Predicting League of Legends Ranked Match Outcomes With Machine Learning

Jihan Yin
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720
jihan_yin@berkeley.edu

Abstract

In this project, we attempt to use machine learning algorithms to predict the outcome of a ranked match before it begins. Our dataset consisted of 70 features that we deemed likely to have an effect on the match's result, collected as public information available to the player in champion select. With this information, we were able to predict a ranked games' result with an accuracy of 60%. We then tested our model's accuracy on subsets of features to discover important factors impacting the game. We found that the ad carry role has the least impact on determining the winner of a match, with the top role being the most inconsistent. We also found that the champion's prowess in the meta does not have as important a role as a player's effectiveness on the champion.

1 Introduction

1.1 League of Legends

League of Legends is a multiplayer online battle arena (MOBA) video game developed and released by Riot Games in 2009. In the standard mode, ten players queue up to play a 5v5 match, with five roles per team and each player selecting their own character (champion). Since its inception, League of Legends has grown into a massive international eSport, with competitive leagues in multiple continents and tens of millions of players and viewers across the world. As hundreds of millions of dollars being invested into the North American League this year, all eyes are on League of Legends, representing the future of eSports.

1.2 Ranked Match Prediction

In League of Legends, players can queue up to play five versus five competitive ranked games. Each player is assigned one of five roles, and can choose their own champion. After every champion has been selected, there is a brief thirty second delay before the game begins, where one can leave the game to avoid playing. Although the system tries its best to create even games, once champions have been selected by all ten players, a common strategy is to look up statistical data on the players and champions to determine how winnable a game is. Thus, applying machine learning models on statistical features can help us discover patterns between such data and ranked game outcomes, allowing one to avoid playing a heavily unfavored match and loss of rank.

2 Data Collection and Methods

2.1 Data Collection

Our dataset is generated from the public APIs of Riot Games[2] and Champion.gg[3], consisting of 14 features corresponding to each of the 5 roles in League of Legends, for a total of 70 features. These features are derived from the specific champion choice of each role and the player statistics in relation to that champion, which we considered impactful on game results. Our data was gathered through crawling through recent ranked matches and gathering data on each match deemed recent enough, generating one sample for each team. The reasoning behind this is that some features we seek are not shown in the match data, and must be found through lookup of player data. Thus, to stay accurate, we only look for very recent matches so our player data is synchronized with the match data. We generated a total of 3400 working training samples to work with.

Role-specific Features (5 per sample)	**Description**
champion_winrate*	Average win rate of the champion
champion_playrate*	Play rate of the champion
champion_percentRolePlayed*	Play rate of the champion specific to its role
champion_overallPerformanceScore*	Champion.gg's champion performance score
champion_matchupWinrate*	Average win rate of the champion in its role matchup
wins_last2	Player win count in the two most recent games
wins_last15	Player win count in the 15 most recent ranked games
gamesPlayed	Player game count on champion in last 300 games
gamesPlayedRanked	Player game count on champion in last 300 ranked games
championPoints	Player champion mastery points
summonerLevel	Player summoner level
rank	Player rank, calculated numerically
lanePlayed	Player game count on lane in last 300 games
lanePlayedRanked	Player game count on lane in last 300 ranked games

***Tier and role specific**

Table 1 - The features for every role and their descriptions

There were potential problems with some samples, as Riot's algorithm for determining the role of a player in their API is not very accurate. To account for this, matches that did not contain all five standard roles were discarded. Furthermore, in some cases, a champion would be played in a role where not enough total games exist to accurately determine its statistics. The data values for the features of this champion would then be NaN, which we accounted for by setting all NaN values to the mean value of their respective feature. We also tested considering only those samples which contained no NaN values in our model selection.

2.2 Feature Importance

Besides modeling ranked match outcomes, we are also interested in determining the importance of certain sets of features and their impact on the game. As such, we would train our models on only the set of features we were interested in, compare their performances. Our motivation is to find out which roles have the most impact on the game, the importance of choosing high win rate and popular champions, and the significance of the player's skill in deciding the outcome of a game.

3 Result

3.1 Model Selection

We leveraged our data on support vector machines, gradient boosting, deep neural networks, random forests, and logistic regression algorithms for binary classification of ranked game outcomes (win/loss). For each model, we performed grid search to find the optimal hyper-parameters, using a randomized 70% of our data as training, with 30% for validation. These are the results after averaging out NaN values.

	SVM	**Neural Net**	**Random Forests**	**Gradient Boosting**	**Logistic Regression**
Sensitivity	56.75%	56.04%	62.86%	63.90%	55.21%
Specificity	53.57%	55.76%	57.70%	57.81%	52.46%
Validation	55.14%	55.91%	60.24%	60.83%	53.80%

Table 2 - Results after training different models with replacing NaN values

These are the results after removing samples which contained NaN values.

	SVM	Neural Net	Random Forests	Gradient Boosting	Logistic Regression
Sensitivity	60.56%	58.26%	62.45%	63.28%	59.95%
Specificity	56.91%	56.73%	57.94%	58.63%	56.52%
Validation	58.72%	57.50%	60.14%	60.93%	58.19%

Table 3 - Results after training different models with removing NaN values

In order to determine if we have enough data to avoid underfitting, we plotted each our of models' average validation accuracies over random subsamples of our data of increasing size. These are the resulting plots:

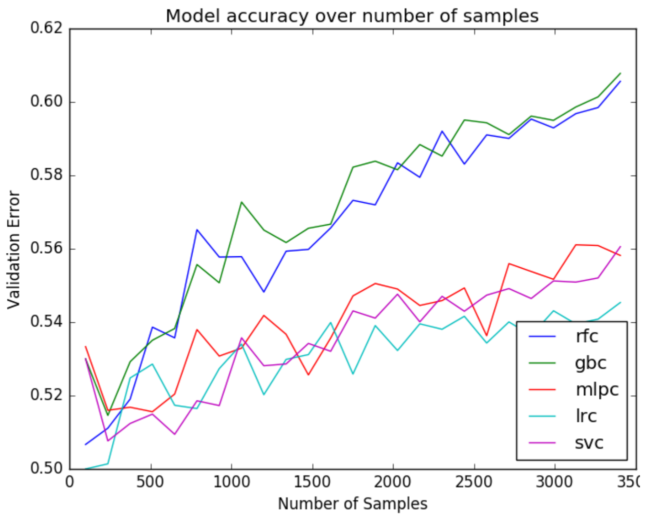

Table 4 - Graph of model accuracy over dataset size for NaN averaging

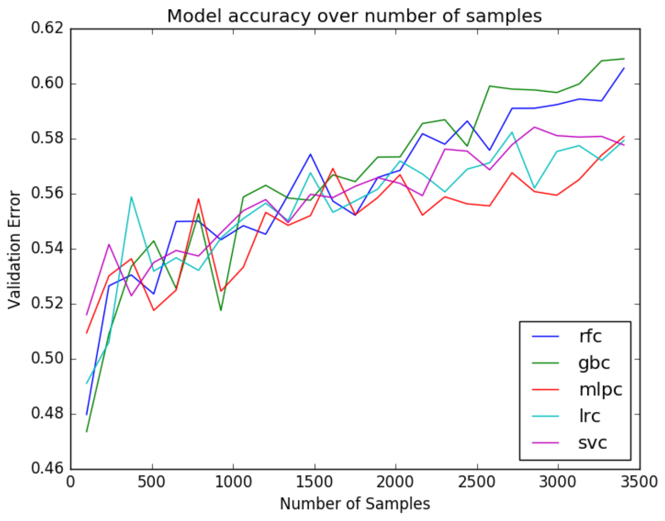

Table 5 - Graph of model accuracy over dataset size for NaN removal

To see potential bias in our data due to uneven rank collecting, we found the percentages of our dataset that consisted of each tier. These are shown in table 6.

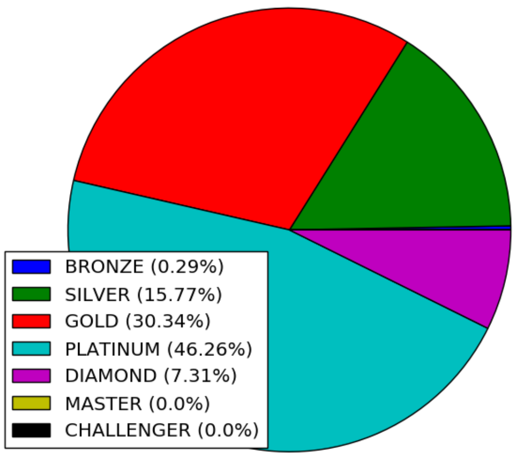

Table 6 - Pie graph of data based on rank tiers

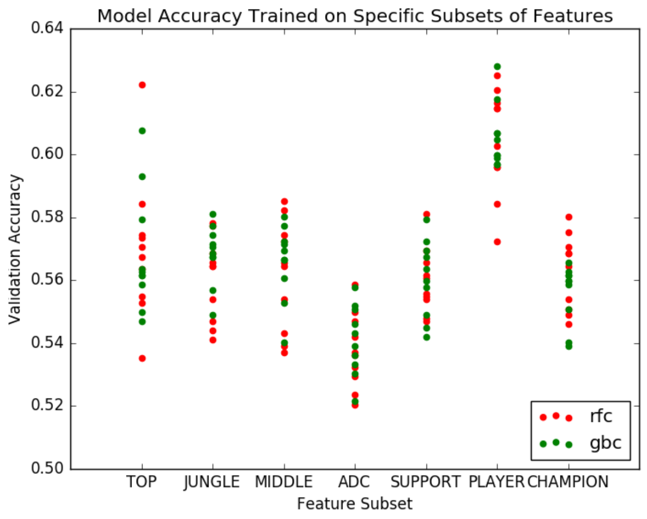

Table 7 - Graph of model accuracy over dataset size for NaN averaging

3.2 Feature Importance

Using our random forests and gradient boosting classifier, we trained on specific subsets of features. We ran twenty samples for each subset, with ten for each model. Our results are shown in table 7.

4 Conclusion

4.1 Model Selection

While SVM, neural nets, and logistic regression improves massively from training on non-NaN samples, gradient boost and random forests see almost no difference in effectiveness. Even with the increase in accuracy, the other models' performance does not match that of gradient boost and random forests. It is also interesting to note that each model performed better on identifying wins rather than losses. As we trained on a dataset with 50% wins and 50% losses, our models may have a higher tolerance to predicting wins.

In table 4 and 5 can see that all of our models are plateauing in performance around 2000 samples, so our dataset size of 3400 samples is sizable enough to get an accurate representation of the overall sample space. Furthermore, we can see that random forests and gradient boosting models outperform the other models for both NaN replacement schemes, and as such we will be using the two aforementioned models in determining feature importance. It is interesting to note the performance improvement of neural net, SVM, and logistics regression models on samples not containing NaN. Random forests and gradient boosting performed around the same for both schemes, however. Thus, for determining feature importance, we will average NaN values, as this provides us more data to work with. All NaN values pertained to champion-specific features, so perhaps the three models with improvement are much more sensitive for those features than random forests and gradient boosting.

As we crawled matches through a platinum-ranked account as a seed, our data may have a bias for ranks near platinum. Indeed, table 6 shows that the majority of our data is from platinum and gold, with almost no data on bronze, master, and challenger. Because of this, the conclusions of this study are limited to the aforementioned ranks.

4.2 Feature Selection

We ran random forests and gradient boost classifiers on the entire dataset with averaging out NaNs, only on a specific subset of features. As we can see from table 7, our models seem to work best when trained only on specific player information, doing significantly better than if trained only on champion specific information. This implies that overall champion strength in the meta is not as important to a match as a player's experience playing on said champion. Furthermore, we can see that training on the top role only has the most variation in performance, whereas training on the ad carry role only has significantly worst performance. This implies that the ad carry role has much less impact on a game as the other roles, with top lane having the most variation in effectiveness.

4.3 Future Extension

All of the data for this study was gathered from matches near the end of the preseason. As such, there could be a potential bias in our models, and a repeat of this study a month into the season would reveal such biases. Furthermore, as we sampled unevenly from different ranked tiers, we could get a more wholistic view by sampling evenly from each tier.

With future updates to Riot's API, one could gather more accurate data for each match to generate samples with less noise. Currently, one must manually calculate a player's win rate on a champion, by requesting information for each match. As this would take too long for this study, we only found the win rate for the last 15 games played. Furthermore, more features could be generated for more accurate predictions. We can find each player's experience playing with and against certain champions by looking in their past games, as well as their average match stats, such as KDA, damage dealt, and wards placed. We can also condition our champion features on how many games the player has on the champion, which would take more request time.

With our already trained and stored models, one could create an application where the user can input the details of their current match, such as their teammates' usernames and champion picks. The application would then use this input to create a sample point, which our models can use to predict a probability of the match being a win or a loss. With a high enough probability of a loss, the user can dodge the match and save time as well as rating.

References

[1] Project Source. https://github.com/arilato/ranked_prediction
[2] Riot Games API. https://developer.riotgames.com
[3] ChampionGG API. http://api.champion.gg