Starcraft Skill Data - Exploration

Sergei Dakov

2023-07-19

The Main Premise - Online Gaming Skill Rating

With advances in internet technology and infrastructure, online gaming is becoming more popular and attractive day by day. One of the main challenges game designers face when designing and maintaining a competitive online game is keeping each match challenging and evenly matched. To achieve this, most games use a skill based ladder system, where players position changes to reflect their skill in the game in order to assist the game to match evenly skilled players against each other each time. Most games rely on a variant of the Elo ranking system, in which players gain or lose points based on their performance against their opponent as well as the ranking of the opponent (beating a higher rated opponent nets more points than beating one rated lower than the player, for example).

While this method generally serves well for most one-on-one games, the system is harder to apply to team based games, as the system does not normally account for individual contributions to a team effort (in fact, it may generally be hard to quantify).

Additionally, the Elo system has a burn-in period for new players, causing some potentially uneven match-ups while the system calibrates to reflect the new players rating. finally, the system struggles to adapt to players taking prolonged hiatuses from the game, as it is hard to assess potential skill decay due to lack of active practice.

While the flaws in the system are not large enough to make that system non-viable, it would be beneficial to find additional ways to assess player skill and make the rating system more reactive and accurate.

Generally, there exist obvious noticeable differences between high and low skill players, these may be expressed in things such as: utilizing more complex strategies, utilizing strategies that require more complex actions, higher precision and confidence in performing actions, and more.

The ability to quantify a players skill based on objective measurable parameters can be used to quickly separate high skill and low skill players, as well as immediately compare a players performance before and after a hiatus to adjust for "rust" and lack of practice.

To test the viability of such an approach, we will use the Skillcraft data set (acquired through: http://archive.ics.uci.edu/dataset/272/skillcraft1+master+table+dataset), which collects data from multiple Starcraft 2 players, their age, in game rank as well as multiple in-game parameters.

The Data

```
## 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.
```

About the Game

Starcraft 2 if a real time strategy (RTS) game, published by Blizzard Entertainment. In it each player is tasked with gathering resources, building a base and producing units with the purpose of defeating all other opponents on the map (by destroying their bases). The game offers multiple competitive online modes, the most popular being the 1v1 mode. As it is the most popular mode, this mode is the main focus of data and the original research.

About the Data

Originally the data was collected for a paper by Thompson et al. in 2013 that explored the decay of cognitive-motor functions with age. The data includes multiple factors which are considered by most of the Starcraft community as indicators of skill (such as: Actions Per Minute, usage of complex units, hotkeys and command groups).

Purpose of Analysis

We will use the data set with multiple prediction models to attempt predict the ladder placement of players based on their in game performance metrics. As a secondary goal, we will compare the performance of the different models, and aim to compare the different models based on their performance vs. their complexity.

We will focus on the ROC-AUC metric (One vs. Rest setting) as the main parameter as the main metric for the comparison, as it is the best metric for comparing the overall quality of a classification model. Accuracy will be used as a secondary metric for comparison as in a real world prediction setting accuracy would be the main point of interest

The Models

In this project we will be performing analysis using the following models: - Linear Regression: as the data uses numeric encoding for the different ranks, we will attempt to treat them as a numeric parameter, predict it using standard liner regression, and then use a rounding decision rule to return back to whole value to be compared to the relevant class.

- Logistic Regression: as the task is a classification type problem, logistic regression is a prime candidate to be a useful yet simple model. Since the target parameter has more than two possible value, multinomial logistic regression will be used.
- Decision Trees: for the purpose of prediction, a decision tree makes for another relatively simple model, with the benefit of it having minimal assumptions about the data.
- Boosting: a more complex algorithm, but one that iteratively strives to minimize prediction error. This is the most complex of the models, but usually the one with strongest prediction ability.

Linear Regression

To perform linear regression on the data, first we convert the classes (1-8) to numerical values, this of course comes with the implicit assumption that the different ranks are equidistant from each-other (that is, the difference in skill between each to adjacent ranks is the same).

Of course, linear regression can produce non whole numbers, which cannot be easily converted back to classes, thus a rounding rule needs to be selected to deal with such values.

While the linear regression model is not the ideal model to deal with this type of data; it is simple and easy to compute, thus can serve as a good baseline for comparison.

Data Processing

- The data has some missing values; these values will need to be imputed or removed, as the linear regression model cannot handle missing values.
- Many of the variables are not normally distributed (which is an assumption of the linear model), thus the Yeo-Johnson transformation will be used.
- Several possible magnitudes of interaction have been tested

(Detailed model selection can be viewed in a separate file here).

```
#Load the preprocessing steps
regression_rec <- readRDS("regression_model")
# Define the model
mod_reg <- linear_reg(engine="glm",penalty=0.1)

#the linear regression model required separating out the missing values and forcing them into one class
regression_test_sep <- regression_test %>% drop_na()
regressoin_test_rest <- regression_test %>%
  anti_join(regression_test_sep) %>%
  select(LeagueIndex) %>%
  mutate("prediction"=8)
```

```
## Joining with ‘by = join_by(GameID, LeagueIndex, HoursPerWeek, TotalHours, APM,
## SelectByHotkeys, AssignToHotkeys, UniqueHotkeys, MinimapAttacks,
## MinimapRightClicks, NumberOfPACs, GapBetweenPACs, ActionLatency, ActionsInPAC,
## TotalMapExplored, WorkersMade, UniqueUnitsMade, ComplexUnitsMade,
## ComplexAbilitiesUsed)‘
```

```
regression_train_sep <- regression_train %>% drop_na()

#make the prediction
train_final <- bake(regression_rec,NULL)
test_final <- bake(regression_rec,new_data = regression_test_sep)
fit_final <- fit(mod_reg,as.numeric(LeagueIndex)~.,train_final)
predicted_final <- predict(fit_final,test_final)
predicted_final <- cutoff(predicted_final$.pred,0.354)
predicted_final <- bind_cols("LeagueIndex" = regression_test_sep$LeagueIndex,"prediction"=predicted_fina
mean(predicted_final$prediction==predicted_final$LeagueIndex)
```

```
## [1] 0.39953
```

```
OnevRest(predicted_final,"LeagueIndex","prediction")
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use ‘all_of()‘ or ‘any_of()‘ instead.
##   # Was:
##   data %>% select(guess)
##
##   # Now:
##   data %>% select(all_of(guess))
```

```
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## [1] 0.6841414
```

```
#create a collected AUC vector, to compare different models later
AUC_collected <- c("Regression" = OnevRest(predicted_final,"LeagueIndex","prediction"))


# create a dataframe of predictions to compare different models later
results_collected <- predicted_final %>% arrange(LeagueIndex) %>% rename("regression_guess" = predictio
```

Using the final selected model for linear regression, the accuracy of the model is 0.400, and an AUC of 0.701
.

Multinomial Logistic Regression

As the problem is a classification problem, we can try predict the probability that each data point belongs in each class (rank in game) using multinomial regression.

While this model is somewhat more complex than linear regression, it is still simple enough for run-time not to be a major issue.

Data Processing

- Similarly to the linear regression model, missing data was imputed and data was normalized using a Yeo-Johnson transformation.
- Multiple settings of interactions were tested.
- Due to the scarcity of some of the classes, options of rebalancing the data set have been tested.

The full model selction process can be viewed here.

```
mult_reg <- readRDS("multinomial_model")

mod_multi_final <- multinom_reg(mode="classification",engine = "glmnet",mixture = 0,penalty=0.006)
fit_final <- mult_reg %>% bake(new_data=NULL)
fit_final <- fit(mod_multi_final,LeagueIndex~.,fit_final)
pred_final <- predict(fit_final,new_data=bake(mult_reg,new_data=regression_test),type="prob")
pred_final <- pred_final %>% mutate(truth = regression_test$LeagueIndex)
pred_guess <- predict(fit_final,new_data=bake(mult_reg,new_data=regression_test))
pred_final <- pred_final %>% mutate(guess = pred_guess$.pred_class)
roc_auc(pred_final,"truth",starts_with(".pred"))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.867
```

```
accuracy(pred_final,"truth","guess")
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.438
```

```
AUC_collected <- c(AUC_collected,"Miltinomial" = roc_auc(pred_final,"truth",starts_with(".pred"))$.esti


pred_multi <- pred_final %>% arrange(truth) %>% pull(guess)

results_collected <-results_collected %>% bind_cols("multinomial_guess" = pred_multi )
```

For the best selected model the values are AUC of 0.867, and accuracy of 0.438 .

Decision Tree

As shown by the previous regression models, the data does not naturally conform to many of the assumptions of the models.

The tree model has little assumptions regarding the data, and can even handle missing values without additional processing.

Data Processing

- To avoid information getting "overlooked" due to imbalance in data, different approached to rebalancing were tested.
- PCI and dimensionality reduction were tested.

(full model selection can be viewed here)

```
tree_rec <- readRDS("tree_model")

mod_tree <- decision_tree(mode="classification",cost_complexity = 0.01)
train_final <- bake(tree_rec,new_data = NULL)
test_final <- bake(tree_rec,new_data = regression_test)

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

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

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.835
```

```
accuracy(pred_final,"truth","guess")
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.317
```

```
AUC_collected <- c(AUC_collected,"Tree" = roc_auc(pred_final,"truth",starts_with(".pred"))$.estimate)


pred_tree <- pred_final %>% arrange(truth) %>% pull(guess)

results_collected <-results_collected %>% bind_cols("tree_guess" = pred_tree )
```

The final performance of the tree model is: AUC = 0.835, Accuracy = 0.317 .

#Boosting

The boosting algorithm operates iteratively on the data, attempting to minimize the error of each previous step. Computationally this is the most time and resource consuming of the models explored here, ideally the complexity/precision trade-off being significant enough to make it worth while.

Data Processing

- As the boosting model is based on a tree model, the same processing steps are performed.
- The complexity of the boosting process itself is tuned.

(full model selection can be viewed here).

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

boosting_model <- readRDS("boosting_model")

train_final <- bake(boosting_model,new_data = NULL)
test_final <- bake(boosting_model,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)

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

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

```
accuracy(pred_final,"truth","guess")
```

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

```
AUC_collected <- c(AUC_collected,"Boosting" = roc_auc(pred_final,"truth",starts_with(".pred"))$.estimat

pred_boost <- pred_final %>% arrange(truth) %>% pull(guess)

results_collected <-results_collected %>% bind_cols("boosting_guess" = pred_boost )
```

The resulting AUC is 0.851, and accuracy of 0.353 .

Comparison of Results

Collating the results of all the models together we can see that none of the models performed exceptional well, and the more complicated models have not justified the investment.

```
AUC_collected
```

```
##  Regression Miltinomial        Tree    Boosting
##   0.6841414   0.8668581   0.8350672   0.8513702
```

We can see that the multinomial model performed the best, followed by the boosting model. But as the multinomial model is both simpler and faster, there is no justification in using a more complex model.

It is worth noting that all the models do outperform the naive estimators of uniform guessing and selection of the mode, thus the models do have some merit to them.

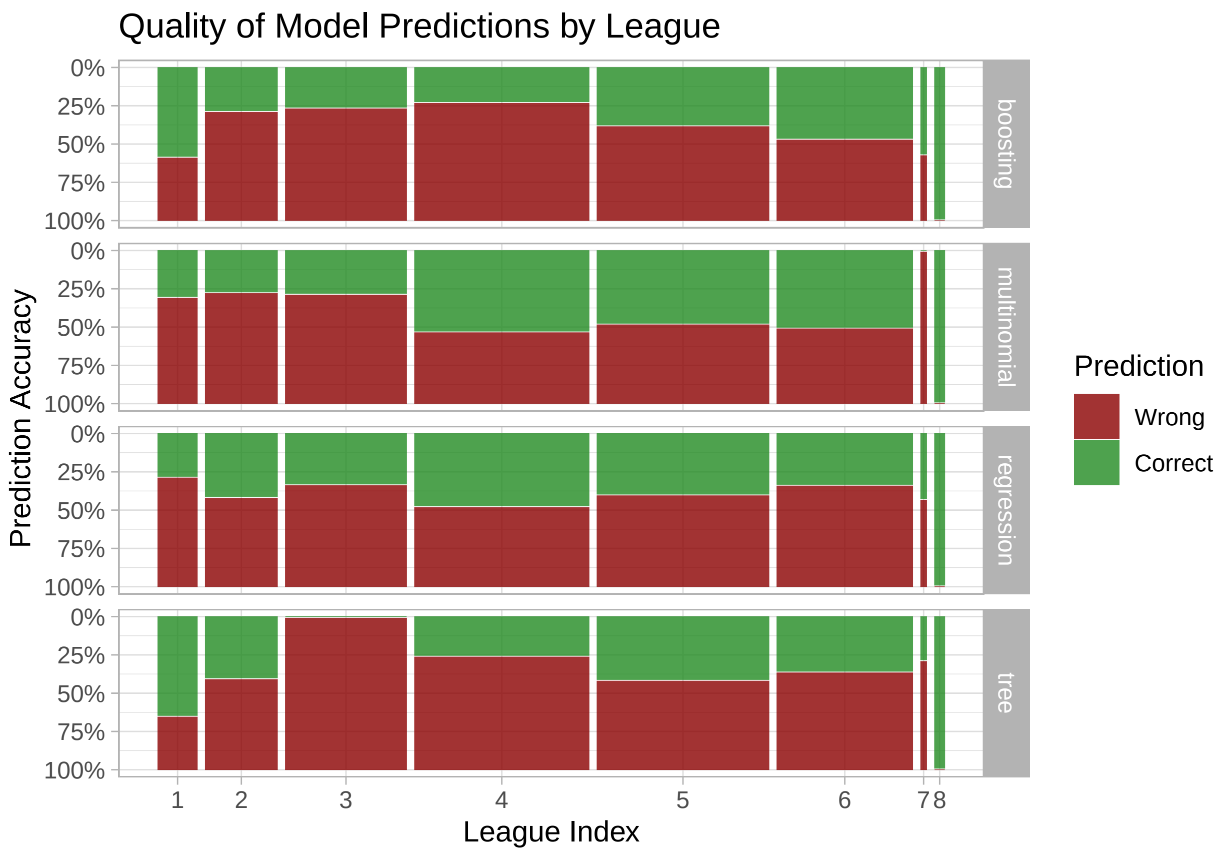
By plotting the prediction accuracy of each of the models we can also analyze the strengths and weaknesses of each model:

```
library(ggplot2)
library(ggmosaic)



results_collected %>%
  mutate(across(c(1,3:5),~as.numeric(.x))) %>%
  pivot_longer(cols = ends_with("guess"),names_to = "model") %>%
  mutate(model = str_remove_all(model,"_guess")) %>%
  mutate(match = LeagueIndex==value) %>%
  ggplot() +
  geom_mosaic(aes(x=product(match,LeagueIndex),fill=match)) +
  facet_grid("model")+
  labs(x="League Index",y="Prediction Accuracy",title = "Quality of Model Predictions by League",fill="
  theme_light() +
  scale_fill_manual(labels = c("Wrong","Correct"),values = c("dark red","forest green"))+
  scale_y_continuous(breaks=c(0,0.25,0.5,0.75,1),labels =c("100%","75%","50%","25%","0%") )
```

```
## 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.
```



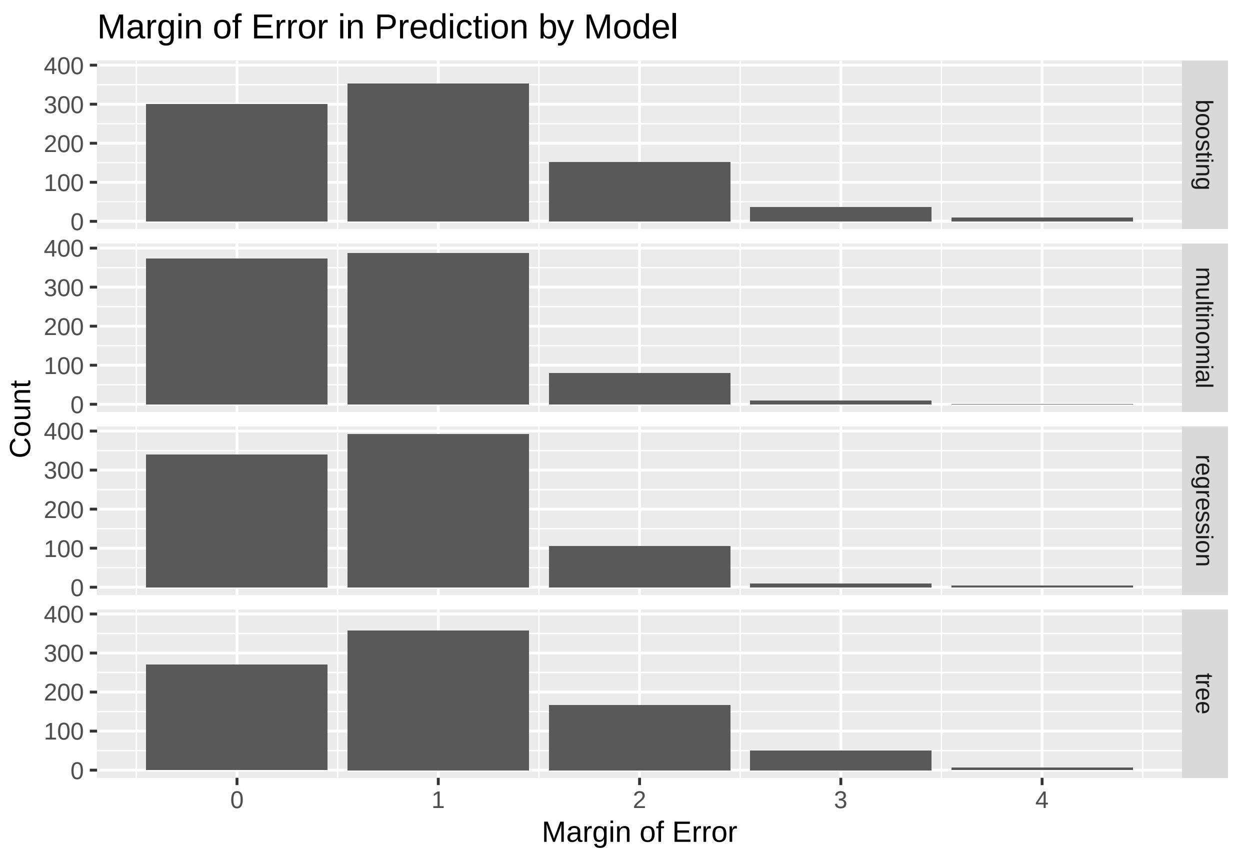
The multinomial model does well with the most populated classes, and gets all the "pro players" correctly, struggling more with the lower skill brackets.

The linear regression preforms quite well in the center, but struggles much more towards the edges, likely due to the non linear relation between the predictors and the outcome variable. The tree, and especially the boosting performed the best on edge cases while struggling more towards the center, with the boosting model predictably improving on the single tree model.

From these results, we can speculate that it is possible to create an ensemble model of the boosting and multinomial regression models, giving them weighted confidence levels depending on the range of prediction (if the models predict a middle level player, the multinomial prediction takes precedence; while the boosting model takes precedence in the edges).

We can also inspect the margin of error frequency of each of the models. Since the models all perform somewhat similar, selecting the model that has the smallest margin of error is the optimal choice:

```r
results_collected %>%
  mutate(across(c(1,3:5),~as.numeric(.x))) %>%
  pivot_longer(cols = ends_with("guess"),names_to = "model") %>%
  mutate(model = str_remove_all(model,"_guess")) %>%
  mutate(difference = abs(LeagueIndex-value)) %>%
  ggplot() +
  geom_bar(aes(x = difference)) +
  facet_grid("model") +
  labs(x="Margin of Error",y="Count",title = "Margin of Error in Prediction by Model")
```



In this case, we can see that the tree and boosting models tend to have the largest margin of error, while both the regression and multinomial regression are mostly off by a single rank with very few large errors.

Overall, the multinomial logistic regression model perform the best, while the tree model struggled the most. Also, while not precise all models were good at estimating the rough skill level of a player (it is uncommon to see a high skill player predicted to be of low skill or vice versa).

Conclusion

While all of the models have outperformed the naive estimators, none of the models has performed exceptionally well. We can infer that there is a correlation between these parameters and player skill level, but it is not strong enough to base decisions on it completely.

some possible explanations for this may be that there are more parameters that are not handled by the models, but a more likely explanation is that skill level in a game is governed by more than just measurable mechanical skill, and includes hard to gauge parameters like game knowledge, awareness and strategical thinking.

Nonetheless, the fact that the models margin of error is not very large even for failed predictions shows that it is possible to use easily obtainable metrics to roughly estimate a player's skill level; thus, a purely mechanical method can be used for the initial seeding of a new player joining the ladder (such metrics can be obtained from some practice or tutorial matches before the player joins the matchmaking system) and then further fine tuned using conventional methods.