Project 003 - Nonlinear Predictors

Kyle Brewster

3/13/2022

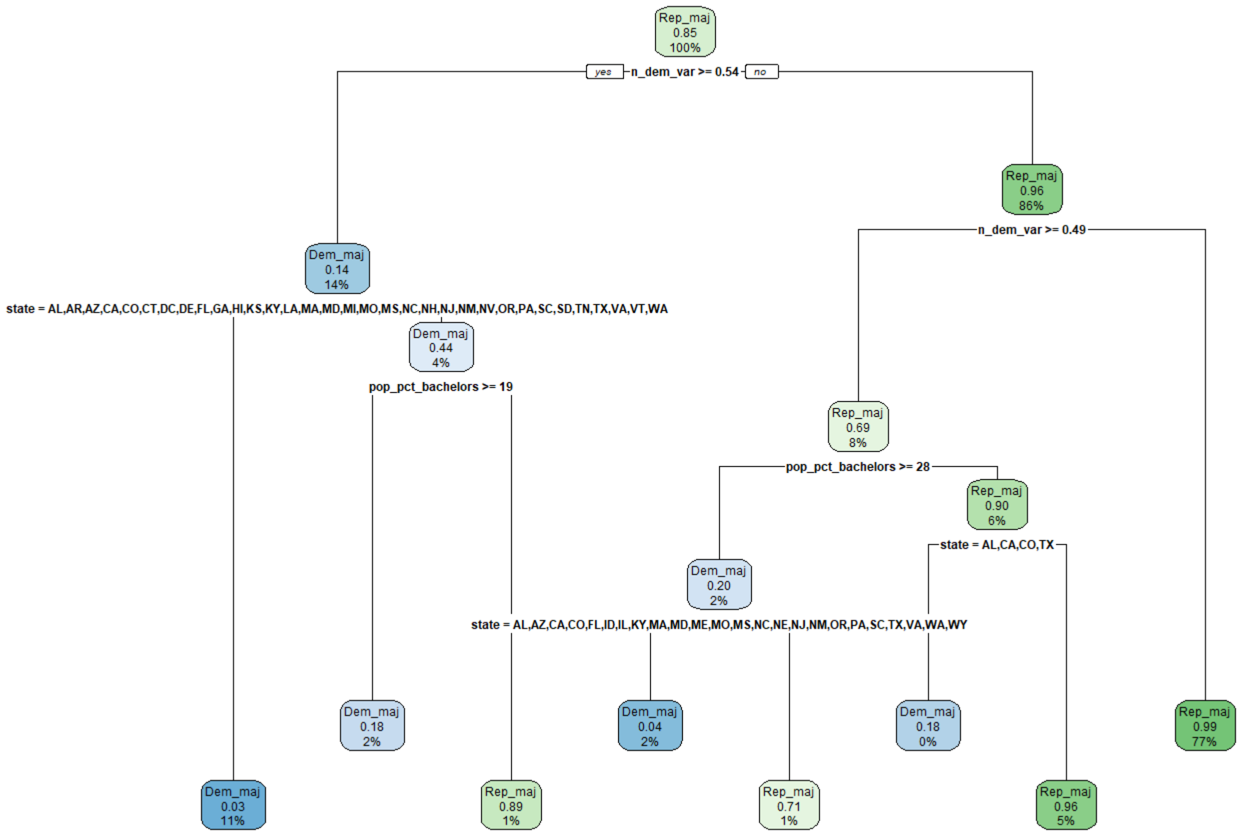
Part 0: Prep Work

Loading packages

```r
library(pacman)
p_load(readr,      # Reading csv
       dplyr,      # Syntax
       magrittr,   # Piping
       tidymodels, # Modeling
       rpart.plot,  # Plotting
       baguette,   # Bagging trees
       randomForest,      # Random forests
       caret,      # General model fitting
       rpart,
       parsnip,
       ipred)

election = read_csv("election_2016.csv")

## Cleaning

# Adding a variable found to be significant from last time
election %<>%
  mutate(log_inc_hh = log(income_median_hh),
         log_home_med = log(home_median_value),
         intrxt_var = (log_inc_hh*log_home_med),
         n_dem_var = (n_votes_democrat_2012/n_votes_total_2012),
         n_rep_var = (n_votes_republican_2012/n_votes_total_2012),
         i_republican_2012 = if_else(i_republican_2012==1,
                                     "Rep_maj", "Dem_maj"),
         i_republican_2016 = if_else(i_republican_2016==1,
                                     "Rep_maj", "Dem_maj"),
         state = usdata::state2abbr(election$state))
# Last line to help save space for plotting trees

set.seed(123)
# Creating Train/Test Splits
train_elect = election %>%  sample_frac(0.8)
test_elect = anti_join(election, train_elect, by = 'fips')

# Removing 'fips' since it is an indicator value
train_elect %<>%  select(-c('fips'))
test_elect %<>% select(-c('fips'))
election %<>% select(-c('fips'))

# Duplicating for consequence-free sandboxing and removing county for better results
train_1 <- train_elect %>% select(-c("county"))
test_1 <- test_elect %>% select(-c("county"))
```

Individual Decision Trees

```r
default_cv = train_1 %>% vfold_cv(v =5)

# Define the decision tree
default_tree = decision_tree(mode ="classification",
                             cost_complexity = tune(),
                             tree_depth = tune()) %>%
               set_engine("rpart")

# Defining recipe
default_recipe = recipe(i_republican_2016 ~., data = train_1)

# Defining workflow
default_flow = workflow() %>%
  add_model(default_tree) %>%
  add_recipe(default_recipe)

# Tuning
default_cv_fit = default_flow %>%
  tune_grid(
    default_cv,
    grid = expand_grid(
      cost_complexity = seq(0, 0.15, by = 0.01),
      tree_depth = c(1,2,5,10),
    ),
    metrics = metric_set(accuracy, roc_auc)
  )

# Fitting the best model
best_flow = default_flow %>%
  finalize_workflow(select_best(default_cv_fit, metric = "accuracy")) %>%
  fit(data = train_1)

# Choosing the best model
best_tree = best_flow %>% extract_fit_parsnip()

# Plotting the tree
best_tree$fit %>% rpart.plot::rpart.plot(roundint=F)
```




```
# Printing summary statistics
printcp(best_tree$fit)
```

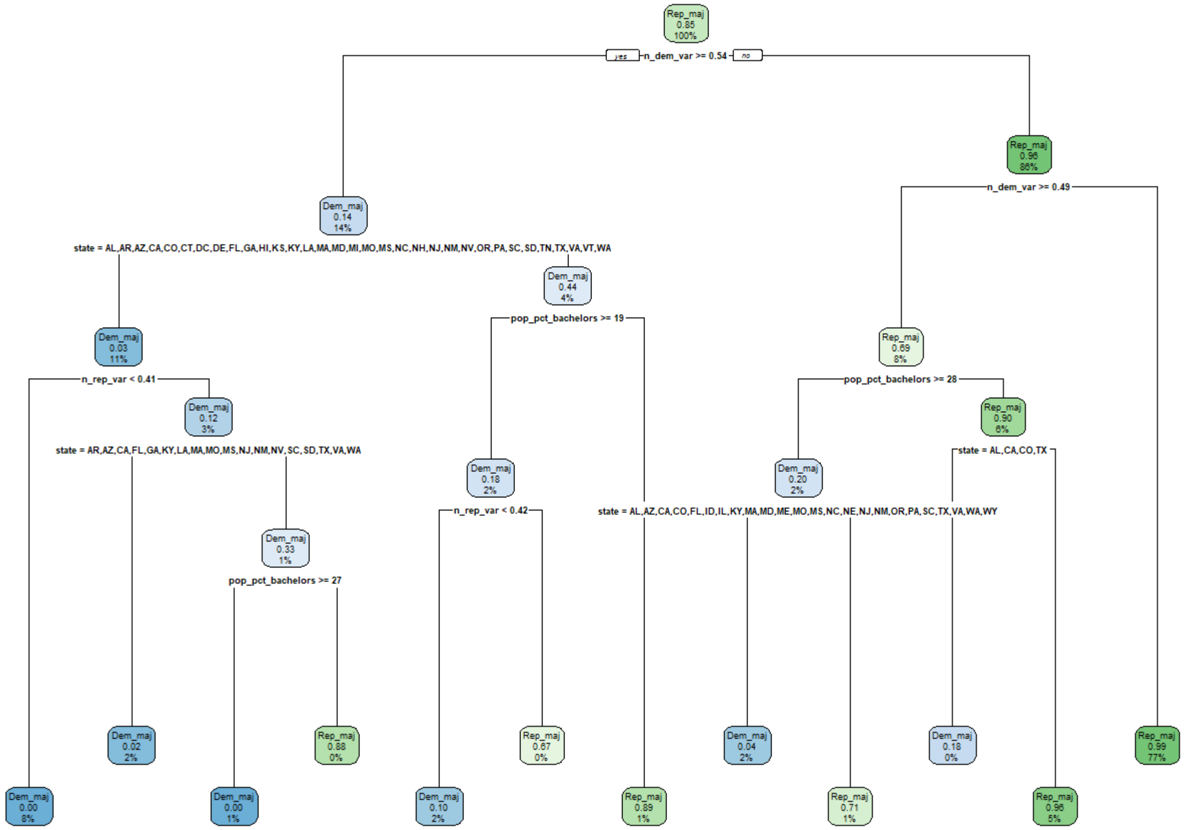
```
##
## Classification tree:
## rpart::rpart(formula = ..y ~ ., data = data, cp = ~0.01, maxdepth = ~5)
##
## Variables actually used in tree construction:
## [1] n_dem_var         pop_pct_bachelors state
##
## Root node error: 386/2493 = 0.15483
##
## n= 2493
##
##         CP nsplit rel error  xerror     xstd
## 1 0.670984      0   1.00000 1.00000 0.046793
## 2 0.047927      1   0.32902 0.34715 0.029172
## 3 0.034974      3   0.23316 0.33161 0.028548
## 4 0.018135      5   0.16321 0.26684 0.025744
## 5 0.015544      6   0.14508 0.27202 0.025981
## 6 0.010000      7   0.12953 0.24611 0.024765
```

```
best_tree$fit$variable.importance
```

```
##            n_dem_var            n_rep_var     i_republican_2012
##          452.1463698          414.2622418           228.8220340
##        pop_pct_white        pop_pct_black n_votes_democrat_2012
##          116.0882558           91.9720204            68.2476335
##    pop_pct_bachelors                state     home_median_value
##           64.0394010           45.6071438            34.6851294
##         log_home_med            income_pc            intrxt_var
##           34.6851294           22.8449933            22.8449933
##      income_median_hh   n_votes_other_2012              n_firms
##           21.4604483           16.8250475            12.4633011
##       pop_pct_foreign    pop_pct_nonenglish        persons_per_hh
##            8.0334308            6.6427714             6.0344663
##      pop_pct_hispanic        pop_pct_asian       pop_pct_pacific
##            5.5356429            4.4285143             2.0859891
##    n_votes_total_2012     pop_pct_homeowner        pop_pct_native
##            1.6518638            1.3906594             0.9976469
```

```
# Creating new df to hold predicted values for later comparison
comp_df = train_1 %>% select(c(i_republican_2016))

comp_df$one_tree_1 = predict(best_tree, new_data=train_1)
```

```r
# Defining another tree with tuning adjustments
default_tree2 = decision_tree(mode ="classification",
                              cost_complexity = 0.005,
                              tree_depth = 10) %>%
               set_engine("rpart")

# Defining recipe
default_recipe = recipe(i_republican_2016 ~., data = train_1)

# Defining workflow
default_flow = workflow() %>%
  add_model(default_tree2) %>%
  add_recipe(default_recipe)

# Tuning
default_cv_fit = default_flow %>%
  tune_grid(
    default_cv,
    grid = expand_grid(
      cost_complexity = seq(0, 0.15, by = 0.01),
      tree_depth = c(1,2,5,10),
    ),
    metrics = metric_set(accuracy, roc_auc)
  )

# Fitting the best model
best_flow = default_flow %>%
  finalize_workflow(select_best(default_cv_fit, metric = "accuracy")) %>%
  fit(data = train_1)

# Choosing the best model
best_tree = best_flow %>% extract_fit_parsnip()

# Plotting the tree
best_tree$fit %>% rpart.plot::rpart.plot(roundint=F)
```




```r
# Printing summary statistics
printcp(best_tree$fit)
```

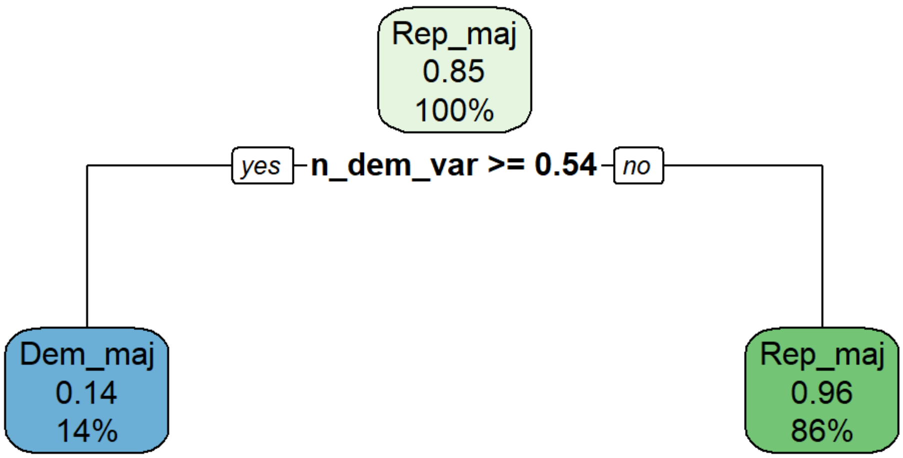
```
##
## Classification tree:
## rpart::rpart(formula = ..y ~ ., data = data, cp = ~0.005, maxdepth = ~10)
##
## Variables actually used in tree construction:
## [1] n_dem_var         n_rep_var         pop_pct_bachelors state
##
## Root node error: 386/2493 = 0.15483
##
## n= 2493
##
##          CP nsplit rel error  xerror     xstd
## 1 0.6709845      0   1.00000 1.00000 0.046793
## 2 0.0479275      1   0.32902 0.33420 0.028653
## 3 0.0349741      3   0.23316 0.27720 0.026217
## 4 0.0181347      5   0.16321 0.23834 0.024386
## 5 0.0155440      6   0.14508 0.23575 0.024258
## 6 0.0077720      7   0.12953 0.22021 0.023474
## 7 0.0051813      8   0.12176 0.21503 0.023206
## 8 0.0050000     11   0.10622 0.22539 0.023739
```

```r
best_tree$fit$variable.importance
```

```
##               n_dem_var               n_rep_var       i_republican_2012
##            454.47677685            420.72445213            228.82203400
##           pop_pct_white           pop_pct_black       pop_pct_bachelors
##            116.99264860             92.48881630             71.75193327
##    n_votes_democrat_2012                   state       home_median_value
##             68.24763352             48.94101807             38.47679605
##            log_home_med               income_pc               intrxt_var
##             34.68512938             27.58457666             27.58457666
##        income_median_hh      n_votes_other_2012                 n_firms
##             21.46044828             16.82504752             12.46330111
##           pop_pct_asian         pop_pct_foreign      pop_pct_nonenglish
##              8.22018095              8.03343079              6.64277143
##           persons_per_hh        pop_pct_hispanic          pop_pct_change
##              6.03446631              5.79404079              5.68750000
##         pop_pct_pacific      n_votes_total_2012       pop_pct_homeowner
##              2.08598905              1.65186379              1.39065937
##         pop_pct_below18          pop_pct_native           land_area_mi2
##              1.06646907              0.99764690              0.04734848
##         pop_pct_poverty n_votes_republican_2012
##              0.04734848              0.02367424
```

```r
# Adding prediction to comparison data frame
comp_df$one_tree_2 = predict(best_tree, new_data=train_1)
```

```r
# And another with different tuning
default_tree3 = decision_tree(mode ="classification",
                              cost_complexity = 0.05,
                              tree_depth = 5) %>%
               set_engine("rpart")

# Defining recipe
default_recipe = recipe(i_republican_2016 ~., data = train_1)

# Defining workflow
default_flow = workflow() %>%
  add_model(default_tree3) %>%
  add_recipe(default_recipe)

# Tuning
default_cv_fit = default_flow %>%
  tune_grid(
    default_cv,
    grid = expand_grid(
      cost_complexity = seq(0, 0.15, by = 0.01),
      tree_depth = c(1,2,5,10),
    ),
    metrics = metric_set(accuracy, roc_auc)
  )

# Fitting the best model
best_flow = default_flow %>%
  finalize_workflow(select_best(default_cv_fit, metric = "accuracy")) %>%
  fit(data = train_1)

# Choosing the best model
best_tree = best_flow %>% extract_fit_parsnip()

# Plotting the tree
best_tree$fit %>% rpart.plot::rpart.plot(roundint=F)
```




```
# Printing summary statistics
printcp(best_tree$fit)
```

```
##
## Classification tree:
## rpart::rpart(formula = ..y ~ ., data = data, cp = ~0.05, maxdepth = ~5)
##
## Variables actually used in tree construction:
## [1] n_dem_var
##
## Root node error: 386/2493 = 0.15483
##
## n= 2493
##
##        CP nsplit rel error  xerror     xstd
## 1 0.67098      0   1.00000 1.00000 0.046793
## 2 0.05000      1   0.32902 0.33679 0.028758
```

```
best_tree$fit$variable.importance
```

```
##            n_dem_var            n_rep_var     i_republican_2012
##            417.95279            384.19059             200.24479
##        pop_pct_white        pop_pct_black n_votes_democrat_2012
##            114.09296             88.48026              54.71805
```

```
comp_df$one_tree_3 = predict(best_tree, new_data=train_1)
```

Part 2: Bagging

```
# Define the decision tree
default_treebag = bag_tree(mode ="classification",
                             cost_complexity = 0,
                              min_n = 2) %>%
               set_engine(engine = "rpart", times = 10)

# Defining workflow
default_flow = workflow() %>%
  add_model(default_treebag) %>%
  add_recipe(default_recipe)

fitt = default_flow %>% fit(train_1)

fitt
```

```
## ══ Workflow [trained] ══════════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: bag_tree()
##
## ── Preprocessor ────────────────────────────────────────────────────────────────
## 0 Recipe Steps
##
## ── Model ───────────────────────────────────────────────────────────────────────
## Bagged CART (classification with 10 members)
##
## Variable importance scores include:
##
## # A tibble: 35 × 4
##    term                     value std.error  used
##    <chr>                    <dbl>     <dbl> <int>
##  1 n_dem_var                464.      10.6     10
##  2 n_rep_var                431.       9.90    10
##  3 i_republican_2012        258.      22.0     10
##  4 pop_pct_white            140.       3.96    10
##  5 pop_pct_black             99.1      8.63    10
##  6 n_votes_democrat_2012     85.7     13.8     10
##  7 state                     72.4      7.82    10
##  8 pop_pct_bachelors         44.4      6.03    10
##  9 n_votes_republican_2012   38.7      4.48    10
## 10 n_votes_total_2012        34.5      5.84    10
## # … with 25 more rows
```

```
comp_df$pred_bag = predict(fitt, new_data=train_1)

# out-of-bag estimate
mean(predict(fitt, new_data=train_1) != train_1$i_republican_2016)
```

```
## [1] 0.0008022463
```

Part 3: Forests

```
train_1 %<>%
   mutate(i_republican_2016 = as.factor(i_republican_2016),
          i_republican_2012 = as.factor(i_republican_2012),
          state = as.factor(state))

class_rf =  randomForest(formula = i_republican_2016 ~ .,
                        data = train_1,
                        importance = TRUE,
                        ntree = 50)
importance(class_rf)
```

```
##                          Dem_maj    Rep_maj MeanDecreaseAccuracy
## state                  5.7827920  3.2732210            6.7297826
## n_votes_republican_2012  0.3775585  3.0508479            3.6329311
## n_votes_democrat_2012    3.9973235  2.7527855            4.0766066
## n_votes_other_2012       2.3233129  1.9414456            2.9046628
## n_votes_total_2012       3.0823832  2.4342568            3.0867187
## i_republican_2012        5.0691475  5.3070833            5.6658702
## pop                      1.6296081  1.9995939            2.4183275
## pop_pct_change           2.4073160  3.0010192            3.8472630
## pop_pct_below18          2.5273069 -0.1327897            2.0920663
## pop_pct_above65          2.5904300  2.1438472            3.8430079
## pop_pct_female           1.3215009  2.4557002            2.8852927
## pop_pct_asian            5.0449115  2.0252417            4.5697654
## pop_pct_black            3.9989266  2.3406239            3.6716905
## pop_pct_native          -0.2037754  3.9368324            2.3410602
## pop_pct_pacific         -0.7922786  1.4172684            0.2310305
## pop_pct_white            4.6616620  3.3899703            5.6271140
## pop_pct_multiracial      1.9661917 -0.2641079            1.9409657
## pop_pct_hispanic         1.5943358  3.9908033            4.0988741
## pop_pct_foreign          1.9897345  2.0316928            2.7330599
## pop_pct_nonenglish       3.3227824  2.4623419            3.8031794
## pop_pct_bachelors        4.5862724  3.7306547            5.4026202
## pop_pct_veteran          1.2172802 -0.1207814            1.5329598
## pop_pct_homeowner        5.1503752  2.7177746            5.4510206
## pop_pct_poverty          1.1075750  2.5856720            2.8040001
## home_median_value        3.9774436  3.6583523            5.5103518
## persons_per_hh           2.0583083  2.6709105            3.3611494
## income_pc                1.8890909  2.2699118            3.0970804
## income_median_hh         3.8327118  2.1094371            4.2042322
## n_firms                  1.9952310  2.8771965            3.4285931
## land_area_mi2           -0.6404026  2.0694997            1.8864233
## log_inc_hh               2.8024581  2.1939649            3.7798155
## log_home_med             3.0526006  2.8316391            3.7405009
## intrxt_var               3.1084602  3.2945792            4.5482069
## n_dem_var                9.9562158  6.6807609            9.3077594
## n_rep_var                8.3340291  6.5111767            8.1390245
##                         MeanDecreaseGini
## state                          36.390226
## n_votes_republican_2012         6.664154
## n_votes_democrat_2012          25.039407
## n_votes_other_2012              7.519856
## n_votes_total_2012             16.947185
## i_republican_2012              74.225091
## pop                            11.318793
## pop_pct_change                  3.731598
## pop_pct_below18                 3.124547
## pop_pct_above65                 7.723239
## pop_pct_female                  3.240860
## pop_pct_asian                  24.837747
## pop_pct_black                  16.440402
## pop_pct_native                  2.919690
## pop_pct_pacific                 1.046749
```

```
## pop_pct_white                   23.819835
## pop_pct_multiracial              2.912118
## pop_pct_hispanic                 3.931027
## pop_pct_foreign                  5.892864
## pop_pct_nonenglish               5.110149
## pop_pct_bachelors               15.367654
## pop_pct_veteran                  6.258728
## pop_pct_homeowner               14.299832
## pop_pct_poverty                  5.930266
## home_median_value                8.116033
## persons_per_hh                   4.765856
## income_pc                        5.253349
## income_median_hh                 5.561976
## n_firms                         11.278397
## land_area_mi2                    4.925648
## log_inc_hh                       4.899006
## log_home_med                     7.490974
## intrxt_var                       7.400742
## n_dem_var                      112.270620
## n_rep_var                      157.326737
```

```
comp_df$pred_rf = predict(class_rf, type="response", newdata = train_1)

confusion_mtrx = table(train_1$i_republican_2016, comp_df$pred_rf)
confusion_mtrx # Printing confusion matrix
```

```
##
##             Dem_maj Rep_maj
##   Dem_maj     386       0
##   Rep_maj       0    2107
```

I had originally set `n = 50` so that I could get the model to function properly and had planned to increase the value once I was confident in the functionality of the code, but turns out that 50 was a good value and resulted in great model performance.

Part 4: Boosting

```
default_boost = boost_tree(mode ="classification",
                           engine = "xgboost")

predy = fit(default_boost, formula = i_republican_2016~.,control = control_parsnip(), data = tra
in_1)

comp_df$pred_boost = predict(predy, new_data=train_1)

vec = comp_df %>% pull(pred_boost) %>% as.data.frame(.)
vec2=as_tibble(comp_df$i_republican_2016) %>% as.data.frame(.)

df = vec %>%
   select(.pred_class)%>%mutate(predss = vec$.pred_class) %>%
   mutate(predss = as.character(predss),
          og_val = if_else(
             vec2$value=="Rep_maj","Rep_Maj","Dem_maj"))

confusion_mtrx2 = table(df$predss,df$og_val)
confusion_mtrx2
```

```
##
##           Dem_maj Rep_Maj
##   Dem_maj     385       0
##   Rep_maj       1    2107
```

```
predy
```

```
## parsnip model object
##
## ##### xgb.Booster
## raw: 33.8 Kb
## call:
##   xgboost::xgb.train(params = list(eta = 0.3, max_depth = 6, gamma = 0,
##     colsample_bytree = 1, colsample_bynode = 1, min_child_weight = 1,
##     subsample = 1, objective = "binary:logistic"), data = x$data,
##     nrounds = 15, watchlist = x$watchlist, verbose = 0, nthread = 1)
## params (as set within xgb.train):
##   eta = "0.3", max_depth = "6", gamma = "0", colsample_bytree = "1", colsample_bynode = "1",
min_child_weight = "1", subsample = "1", objective = "binary:logistic", nthread = "1", validate_
parameters = "TRUE"
## xgb.attributes:
##   niter
## callbacks:
##   cb.evaluation.log()
## # of features: 86
## niter: 15
## nfeatures : 86
## evaluation_log:
##     iter training_logloss
##        1       0.45178175
##        2       0.31710814
## ---
##       14       0.01913395
##       15       0.01648365
```

Part 5: Reflection

All of the models above suggested that certain variables we more explanatory than others for predicting the outcome variable `i_republican_2016`

Looking at modeling using a single decision tree, we can see the variation that can arise from tuning the hyperparameters. Each of the individually planted trees had a root node error of 0.15483. This means that these models were incorrect at assigning a given observation to the correct path/spit at the first splitting node. While the end of a split might still result in the correct prediction, that is because we attempting to predict a binary variable; an incorrect assignment at the first split when attempting to predict an outcome that is continuous or with multiple levels. In such cases, more information will be lost with an inaccurate initial assessment.

The last single decision tree that was plotted above provides a fitting visual for this concern from single tree modeling. Since the variable of greatest importance is `n_dem_var` in all of the models, the first split of the tree will be the same for all correct and incorrect assignments.

The out-of-bag error rate estimate for bagging model was 0.00080, which suggests that this model performs well at predicting the outcome variable.

I found it intesting that the `state` variable was not among the top-ranked variables in terms of importance for the single decision tree models, but was ranked as the highest variable for the random forest modeling. When looking at the results of `importance(class_rf)` in part 3 of the code above, we can see that the mean decrease in accuracy is high for the `state` variable as well as many of the other variables that were also considered important in earlier models. A higher value tells us the degree to which the model will loose accuracy if the given variable is excluded from modeling.

Similarily, we can see high values of the mean decrease in Gini coefficient for the variables that this model selected as important. This value provides a measurements for the degree to which each variable contributes to homogeneity of a region (i.e. its purity). If a region is very homogeneous, then the Gini index will be small. In this presentation of summarizing statistics, a lower Gini is represented by a higher decrease in mean Gini, meaning that the given predictor variables plays a greater role in separating the data into the classes defined in the model.

Looking at the confusion matrix for the predicted values of the random forest, it was able to achieve 100% accuracy from the given data. The boosted tree model performed not quite as well, but had strong accuracy in predicting values nonetheless.

```
# Equalizing classes of train and test set
xtest <- rbind(train_1[1, ] , train_1)
xtest <- xtest[-1,]

# Predicting on testing set with model believed to be strongest
xtest$p = predict(class_rf, newdata = xtest, type = "class")

# Cleaning for matri
df = xtest %>%
   mutate(p = as.character(p),
          i_republican_2016 = as.character(i_republican_2016))

# Confusion Matrix
table(df$i_republican_2016, df$p)
```

```
##
##           Dem_maj Rep_maj
##   Dem_maj     385       1
##   Rep_maj       0    2107
```

Part 6: Review

14. Why are boosted-tree ensembles so sensitive to the number of trees (relative to the bagged-tree ensembles and random forests)?

Boosted-tree ensembles are more sensitive to the number of trees compared to bagging or random forests because boosting allows trees to pass information to other trees. Since trees in boosting are trained on residual values from previous trees during the modeling process.

15. How do individual decision trees guard against overfitting?

One way you can guard against over fitting with individual trees is by tuning the number of splits. A higher number of splits for the final selected modeling may result in better model performance for the initial data set, but would become less flexible when using on other data and can result in less interpretability.

We can address these issues by pruning our selected trees. If a variation of the modeling increases variance at a higher rate than it reduces bias (i.e. the bias-variance trade off), then pruning to remove those regions can improve performance in terms of testing MSE.

16. How do ensembles trade between bias and variance?

For ensemble methods, an estimators variance typically decreases as the selected sampling size increases. With this in mind, including a higher number of trees when bagging or growing forests will result in individual trees that are very flexible and noisy, but an aggregate that stabilizes.

17. How do trees allow interactions?

Utilizing methods involving decision trees in prediction allows for models to consider interaction that may be occurring between variables that is much more difficult to capture with a simple linear model. It might be possible to use a simple regression model to fit the training data, but it will likely be overfitting and have poor performance during testing or with new data (or might suggest that perhaps decision trees aren't going to be the best option for modeling a trend).

As a result, trees are able to replicate nonlinear boundaries in data better than other methods and are simple to explain, interpret, and provide graphical visualizations to describe the model.