Statistical Machine Learning Final Project

John Lain

2024-02-22

Figure 1: United States Capitol

I use the following packages in my project:

```
options(scipen = 999)
library(tidyverse)
library(tidymodels)
library(ggplot2)
library(corrplot)
library(discrim)
library(ranger)
library(xgboost)
library(vip)
```

Setting a seed:

```
set.seed(2222)
```

Introduction

Congressional Districts are 435 regions (and an additional region for Washington DC) in the United States which each have a population that votes for a single congressional representative. This representative becomes a member of the United States House of Representatives. Every 10 years, redistricting occurs which redefines these regions. The goal when designing a region is to appoint a number of representatives to each state which is proportional to the relative population of that state compared to the US as a whole. However, each state is also guaranteed a single representative. The United States has two major political parties which run candidates in each of these districts, the Democratic Party and the Republican Party, as well as some other third parties that also participate. Every two years, a representative from each region is chosen in an election.

My goal in this project is to predict the party of United States House of Representatives 118th Congress representatives, using demographic information about the representative's congressional district. The 118th Congress was electing in the 2022 midterm elections, began serving in 2023, and will end their service in 2025, after the 2024 elections. A secondary goal is to explore what variables are most important when predicting the party of representatives.

Introduction to the Data

My data set is a collection of economic and demographic population data from 2022 for United States congressional districts, as well as the party of the representative for each district in the 118th Congress. The population data comes from the Unites States Census Bureau's data website: https://data.census.gov/. The data was collected from the American Community Survey, sorted by congressional district, and it is specifically from S1811, the 2022 5-year Estimates Selected Economic Characteristics table and B02001, the 2022 5-year Estimates Race table. I collected the party of the representatives from the 118th Congress Wikipedia page, https://en.wikipedia.org/wiki/118th_United_States_Congress.

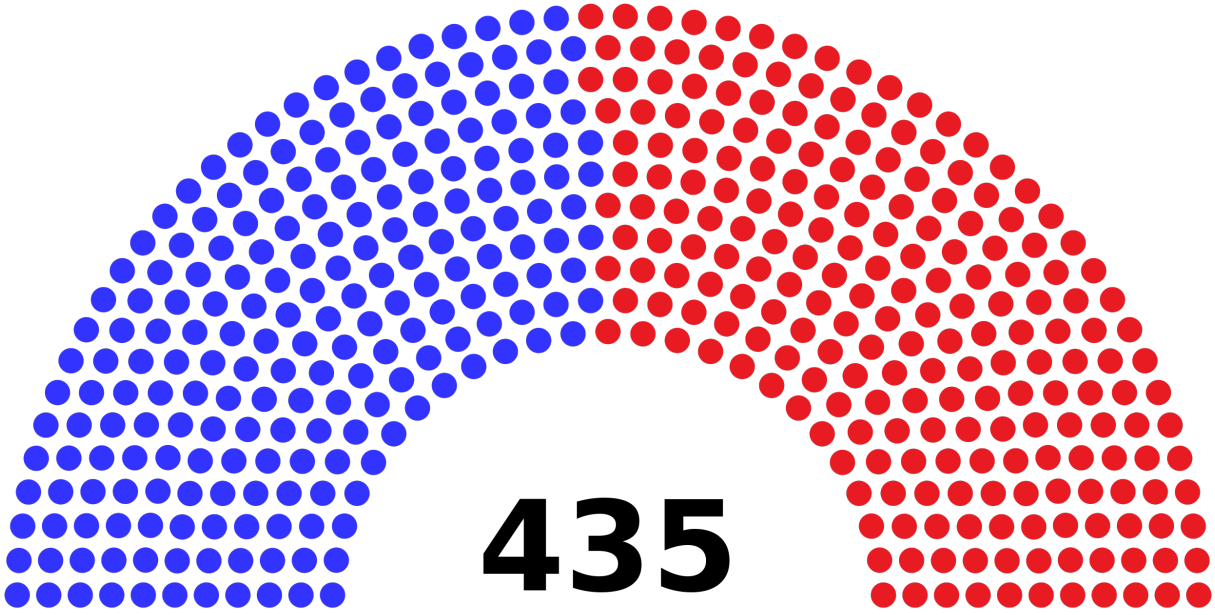


Figure 2: A graphic from the 118th US Congress Wikipedia showing representatives from each party; blue shows Democrats and red shows Republicans.

I first cleaned the data in excel, such as changing variable names for readability and creating the column for the party of the representative, and created a csv:

```
CongressData = read_csv("FinalData118thCongress.csv")
```

```
colnames(CongressData)[1] <- "District"
```

My data was formatted as strings with commas and percentage signs, so I converted it to numeric values for analysis:

```
CongressData[3:62] <- data.frame(sapply(CongressData[3:62], function(x) as.numeric(gsub("%", "", x))))
```

```
CongressData$Party = factor(CongressData$Party)
```

```
dim(CongressData)
```

```
## [1] 436  62
```

This data set includes one observation for each congressional district, including Washington DC, for a total of 436 observations, as we can see in the dimensions. There are 61 variables, as well as one column for the congressional district name which is not used in my models. Some of the variables are numerical values that count populations. Others are percentages which group population, for example, the percentage of the total population of the district that is in a given income bracket. Also, my data has a single categorical variable which is the party of the congressional representative, and takes the values "D" for democrat or "R" for republican (No independents/third party members are in the U.S House of Representative 118th Congress).

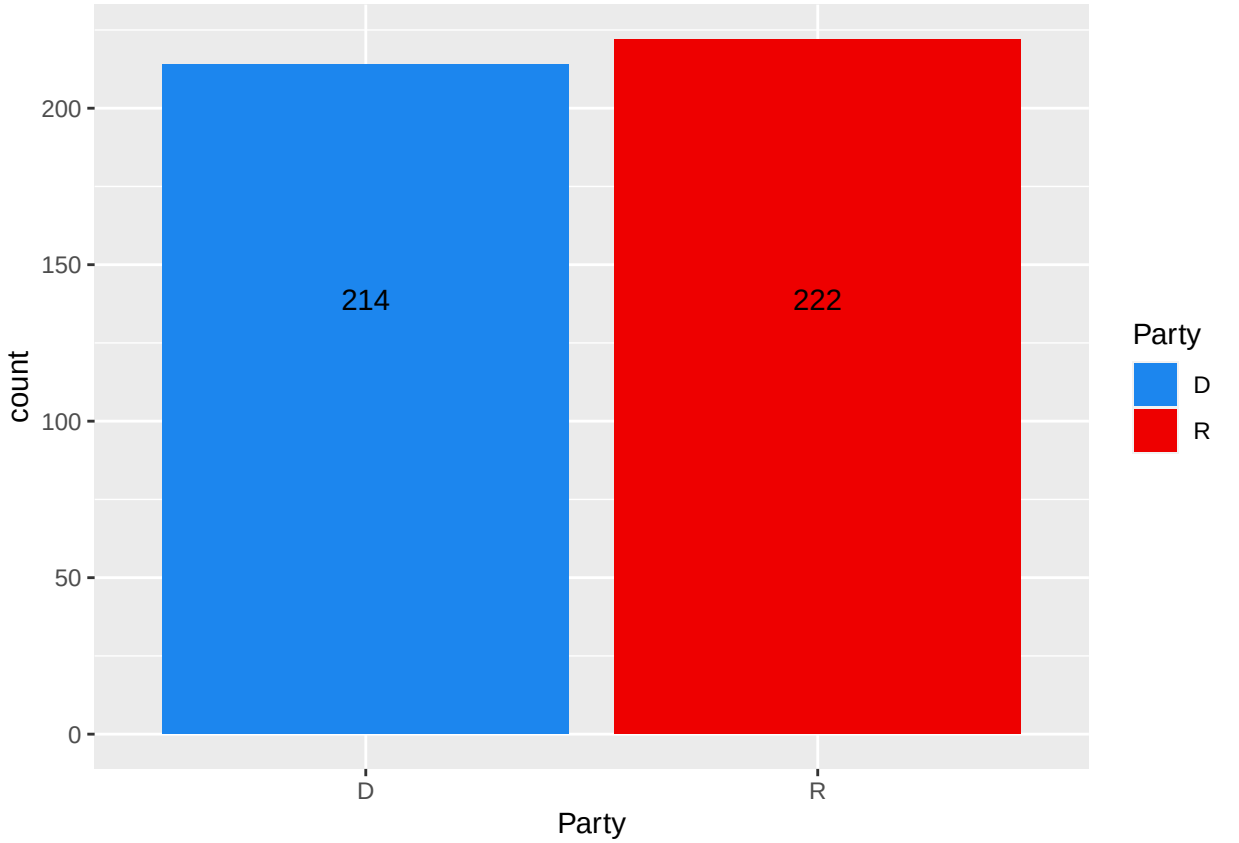
Using the following function, we can see that no data is missing, as the following output would be non-zero if any data was `NA`:

```
sum(colSums(is.na(CongressData)))
```

```
## [1] 0
```

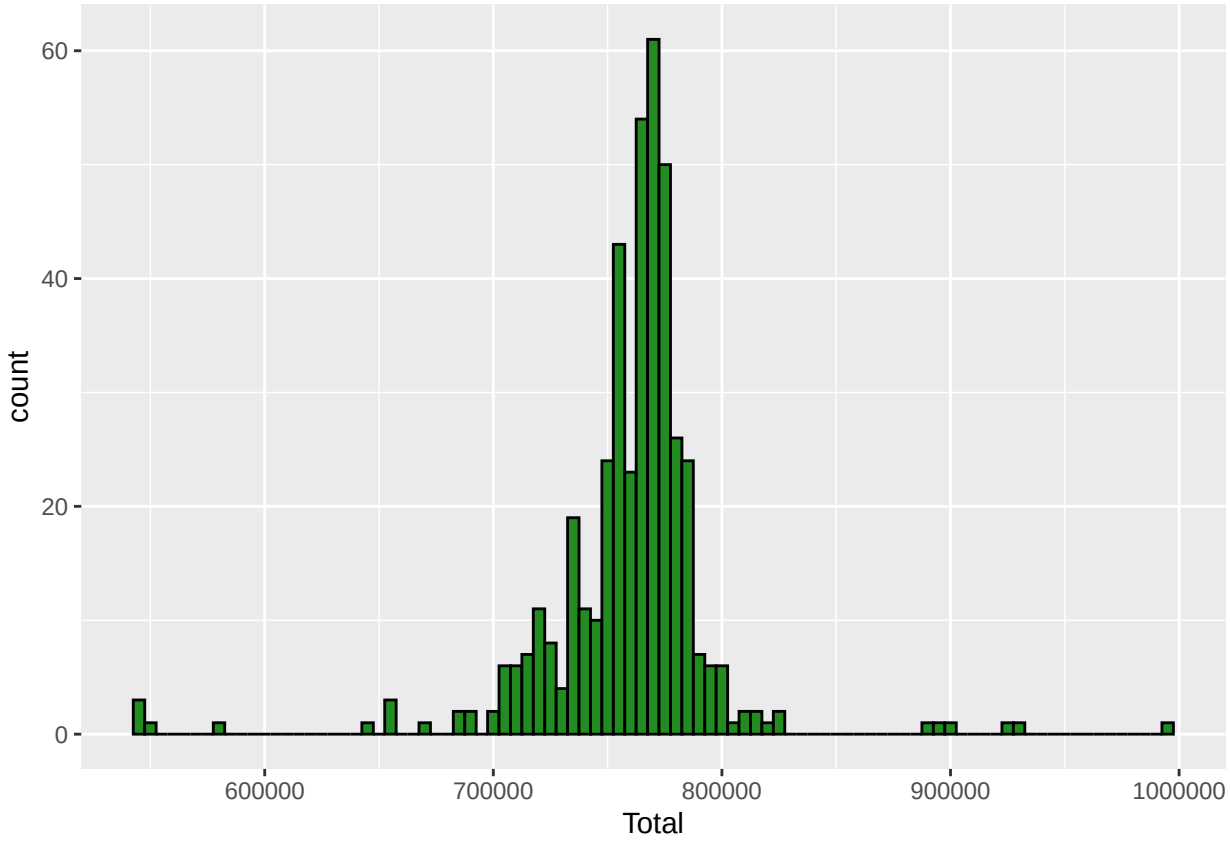
Exploratory Data Analysis (EDA)

First, I want to explore the proportion of representatives from each party:



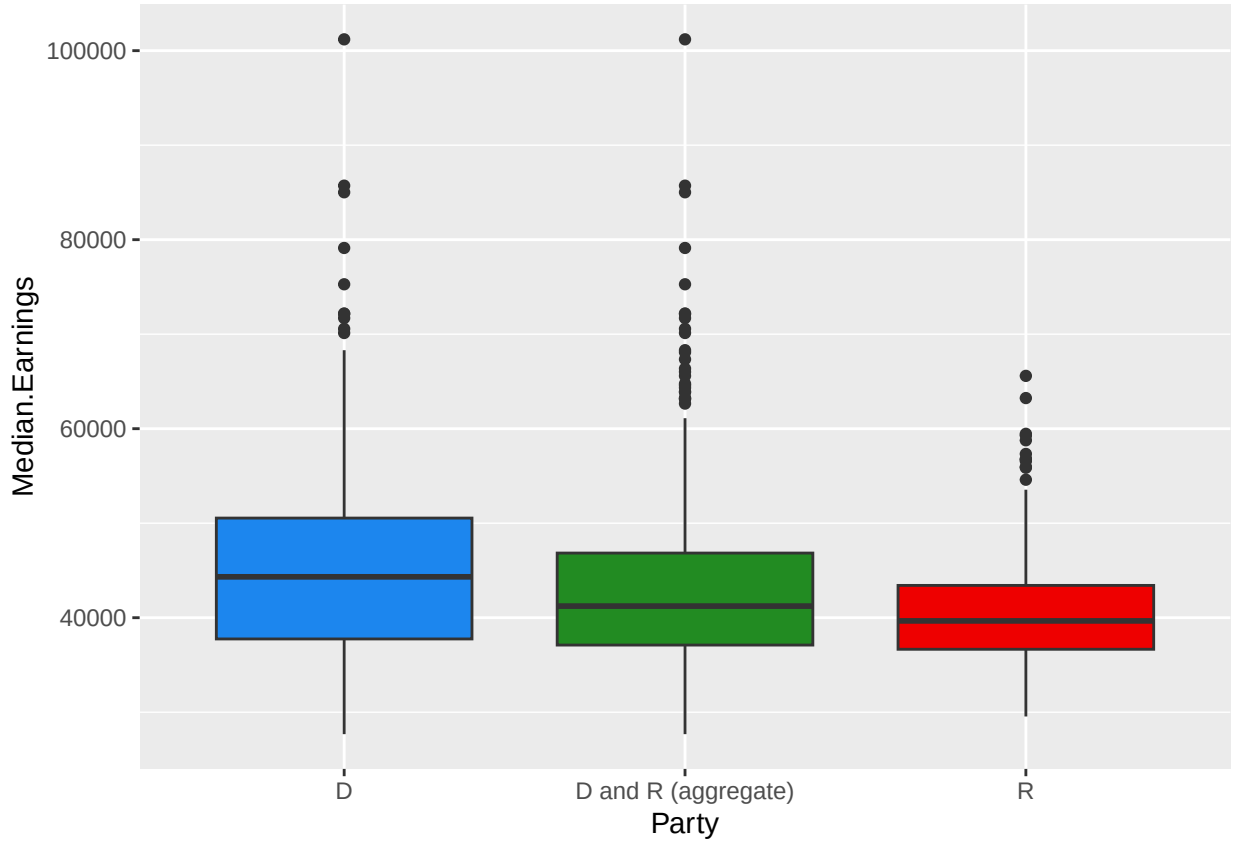
From these bar charts we can see that the Republican party had slightly more seats in the House of Representatives than the Democrats after the 2022 midterms.

Then, we explore the distribution of total population in the congressional districts:



Districts are zoned as to have a similar population in each district, and the graph above shows that most districts have between 700,000 to 800,000 population. The distribution of population that appears Gaussian, or possibly left skewed.

We can also explore the median earnings of the districts, categorized by party representative:



The above box plots show that districts with a democratic representative have higher average median earnings, while also having a larger spread of earnings than districts with a republican representative. The districts with high, outlier earnings also mostly have a democrat representative.

The top 10 highest earning districts (and then bottom 10) are as follows:

```
##         District Median.Earnings Party
## 1    12.New.York          101197     D
## 2  17.California           85716     D
## 3  11.California           85014     D
## 4  16.California           79122     D
## 5    atLarge..DC           75280     D
## 6    10.New.York           72188     D
## 7  10.California           72151     D
## 8    11.Virginia           71704     D
## 9     8.Virginia           70567     D
## 10 36.California           70125     D
```

```
##         District Median.Earnings Party
## 1      7.Arizona           32127     D
## 2       29.Texas           31948     D
## 3   2.New.Mexico           31836     D
## 4     5.Kentucky           31444     R
## 5      7.Alabama           31414     D
## 6       16.Texas           30495     D
## 7       15.Texas           30324     R
```

```
## 8  2.Mississippi           30322     D
## 9  22.California           29548     R
## 10      34.Texas           27676     D
```

The top 10 highest earning districts are all represented by Democrats. The bottom 10 districts in terms of earning are represented by a mix, with 7 being represented by Democrats and 3 being represented by Republicans

Data Splitting

I chose to split my data with 70% for training and 30% for testing. With only 436 observations, I needed to find a balance of training my model with an adequate amount of observations, while still leaving data leftover for testing.

I also set up 5 folds, for a 5-fold cross-validation, in order to predict test MSE.

```
congress_split <- initial_split(CongressData, prop = 0.70,
                                strata = Party)
congress_train <- training(congress_split)
congress_test <- testing(congress_split)

congress_fold <- vfold_cv(congress_train, v = 5, strata = Party)
```

I used stratified sampling for both the split into testing and training data, as well as for my folds, with the Party variable being the strata.

Preping and Baking Recipe

Because all of my predictors are numeric, I did not need to dummy code. I did have to remove the `District` variable, as each observation is unique and I only am using it for bookkeeping. I also chose to center and scale the features.

```
congress_recipe <-
  recipe(Party ~ ., data = congress_train)%>%
  step_rm(District) %>%
  step_center()%>%
  step_scale()
```

Prepping and Baking my recipe:

```
prep(congress_recipe) %>%
  bake(new_data = congress_train)
```

```
## # A tibble: 304 x 61
##     Total White.alone Black.or.African.Amer~1 American.Indian.and.~2 Asian.alone
##     <dbl>       <dbl>                   <dbl>                  <dbl>       <dbl>
##  1 726993      276830                  409862                   1570        8868
##  2 734821      450472                   23395                 104957       47464
##  3 801287      538066                   46599                  20266       43619
##  4 806472      434731                   30676                  33943       14805
##  5 763623      560364                   12640                  15452       31851
```

```
##  6 755594      422384                  72260                  6113       93159
##  7 753515      307553                  77961                  5299      168759
##  8 752780      252570                 111708                  7083      135383
##  9 762122      329498                  52899                  7397      126159
## 10 755993      408067                  40661                  3828      167867
## # i 294 more rows
## # i abbreviated names: 1: Black.or.African.American.alone,
## #   2: American.Indian.and.Alaska.Native.alone
## # i 56 more variables: Native.Hawaiian.and.Other.Pacific.Islander.alone <dbl>,
## #   Some.Other.Race.alone <dbl>, Two.or.More.Races. <dbl>,
## #   Two.races.including.Some.Other.Race <dbl>,
## #   Two.races.excluding.Some.Other.Race.and.three.or.more.races <dbl>, ...
```

The goal of this project is to predict the Party of the representative, so it is a classification problem. I chose to use a KNN model, an elastic net logistic regression, a random forest, and a boosted tree.

K-Nearest Neighbors

The first model I chose to use is K-Nearest Neighbors.

First, I set up my model and workflow. I use `tune()` in place of a specific neighbor parameter in order to find the best parameter by tuning with my folds.

```
knn_model <- nearest_neighbor(neighbors = tune()) %>%
  set_engine("kknn") %>%
  set_mode("classification")
```

```
knn_workflow <- workflow() %>%
  add_model(knn_model) %>%
  add_recipe(congress_recipe)
```

Then, I define a grid of values for neighbors that I can use to select the best hyper-parameter value. I chose a grind of 1 to 25 neighbors in order to fit a large number of models.

```
knn_grid<- grid_regular(neighbors(range = c(1,25)),levels = 25)
```

Next, I fit a set of models with the values of neighbors in the grid I defined. I will use the folds in my training data in order to retrieve an estimate for the testing MSE for each model.

```
tune_knn <- tune_grid(
knn_workflow,
resamples = congress_fold,
grid = knn_grid
)
```

I chose to save the resulting models so that my code would not need to be re-run each time I knit it.

```
save(tune_knn, file = "tune_knn.rda")
```

Loading my data into the project:

```
load("tune_knn.rda")
```

Elastic Net Logistic Regression

The next model I chose to use is a Elastic Net Logistic Regression.

First, I set up my model and workflow. I use `tune()` in place of mixture and penalty hyper-parameters in order to find the best combination of values by fitting to the data folds.

```
en_model<- multinom_reg(mixture = tune(),
                                penalty = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")
```

```
en_workflow <- workflow() %>%
  add_model(en_model) %>%
  add_recipe(congress_recipe)
```

Then, I define a grid of values for mixture and range that I can use to select the best hyper-parameter.

```
en_grid <- grid_regular(penalty(range = c(0.01, 3),
  trans = identity_trans()),mixture(range = c(0, 1)),levels = 10)
```

I then fit my model to all combinations of values in the ranges for mixture and penalty:

```
tune_en <- tune_grid(
  en_workflow,
  resamples = congress_fold,
  grid = en_grid
)
```

Like with the KNN, I saved my models and now load them:

```
save(tune_en, file = "tune_en.rda")
```

```
load("tune_en.rda")
```

Random Forest

I setup my random forest model and workflow:

```
rf_model <- rand_forest(mtry = tune(),
                            trees = tune(),
                            min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

rf_workflow <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(congress_recipe)
```

Then, I define a grid of values for each of my hyper-parameters:

```
rf_grid <- grid_regular(mtry(range = c(5, 15)),
                        trees(range = c(200, 600)),
                        min_n(range = c(10, 20)),
                        levels = 10)
```

Then, I fit my model to each combination of hyper-parameters:

```
tune_rf <- tune_grid(
  rf_workflow,
  resamples = congress_fold,
  grid = rf_grid
)
```

I saved my models and now load them.

```
save(tune_rf, file = "tune_rf.rda")
```

```
load("tune_rf.rda")
```

Boosted Tree

I setup my boosted tree model and workflow:

```
bt_model<- boost_tree(mtry = tune(),
                          trees = tune(),
                          learn_rate = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

bt_workflow <- workflow() %>%
  add_model(bt_model) %>%
  add_recipe(congress_recipe)
```

Then, I define a grid of values for each of the hyper-parameters:

```
bt_grid <- grid_regular(mtry(range = c(1, 10)),
                        trees(range = c(200, 600)),
                        learn_rate(range = c(-10, -1)),
                        levels = 5)
```

Then, I fit my model to each combination of hyper-parameters:

```
tune_bt <- tune_grid(
  bt_workflow,
  resamples = congress_fold,
  grid = bt_grid
)
```

Saving and loading models:

```
save(tune_bt, file = "tune_bt.rda")
```

```
load("tune_bt.rda")
```

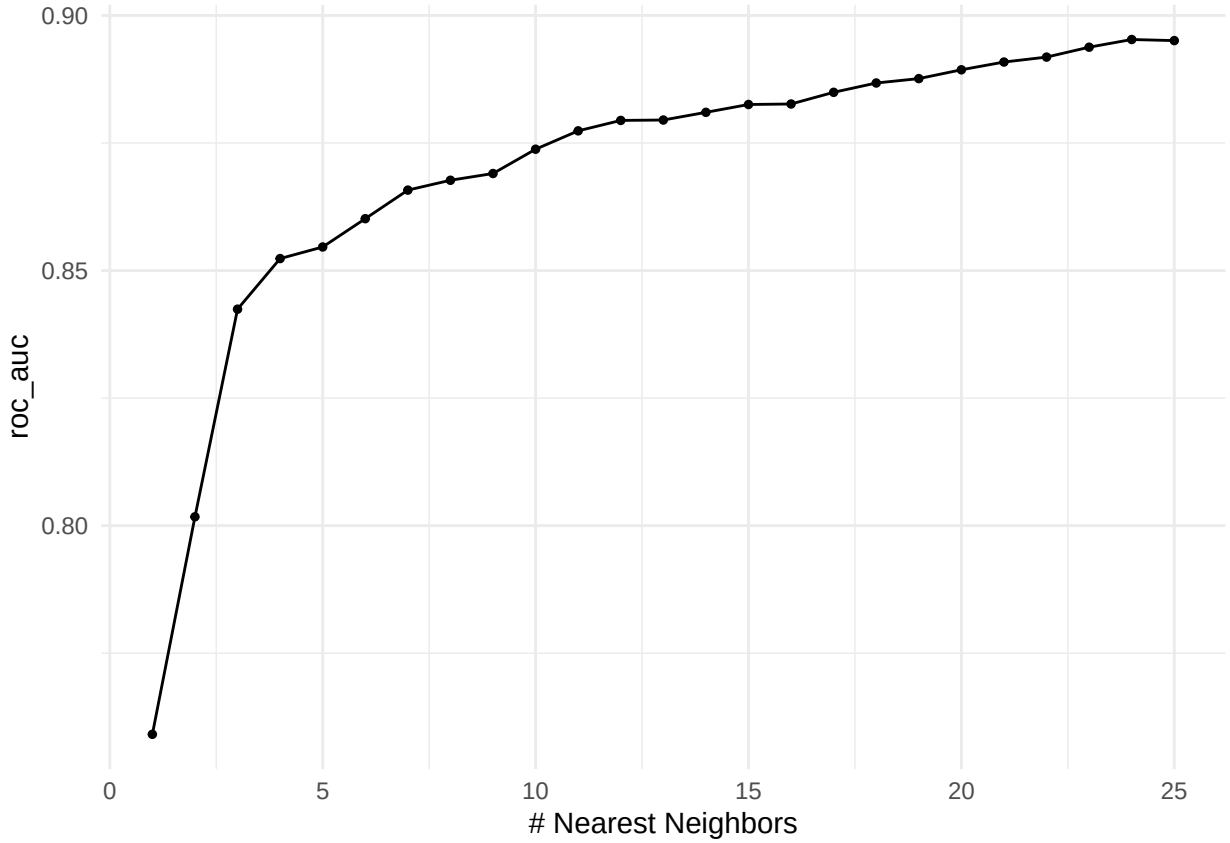
Model Performance:

I will use area under the ROC curve as the metric for model performance. A larger estimate for the testing ROC AUC indicates a better model.

The following plots show the estimate for the test ROC AUC of each model.

KNN Performance:

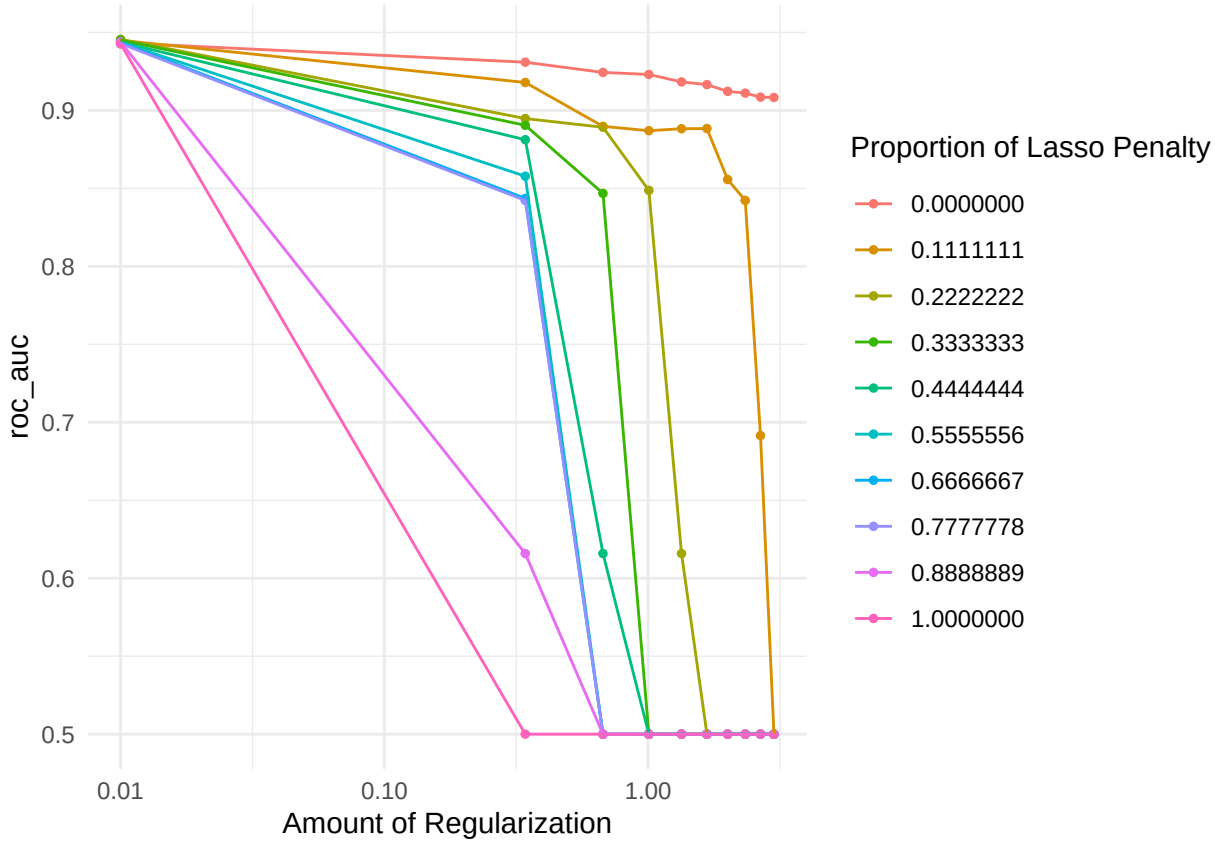
```
autoplot(tune_knn, metric = "roc_auc") + theme_minimal()
```



As the number of neighbors increases the model performs better, until their are 24 neighbors being used.

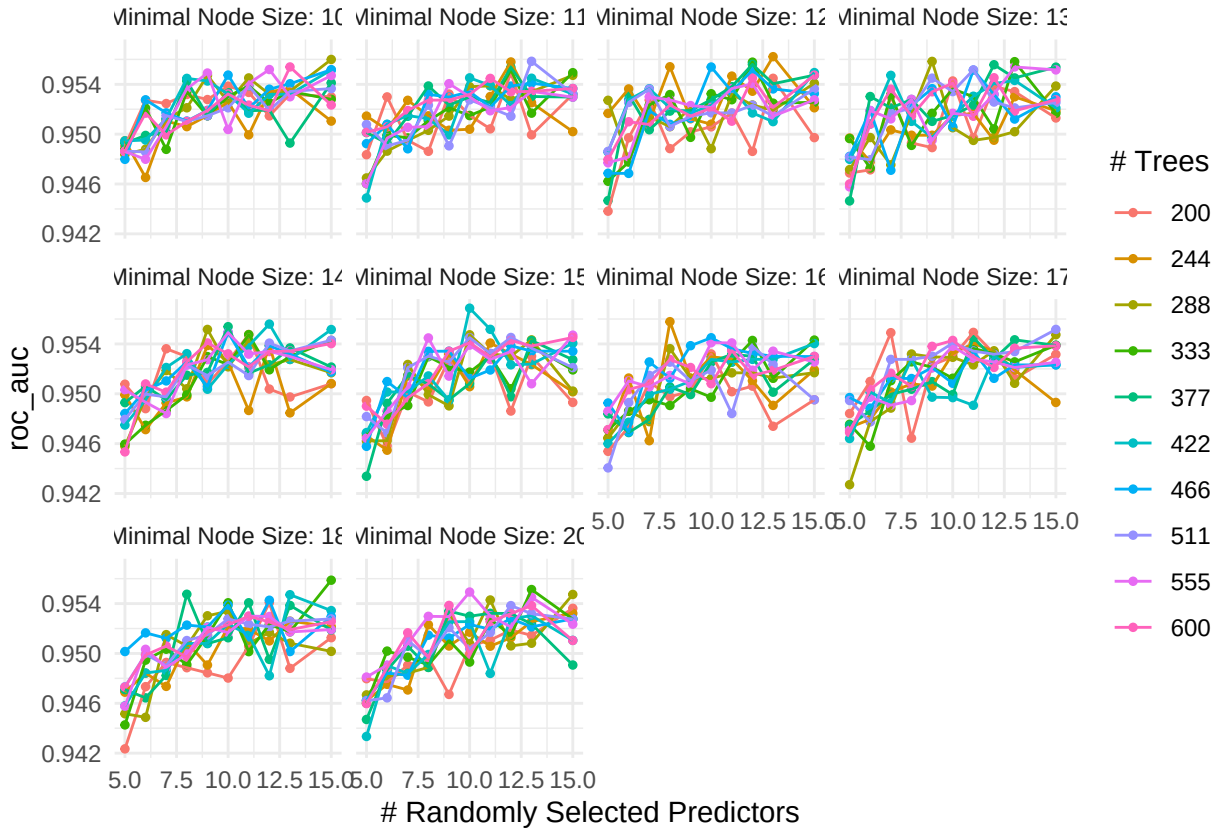
Elastic Net Logistic Regression Performance:

```
autoplot(tune_en, metric = "roc_auc") + theme_minimal()
```



Random Forest Performance:

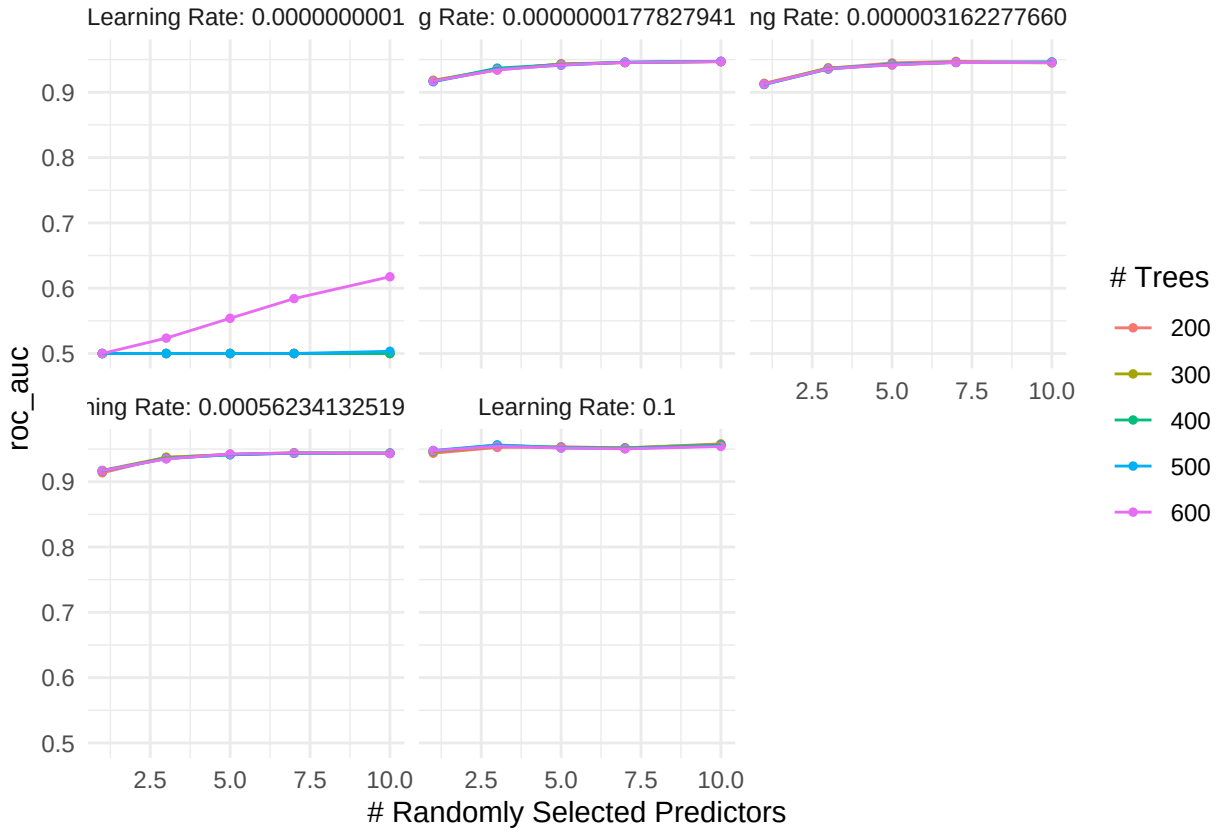
```
autoplot(tune_rf, metric = "roc_auc") + theme_minimal()
```



Generally, as number of randomly selected predictors increased, the roc auc did as well. Based on the graphs, it is hard to determine the effect of the other hyper-parameters values.

Boosted Tree Performance:

```
autoplot(tune_bt, metric = "roc_auc") + theme_minimal()
```



Based on these graphs, it is hard to determine the effect of the hyper-parameters values.

The best KNN models:

```
show_best(tune_knn, metric = "roc_auc")[c(-3, -5, -7)]
```

```
## # A tibble: 5 x 4
##   neighbors .metric  mean std_err
##       <int> <chr>   <dbl>   <dbl>
## 1        24 roc_auc 0.895  0.0139
## 2        25 roc_auc 0.895  0.0144
## 3        23 roc_auc 0.894  0.0136
## 4        22 roc_auc 0.892  0.0137
## 5        21 roc_auc 0.891  0.0135
```

The best estimate for our test ROC AUC was produced when our model was fit with 24 neighbors.

The best elastic net models:

```
show_best(tune_en, metric = "roc_auc")[c(-4, -6, -8)]
```

```
## # A tibble: 5 x 5
##   penalty mixture .metric  mean std_err
##     <dbl>   <dbl> <chr>   <dbl>   <dbl>
## 1    0.01   0.222 roc_auc 0.946  0.0192
## 2    0.01   0.333 roc_auc 0.945  0.0196
```

```
## 3    0.01   0.111 roc_auc 0.945  0.0190
## 4    0.01   0.889 roc_auc 0.944  0.0213
## 5    0.01   0.444 roc_auc 0.944  0.0208
```

The best estimate for our test ROC AUC was produced when our model was fit with a penalty value of 0.01 and a mixture value of 0.2222.

The best random forest models:

```
show_best(tune_rf, metric = "roc_auc")[c(-5, -7, -9)]
```

```
## # A tibble: 5 x 6
##    mtry trees min_n .metric  mean std_err
##   <int> <int> <int> <chr>   <dbl>   <dbl>
## 1    10   422    15 roc_auc 0.957  0.0116
## 2    13   244    12 roc_auc 0.956  0.0121
## 3    15   288    10 roc_auc 0.956  0.0118
## 4    15   333    18 roc_auc 0.956  0.0115
## 5    13   511    11 roc_auc 0.956  0.0107
```

```
best_rf = select_best(tune_rf, metric = "roc_auc")
```

The best estimate for our test ROC AUC was produced when our model was fit with values of 10 for mtry, 422 trees, and 15 min_n.

The best boosted tree models:

```
show_best(tune_bt, metric = "roc_auc")[c(-5, -7, -9)]
```

```
## # A tibble: 5 x 6
##    mtry trees learn_rate .metric  mean std_err
##   <int> <int>      <dbl> <chr>   <dbl>   <dbl>
## 1    10   200        0.1 roc_auc 0.958 0.00805
## 2    10   300        0.1 roc_auc 0.957 0.00822
## 3     3   500        0.1 roc_auc 0.956 0.00805
## 4     3   400        0.1 roc_auc 0.956 0.00875
## 5    10   400        0.1 roc_auc 0.956 0.00800
```

```
best_bt = select_best(tune_bt, metric = "roc_auc")
```

The best estimate for our test ROC AUC was produced when our model was fit with 10 mtry, 200 trees, and a learning rate of 0.1.

Best Models

The boosted tree with mrty 10, 200 trees, and a learning rate of 0.1 had the highest estimate area under the ROC curve. The best random forest produced similar results when estimating test ROC AUC, so we will fit this model to our training data as well.

So, we fit these chosen models to our whole training data:

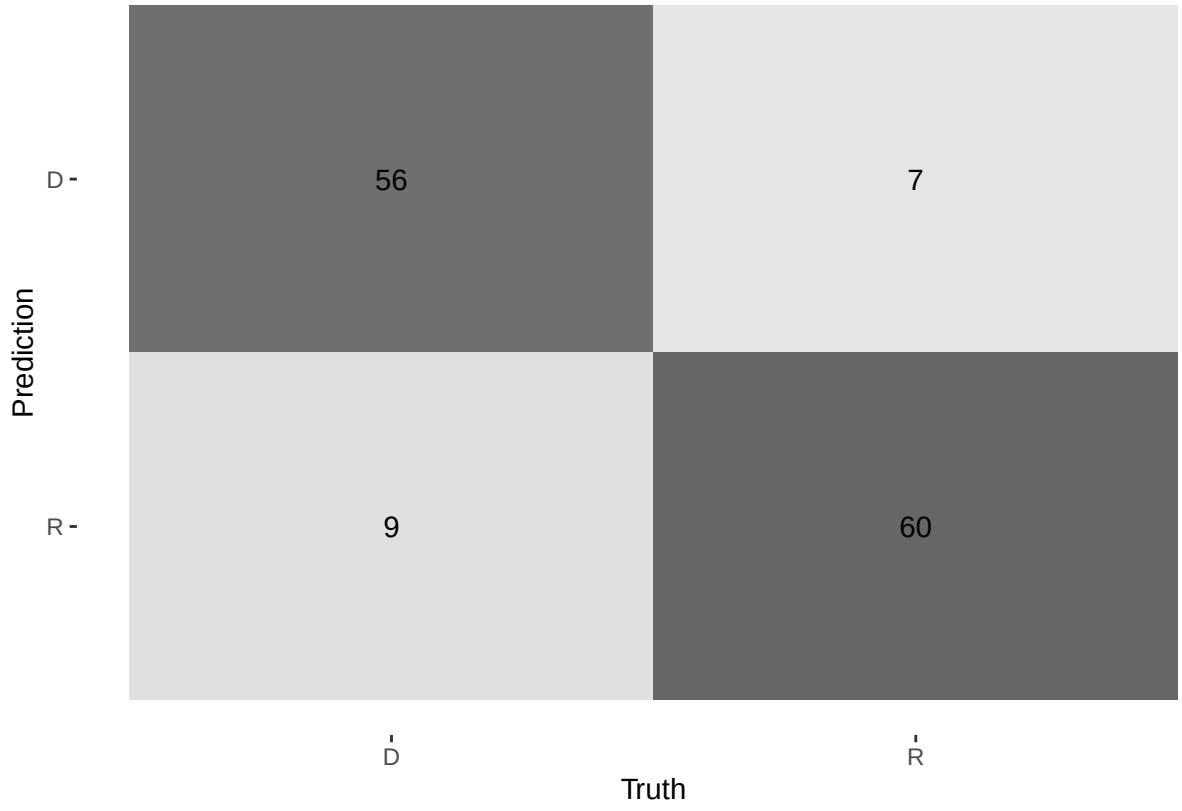
```
final_rf_model <- finalize_workflow(rf_workflow, best_rf)
final_rf_model <- fit(final_rf_model, congress_train)
```

```
final_bt_model <- finalize_workflow(bt_workflow, best_bt)
final_bt_model <- fit(final_bt_model, congress_train)
```

Then, we use our test data to see if the model produces accurate predictions about the party of congressional representatives:

A heat map of the results of applying the best boosted tree to the testing data:

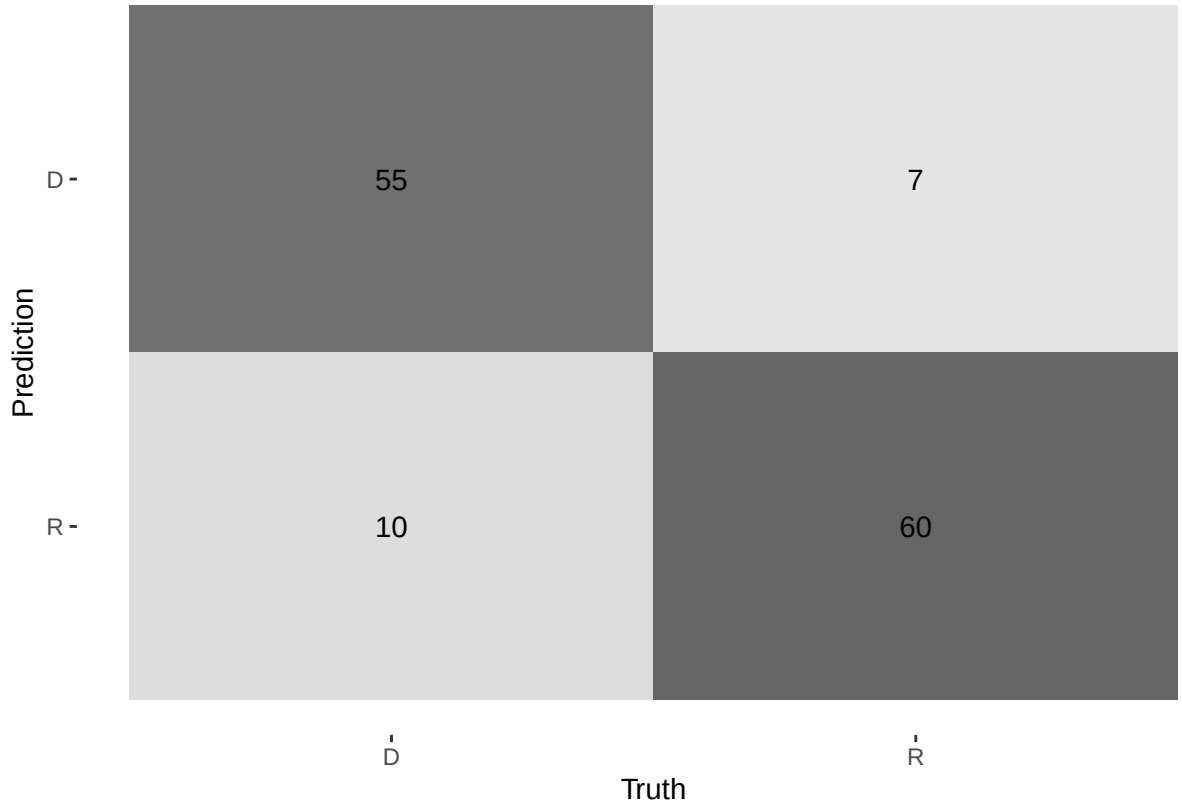
```
augment(final_bt_model, new_data = congress_test) %>%
  conf_mat(truth = Party, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```



The model correctly predicted 56 of the Democrat representatives and 60 of the Republican representatives. It incorrectly predicted that 9 Democrat representatives were Republican. It also incorrectly predicted that 7 Republican representatives were Democrats.

A heat map of the results of applying the best random forest to the testing data:

```
augment(final_rf_model, new_data = congress_test) %>%
  conf_mat(truth = Party, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```

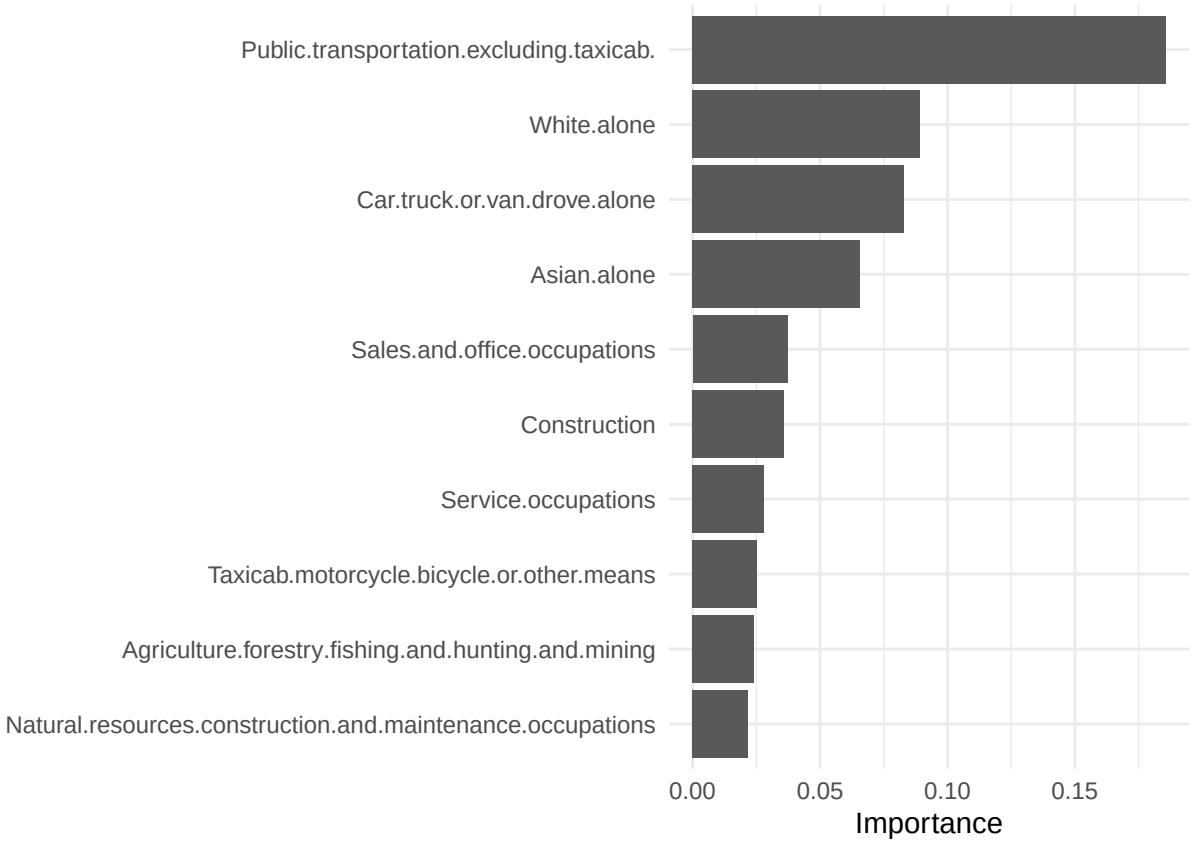


The model correctly predicted 55 of the Democrat representatives and 60 of the Republican representatives. It incorrectly predicted that 10 Democrat representatives were Republican. It also incorrectly predicted that 7 Republican representatives were Democrats.

The boosted tree had one more correct prediction than the random forest, so both models performed similarly.

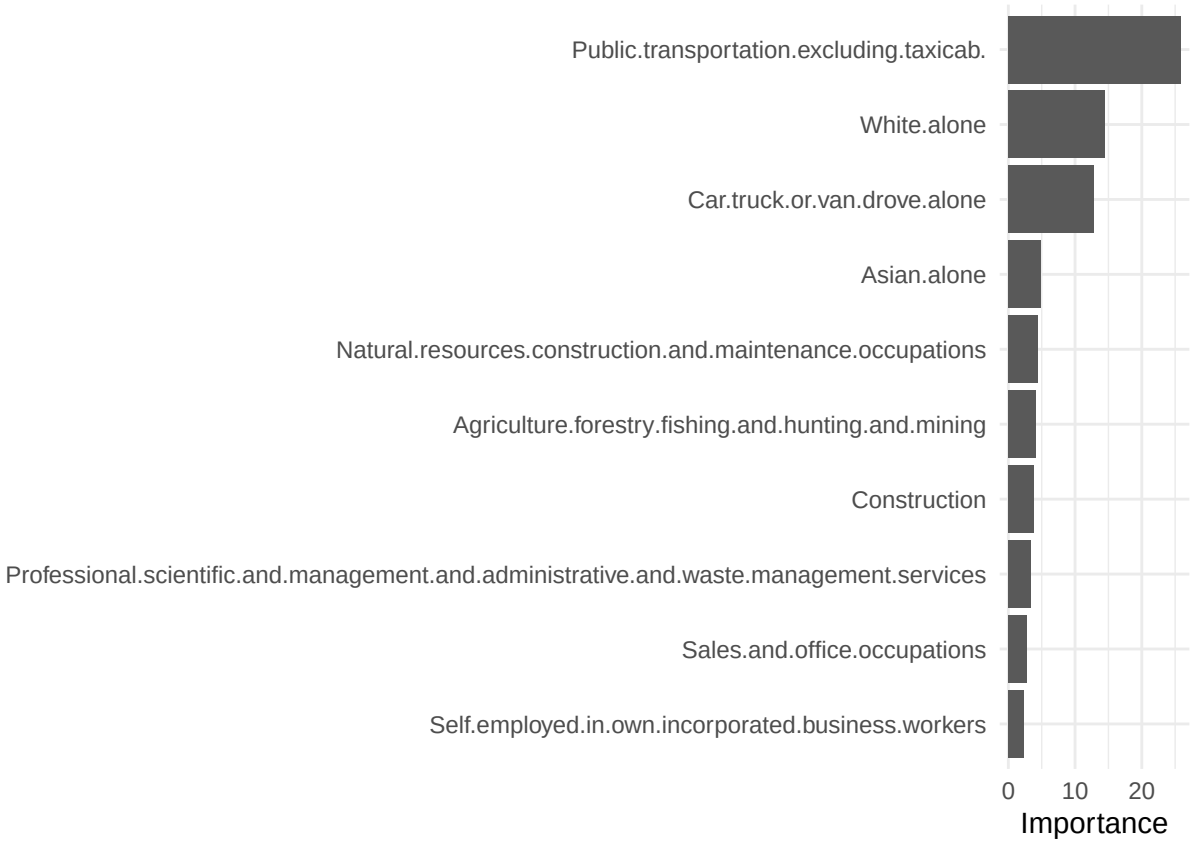
An influence plot showing the most influential predictors in the boosted tree model:

```
final_bt_model %>% extract_fit_parsnip() %>%
vip() +
theme_minimal()
```



An influence plot showing the most influential predictors in the random forest model:

```
final_rf_model %>% extract_fit_parsnip() %>%
vip() +
theme_minimal()
```



In both of these models, the most influential predictor was the proportion of the population which took public transportation to work. The second most influential predictor was the total number of individuals in the district whole only identified as White.

Conclusion

Both the tuned Boosted Tree model and the tuned Random Forest model were able to predict the party of US House representatives with high accuracy. The Boosted Tree did perform slightly better. The KNN and Elastic Net models did not perform as well during our hyper parameter tuning, and thus were never applied to our test data. We also discovered a few of the most important variables used in the models. However, because we used boosted trees and random forests, it is hard to determine the exact effect these variables had on our models predictions.

The data I used was collected from the US Census Bureau. More congressional district demographic data is available going back many years. It would also be possible to find the party representatives in congress over the years. A model that uses more of this data could possibly produce better results, however it is also complicated by the fact that this data would span a length of time. Another interesting project could be to apply my models with demographic data prior to an election, and see how it performs. However I would expect this to not perform as well due to changes in voter preferences over time.