Homework 2: Classification

University of Chicago
MACS 30100: Perspectives on Computational Modeling

```
library(knitr)
```

Overview

For each of the following prompts, produce responses *with* code in-line. While you are encouraged to stage and draft your problem set solutions using any files, code, and data you'd like within the private repo for the assignment, *only the final, rendered PDF with responses and code in-line will be graded.*

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 $j \in \{1, \ldots, J\}$, 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.
b. Randomly split your data into 80/20% training/test sets, respectively.
c. Train LDA and QDA classifiers.
d. Calculate each model's training and test error rate, based on your trained model from the previous step.
e. Present results (error rates for both sets of data and both classifiers) **visually** and **numerically**
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.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(here) # for loading data; this is *optional*
```

```
## here() starts at /Users/egemenpamukcu
```

```
library(tidymodels) # for accuracy, splitting, etc.
```

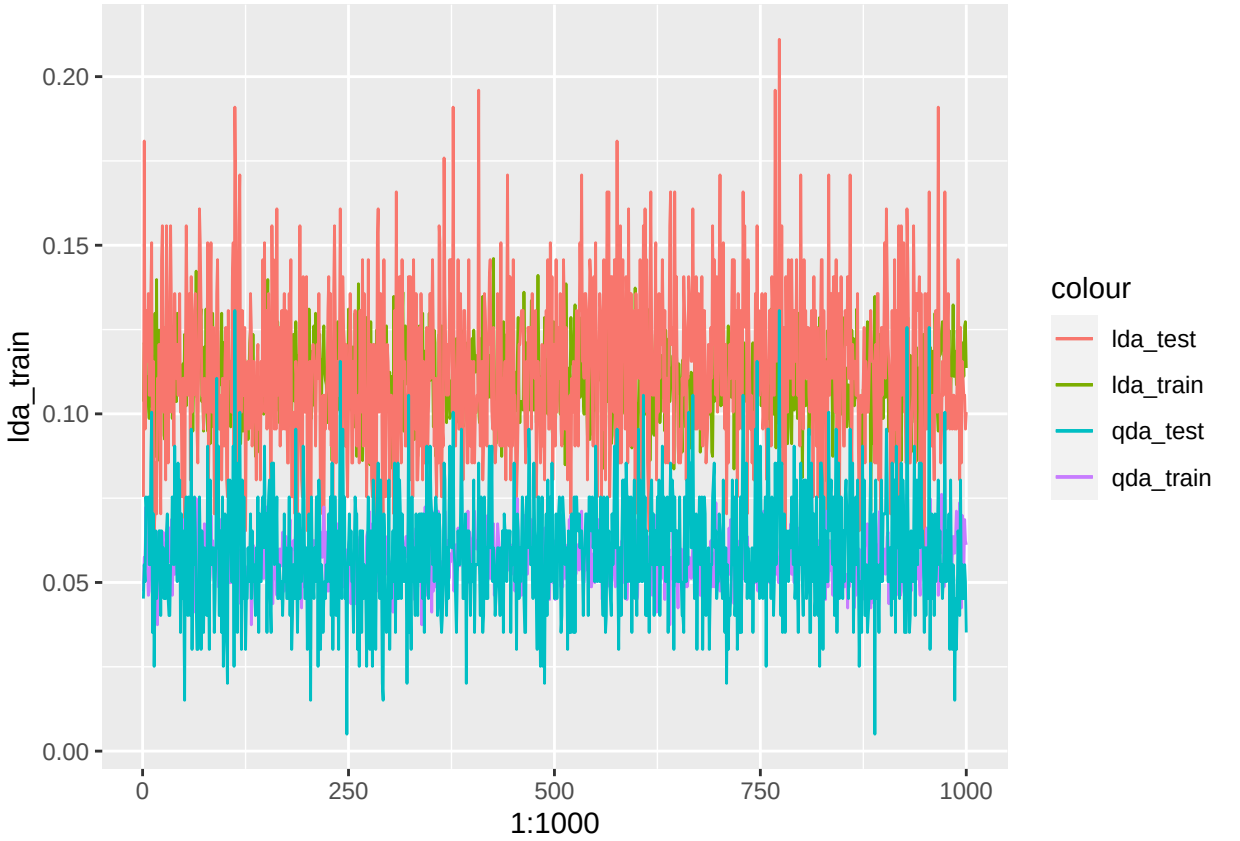
```
## -- Attaching packages --------------------------------------- tidymodels 0.1.2 --

## v broom     0.7.3      v recipes   0.1.15
## v dials     0.0.9      v rsample   0.0.8
## v infer     0.5.4      v tune      0.1.2
## v modeldata 0.1.0      v workflows 0.2.1
## v parsnip   0.1.4      v yardstick 0.0.7

## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
```

```
library(foreign) # for the (stata) data
library(class) # for knn()
library(MASS) # for lda() and qda()
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
#install.packages('klaR')
#install.packages('kknn')
library(klaR)
library(kknn)
```

```r
run1000 <- function(){
  sim_data <- tibble(x1 = runif(n=1000, min=-1, max=1), x2 = runif(n=1000, min=-1, max=1)) %>%
    mutate(y = as.factor(ifelse(x1 + x1^2 + x2 + x2^2 >= 0, TRUE, FALSE)))

  split <- initial_split(sim_data, prop=4/5)
  sim_train <- training(split)
  sim_test <- testing(split)

  lda_mod <- lda(y ~ x1 + x2,
                 data = sim_train)

  sim_train$.pred <- predict(lda_mod, sim_train)$class
  sim_test$.pred <- predict(lda_mod, sim_test)$class

  lda_train_error <- sum(sim_train$.pred != sim_train$y) / length(sim_train$y)
  lda_test_error <- sum(sim_test$.pred != sim_test$y) / length(sim_test$y)

  qda_mod <- qda(y ~ x1 + x2,
                 data = sim_train)

  sim_train$.predqda <- predict(qda_mod, sim_train)$class
  sim_test$.predqda <- predict(qda_mod, sim_test)$class

  qda_train_error <- sum(sim_train$.predqda != sim_train$y) / length(sim_train$y)
  qda_test_error <- sum(sim_test$.predqda != sim_test$y) / length(sim_test$y)

  c(lda_train_error, lda_test_error, qda_train_error, qda_test_error)
}

errors <- tibble(lda_train=numeric(), lda_test=numeric(), qda_train=numeric(), qda_test=numeric())

for (i in 1:1000){
    er <- run1000()
    errors <- errors %>%
      add_row(lda_train = er[1], lda_test = er[2], qda_train = er[3], qda_test = er[4])
}

ggplot(errors) +
  geom_line(aes(x = 1:1000, y = lda_train, col='lda_train')) +
  geom_line(aes(x = 1:1000, y = lda_test, col='lda_test')) +
  geom_line(aes(x = 1:1000, y = qda_train, col='qda_train')) +
  geom_line(aes(x = 1:1000, y = qda_test, col='qda_test'))
```




```
head(errors)
```

```
## # A tibble: 6 x 4
##   lda_train lda_test qda_train qda_test
##       <dbl>    <dbl>     <dbl>    <dbl>
## 1     0.104   0.0754    0.0474   0.0452
## 2     0.121   0.181     0.0574   0.0553
## 3     0.117   0.0955    0.0574   0.0503
## 4     0.120   0.126     0.0724   0.0754
## 5     0.104   0.131     0.0549   0.0704
## 6     0.111   0.116     0.0537   0.0704
```

QDA model has almost half the error rate of the LDA model. Because the data generating model was not linear, the linear decision boundary generated by the LDA model had a relatively higher error rate than the more complex QDA model which does not assume a common covariance for each response class. Also the testing error seems to be a lot more volatile compared to the training error for both of the models, which is expected.In short, QDA does a better job of caoturing the nonlinear relationship.

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
c. Set the seed, and split the data into training (0.8) and testing (0.2) sets.
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
e. Evaluate each model's performance using the test set. Select the best model based on the test set performance via:
    - Error rate
    - ROC curve
    - Area under the curve (AUC)
f. Once you select the best model (from your perspective)...
    - calculate your final estimate of the test error rate using the test set. In other words, take your best model and re-fit it using the entire training set (i.e. no cross-validation)
    - calculate performance metrics using the original test set
    - report (numerically *and* visually) and discuss results

```
anes <- read_csv('data/anes_pilot_2016.csv')
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   version = col_character(),
##   pid2d = col_character(),
##   pid2r = col_character(),
##   other10_open = col_character(),
##   race_other = col_character(),
##   employ_t = col_character(),
##   religpew_t = col_character(),
##   disc_fed_disc_police_rnd = col_character(),
##   white_sections_rnd = col_character(),
```

```
##   lazy_violent_rnd = col_character(),
##   FEELING_THERMOMETER_rnd = col_character(),
##   meet_rnd = col_character(),
##   givefut_rnd = col_character(),
##   info_rnd = col_character(),
##   ISSUES_OC14_rnd = col_character(),
##   disc_selfsex_rnd = col_character(),
##   lazy_col_rnd = col_character(),
##   lazy_row_rnd = col_character(),
##   violent_col_rnd = col_character(),
##   violent_row_rnd = col_character()
##   # ... with 9 more columns
## )
## i Use `spec()` for the full column specifications.
```

```
#install.packages("discrim")
library(discrim)
```

```
##
## Attaching package: 'discrim'

## The following object is masked from 'package:dials':
##
##     smoothness
```

```
anes_short <- anes%>%
  dplyr::select(fttrump, ftobama, fthrc, ftrubio, ideo5, pid3) %>%
  mutate(democrat = as.factor(ifelse(pid3 == 1, 1, 0))) %>%
  na.omit() %>%
  dplyr::select(-c(pid3))

convert_na <- function(x) {
  b <- mutate(.data = anes_short, x = replace(x, x > 100, NA))
  return(b$x)
}

anes_short <- anes_short %>%
  mutate_all(funs(convert_na(.))) %>%
  drop_na()
```

```
## Warning: `funs()` is deprecated as of dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## Warning: Problem with `mutate()` input `democrat`.
## i Problem with `mutate()` input `x`.
## i '>' not meaningful for factors
## i Input `x` is `replace(x, x > 100, NA)`.
## i Input `democrat` is `convert_na(democrat)`.

## Warning: Problem with `mutate()` input `x`.
## i '>' not meaningful for factors
## i Input `x` is `replace(x, x > 100, NA)`.

## Warning: Problem with `mutate()` input `democrat`.
## i '>' not meaningful for factors
## i Input `democrat` is `convert_na(democrat)`.

## Warning in Ops.factor(x, 100): '>' not meaningful for factors
```

```r
set.seed(1234)

split <- initial_split(data = anes_short, prop = 4/5)
anes_train <- training(split)
anes_test <- testing(split)

cv_train <- vfold_cv(anes_train,
                     v = 10)

recipe <- recipe(democrat ~ .,
                 data = anes_short)

logmod <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

ldamod <- discrim_linear() %>%
  set_engine("MASS") %>%
  set_mode("classification") %>%
  translate()

qdamod <- discrim_regularized(frac_common_cov = 0) %>%
  set_engine("klaR") %>%
  set_mode('classification')

knn_mod1 <- nearest_neighbor(neighbors = 1) %>%
  set_engine("kknn") %>%
  set_mode("classification")

knn_mod2 <- nearest_neighbor(neighbors = 2) %>%
  set_engine("kknn") %>%
  set_mode("classification")

knn_mod3 <- nearest_neighbor(neighbors = 3) %>%
  set_engine("kknn") %>%
  set_mode("classification")
```

```
knn_mod4 <- nearest_neighbor(neighbors = 4) %>%
  set_engine("kknn") %>%
  set_mode("classification")

knn_mod5 <- nearest_neighbor(neighbors = 5) %>%
  set_engine("kknn") %>%
  set_mode("classification")

knn_mod6 <- nearest_neighbor(neighbors = 6) %>%
  set_engine("kknn") %>%
  set_mode("classification")

knn_mod7 <- nearest_neighbor(neighbors = 7) %>%
  set_engine("kknn") %>%
  set_mode("classification")

knn_mod8 <- nearest_neighbor(neighbors = 8) %>%
  set_engine("kknn") %>%
  set_mode("classification")

knn_mod9 <- nearest_neighbor(neighbors = 9) %>%
  set_engine("kknn") %>%
  set_mode("classification")

knn_mod10 <- nearest_neighbor(neighbors = 10) %>%
  set_engine("kknn") %>%
  set_mode("classification")

compute_metrics_auc <- function(mod){
  workflow <- workflow() %>%
    add_recipe(recipe) %>%
    add_model(mod)
  res <- workflow %>%
    fit_resamples(resamples = cv_train,
                  metrics = metric_set(roc_auc))
  final <- res %>%
    select_best(metric = "roc_auc")
  workflow <- workflow %>%
    finalize_workflow(final)
  final_mod <- workflow %>%
    last_fit(split)
  final_mod$.metrics
}

compute_metrics_accuracy <- function(mod){
  workflow <- workflow() %>%
    add_recipe(recipe) %>%
    add_model(mod)
  res <- workflow %>%
    fit_resamples(resamples = cv_train,
                  metrics = metric_set(accuracy))
  final_mod <- res %>%
    select_best(metric = "accuracy")
```

```
  workflow <- workflow %>%
    finalize_workflow(final_mod)
  final_mod <- workflow %>%
    last_fit(split)
  x <- final_mod$.metrics
}
```

```
compute_metrics_auc(logmod)
```

```
##
## Attaching package: 'rlang'

## The following objects are masked from 'package:purrr':
##
##     %@%, as_function, flatten, flatten_chr, flatten_dbl, flatten_int,
##     flatten_lgl, flatten_raw, invoke, list_along, modify, prepend,
##     splice

##
## Attaching package: 'vctrs'

## The following object is masked from 'package:dplyr':
##
##     data_frame

## The following object is masked from 'package:tibble':
##
##     data_frame

## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.831 Preprocessor1_Model1
## 2 roc_auc  binary         0.889 Preprocessor1_Model1
```

```
compute_metrics_auc(ldamod)
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.818 Preprocessor1_Model1
## 2 roc_auc  binary         0.888 Preprocessor1_Model1
```

```
compute_metrics_auc(qdamod)
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.809 Preprocessor1_Model1
## 2 roc_auc  binary         0.886 Preprocessor1_Model1
```

```
compute_metrics_auc(knn_mod1)
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.733 Preprocessor1_Model1
## 2 roc_auc  binary         0.714 Preprocessor1_Model1
```

```
compute_metrics_auc(knn_mod2)
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.733 Preprocessor1_Model1
## 2 roc_auc  binary         0.795 Preprocessor1_Model1
```

```
compute_metrics_auc(knn_mod3)
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.733 Preprocessor1_Model1
## 2 roc_auc  binary         0.827 Preprocessor1_Model1
```

```
compute_metrics_auc(knn_mod4)
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.733 Preprocessor1_Model1
## 2 roc_auc  binary         0.835 Preprocessor1_Model1
```

```
compute_metrics_auc(knn_mod5)
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.763 Preprocessor1_Model1
## 2 roc_auc  binary         0.842 Preprocessor1_Model1
```

```
compute_metrics_auc(knn_mod6)
```

```
## [[1]]
## # A tibble: 2 x 4
```

```
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.780 Preprocessor1_Model1
## 2 roc_auc  binary         0.850 Preprocessor1_Model1
```

```
compute_metrics_auc(knn_mod7)
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.780 Preprocessor1_Model1
## 2 roc_auc  binary         0.858 Preprocessor1_Model1
```

```
compute_metrics_auc(knn_mod8)
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.805 Preprocessor1_Model1
## 2 roc_auc  binary         0.865 Preprocessor1_Model1
```

```
compute_metrics_auc(knn_mod9)
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.818 Preprocessor1_Model1
## 2 roc_auc  binary         0.871 Preprocessor1_Model1
```

```
compute_metrics_auc(knn_mod10)
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.818 Preprocessor1_Model1
## 2 roc_auc  binary         0.873 Preprocessor1_Model1
```

```
compute_metrics_accuracy(logmod)
compute_metrics_accuracy(ldamod)
compute_metrics_accuracy(qdamod)
compute_metrics_accuracy(knn_mod1)
compute_metrics_accuracy(knn_mod2)
compute_metrics_accuracy(knn_mod3)
compute_metrics_accuracy(knn_mod4)
compute_metrics_accuracy(knn_mod5)
compute_metrics_accuracy(knn_mod6)
```

```
compute_metrics_accuracy(knn_mod7)
compute_metrics_accuracy(knn_mod8)
compute_metrics_accuracy(knn_mod9)
compute_metrics_accuracy(knn_mod10)
```

Best performing model, in terms of both AUC_ROC and accuracy seems to be the logistic regression model. Usually models seem to be doing worse as the model complexity increases. The LDA performed slightly better than the more complex QDA model. KNN models also performed the best when K=10 and worst when K=1. The area under the ROC curve shrunk as model complexity increased. Which means that model started to do a worse job in identifying positives (true positive rate) or minimizing false positive rate or perhaps both as the complexity rises. The best performing mode, logistic regression, has an accuracy rate of ~83% which means that it classified the participants (observations) in the test set with the right party identification–guessing whether they are a democrat or not right 83% of the time.

```
#best performer is the logistic regression
logit <- logmod %>%
  fit(democrat ~ .,
      data = anes_train)

logit %>%
  predict(anes_test) %>%
  bind_cols(anes_test) %>%
  metrics(truth = democrat,
          estimate = .pred_class)
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.831
## 2 kap      binary         0.641
```

```
logit
```

```
## parsnip model object
##
## Fit time:  6ms
##
## Call:  stats::glm(formula = democrat ~ ., family = stats::binomial,
##     data = data)
##
## Coefficients:
## (Intercept)      fttrump      ftobama        fthrc      ftrubio        ideo5
##   -1.589427    -0.010152     0.020157     0.025187    -0.007489    -0.218310
##
## Degrees of Freedom: 945 Total (i.e. Null);  940 Residual
## Null Deviance:       1263
## Residual Deviance: 755.2     AIC: 767.2
```

```
logit %>%
  predict(anes_test) %>%
  mutate(model = "logit",
         truth = anes_test$democrat) %>%
```

```
mutate(correct = if_else(.pred_class == truth, "Yes", "No")) %>%
ggplot() +
geom_bar(alpha = 0.8, aes(correct, fill = correct)) +
labs(x = "Correct?",
     y = "Count",
     fill = "Correctly\nClassified") +
theme_minimal()
```