Election Predictions

Nataniel Moreau

2023-07-27

Predicting 2016 presidential outcomes by county using trees & random forest

```
knitr::opts_chunk$set(echo = TRUE)
# Prep ----------------------------------------------------------------------
#load packages
library(pacman)
p_load(tidyverse, modeldata, skimr, janitor,
               kknn, tidymodels, magrittr, glmnet, baguette, data.table, parallel, xgboost)
# load data
elec_data <- read_csv("election-2016.csv")
```

```
## Rows: 3116 Columns: 33
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (3): fips, county, state
## dbl (30): i_republican_2016, n_votes_republican_2012, n_votes_democrat_2012,...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# clean data
elec_data %<>% clean_names()

# outcome to factor
elec_data %<>% mutate(
  i_republican_2016 = ifelse(i_republican_2016 == 1,"Rep","Dem")
)

set.seed(13579)
```

Part 1: One Tree

01. I think that precision is the best metric. From my last project: "Assuming each party thinks of Y = 1 as leaning towards themselves, there is not as much consequence from not correctly guessing y = 0 as an incorrect guess still works in the favor of that party. On the other hand, having low precision will lead to misplaced confidence in the strength of their voter base which could lead to miss allocation of time and money when campaigning and a bit of a shock come election day when counties assumed to be in favor of you lean the other direction."

```r
# Part 1: Single Tree ----------------------------------------------------
# splits for CV
elec_cv = elec_data %>% vfold_cv(v = 5)

# create recipe for single tree
elec_recipe_single = recipe(
  i_republican_2016 ~ ., data = elec_data
) %>%
  update_role(c(fips, state, county), new_role = "id variable")

# define tree
elec_tree = decision_tree(
  mode = "classification",
  cost_complexity = tune(),
  tree_depth = 5,
  min_n = tune()
) %>% set_engine("rpart")

# define workflow
single_workflow = workflow() %>%
  add_model(elec_tree) %>%
  add_recipe(elec_recipe_single)

# tune hyper parameters
single_tree_cv = single_workflow %>%
  tune_grid(
    elec_cv,
    grid = expand_grid(
      cost_complexity = seq(0,.15, by = .01),
      min_n = seq(1,10, by = 1)
    ), metrics = metric_set(accuracy, precision))
```

```r
# best models
show_best(single_tree_cv, metric = "accuracy")
```

02.

```
## # A tibble: 5 x 8
##   cost_complexity min_n .metric  .estimator  mean     n std_err .config
##             <dbl> <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1            0.01     1 accuracy binary     0.960     5 0.00199 Preprocessor1_M~
## 2            0.01     2 accuracy binary     0.960     5 0.00199 Preprocessor1_M~
## 3            0.01     3 accuracy binary     0.960     5 0.00199 Preprocessor1_M~
## 4            0.01     4 accuracy binary     0.960     5 0.00199 Preprocessor1_M~
## 5            0.01     5 accuracy binary     0.960     5 0.00199 Preprocessor1_M~
```

```r
#show_best(single_tree_cv, metric = "precision")
select_best(single_tree_cv, metric = "accuracy")
```

```
## # A tibble: 1 x 3
```

```
##   cost_complexity min_n .config
##             <dbl> <dbl> <chr>
## 1            0.01     1 Preprocessor1_Model011
```

```
select_best(single_tree_cv, metric = "precision")
```

```
## # A tibble: 1 x 3
##   cost_complexity min_n .config
##             <dbl> <dbl> <chr>
## 1               0     9 Preprocessor1_Model009
```

For both performance metrics the chosen compelxity was the smallest possible at .01 and the minimum number of obs was the lowest value at 1

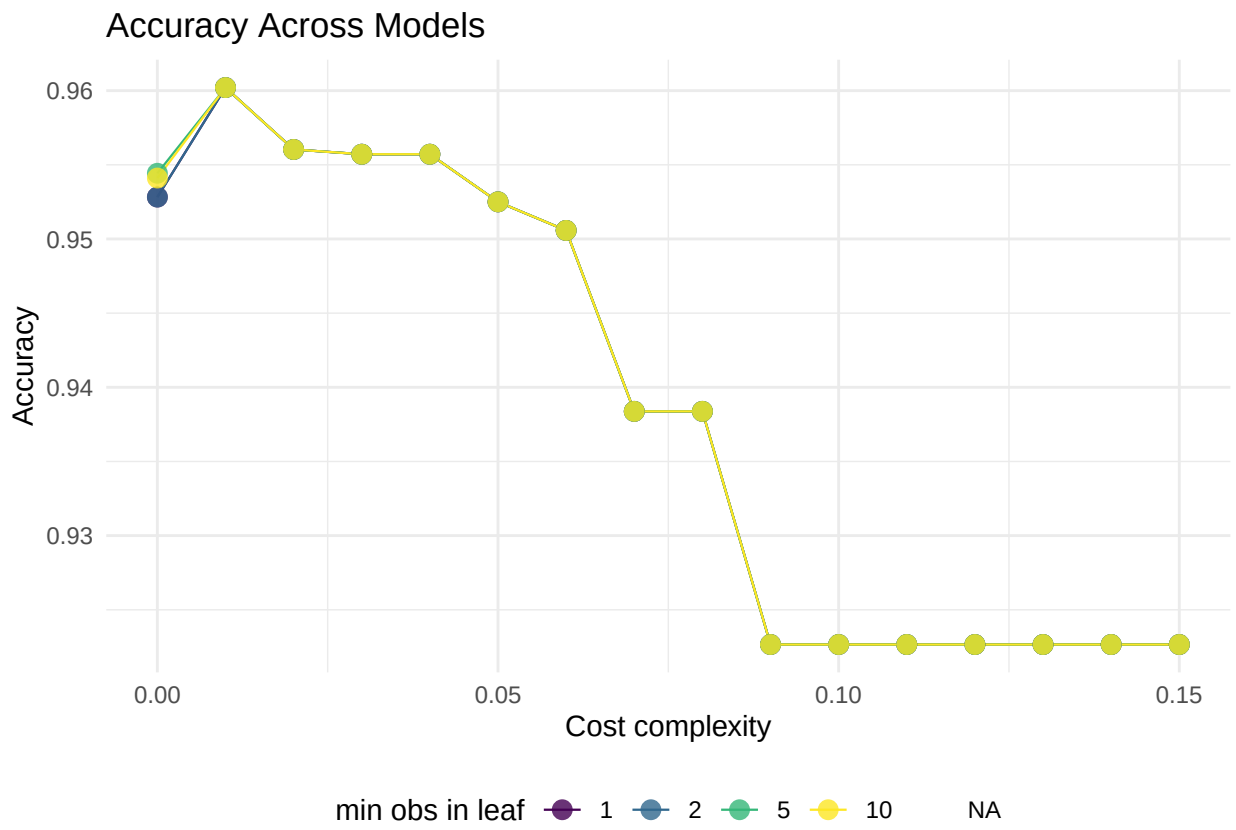
```
# plot models
single_tree_cv %>%
  collect_metrics() %>%
  filter(.metric == "accuracy") %>%
  ggplot(aes(x=cost_complexity, y=mean,
             color = min_n %>% factor(
               levels = c(1,2,5,10), ordered = T
             ))) +
  geom_line(size = .4)+
  geom_point(size = 3, alpha = .8)+
  labs(color = "min obs in leaf",
       title = "Accuracy Across Models")+
  scale_x_continuous('Cost complexity')+
  scale_y_continuous('Accuracy')+
  theme_minimal()+
  theme(legend.position = 'bottom')
```

03.

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
## Warning: Removed 96 rows containing missing values (‘geom_line()‘).
```

```
## Warning: Removed 96 rows containing missing values (‘geom_point()‘).
```



My best model has an accuracy of 95.6%. The performance of all my models were pretty similar across min_n with only cost_complexity having a large effect as accuracy drops by around 3-4 pp around a cost of .1

Part 2: Bag O' Trees

```
# Part 2: Bag o' trees ------------------------------------------------------
# define model
bag_all = bag_tree(
  mode = "classification",
  cost_complexity = tune(),
  tree_depth = tune(),
  min_n = 1,
  class_cost = NULL) %>%
  set_engine("rpart",
             times = 50)
# workflow
bag_all_wkfl =
  workflow() %>%
  add_model(bag_all) %>%
  add_recipe(elec_recipe_single)

# fit using cv
```

```
bag_all_fit = bag_all_wkfl %>%
  tune_grid(
    elec_cv,
    grid = expand_grid(
      cost_complexity = seq(0,.1,by = .01),
      tree_depth = seq(1,5,by = 1)
    ),
    metrics = metric_set(accuracy)
  )
```

04.

```
# best models
show_best(bag_all_fit, metric = "accuracy")
```

05. & 06.

```
## # A tibble: 5 x 8
##   cost_complexity tree_depth .metric  .estimator  mean     n std_err .config
##             <dbl>      <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1            0.01          5 accuracy binary     0.964     5 0.00286 Preprocess~
## 2            0             5 accuracy binary     0.963     5 0.00292 Preprocess~
## 3            0.02          5 accuracy binary     0.962     5 0.00290 Preprocess~
## 4            0.04          5 accuracy binary     0.962     5 0.00218 Preprocess~
## 5            0.03          5 accuracy binary     0.962     5 0.00388 Preprocess~
```

I decided to tune cost complexity and tree depth. The model stuck with the lowest complexity but leans towards models that allow each tree to learn as much as possible with a tree depth of 5.

The accuracy of my best model showed a slight improvement of around half a pp, going from 95.8 to 96.3.

07. Ensembles should perform better out of sample as they allow for more variance in exchange for some more bais. Within-sample I got a slight increase in accuracy, by about 1pp.

Part 3: Forests

```
# Part 3: Rand Forest ------------------------------------------------------
#define parameter grid
hp_grid = expand_grid(
  mtry = 1:29,
  min_n = 1:10
)
# write function for rand forest
rf_elec = function(i){

  # define model
```

```
  rf_all = rand_forest(mode = "classification",
                       mtry = hp_grid$mtry[i],
                       min_n = hp_grid$min_n[i],
                       trees = 50) %>%
    set_engine(engine = "ranger",
               splitrule = "gini")

  # define workflow
  rf_all_wkfl = workflow() %>%
    add_model(rf_all) %>%
    add_recipe(elec_recipe_single)

  # fit using cv
  rf_all_fit = rf_all_wkfl %>% fit(elec_data)

  # store parameters & OOB error
  tibble(mtry = hp_grid$mtry[i],
         min_n = hp_grid$min_n[i],
         oob_er = rf_all_fit$fit$fit$fit$prediction.error)
}

# fit forests
rf_tuned = mclapply( X = 1:nrow(hp_grid),
                     FUN = rf_elec,
                     mc.cores = 1) %>%
  rbindlist()
```

08.

```
# best models
rf_tuned %<>% mutate(accuracy = 1- oob_er) %>% arrange(desc(accuracy))

head(rf_tuned)
```

09.

```
##    mtry min_n     oob_er  accuracy
## 1:   10    10 0.02502276 0.9749772
## 2:   10     9 0.02514094 0.9748591
## 3:   15    10 0.02523792 0.9747621
## 4:   16     8 0.02526609 0.9747339
## 5:    9     8 0.02535417 0.9746458
## 6:   16     4 0.02540532 0.9745947
```

The model chose values of 13 variables per tree and a minimum of 1 obs per leaf which is consistent with previous models.

10. There was a slight increase in performance moving from bagged trees to a random forest. This means that decorrelating the trees does allow them to learn a little bit more. The chosen value of 13 for mtry lies on the middle of the possible values (1:29) which means that variation and stability are moderately valued by the model.

Part 4: Boosting

```
# Part 4: Boosting ------------------------------------------------
# define model
boost_all = boost_tree(mode = "classification",
                       mtry = 8,
                       min_n = 8,
                       trees = tune(),
                       tree_depth = tune(),
                       learn_rate = tune()) %>%
  set_engine("xgboost")

#define workflow
boost_all_wkfl = workflow() %>%
  add_model(boost_all) %>%
  add_recipe(elec_recipe_single)

#fit using cv
boost_all_fit = boost_all_wkfl %>%
  tune_grid( elec_cv,
             grid = expand_grid(tree_depth = seq(1,10, by = 1),
                                learn_rate = seq(.01,.1, by = .01),
                                trees = seq(50,100,10)),
             metrics = metric_set(accuracy))
```

11.

```
#best models
show_best(boost_all_fit, metric = "accuracy")
```

12.

```
## # A tibble: 5 x 9
##   trees tree_depth learn_rate .metric  .estimator  mean     n std_err .config
##   <dbl>      <dbl>      <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1    80          5       0.1  accuracy binary     0.970     5 0.00188 Preproces~
## 2    90          5       0.1  accuracy binary     0.970     5 0.00194 Preproces~
## 3    70          5       0.1  accuracy binary     0.970     5 0.00166 Preproces~
## 4   100          5       0.1  accuracy binary     0.970     5 0.00194 Preproces~
## 5    70          5       0.06 accuracy binary     0.970     5 0.00138 Preproces~
```

Yes and no. The learning rate was on the higher end, so the model is geting updated quite fast. similarly the tree depth is high so the model should be learning faster/ more per tree. On the other hand, the high number of trees for all the best models means that the model is taking a long time to learn all it can.

Part 5: Reflection

13. Within this project my random forest model slightly out performed the rest, just barely beating out the boosted trees model by .005%. Across projects relaxing linearity did very little to nothing. Most of my

best models were my original linear ones from the first project, only the best of the more complex models (rand forest & boost) are with 1pp performance of the linear models (~97-98%). That being said, the early linear models were in a regression setting and used rmse as the performance metric to predict a classification problem.

Part 6: Review

14. When using a boosted model, the trees are learning from each other via the residuals, allowing more trees allows more intercommunication and learning so it makes sense that this model would be more sensitive the how many trees there are than models were they learn in isolation.

15. Individual trees are about as biased as models can be and so do not account for a large amount of variation within the training data, increasing variation to reduce bias is what may lead to overfitting.

16. Ensembles increase variance in an attempt to learn more from the fluctuations within the training data. Making the model more flexible reduces the potential bias that may be present in the model as it pulls the model away from what ever value(s) it has a tendency to report.

17. Trees can allow interactions via methods like boosting where trees are allowed to learn from their predecessors and therefore not make the same errors.