Stats 101C Final Project

Aryan Mistry

7/19/2022

Introduction

This notebook attempts to predict the percentage of voters that voted for Joe Biden in the 2020 US Presidential Election. The data for this project comes from two sources; all voting data comes from the MIT Election Lab, while all demographic and census data comes from the US Census Bureau.

The data represents a total of 3,111 counties across 49 US states (Alaska was excluded due to inconsistencies between the number of Alaskan counties that voted, and the number represented by the census data). For each county, there are columns describing percentage values and actual counts for the number of voters separated by race, gender, age, and education level.

The aim is to use various statistical models and machine learning methods (with tuned hyperparameters as needed) in order to create a model that minimizes the residual mean squared error against the testing set. For this class specifically, after a model was chosen, the predictions are written to a CSV file and uploaded to Kaggle for a competition-style scenario in which the highest-ranked students received the highest project grades.

The machine learning models used in this project are implemented via Tidymodels.

Cleaning and Preprocessing

We load in the training and testing data, which is split 70/40.

```
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 0.2.0 --

## v broom        0.8.0     v recipes      1.0.0
## v dials        1.0.0     v rsample      1.0.0
## v dplyr        1.0.9     v tibble       3.1.7
## v ggplot2      3.3.6     v tidyr        1.2.0
## v infer        1.0.2     v tune         0.2.0
## v modeldata    0.1.1     v workflows    0.2.6
## v parsnip      1.0.0     v workflowsets 0.2.1
## v purrr        0.3.4     v yardstick    1.0.0

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x recipes::step()  masks stats::step()
## * Search for functions across packages at https://www.tidymodels.org/find/
```

```
library(dplyr)
df <- read.csv('train.csv')
df<-na.omit(df)
```

```r
test <- read.csv('test.csv')
```

Since the data includes both percent estimates and actual value counts for each column, we choose to select only the percent estimates, since we do not need both.

```r
df <- df %>% select(contains("PE"))
head(df)
```

```
##   percent_dem X0002PE X0003PE X0005PE X0006PE X0007PE X0008PE X0009PE X0010PE
## 1   0.3921660    49.0    51.0     5.5     5.6     5.4    10.1    18.7    12.4
## 2   0.3471477    50.6    49.4     7.1     6.9     8.2     7.2     6.3    13.3
## 3   0.5094882    47.1    52.9     5.9     6.0     7.0     6.8     6.0    12.0
## 4   0.5745893    48.9    51.1     5.8     6.1     5.9     5.8     5.8    14.6
## 5   0.2971902    44.8    55.2     5.2     6.2     6.4     9.2    10.9    11.3
## 6   0.2644473    48.9    51.1     6.7     6.3     8.2     6.8     6.2    12.9
##   X0011PE X0012PE X0013PE X0014PE X0015PE X0016PE X0017PE X0019PE X0020PE
## 1     9.9     9.1     5.3     4.3     7.7     4.0     1.9    20.0    82.7
## 2    13.0    11.6     6.0     5.5     8.8     4.4     1.7    26.9    76.2
## 3    11.4    12.7     6.8     7.1    10.8     5.3     2.0    23.0    79.5
## 4    13.5    12.2     6.3     6.7    10.8     4.7     1.8    21.4    81.1
## 5     8.1    13.4     4.2     7.0    10.0     5.9     2.1    20.7    81.1
## 6    12.0    12.4     6.5     5.8     9.3     5.0     1.9    25.5    77.3
##   X0021PE X0022PE X0023PE X0024PE X0025PE X0026PE X0027PE X0029PE X0030PE
## 1    80.0    68.1    15.9    13.6   37692    48.3    51.7    6422    42.8
## 2    73.1    69.6    18.0    14.9   48968    50.0    50.0    9948    46.3
## 3    77.0    73.1    22.7    18.2   62830    46.3    53.7   14810    43.1
## 4    78.6    75.3    21.3    17.3  223715    48.2    51.8   49113    44.7
## 5    79.3    71.5    21.8    18.0    5139    45.1    54.9    1168    40.5
## 6    74.5    70.6    19.7    16.2   64922    48.4    51.6   14084    44.0
##   X0031PE X0033PE X0034PE X0035PE X0036PE X0037PE X0038PE X0039PE X0040PE
## 1    57.2   47118    98.4     1.6    98.4    54.4    40.3     0.1     0.0
## 2    53.7   66969    93.1     6.9    93.1    87.5     1.7     0.2     0.1
## 3    56.9   81579    96.9     3.1    96.9    49.9    39.8     0.5     0.1
## 4    55.3  284698    92.2     7.8    92.2    79.5     3.0     1.2     0.0
## 5    59.5    6477    96.1     3.9    96.1    88.5     6.0     0.1     0.0
## 6    56.0   87119    96.8     3.2    96.8    77.4    15.1     0.3     0.0
##   X0041PE X0042PE X0043PE X0044PE X0045PE X0046PE X0047PE X0048PE X0049PE
## 1       0       0     0.0     1.6     0.2     0.6     0.1     0.1     0.0
## 2       0       0     0.0     0.9     0.1     0.2     0.2     0.0     0.3
## 3       0       0     0.0     0.6     0.2     0.1     0.0     0.0     0.1
## 4       0       0     0.1     5.7     0.3     0.6     1.3     0.3     1.1
## 5       0       0     0.0     0.4     0.0     0.0     0.0     0.0     0.0
## 6       0       0     0.0     1.1     0.3     0.1     0.4     0.0     0.0
##   X0050PE X0051PE X0052PE X0053PE X0054PE X0055PE X0056PE X0057PE X0058PE
## 1     0.4     0.1     0.0     0.0     0.0     0.0     0.0     2.0     1.6
## 2     0.0     0.1     0.1     0.0     0.0     0.1     0.0     2.7     6.9
## 3     0.1     0.2     0.0     0.0     0.0     0.0     0.0     6.0     3.1
## 4     1.0     1.1     0.9     0.2     0.3     0.2     0.2     1.9     7.8
## 5     0.4     0.0     0.3     0.3     0.0     0.0     0.0     0.8     3.9
## 6     0.1     0.1     0.0     0.0     0.0     0.0     0.0     2.8     3.2
##   X0059PE X0060PE X0061PE X0062PE X0064PE X0065PE X0066PE X0067PE X0068PE
## 1     0.5     0.5     0.1     0.1    55.8    41.0     0.7     1.7     0.0
## 2     1.0     0.3     0.4     0.0    94.2     2.9     0.8     1.5     0.3
## 3     1.0     0.2     0.7     0.2    52.7    41.3     1.0     1.3     0.1
```

```
## 4      1.6      1.4      2.0      0.0     86.8      5.3      3.1      8.5      1.5
## 5      0.2      3.3      0.1      0.0     92.2      6.4      3.6      0.6      0.3
## 6      0.4      0.5      0.0      0.0     80.6     15.6      1.1      1.2      0.0
##   X0069PE X0071PE X0072PE X0073PE X0074PE X0075PE X0076PE X0077PE X0078PE
## 1      2.4      3.1      2.1      0.4      0.2      0.4     96.9     53.2     40.1
## 2      7.7     58.1     54.5      0.2      0.2      3.2     41.9     37.9      1.5
## 3      6.9     10.5      7.8      0.4      0.4      1.9     89.5     46.6     39.4
## 4      3.6      9.3      5.9      0.9      0.1      2.3     90.7     73.9      2.9
## 5      1.0      1.4      0.5      0.0      0.0      0.9     98.6     87.8      6.0
## 6      5.0     22.3     20.1      0.3      0.1      1.8     77.7     60.2     15.1
##   X0079PE X0080PE X0081PE X0082PE X0083PE X0084PE X0085PE X0088PE X0089PE
## 1      0.1      1.6      0.0      0.7      1.3      0.0      1.3     47.7     52.3
## 2      0.1      0.9      0.1      0.1      1.4      0.1      1.3     49.4     50.6
## 3      0.3      0.6      0.0      0.1      2.4      0.3      2.1     45.8     54.2
## 4      1.0      5.7      0.9      0.3      6.1      0.4      5.6     48.6     51.4
## 5      0.1      0.4      0.3      0.1      3.8      0.1      3.7     45.6     54.4
## 6      0.1      1.1      0.0      0.2      1.1      0.0      1.1     48.5     51.5
```

```
dups <- duplicated(as.list(df))

df <- df[!dups]
```

We use a recipe to center and scale the data.

```
election_recipe <- recipe(percent_dem ~., data = df) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())
keep_pred <- control_resamples(save_pred = TRUE)
```

Model Building

After several previous trials and experimentation attempting to determine the best possible model for this regression problem, it was decided that a gradient-boosted tree model provided best results. The code below describes how the model was created.

We first use Tidymodels' boosted tree model to create a specification. We indicate that we wish to tune the model's hyperparameters, such as the number of trees, their depth, the learning rate, etc.

```
xgb_spec <- boost_tree(
  trees = tune(),
  tree_depth = tune(), min_n = tune(),
  loss_reduction = tune(),
  sample_size = tune(), mtry = tune(),
  learn_rate = tune(),
) %>%
  set_engine("xgboost") %>%
  set_mode("regression")

xgb_spec
```

```
## Boosted Tree Model Specification (regression)
##
## Main Arguments:
##   mtry = tune()
##   trees = tune()
##   min_n = tune()
```

```
##   tree_depth = tune()
##   learn_rate = tune()
##   loss_reduction = tune()
##   sample_size = tune()
##
## Computational engine: xgboost
```

We employ a grid-search method to fill the hyperparameter space.

```
xgb_grid <- grid_latin_hypercube(
  trees(),
  tree_depth(),
  min_n(),
  loss_reduction(),
  sample_size = sample_prop(),
  finalize(mtry(), df),
  learn_rate(),
  size = 30
)
```

We now create a workflow and specify the predictors we wish to employ. In this case, predictors were chosen via PCA and ANOVA tables.

```
xgb_wf <- workflow() %>%
  add_formula(percent_dem ~ X0037PE * X0067PE * X0077PE * X0025PE * X0064PE * X0029PE * X0046PE * X0044I
  add_model(xgb_spec)

xgb_wf
```

```
## == Workflow ====================================================================
## Preprocessor: Formula
## Model: boost_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## percent_dem ~ X0037PE * X0067PE * X0077PE * X0025PE * X0064PE *
##     X0029PE * X0046PE * X0044PE * X0065PE * X0019PE
##
## -- Model -----------------------------------------------------------------------
## Boosted Tree Model Specification (regression)
##
## Main Arguments:
##   mtry = tune()
##   trees = tune()
##   min_n = tune()
##   tree_depth = tune()
##   learn_rate = tune()
##   loss_reduction = tune()
##   sample_size = tune()
##
## Computational engine: xgboost
```

```
set.seed(123)
folds <- vfold_cv(df, strata = percent_dem, v = 5)

folds
```

```
## #  5-fold cross-validation using stratification
```

```
## # A tibble: 5 x 2
##   splits             id
##   <list>             <chr>
## 1 <split [1862/468]> Fold1
## 2 <split [1862/468]> Fold2
## 3 <split [1864/466]> Fold3
## 4 <split [1866/464]> Fold4
## 5 <split [1866/464]> Fold5
```

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'

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

```
doParallel::registerDoParallel()

set.seed(234)
xgb_res <- tune_grid(
  xgb_wf,
  resamples = folds,
  grid = xgb_grid,
  control = control_grid(save_pred = TRUE)
)

xgb_res
```

```
## # Tuning results
## # 5-fold cross-validation using stratification
## # A tibble: 5 x 5
##   splits             id    .metrics           .notes           .predictions
##   <list>             <chr> <list>             <list>           <list>
## 1 <split [1862/468]> Fold1 <tibble [60 x 11]> <tibble [1 x 3]> <tibble>
## 2 <split [1862/468]> Fold2 <tibble [60 x 11]> <tibble [1 x 3]> <tibble>
## 3 <split [1864/466]> Fold3 <tibble [60 x 11]> <tibble [1 x 3]> <tibble>
## 4 <split [1866/464]> Fold4 <tibble [60 x 11]> <tibble [1 x 3]> <tibble>
## 5 <split [1866/464]> Fold5 <tibble [60 x 11]> <tibble [1 x 3]> <tibble>
##
## There were issues with some computations:
##
##   - Warning(s) x5: A correlation computation is required, but `estimate` is constant...
##
## Use `collect_notes(object)` for more information.
```

The model output is stored in tibbles. We can examine the metrics in more details below.

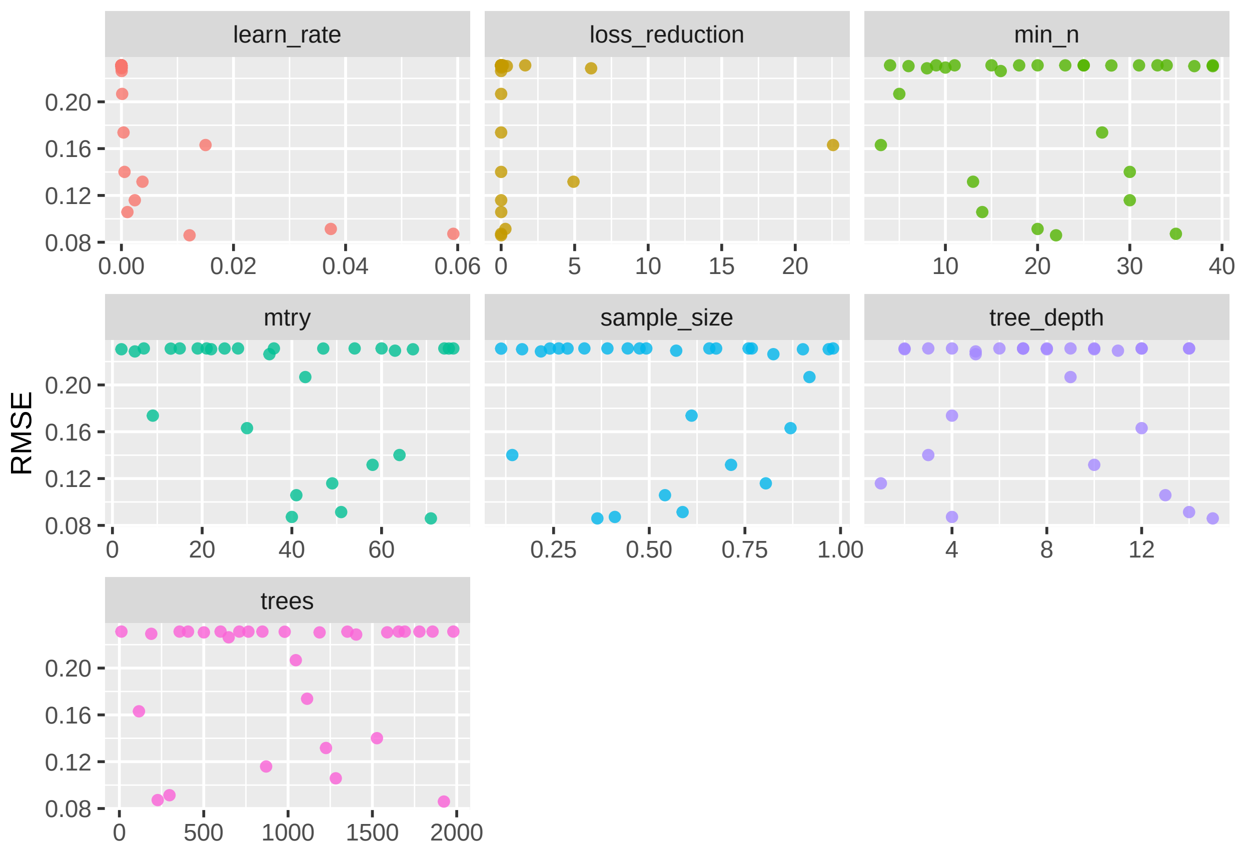
```
collect_metrics(xgb_res)
```

```
## # A tibble: 60 x 13
##     mtry trees min_n tree_depth    learn_rate loss_reduction sample_size .metric
##    <int> <int> <int>      <int>         <dbl>          <dbl>       <dbl> <chr>
##  1    49   870    30          1 0.00239            2.69e- 3       0.805 rmse
##  2    49   870    30          1 0.00239            2.69e- 3       0.805 rsq
##  3    22  1588     6          2 0.00000251         1.16e- 3       0.969 rmse
```

```
## 4     22  1588     6          2 0.00000251         1.16e- 3       0.969 rsq
## 5     75   765    25          2 0.000000408        2.20e- 9       0.287 rmse
## 6     75   765    25          2 0.000000408        2.20e- 9       0.287 rsq
## 7     36  1692     9          3 0.00000000173      2.00e-10       0.657 rmse
## 8     36  1692     9          3 0.00000000173      2.00e-10       0.657 rsq
## 9     64  1527    30          3 0.000555           2.09e- 7       0.142 rmse
## 10    64  1527    30          3 0.000555           2.09e- 7       0.142 rsq
## # ... with 50 more rows, and 5 more variables: .estimator <chr>, mean <dbl>,
## #   n <int>, std_err <dbl>, .config <chr>
```

Let's examine the distributions for the hyperparameters to determine how varying them affects our RMSE (root mean squared error).

```
xgb_res %>%
  collect_metrics() %>%
  filter(.metric == "rmse") %>%
  select(mean, mtry:sample_size) %>%
  pivot_longer(mtry:sample_size,
               values_to = "value",
               names_to = "parameter"
  ) %>%
  ggplot(aes(value, mean, color = parameter)) +
  geom_point(alpha = 0.8, show.legend = FALSE) +
  facet_wrap(~parameter, scales = "free_x") +
  labs(x = NULL, y = "RMSE")
```



We can see that there is a significant decrease in RMSE as the learning rate increases. Let's now look at the top 3 combinations of hyperparameter values that lead to the lowest RMSE. Let us also select the very best out of these combinations.

```
head(show_best(xgb_res, "rmse"), 3)
```

```
## # A tibble: 3 x 13
##    mtry trees min_n tree_depth learn_rate loss_reduction sample_size .metric
##   <int> <int> <int>      <int>      <dbl>          <dbl>       <dbl> <chr>
## 1    71  1924    22         15     0.0122   0.0000000255       0.364 rmse
## 2    40   227    35          4     0.0592   0.000000295        0.410 rmse
## 3    51   297    20         14     0.0374   0.290              0.587 rmse
## # ... with 5 more variables: .estimator <chr>, mean <dbl>, n <int>,
## #   std_err <dbl>, .config <chr>
```

```
best_rmse <- select_best(xgb_res, "rmse")
best_rmse
```

```
## # A tibble: 1 x 8
##    mtry trees min_n tree_depth learn_rate loss_reduction sample_size .config
##   <int> <int> <int>      <int>      <dbl>          <dbl>       <dbl> <chr>
## 1    71  1924    22         15     0.0122   0.0000000255       0.364 Preprocess~
```

```
final_xgb <- finalize_workflow(
  xgb_wf,
  best_rmse
)

final_xgb
```

```
## == Workflow ==================================================================
## Preprocessor: Formula
## Model: boost_tree()
##
## -- Preprocessor --------------------------------------------------------------
## percent_dem ~ X0037PE * X0067PE * X0077PE * X0025PE * X0064PE *
##     X0029PE * X0046PE * X0044PE * X0065PE * X0019PE
##
## -- Model ---------------------------------------------------------------------
## Boosted Tree Model Specification (regression)
##
## Main Arguments:
##   mtry = 71
##   trees = 1924
##   min_n = 22
##   tree_depth = 15
##   learn_rate = 0.0121583885710043
##   loss_reduction = 2.55272002918185e-08
##   sample_size = 0.364445822262205
##
## Computational engine: xgboost
```

We can now finalize our workflow and use our model to predict the percentage of voters in a county that voted for Biden in the 2020 election.

```
fitted_final_xgb <- final_xgb %>%
  fit(data = df)
```

```r
xgb_preds_train <- df %>%
  bind_cols(predict(fitted_final_xgb, new_data = df))
metrics(xgb_preds_train, percent_dem, .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0305
## 2 rsq     standard      0.966
## 3 mae     standard      0.0209
```