Predicting Winners in League of Legends

Parth Mishra

University of Colorado Boulder
parth.mishra@colorado.edu

December 7, 2016

Abstract

This paper attempts to generate an accurate model for predicting the winner of a League of Legends game. Utilizing Riot Game's API, a dataset of 700 samples corresponding to individual games was prepared. Each sample records pregame information such as the champions each team chose as well as the average skill of each team with their respective picks. Several classifiers were trained on the dataset with the decision tree being the most accurate with 61% accuracy. Optimizations to the parameters of the decision tree did not result in any accuracy improvements. Various suggestions for improving the process are then made, the most prominent one being the need for more samples.

1. Definition

1.1. Project Overview

League of Legends[1] in an online multiple player battle arena (MOBA) game that is currently the world's most popular online game with 100 million monthly players worldwide. Like the genre name suggests, League is very much a team game with Role-Playing-Games (RPG) and Real Time Strategy (RTS) influences. There are 10 players total with 5 on each team who are all using a unique in-game avatar known as a "Champion" with various special abilities. Using these Champions, players fight against each other and attempt to take control of each other's respective base and thereby winning the game. Prior to the game actually starting, players on both teams engage in a "draft" where each player selects a champion to use in the game. The choice of champion is very important as a means to best play to the strengths of a particular player as well as select a champion that best fits the overall team composition. There are 133 unique champions in the game, all with their own strengths and weaknesses, thus creating a winning team composition somewhat of an art. In the professional scene, the importance of selecting a good team composition during the draft (also know as "pick-ban phase") is paramount to a team's success. This is due to the fact that at the professional level, individual player skill tends to not vary as much as it does in amateur play. It is still important at lower levels of play since in competitive matchmaking ("ranked") the players are grouped by relative skill.

The problem many players face when entering competitive games is knowing how their team composition will fare against the enemy's team composition. Beyond just the players themselves, coaches, analysts, and spectators often want to have some idea of who is most likely to win a match based on the selected team composition. Assuming relatively similar skill levels, team composition and player mastery present the biggest uncertainties for predicting a team's win or loss. Knowing how likely you are to win with a given team composition and player mastery mitigates this uncer-

[1] This video is a quick, high level introduction to the game: https://youtu.be/BGtROJeMPeE

tainty to some extent. If a team optimized its picks based on maximizing the overall win rate, then they should logically expect to win more in the long run. Using a supervised learner on previously played games, it would be possible to learn what team compositions are more likely to win.

Due to the niche nature of this problem, there are not very many publicly available data sets to use to tackle this problem. However, through use of the publicly usable Riot Games API, I sampled ranked game data for players of various skill rankings. Ranked game data refers to the subset of games played that count towards each player's ranking on the server ladder. Ranked data is going to be more indicative of true patterns/results as this the one serious game play mode available to the majority of players[2]. The queries of these previously played ranked games return JSON formatted data that can be then transformed into a standard CSV format. Queries on player game data returns numerous categorical data that needs to preprocessed in order to reduce the amount of time needed to process these data in order to get it down to each champion and their respective player's mastery of that champion.

1.2. Problem Statement

The problem here is that, given perfect pregame information about every champion and player on both teams, which team is most likely to win the game? In order to come up with a reasonable answer to this question, it is possible to use numerous previously played games along with their results to learn over time how well a given team composition will do in a given match. In addition to just team compositions however, player skill/experience can also be factored in using Riot's "Mastery" system which is essentially a separate ranking independent of overall player ranking that numerically (0-7) indicates their mastery with a given champion. Player mastery can be used as a sort of "weight" on individual champion picks that can help in instances where a champion might not be very useful by itself, but if the individual player happens to be very good at using them, it could offset to varying degrees. Note that this is unrelated to a player's win rate on a certain champion. Since this problem regards predicting a binary outcome of winning or losing, this is a supervised learning task. Thus, given a particular game sample that includes every champion picked and their corresponding mastery, it should be possible to predict the winner with reasonable accuracy.

[2]The other game mode, normals, is more likely to have erratic play due to being a casual atmosphere which could potentially skew some results

1.3. Metrics

Using a simple accuracy evaluation metric (# of correct classifications / # of total classifications) is a basic but useful metric for determining the usefulness of a binary classifier. This assumes that champions and player skill on their champion vary enough that in a game with all players being close in rank, is enough of an indicator of potential team success. Additionally, it is expected that the win-loss ratio for each team is balanced with the underlying assumption that the side of the map, and thus team color, is irrelevant in determining the outcome of the game.

2. Analysis

2.1. Data Exploration

As previously mentioned, player mastery is a ranking from 0-7 that indicates how much experience a player has on a champion. This is a more useful metric in this situation compared to win rate since it is independent of how many times you win or lose. Win rates are often low (less than 50%) across the board for players that are ranked lower, thus making it harder to distinguish a winner. Mastery is only achieved by using the particular champion often. Mastery distinguishes itself from pick rate by also having a requirement for occasional good performances on a champion in game prior to advancing mastery level. For its

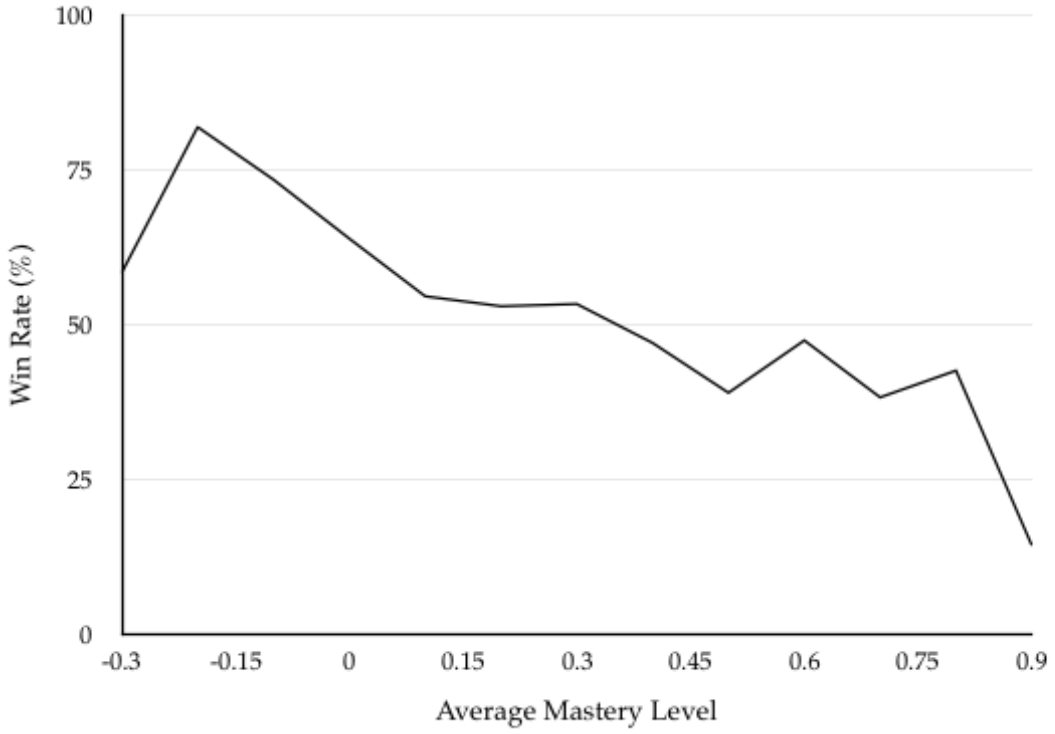

Figure 1: *Win rates (%) per average mastery level*

inclusion in this dataset, I used the normalized, average champion mastery level rather than including each individual mastery level.

In order to represent the desired information on each team's picks and mastery level, the dataset is composed using data for 269 features. For each team, 133 features are present that correspond to each unique champion in the game, thus making 266 features total. Two of the remaining 3 features are to indicate the average mastery level of each team. The last feature is a binary target label that simply presents the winner of the match with a blue team win corresponding to 0 and a red team win corresponding to 1. Overall, the dataset is slightly imbalanced with 47% blue team wins and 53% red team wins. Ideally, the dataset would be balanced for win-loss ratio to avoid using the side of the map as a indicator of success.

In order to get a better idea of what the dataset looks like, a very truncated sample showing only 2 of the 133 possible champions would look similar to the following:

C1_1	C2_1	C1_2	C2_2	BM	RM	W
1	0	0	1	0.53	0.49	1
0	1	1	0	0.35	0.74	0

The feature names are abbreviated as follows:

- C1_1 = "Champion 1 for blue team"
- C2_1 = "Champion 2 for blue team"
- C1_2 = "Champion 1 for red team"
- C2_2 = "Champion 2 for red team"
- BM = "Blue Team Average Mastery Level"
- RM = "Red Team Average Mastery Level"
- W = "Winner"

Note that in the actual data set, there would be entries for C1 through C133, each corresponding to a unique champion and each team. A value of 1 in a C*N*_*N* feature indicates its

presence in that particular game. Every pick is unique such that a 1 for one champion on a team precludes it from being present for the other team e.g. if CN_1 is 1, then CN_2 has to be 0 since the same champion can not be on both teams for a particular game. Overall, out of the 266 champion features, only 10 of them will have a 1 as their value.

The large dimensionality of the feature space requires a large number of samples as well. For this dataset, there are a total of 700 samples, 100 from each skill tier in the game. Ideally, a larger number of samples should be used but producing the dataset takes a very long time due to Riot's API rate limiting and the need to find all 10 players individual mastery score for every single game/sample.

2.2. Exploratory Visualization

One area of particular interest in this dataset is to see how win rates are affected based on average champion mastery. The underlying assumption for this entire experiment is that a team's skill has some effect on the outcome of the game that is distinguishable from the strength of the champions picked. Seeing how well correlated win rates and average champion mastery can give a precursory idea of what might be expected when running the classifiers later. Looking at Figure 1, we can see how win rates vary with average champion mastery:

Interestingly enough, there seems to be a clear downward trend in win percentage as a team's average mastery level goes up. This seems counterintuitive as the team that has demonstrated greater skill with their respective picks would intuitively be the more likely winner. As far as what these results may suggest about the performance of a classifier, any strong correlation, whether intuitive or not, could potentially make mastery level a useful predictor for the classifier.

2.3. Algorithms and Techniques

As previously mentioned, the compiled dataset has a very large feature space and requires some dimensionality reduction in order to use it effectively with some classifiers. However, loss of information during dimensionality reduction can negatively impact results as it is imperative to know what champions were picked and not picked. In this light, these data are a prime target for Principal Component Analysis (PCA) which is useful for situations in which dimensionality reduction is desired but not at the expense of information loss. PCA creates new features that maximize variance in the data which results in minimal information loss while reducing the feature space to essential components.

Given the nature of this problem, predicting the binary outcome of a game instance, this problem is a classification task that necessitates the usage of classifiers. Ideally the chosen algorithms would handle a large feature space. Training time is not a huge concern as the dataset is small enough that performance is negligible. With these characteristics, there are not really any classifiers that stand out as being more so suited to this problem than another classifier. Thus, an array of classifiers were tested to gain an initial idea on each classifier's performance, specifically:

- Support Vector Machines

  Support Vector Machines create boundary lines by maximizing the distance from points to the decision boundary in a classification scenario. SVMs work well with highly dimensional data such as ours due to its ability to project a hyperplane in multiple dimensions.

- Decision Trees

  Decision Trees are a very simple classifier that can work well on many classification tasks which makes it a good candidate for an initial run. Good initial results from a decision tree must be taken with caution as they tend to overfit the data. Some parametric tuning of the tree as well testing robustness with another test set may be able to alleviate these problems.

- Random Forests

  Random Forests are an ensemble classification method that averages a series of decision trees to reduce variance of singular decision trees and compensate for the high bias that trees and forests tend to have. Random forests, like decision trees, have a wide variety of tuning options should there be a good initial performance.

Then, the one that exhibits the best performance in the initial run will be selected for further refinement of its parameters in order to develop a final model.

2.4. Benchmark

The benchmark model for this experiment is a simple classifier that takes the win rates[3] for each champion picked, averages them together, and outputs the winner as the team with the higher average win rate. The overall accuracy of that this model produces will then be used a point of comparison for determining how well the final model works. This model is useful as a benchmark because it is player/skill agnostic and focuses only on overall strength of champion picks. Using this model, the accuracy on the test set of the data was 52% which is just above naive guessing.

If the accuracy is low for the final model compared to the benchmark model, this may indicate that player skill for a certain champion is not as important as the champion pick itself or that there's some other underlying relationship that's not accurately being modeled. Given the numerous factors that can influence a game, of which not all are statistical representable e.g. player behavior towards teammates, accuracy is not expected to be super high for any model applied to this problem, but at least enough to be better than random chance.

[3]Win rates from http://champion.gg/statistics/

3. Methodology

3.1. Data Preprocessing

From the initial dataset, a training and testing set are determined using 80% as the training set and the remaining 20% as the validation set. Additionally, the average mastery for each team was normalized to the range [-1,1] using a min-max scalar to prevent any one feature from dominating the otherwise high amount of binary attributes.

As previously mentioned, prior to applying the classifiers, the dataset dimensionality was reduced using the PCA technique. Various number of components were tested to see how accuracy changed for the classifiers by changing the number of components until accuracy converged for each classifier. Through this process of trial and error, the final number of components to be used is 30 components.

3.2. Implementation

Much of the implementation of the classifiers as well as preprocessing of the dataset were accomplished using the scikit-learn Python library which vastly reduced the complications of the coding process for data preprocessing and the classifiers themselves. After creating the training and testing sets with 80% of the 700 samples for training and 20% for testing, the classifiers were then trained on the PCA reduced training set and created predictions for the winners in the test set. From there, the accuracy of each classifier was calculated. These initial results were obtained without changes to default parameters.

3.3. Refinement

As previously mentioned, the process of selecting the final model first relies upon an initial run of the various classifiers with default parameters. From there, there, the classifier with the strongest initial performance will be optimally tuned using a grid search method from scikit-learn's GridSearchCV functionality.

The initial results of the classifiers are shown in the following table:

SVM	RF	DT
0.52	0.58	0.61

From the initial results, it seems that none of the classifiers work particularly well. Relatively speaking, it seems like SVMs is a particularly poor choice while Random Forests and Decision Trees tend to perform better with accuracy of 58% and 61% respectively. Decision trees seem to perform marginally better than the other classifiers on average, therefore making it the final candidate for parameter tuning.

The process of refining the parameters of the decision tree is through systematically moving specific parameters until maximal performance is achieved i.e. right before validation error and training error diverge. This process was automated using the aforementioned GridSearchCV method from scikit which does an exhaustive search on a range of given potential parameter values and uses cross validation as well. For the decision tree, of the parameters to tune are:

- splitter

  This defines the method through which splits in the tree are made (optimal or random-optimal split). The default run utilized the optimal, or 'best' splitting method.

- max features

  This is how many features are targeted per split which can impact performance of the tree. Typically this is done as a function of n input samples, so several methods are tried here such as $n, \sqrt{n}, \log n$, etc. The default run used n features.

- max depth

  The maximum depth (layers) of the tree. Setting this can be useful for optimizing a tree that is over fitting. The initial run did not set a maximum depth.

Using a range of values for each parameter, including the default values, the exhaustive grid search returned the default parameters as the optimal ones. For the splitter, the "random" split method was tried along with the default "best" splitting method. For the maximum features, a range of input functions mentioned above were tried. For the maximum depth of the tree, many values were tried from 1 to 10 as well as the default option of no maximum depth set. Changes to these parameters at best resulted in the same accuracy. More often than not, accuracy for the decision tree dropped to 50-55% when changing these values.

4. Results

4.1. Model Evaluation and Validation

The final model made from the decision tree classifier ended up with an average of 61% accuracy and ultimately did not perform better with further optimizations to the splitting method, max features per split, and maximum depth of the tree. Although the optimizations did not improve the final model beyond the initial model, the results are still in line with the expected results of any final model produced by this experiment.

Using a separate test set of 50 matches not found in the original dataset, the model was able to achieve an average of 58% accuracy which suggests that the model was not over fitting to the original data. However it still exhibits traits of underfitting to unseen data given the low accuracy.

4.2. Justification

As stated earlier, the accuracy of any model applied to this problem was likely not to return high accuracy due to the presence of a multitude of potentially influencing factors whose inclusion into the dataset would dramatically

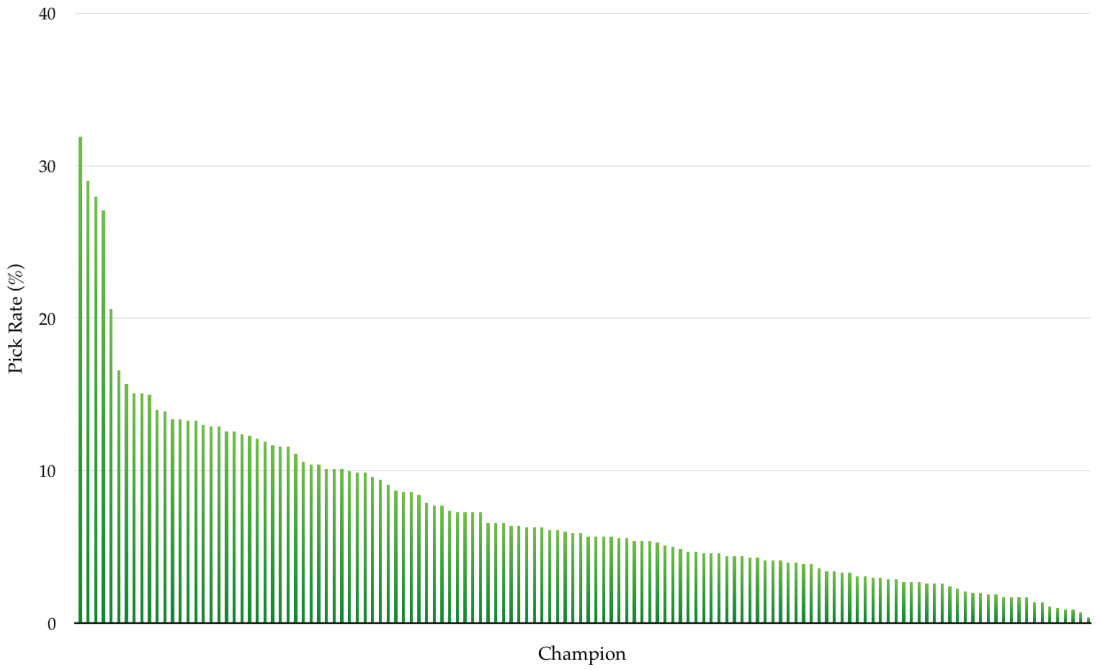

Figure 2: *Pick rate (%) for all 133 champions*

increase the dimensionality of the dataset. Additionally, the presence of important yet statistically unrepresentable data such as player behavior may reduce accuracy of predictions.

The initial problem that this experiment is trying to solve is the question of what team will win a given game, given pre-match data, namely: champion picks and associated player skill with champion picks. The benchmark model took one aspect of this problem, the champion picks, and evaluated the average win rates of the selected champions by each team in order to predict a winner. The accuracy of the benchmark model was an average of 53% compared to the 61% of the decision tree which suggests a marked improvement in accuracy, lending credence to the idea that player skill does indeed have some affect on the outcome of the game. This is in line with the intuition gained from the exploratory visualization which showed a strong negative correlation of winrates and mastery level.

While there is an improvement from the benchmark model, the accuracy of both the benchmark and final models suggest they are a poor predictor of winners and do not perform reliably better than random guessing (assuming win rates of players and champions are close to 50%).

5. Conclusion

5.1. Free-Form Visualization

There are a multitude of reasons for the low accuracy rates found in the previous section. One of these reasons can be seen in Figure 2 which shows the distribution of pick rates for all 133 champions in the dataset. From the graph, the distribution is positively skewed which presents some problems for the classifier. The figure shows only a few champions appearing in a disproportionately large amount of game samples. This means that models applied to these data will have a harder time with classification when seeing the majority of

champions. In a dataset this small, this distribution has a greater impact since there are very few samples of a large portion of champions. This distribution would remain similar with more samples (most likely a bit more uniform) but would not pose as much of a problem as the relative differences would lower and more samples would be available for all champions.

An ideal dataset would have a somewhat uniform distribution of champion appearances since it is not the case that higher pick rate means a higher win rate. Controlling for pick rate allows the relative strengths of individual champions to be more prominent of a factor when training a classifier.

5.2. Reflection

The process of answering the question "Can a winner of a game of League of Legends be accurately predicted using available pre-match information?" can be summarized as follows:

1. Generate dataset of match data that shows champion picks for each team, average mastery score for each team, and a target label that indicates the winner of the match
2. Apply PCA to the dataset to reduce dimensionality
3. Train several classifiers on the PCA-reduced data
4. Test each trained classifier using a validation set and report the accuracy
5. Select the model with the highest initial accuracy for further refinement
6. Use grid search method along with cross validation to determine optimal combination of parameters that result in the highest accuracy

Prior research into this area indicated that this problem is very challenging to approach with traditional learning machine learning methods. Low accuracy is typical of classifiers that attempt to predict winners. While my method approached this problem from a slightly different angle by incorporating player skill into the predictions, ultimately the final model generated suffered similar issues as previous attempts.

As far the as the process goes, generating the initial dataset proved to be the biggest technical challenge due to strict API query limits and the amount of API calls needed to generate a single data sample (11 unique calls to be exact). The dataset as a whole took many hours to generate leading to the low amount of samples used in the experiment.

The structure of the dataset also posed an interesting challenge in how to represent the desired data in such a way that a classifier could detect the uniqueness of champion picks and there relationship to each other e.g. combinations of certain champions leading to more likely chance of winning or losing. In order to capture the uniqueness of picks, 266 features had to be made with only 10 of them having a 1 (indicating the champion is present for that game) compared to 256 of them having 0s per game sample. The extremely large dimensionality requires many more samples even if dimensionality reduction techniques such as PCA are applied.

5.3. Improvement

As the results suggest, the final model returned from this experiment exhibited high bias and ultimately a low accuracy which made it a poor predictor for success. However, there are many potential optimizations to the process that could potentially result in a better model. To address issues with high bias, the most obvious solution is to simply acquire more data samples. As noted in the previous section, the data has high dimensionality due to the need to preserve uniqueness of champion picks and 700 samples, as it turns out, is not really sufficient. With 133 potential picks per game, it is very hard to gain reasonable information about many champions due to their low pick rate. This presents an issue with generalizing to unseen data, especially when a rarely picked

champion appears. Ideally, the dataset would have several thousands of games to establish more definite patterns and have all champions appear more than a couple of times. This could potentially result in a more uniform distribution of champions which, as explained in the free-form visualization section, could make relative champion strength more prominent of a factor in classification.

Beyond the addition of more data, some additional features could potentially be added to provide a more complete picture of any available pregame information such as if all roles in a team are fulfilled. There are 5 distinct team roles with one for each player on a team. Certain champions are exclusively played in or meant for, certain roles. Thus a situation in which a player who is playing a strong champion but in an "off-role" should greatly decrease a team's chance of victory, regardless of the player's mastery of that champion. Currently, the classifier makes no attempt to distinguish this fact and thus probably made some incorrect classifications of games. This is especially important for games that were taken from low skill games in the dataset. These players are more likely to play random champions in roles compared to higher level players who stick to the "meta", or optimal combination of champion picks per role.

Even with these changes, the model most likely still would not reach really high accuracy levels but an accuracy of 70%+ is not unfeasible. At that point, the model could be fairly reliable for predicting winners.