>%
  bind_rows() %>%
  unnest(.predictions) %>%
  mutate(model = paste0("KNN", rep(1:10, each = 71))) %>%
  mutate(model = as_factor(model))

# ROC CURVE for Logistic, LDA, QDA
roc_curve1 <- logistic_res %>%
  unnest(.predictions) %>%
  mutate(model = "logistic") %>%
  bind_rows(lda_res %>%
              unnest(.predictions) %>%
              mutate(model = "LDA")) %>%
  bind_rows(qda_res %>%
              unnest(.predictions) %>%
              mutate(model = "QDA")) %>%
  group_by(model) %>%
  roc_curve(democrat, .pred_0) %>%
  autoplot() +
  labs(
    title = "The Average ROC Curve For Logistic, LDA, QDA",
    color = "Model"
  )

# Display ROC CURVE 1
roc_curve1
```




```
# ROC CURVE for KNNs
roc_curve2 <- knn_curve %>%
  group_by(model) %>%
  roc_curve(democrat, .pred_0) %>%
  autoplot() +
  labs(
    title = "The Average ROC Curve for KNNs",
    color = "Model"
  )

# Display ROC CRUVE 2
roc_curve2
```



- Area under the curve (AUC)

Answer: the Area under the curve (AUC) is intimately related to the ROC Curve. In fact, the AUC is the area under the ROC Curve. These AUC metrics makes it easy to compare the performance between the models in precise manners. That is, AUC provides a comprehensive measure of performance across every possible classification threshold. In this problem, the LDA model shows the highest AUC followed by QDA and Logistic model. As far as AUC is concerned, the LDA model performs better than QDA and Logistic model

```
# Collecting all the KNN models into a dataframe
knn_df <- knn_res %>%
  map(collect_metrics) %>%
  bind_rows() %>%
  mutate(model = paste0("KNN", rep(1:10, each = 2)))

# Saving the data frame with AUC
auc_df <- logistic_res %>%
  collect_metrics() %>%
  mutate(model = "Logistic") %>%
  bind_rows(
    collect_metrics(lda_res) %>% mutate(model = "LDA")
  ) %>%
  bind_rows(
    collect_metrics(qda_res) %>% mutate(model = "QDA")
  ) %>%
  bind_rows(knn_df) %>%
  filter(.metric == "roc_auc") %>%
  dplyr::select(model, everything())
```

```
# Displaying the results
auc_df %>%
  dplyr::select(-7) %>%
  mutate(model = str_to_upper(model)) %>%
  set_names(str_to_title(names(.))) %>%
  knitr::kable(
    caption = "The Area Under the Curve (AUC) for Different Models"
  )
```

Table 3: The Area Under the Curve (AUC) for Different Models

Model	.Metric	.Estimator	Mean	N	Std_err
LOGISTIC	roc_auc	binary	0.8629630	9	0.0393600
LDA	roc_auc	binary	0.8740741	9	0.0349284
QDA	roc_auc	binary	0.8629630	9	0.0328410
KNN1	roc_auc	binary	0.6009259	9	0.0612915
KNN2	roc_auc	binary	0.6990741	9	0.0659019
KNN3	roc_auc	binary	0.7472222	9	0.0556423
KNN4	roc_auc	binary	0.7666667	9	0.0478512
KNN5	roc_auc	binary	0.7666667	9	0.0483525
KNN6	roc_auc	binary	0.7685185	9	0.0442221
KNN7	roc_auc	binary	0.7759259	9	0.0393927
KNN8	roc_auc	binary	0.7981481	9	0.0402964
KNN9	roc_auc	binary	0.8166667	9	0.0341339
KNN10	roc_auc	binary	0.8259259	9	0.0266596

f. Once you select the best model (from your perspective)...

Answer: While there are many ways to select the best model, I have used the accuracy rates as the primary criteria for deciding the best model. The logistic model and the LDA model showed the best performance in terms of accuracy. However, the LDA model showed higher performance in regards to AUC. Taken together, I decided to use the LDA model as the best model.

- calculate your final estimate of the test error rate using the test set. In other words, take your l

```
# Selecting the best model: LDA (highest roc_auc and accuracy)

# Fitting the model using the entire training set
final_mod <- mod_lda %>%
  fit(democrat ~., data = train)
```

- calculate performance metrics using the original test set

```
# Calculating performance metrics using the original test set

# Calculating & Saving the accuracy rates
final_accuracy <- final_mod %>%
  predict(new_data = test) %>%
  bind_cols(test) %>%
  metrics(truth = democrat,
          estimate = .pred_class)
```

```r
# Calculating & Saving the roc_auc
final_auc <- final_mod %>%
  predict(new_data = test, type = "prob") %>%
  bind_cols(test) %>%
  roc_auc(democrat, .pred_0)
```

```
  - report (numerically *and* visually) and discuss results
```

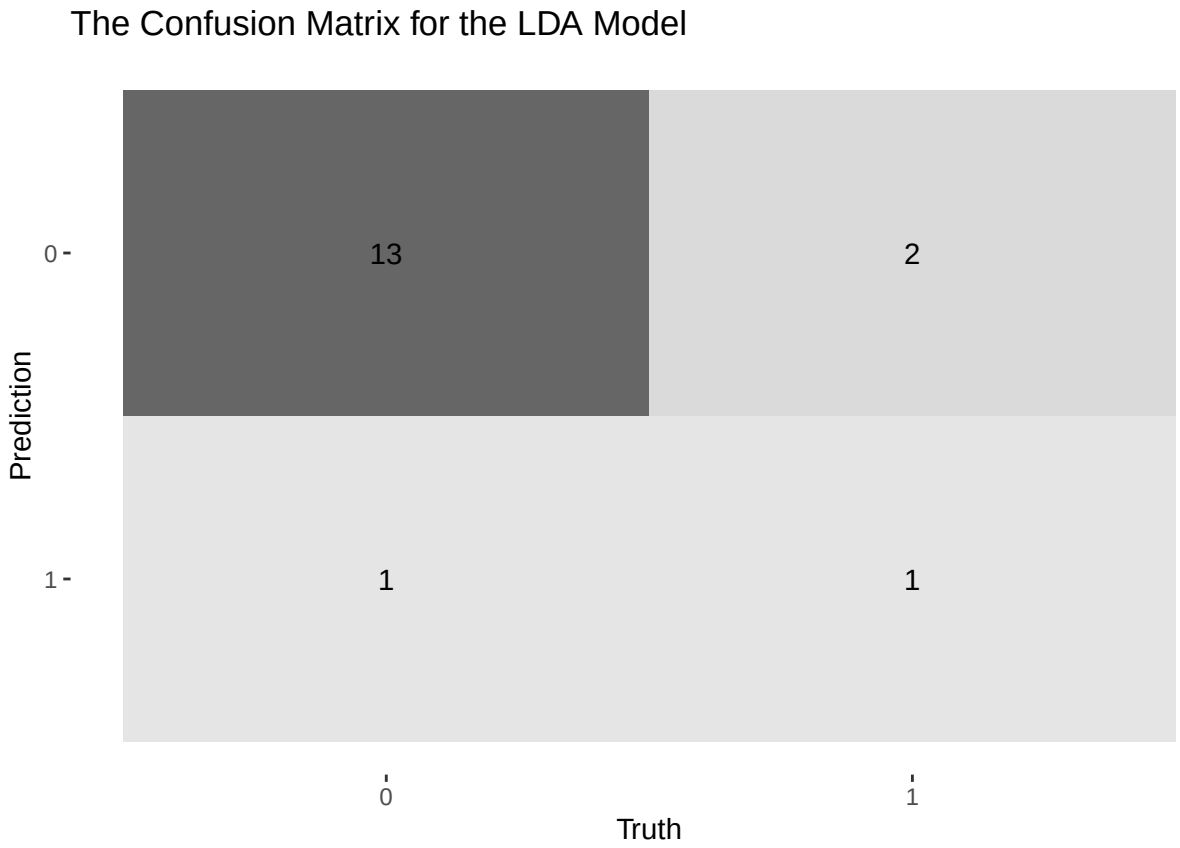
Answer: The follwing results show that the test accuracy rates are 0.832 and the ACU is 0.8333. Although there are subjective elements to understanding these figures, these statistics show that LDA is a good model. You can also verify these by looking at the confusion matrix and the ROC curve. Thinking back to our original assumption about party affiliations, we can see that party affiliation can be accurately predicted using feeling thermometers on the given data set at a a considerable degree. This is both surprising and terrifying because this implies that we can unwittingly expose our political affiliation just by showing feelings. On the technical notes, the results show that these estimates are very close to the estimates we obtained using the cross-validation. It is important to recall that we use the CV method to estimate the test error rate and select the best model among other models such as QDA, Logistic Regression, and KNNs. Although we have re-refitted the model using the entire training set and tested on the test set, we only witnessed small difference between the two different sets of estimates. This indicates that our cv method worked greatly!

```r
# Numerical Results
# Displaying the performance metrics using the original test set
final_accuracy %>%
  bind_rows(final_auc) %>%
  set_names(str_to_title(names(.)) %>% substr(2, 10)) %>%
  knitr::kable(
    caption = "The Performance Metrics For the LDA Model"
  )
```

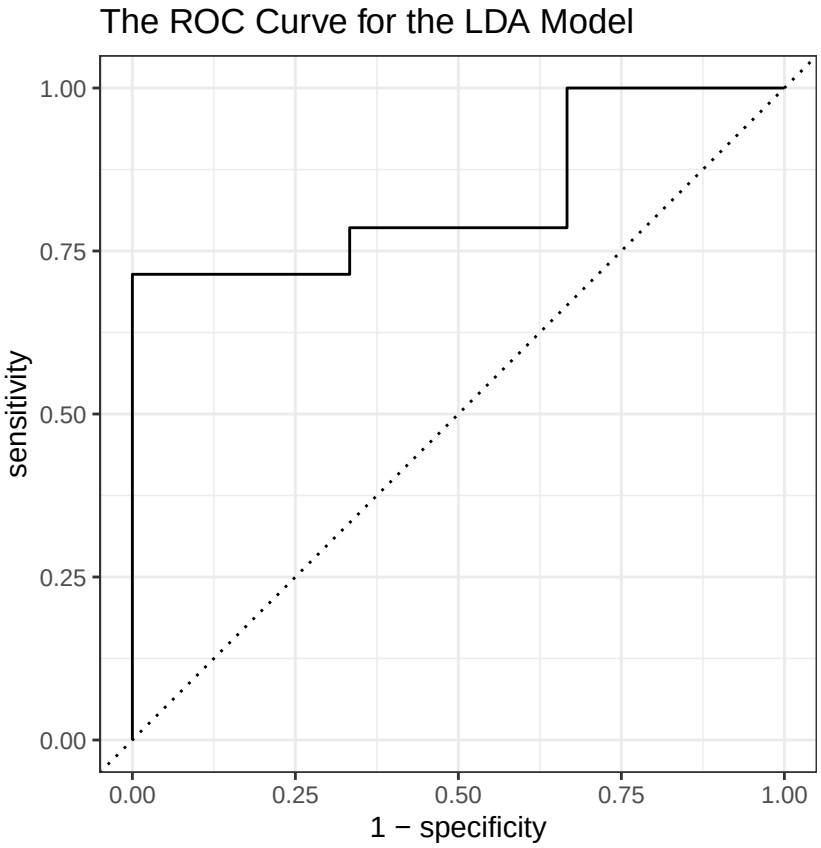
Table 4: The Performance Metrics For the LDA Model

Metric	Estimator	Estimate
accuracy	binary	0.8235294
kap	binary	0.3013699
roc_auc	binary	0.8333333

```r
# The final result
predict(final_mod, new_data = test) %>%
  bind_cols(test) %>%
  conf_mat(truth = democrat,
           estimate = .pred_class,
           dnn = c("Pred", "Truth")) %>%
  autoplot("heatmap") +
  labs(
    title = "The Confusion Matrix for the LDA Model"
  )
```




```
# Setting the type argument to prob in the prediction lets you draw a roc curve!
final_mod %>%
  predict(new_data = test, type = "prob") %>%
  bind_cols(test) %>%
  roc_curve(truth = democrat,
            estimate = .pred_0) %>%
  autoplot() +
  labs(
    title = "The ROC Curve for the LDA Model"
    )
```


The ROC Curve for the LDA Model
sensitivity
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1 − specificity