Starcraft Skill Data - Exploration

Boosting Model Selection

Sergei Dakov

2023-07-19

Data Preparation

First, load in the libraries used in model selection:

```
library(tidyverse)
library(rsample)
library(parsnip)
library(recipes)
library(themis)
library(glmnet)
library(ggplot2)
library(ggmosaic)
library(yardstick)
```

Second, load in some helper functions:

```
source("model_selection_skeleton.r")
```

Next, load in the data.

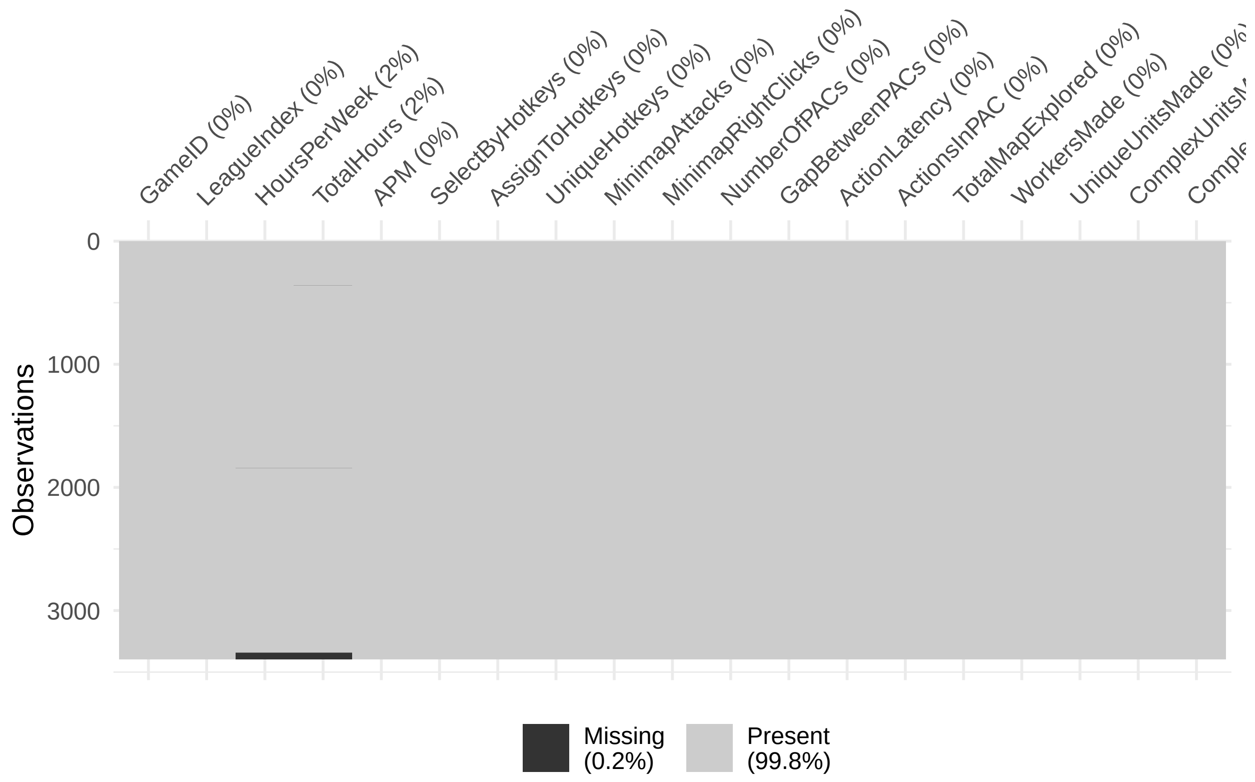
Some changes need to be made to be data so the model selection process works, such as converting the LeagueIndex column to a factor, and removing the age variable (as it is not a measurable metric by the game).

```
skillData <- read_csv("SkillCraft1_Dataset.csv") %>%
  mutate(LeagueIndex = factor(LeagueIndex)) %>% select(-Age) %>%
  mutate(across(c("HoursPerWeek","TotalHours"),~as.numeric(.x)))
```

```
## Rows: 3395 Columns: 20
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (3): Age, HoursPerWeek, TotalHours
## dbl (17): GameID, LeagueIndex, APM, SelectByHotkeys, AssignToHotkeys, Unique...
##
## 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.
## Warning: There were 2 warnings in ‘mutate()‘.
## The first warning was:
## i In argument: ‘across(c("HoursPerWeek", "TotalHours"), ~as.numeric(.x))‘.
## Caused by warning:
## ! NAs introduced by coercion
## i Run ‘dplyr::last_dplyr_warnings()‘ to see the 1 remaining warning.
```

Visualize the missing values in the data:

```
library(naniar)
vis_miss(skillData)
```



As both the single tree and the multinomial regression struggled with producing a strong predictor (in terms of accuracy) we need to try a more complex model to improve prediction quality.

#Boosting

the boosting algorithm improves prediction quality of a model by repeatedly fitting a model based on the errors of the previous step, thus each step of the algorithm strives to minimize the error of previous steps.

In this case, we are going to be using the XGBoost implementation of the boosting algorithm:

```
library(xgboost)
```

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

To start, we will have the algorithm repeat 20 times, which is large enough to see a proper effect, without making the model too cumbersome to run.

```
mod_boost <- boost_tree(mode="classification", trees = 20)
```

Perform the initial split of the data:

```
set.seed(1234)
reg_split <- initial_split(skillData,strata = "LeagueIndex")
regression_train <- training(reg_split)
regression_test <- testing(reg_split)
```

As a baseline we will use the predictions of the model with no optimization.

```
fit_boost <- mod_boost %>% fit(LeagueIndex~.,regression_train)
pred_boost <- predict(fit_boost,regression_test,type="prob")
pred_boost <- pred_boost %>% cbind(truth = regression_test$LeagueIndex)
roc_auc(pred_boost,"truth",starts_with(".pred"))
```

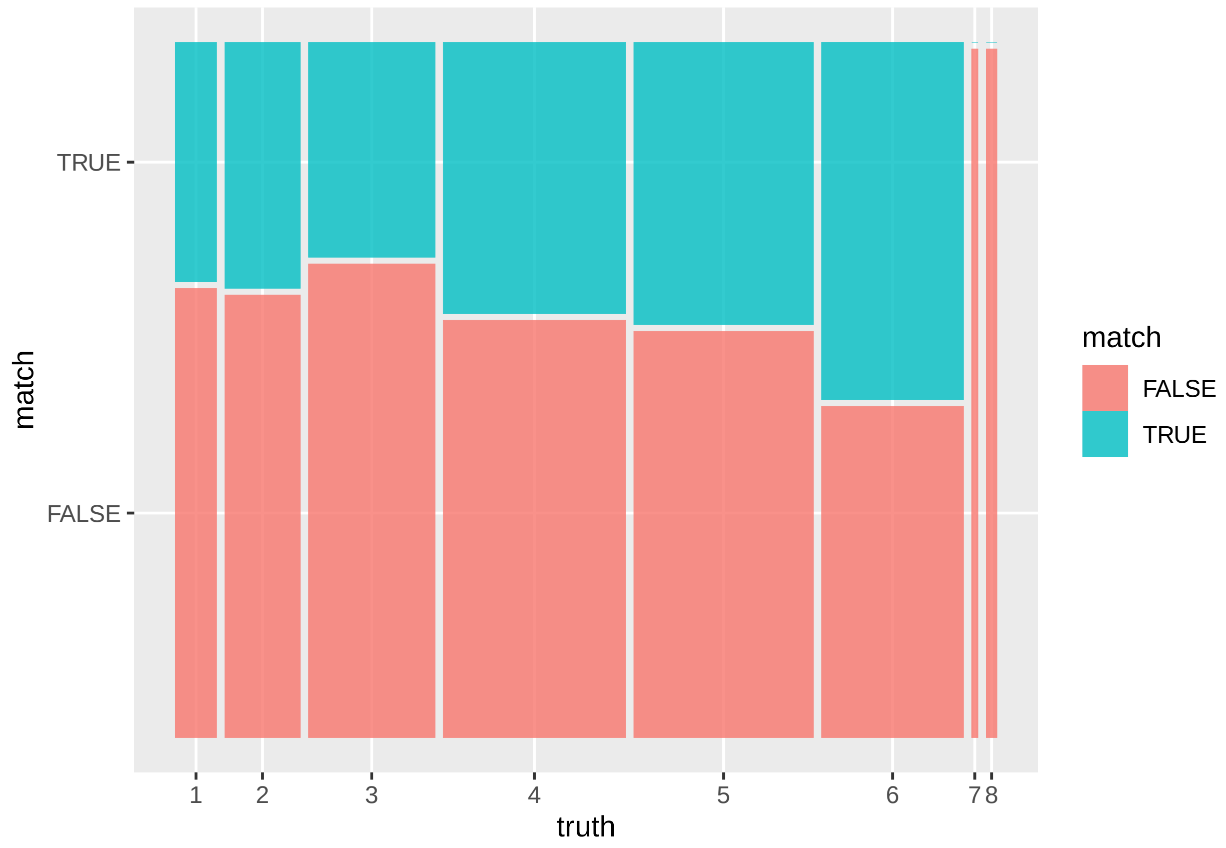
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.779
```

```
guess_boost <- predict(fit_boost,regression_test)
pred_boost <- pred_boost %>% cbind(guess=guess_boost$.pred_class)
accuracy(pred_boost,"truth","guess")
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.392
```

```
pred_boost <- pred_boost %>% mutate(match = (truth==guess))
pred_boost %>% ggplot() + geom_mosaic(aes(x=product(match,truth),fill=match))
```

```
## Warning: `unite_()` was deprecated in tidyr 1.2.0.
## i Please use `unite()` instead.
## i The deprecated feature was likely used in the ggmosaic package.
##   Please report the issue at <https://github.com/haleyjeppson/ggmosaic>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



The results are AUC of 0.779 and accuracy of 0.392

Model Optimisiation

We will use similar split ratio as in the tree model.

It is worth noting that since the data is not too rich, we will be using a smaller number of folds in the cross-validation as to not to stretch it too thin.

```
split_sizes <- c("imbalance"=900,"tuning"=644,"components"=1000)

set.seed(123)
imbalance_split <- initial_split(regression_train,strata="LeagueIndex",prop = ((split_sizes["imbalance"]
train_imbalance <- training(imbalance_split)
rest_split <- testing(imbalance_split)

set.seed(234)
tuning_split <- initial_split(rest_split,strata="LeagueIndex",prop=((split_sizes["tuning"])/nrow(rest_s
train_tuning <- training(tuning_split)
train_components <- testing(tuning_split)
```

Re-Sampling

Since trees do not suffer from the presence of missing values, we do not have to impute any data, But since trees might be weak to class imbalance we will test to see if re-sampling of the data can bring out effects drowned out due to sparsity.

```r
rec_nothing <- recipe(LeagueIndex~.,data=train_imbalance) %>%
  step_rm(GameID)

rec_upsample <- recipe(LeagueIndex~.,data=train_imbalance) %>%
  step_rm(GameID) %>%
  step_upsample(LeagueIndex,over_ratio=1,seed=111)

rec_downsample <- recipe(LeagueIndex~.,data=train_imbalance) %>%
  step_rm(GameID) %>%
  step_downsample(LeagueIndex,under_ratio = 1,seed=111)

rec_smote <- recipe(LeagueIndex~.,data=train_imbalance) %>%
  step_rm(GameID) %>%
  step_impute_mean(all_predictors()) %>%
  step_downsample(LeagueIndex,under_ratio = 1.5,seed = 111)%>%
  step_smotenc(LeagueIndex,over_ratio = 1,seed=111)

rec_puresmote <- recipe(LeagueIndex~.,data=train_imbalance) %>%
  step_rm(GameID) %>%
  step_impute_mean(all_predictors()) %>%
  step_smotenc(LeagueIndex,over_ratio = 1,seed =111)

set.seed(100)
cv_splits <- vfold_cv(train_imbalance,v=5,strata = 'LeagueIndex')


lst_recs <- list("nothing" = rec_nothing,
                 "upsample" = rec_upsample,
                 "downsample" = rec_downsample,
                 "smote" = rec_smote,
                 "only_smote" = rec_puresmote
)

cv_splits <- calculate_splits(cv_splits,lst_recs,mod_boost)
```

```
## Warning: No observations were detected in ‘truth‘ for level(s): ’7’
## Computation will proceed by ignoring those levels.
## No observations were detected in ‘truth‘ for level(s): ’7’
## Computation will proceed by ignoring those levels.
## No observations were detected in ‘truth‘ for level(s): ’7’
## Computation will proceed by ignoring those levels.
## No observations were detected in ‘truth‘ for level(s): ’7’
## Computation will proceed by ignoring those levels.
## No observations were detected in ‘truth‘ for level(s): ’7’
## Computation will proceed by ignoring those levels.
```

```r
cv_res_imbalance <- cv_splits %>% pivot_longer(cols=c(names(lst_recs)),names_to = "recipe",values_to =
  select(id,recipe,AUC) %>%
  group_by(recipe) %>%
  summarise (AUC = mean(AUC)) %>% arrange(-AUC)
```

```
cv_res_imbalance
```

```
## # A tibble: 5 x 2
##   recipe       AUC
##   <chr>      <dbl>
## 1 only_smote 0.848
## 2 smote      0.786
## 3 nothing    0.741
## 4 upsample   0.728
## 5 downsample 0.682
```

The best result was using smote without re-sampling AUC=0.848.

Dimensionality

Tree models naturally considers interactions due to its tiered structure, but sometimes the model may overfit due to the many features. We can lower this effect by lowering dimensionality

```
library(tune)

set.seed(100)
cv_splits <- vfold_cv(train_components,v=5,strata = 'LeagueIndex')

components <- c(seq(5,10,by=1),18)
components <- cbind.data.frame("num_comp" = components)




recipe_components <- recipe(LeagueIndex~.,data=train_components) %>%
  step_rm(GameID) %>%
  step_impute_mean(all_predictors()) %>%
  step_smotenc(LeagueIndex,over_ratio = 1,seed=111) %>%
  step_pca(all_predictors(),num_comp = tune())

formula_res <- mod_boost %>% tune_grid(object = mod_boost,
                                      preprocessor = recipe_components,
                                      resamples = cv_splits,
                                      metrics = metric_set(roc_auc),
                                      grid = components
)
```

```
## Warning: The ‘...‘ are not used in this function but one or more objects were
## passed: ’’
```

```
## > A | warning: No observations were detected in ‘truth‘ for level(s): ’8’
##                Computation will proceed by ignoring those levels.
```

```
## There were issues with some computations   A: x1
## > B | warning: No observations were detected in ‘truth‘ for level(s): ’7’
##                 Computation will proceed by ignoring those levels.

## There were issues with some computations   A: x1There were issues with some computations   A: x1   B
## There were issues with some computations   A: x1   B: x1
```

```
col_metric <- collect_metrics(formula_res)

col_metric %>% arrange(-mean) %>% head(10)
```

```
## # A tibble: 7 x 7
##   num_comp .metric .estimator  mean     n std_err .config
##      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        5 roc_auc hand_till  0.810     5  0.0164 Preprocessor1_Model1
## 2        6 roc_auc hand_till  0.797     5  0.0268 Preprocessor2_Model1
## 3        8 roc_auc hand_till  0.796     5  0.0218 Preprocessor4_Model1
## 4       18 roc_auc hand_till  0.795     5  0.0197 Preprocessor7_Model1
## 5       10 roc_auc hand_till  0.792     5  0.0247 Preprocessor6_Model1
## 6        7 roc_auc hand_till  0.786     5  0.0267 Preprocessor3_Model1
## 7        9 roc_auc hand_till  0.785     5  0.0269 Preprocessor5_Model1
```

```
lst_recs <- list("nothing" = rec_nothing)

cv_splits <- calculate_splits(cv_splits,lst_recs,mod_boost)
```

```
## Warning: No observations were detected in ‘truth‘ for level(s): ’8’
## Computation will proceed by ignoring those levels.

## Warning: No observations were detected in ‘truth‘ for level(s): ’7’
## Computation will proceed by ignoring those levels.
```

```
cv_res_components <- cv_splits %>% pivot_longer(cols=c(names(lst_recs)),names_to = "recipe",values_to =
  select(id,recipe,AUC) %>%
  group_by(recipe) %>%
  summarise (AUC = mean(AUC)) %>% arrange(-AUC)

cv_res_components
```

```
## # A tibble: 1 x 2
##   recipe    AUC
##   <chr>   <dbl>
## 1 nothing 0.716
```

Reducing the dimensions to 5 primary components has the best performance (AUC of 0.81 compared to AUC of 0.763 when not performing PCA).

Tuning

We can tune the tree structure for the boosting algorithm, including the number of iterations and number of variables to consider at each step in the tree model (similarly to a random forest).

```
set.seed(100)
cv_splits <- vfold_cv(train_tuning,v=5,strata = 'LeagueIndex')


num_trees <- expand.grid("trees" = seq(10,90,by=10),"mtry"=1:5)

recipe_final <- recipe(LeagueIndex~.,data=train_tuning) %>%
  step_rm(GameID) %>%
  step_impute_mean(all_predictors()) %>%
  step_smotenc(LeagueIndex,over_ratio = 1,seed=111) %>%
  step_pca(all_predictors(),num_comp = 5) %>%
  prep()

mod_tuning <-  boost_tree(mode="classification",mtry = tune(), trees = tune())
set.seed(100)
formula_res_2 <- tune_grid(object = mod_tuning,
                                       preprocessor = recipe_final,
                                       resamples = cv_splits,
                                       metrics = metric_set(roc_auc),
                                       grid = num_trees
)
```

```
## > A | warning: No observations were detected in ‘truth‘ for level(s): ’8’
##                 Computation will proceed by ignoring those levels.

## There were issues with some computations   A: x1
## There were issues with some computations   A: x1

##
```

```
col_metric <- collect_metrics(formula_res_2)

col_metric %>% arrange(-mean) %>% head(10)
```

```
## # A tibble: 10 x 8
##     mtry trees .metric .estimator  mean     n std_err .config
##    <int> <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     5    30 roc_auc hand_till  0.811     5  0.0197 Preprocessor1_Model39
##  2     3    20 roc_auc hand_till  0.811     5  0.0153 Preprocessor1_Model20
##  3     3    30 roc_auc hand_till  0.810     5  0.0175 Preprocessor1_Model21
##  4     1    30 roc_auc hand_till  0.810     5  0.0125 Preprocessor1_Model03
##  5     5    40 roc_auc hand_till  0.810     5  0.0189 Preprocessor1_Model40
##  6     4    30 roc_auc hand_till  0.810     5  0.0191 Preprocessor1_Model30
##  7     1    40 roc_auc hand_till  0.809     5  0.0132 Preprocessor1_Model04
##  8     4    20 roc_auc hand_till  0.809     5  0.0172 Preprocessor1_Model29
##  9     1    50 roc_auc hand_till  0.809     5  0.0141 Preprocessor1_Model05
## 10     3    40 roc_auc hand_till  0.809     5  0.0163 Preprocessor1_Model22
```

The best result was when using 20 trees, and using all the predictors each time. Mean AUC of 0.814.

#Final prediction

using the optimised model, we can make a prediction again:

```r
mod_boost <-  boost_tree(mode="classification", trees = 30, mtry = 5)

recipe_final <- recipe(LeagueIndex~.,data=regression_train) %>%
  step_rm(GameID) %>%
  step_impute_mean(all_predictors()) %>%
  step_smotenc(LeagueIndex,over_ratio = 1,seed=111) %>%
  step_pca(all_predictors(),num_comp = 5) %>%
  prep()

train_final <- bake(recipe_final,new_data = NULL)
test_final <- bake(recipe_final,new_data = regression_test)

fit_final <- fit(mod_boost,LeagueIndex~.,train_final)
prob_final <- predict(fit_final,new_data = test_final,type = "prob")
pred_final <- predict(fit_final,new_data= test_final)

res_final <- prob_final %>% mutate(truth=test_final$LeagueIndex,guess = pred_final$.pred_class)
roc_auc(res_final,"truth",starts_with(".pred"))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.851
```

```r
accuracy(res_final,"truth","guess")
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.353
```

The model brings us to a final AUC of 0.851, and accuracy of 0.353. Thus, the model selection process was indeed needed and useful.

Finally, we save the pre-processing steps required in a separate file to be accessed in the main summary:

```r
saveRDS(recipe_final,"boosting_model")
```