League of Legends Win Predictor

Kyle White

Michigan State University  
4115 Naubinway rd.  
Okemos, MI 48864  
whiteky8@msu.edu

**ABSTRACT**

The goal of the project was to predict the winner of a match in the video game League of Legends. A match in League of Legends, ‘League’ for short, has two teams of five playing against each other. All ten players are on the same map, and play a character that can move anywhere on the map. The two teams start on opposite sides of the map, and the goal is to destroy the other team’s base to win the game. In order to do this, you have to destroy a series of turrets to get to the base. Players can ‘kill’ players on the other team, but the player that died will respawn in the next 20-60 seconds. So, to win a game, one team needs to kill the other team so that they have time to destroy the turrets and eventually get to the base and destroy it.

If the model preforms well enough, the project can contribute to the League community by giving players a prediction on whether or not they will win a game. The model could help a player decide what champion to play based on the model’s prediction of whether or not they will win with that champion. The model may also be a starting place for someone to create a more advanced model that is more accurate, and be used by professional players.

The model used to predict the winner is a decision tree. The results of the project show that a win predictor for League is possible, but requires an advanced model that requires a tremendous amount of data, as well as a lot of computing power.

**Keywords**

League of Legends – the name of the video game that the model makes predictions about.

Riot Games API – the API used to collect data from the game.

Champions – list of characters that can be played in a match, there are currently 134 champions to choose from.

Role – The role the player is in for the match. Each team has five different roles.

Rank – each player has a rank, and winning a game increases your rank, while losing decreases your rank.

Challenger – the highest rank possible, the top 200 players in the League of Legends.

Master – the second highest rank, the next 600 best players.

Summoner – same thing as a player, persons account name

# INTRODUCTION

Each match in League starts with the 10 players picking a champion to play for the current match, they can pick any champion and the pick is only for one match. This is a very important part of the match, because some champions are better than others, and having the right combination of champions one a team. If a player picks one of the weakest champions in the game, their team has a lower chance of winning the game. Giving players a tool to see how different champions will affect their chances of winning could help players win more.

Riot Games provides an API that can be used to get data from matches, player data, and general game data. [1] The API gives access to most of the useful data from games, but it does have its drawbacks.

Since League is such a popular game, there are many websites that use the Riot Games API to generate statistics for players. One site, champion.gg, provides a very large amount of summary statistics for each champion. Another site, OP.gg, as statistics for each player. It summarizes stats for each champion a player plays, show a list of their recent matches, and can even show if the player is currently in a match. But, oddly enough, there are no major websites that predict the winner of a match. So, while websites using the Riot API are common, there is not a website that predicts the winner of a game.

The project was very challenging for multiple reasons, mainly data collection. The Riot API doesn’t make it easy to collect data for thousands of matches. The best way to collect data is to start with a player, and crawl through their match history, saving data for each match, and then get matches from the players in that match. There is also a rate limit for the API, set at 10 requests every 10 seconds, and 500 requests every 10 minutes. Because of the rate limit, collecting enough data required hours to days of time running. Another problem was receiving errors on requests to the API, which can cause the program to crash. To prevent the data collection program from failing, the script has a function wrapper to skip certain errors.

The accuracy of the model is 52%, which is not very good considering random guessing should be able to get 50% of matches right. Although the model isn’t very successful, it is most likely due to a poor selection of features used to pick the winner. The project is still useful though, because the methods used to collect and process the data can be reused with a different, bigger set of features to get better accuracy.

# PRELIMINARIES

The goal of the project is to correctly predict the winner of a match before the match starts. The data is used is in the form of a comma separated values file. Each line is a match, containing 52 columns. The first column is the match ID, which isn’t used in predicting the winner, but used to identify the match so it can be paired with other potential data. The last column is the classifier, marking which team won the match. The other 50 columns are for the 10 players, the first 25 for team one, and the second 25 for team two. Since a team has five roles, each team is ordered by row, so 2-6 is always one role, 7-11 is another and so on. For each role, there are five columns, kills, deaths, assists, gold, and win percent. The values for each statistic is inserted from the average values for the champion played in that role.

The predictive model is created using a decision tree classifier method. Random Forest, Bagging, and AdaBoost were also used to in an attempt to make the model more accurate. But, the model has proven to not predict the winner of a match very well at all.

# METHODOLOGY

Creating the csv file which each match on one line took an extensive amount of data collection and preprocessing. Multiple steps were used to collect data and transform it into a useful format. A summary of the order of data collection is:

1. Get a list of the summoner ID of the top 800 players.
2. Get a list of match IDs of the last 200 matches of each player.
3. For each match ID in the list, get statistics for each player in the match.
4. Create a file with the average of each champion in each role.
5. Create final file with each match on one line, and file in columns with data from the file with champion averages.

I decided to collect data from matches of the players from Challenger and Master ranks. Using players of the same skill level should make the average stats of the champions based more on how good the champion is, not how good the player is. The original lane for the project was to make predictions based on the players and stats that they have for each role. However, this proved to be too difficult, because players rarely play with the same people in multiple games, so the data set needed would be extremely large and would need to be constantly added to.

The first step in data collection was getting a list of the top players. There is a python library called Cassiopeia [2] that provides methods that make calling the API easier. Instead of creating URLS to call the API, the library has methods that will do it, as well as making the data returned easier to access. One of the methods returns a list of players from the Challenger and Master ranks. I used this to create a csv with the player ID of every player.

With the file of the player IDs, another program iterates through the list, and gets the match history of each player. IT goes through the match history and gets the match ID of the most recent 200 matches. If a match has already been added to the list, it is not added again. After going through each player there are around 70,000 match IDs. This process took around 5 hours to collect all the match IDs.

If every player never played with each other, there would be 160,000 match IDs. 70,000 indicates that each match had an average of 2 players from the data set. This means, that to do the predictions based on stats for a player, the dataset would have to increase by a factor of 5. But then, the new games would have players in them that aren’t in the data set so more players would need to be added. This would continue, showing that it would be very difficult to do predictive analysis based on the players stats.

With the list of matches, another python script gets all the information from the game. It stores stats for each player in the match on a line, ten lines for each match. Each line has the match ID, player ID, team ID, rank, role, champion, kills, deaths, assists, gold, win, and time when match was played. Some fields weren’t used, like rank, because it returned the rank of the player from last year, which is outdated so it doesn’t have much meaning.

Based on how long it took to collect a smaller number of matches, it would take around two days to collect data for all of the matches. In order to speed up the process, I got four more API keys. Then I split the list of match IDs into five sections, and ran five instances of the script at the same time. This allowed all of the data to be collected over night. With data from all of the matches collected, preprocessing was needed.

There is actually another game type in League of Legends that is 3v3. This game type is not very popular and is played less than one percent as much has the main game type. Since the game type is so unpopular, it is not included in the predictive model at all. However, about 1% of the matches collected were from this 3v3 game type. These games were removed from dataset using a pandas data frame. The data was taken from a csv and put into a data frame, then a frequency count was added to each record that counted the number of times the match ID was in the data. Any record with a number other then 10 was removed from the data.

Another preprocessing step is needed. There is a flaw in the API where it doesn’t return the role the plyer was assigned before the match. Instead, it uses an algorithm to predict what role the player is based on other stats in the game. Sometimes, the algorithm doesn’t work, so there aren’t the 5 different roles on each team. Matches where this happened where removed by skipping matches that couldn’t find the 10 correct roles. After collecting all of the data, two new files were created during experimental evaualtion.

# EXPERIMENTAL EVALUATION

The model used to predict the winner is a decision tree. Experiments run on the model consisted primarily of changing the size of the tree and the size of the training and test data.

## Experimental Setup

The initial data was from 720,000 games, and each champion in the game was one a different line. So, there were ten lines for each match. It took around 40 hours to collect all of the data, but it was split into five different sections, and ran in parallel. The data was then split into two, and one was used to create the file with average stats for each champion. The other was used for testing the model.

The champion average data has the average kills, deaths, assist, gold, and win percentage of each champion in each role. The data set was created by sorting the data by champion and then role, and calculating the average for each subset.

The data used in testing the model contains 52 columns and is in the form of a csv. The first column is the match ID and the last column is the winner. The other 50 columns are for the ten players, the first half for team one, the second half for team two. For each team, the first five are for the top role, second five are for jungle, third is middle, fourth is adc, and fifth is support. There are 15,000 games in the data set. The values of the data for each champion was taken from the champion average dataset, and then standardized so the model would have better results.

Four different methods were used to test the accuracy of the model. Decision tree, random foresting, bagging, and AdaBoost. Tests included changing the size of the tree to fit the data better. A baseline accuracy was calculated using random guessing and compared to the results.

Most of the python scripts were written in Notepad++ a basic text editor, and were run through a command line. Excel was used frequently to easily view the csv files, and do some basic analysis to check for issues in the data. Pandas python library was used for preprocessing. IPython notebook was used to create a model and see the results. See figure 1 for a timeline of the work done for the project.

## Experimental Results

The results of the project showed that the model did not predict the winner very well at all. The accuracy was 52%, which is barely above 50 percent, meaning it is not much better then random guessing. Figure 2 shows how it compared to random guessing. The gray line is the model’s accuracy, and the orange line is for random guessing. It was calculated taking the number of games where team one won multiplied by a variable ranging from 0 to 1. This was done for games team 2 won, and then these two numbers were added together and then divided by the total number of games. This shows the model was better than random guessing.

Figure three shows the accuracy of the four different methods. The graph shows that the four methods produced similar results, but varied slightly. It shows that some methods were actually lower than 50%, and that a decision tree was the best model.

## Discussion

The results show that the features in the model do not do a good job in predicting the winner pf a game. In a large amount of data, the kills, deaths, and assists do not vary enough to be good in the model. Using amount of gold seems good, however it would be better to use an attribute like gold/minute so the attribute isn’t effected by the length of the match. The model could also be improved by adding more statistics from the match, like damage dealt and number of dragons/barons killed.

The main insight firm this project is that the best way to increase the accuracy would be to use statistics about the actual players in each game. Going based of average KDA of a champion doesn’t work because the values are too close to each other from champion to champion. But if you use the players’ stats it will be clearer what champions they are good at and what champions they aren’t. A big indicator would be how many games the player has played on a champion. If a player has over 100 games on a champion, they will almost always beat a player who is playing a champion for the first time.

The initial data set contained the rank of each player in the game. This feature could be good in the model, because a low rank player will probably lose to a higher rank player. It was not included in this project though, because the values collected were the players rank from a year ago, which would be far less accurate because players’ skill levels will change a lot over a year.

The challenge with using players’ data instead of champion specific data is that it is much harder to collect good data. Making a player based model would require a lot of data for each player in the model. There are millions of players and they don’t usually play the same people more than once, so the amount of data needed for the players would be huge. So everyone time a match is added to the dataset, it is likely that ten more players will need to be added also.

# CONCLUSIONS

This project did a good job in laying down the foundation for a League of Legends win predictor. While the model itself didn’t do a very good job predicting the winner, it showed how a model can be created. The code used to collect player IDs, match ID’s, and match data can be used for other predictive models, and can be adapted to collect data for other purposes.

The project showed how to crawl through the Riot Games API to collect large amounts of data. Since this is the best way to collect data from the API, this contribution can be used for many projects using the API. The project also showed how to use the Cassiopeia library in python, which makes using the API much easier. Future projects should make the model based on the players’ data to increase the accuracy.

# REFERENCES

1. Riot Games. "Full API Refernece." Riot Developer Portal. Riot Games, n.d. Web. 30 Apr. 2017.
2. Maldonis, Jason, and Rob Rua. "Cassiopeia Documentation¶." Cassiopeia Documentation — Cassiopeia 0.1.3 Documentation. Meraki Analytics, n.d. Web. 30 Apr. 2017.

Figure 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Write python script that collects 200 match ID’s from each of the top 800 players in North America | For each match ID, get data for the match and save it in csv | Preprocess the match data | Create file with champion averages |
| Completion Date | 3/10/17 | 3/20/17 | 3/30/17 | 4/14/17 |

|  |  |  |
| --- | --- | --- |
| Task | Create test model file, where each match is one line in the csv | Create prediction model, analyze results |
| Completion Date | 4/18/17 | 4/25/17 |

Figure 2

Figure 3

