Classification attempt 1

Liam Martin-McClay

2024-07-31

Set-up

If you want to knit the PDF but don't want to run the code chunks, set eval = FALSE.

```r
knitr::opts_chunk$set(echo = TRUE, eval = FALSE)
```

```r
library(tidyverse)
library(tidymodels)
library(xgboost)
library(vip)
```

```r
train <- read_csv("../Data/train_class.csv")
test <- read_csv("../Data/test_class.csv")

train <- train %>%
  select(-'name') %>%
#  mutate(across('x2013_code', as.factor)) %>%
  mutate_if(is.character, factor)

test <- test %>%
#  mutate(across('x2013_code', as.factor)) %>%
  mutate_if(is.character, factor)
```

```r
#visualization that I want to come back to
```

Recipe

```r
xgb_recipe <- recipe(winner ~ ., data = train) %>%
  step_log('total_votes', offset = 1) %>%
  step_rm('id') %>% # Don't use these as predictors
  step_dummy('x2013_code') %>% # Make sure to normalize after encoding
  step_impute_mean(all_numeric()) %>%
  step_normalize(all_numeric()) # if we don't drop id then don't forget to avoid normalizing it with `a
```

```r
#no data preprocessing. The exports say no preprocessing, but takes times to tune
#I need to read up on exactly what each of these hyperparamaters do
#min_n, loss, and tree_depth has to deal with model complexity
#sample_size, and mtry are related to the randomness involved
xgb_spec <- boost_tree(trees = 1000,
           tree_depth = tune(), min_n = tune(),
           loss_reduction = tune(), sample_size = tune(),
           mtry = tune(),learn_rate = tune()
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

```r
#grid_regular method would take too long with this many tuned parameters
#grid_latin_hypercube evenly spaces out different models in the n(six in this case) dimensional space,

xgb_grid <- grid_latin_hypercube(
  tree_depth(),
  min_n(),
  loss_reduction(),
  sample_size = sample_prop(),
  learn_rate(),
  finalize(mtry(), train),
  size = 10
)
```

```r
xgb_wf <- workflow() %>%
  add_formula(winner~.) %>%
  add_model(xgb_spec)
xgb_wf

xgb_recipe_wf <- workflow() %>%
  add_recipe(xgb_recipe) %>%
  add_model(xgb_spec)
```

Cross validation

```r
set.seed(101)
vb_folds <- vfold_cv(train, strata = winner)
```

```r
#doParallel::registerDoParallel()

set.seed(202)

xgb_res <- tune_grid(
  xgb_wf, # Change this line to desired workflow
  resamples = vb_folds,
  grid = xgb_grid,
  control = control_grid(save_pred = TRUE, verbose = TRUE)
)
```

```r
xgb_res %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  select(mean,mtry:sample_size) %>%
  pivot_longer(mtry:sample_size,
               names_to = "parameter",
               values_to = "value") %>%
  ggplot(aes(value, mean, color = parameter)) +
  geom_point(show.legend = FALSE) +
facet_wrap(~parameter, scales = "free_x")
```

```r
#just helpful to see
show_best(xgb_res, metric = "roc_auc")
best_auc <- select_best(xgb_res, metric = "roc_auc")
final_xgb <- finalize_workflow(xgb_wf, best_auc)
```

```r
final_xgb
```

```r
#useless graph, too many predictors
final_xgb %>%
  fit(data = train) %>%
  pull_workflow_fit() %>%
  vip(geom = "point")
```

```r
final_res <- final_xgb %>%
  fit(data = train)
predictions <- final_res %>%
  predict(new_data = test)
```

```r
pred_table_2 <- bind_cols(test %>% select(id), predictions)

write_csv(pred_table_2, "xgb_predictions_tuned.csv")
```