# *League of Legends* Match Outcome Prediction

Lucas Lin[1]

***Abstract*— We use gradient boosted trees and gradient boosted trees with logistic regression to predict the match outcomes of the popular online multiplayer game, *League of Legends*. Features are extracted from the data that Riot Games API exposes–including champions picked for the game, player role information, and mastery levels for the players' champions (pre-game knowledge) as well as in-game player statistics. One-hot encoding is used to transform this mostly categorical data of pre-game knowledge into a feature vector that is used in the machine learning algorithms. We find that using only pre-match knowledge (champions, masteries, roles, spells) from the very start is only a weak predictor of match outcome but using in-game statistics, the model becomes a strong predictor.**

## I. INTRODUCTION

*League of Legends* is a multiplayer online battle arena (MOBA) game developed by Riot Games where players participate with four teammates in head-to-head matches and the goal is to destroy the opposing teams nexus. Each player in a match–usually lasting between twenty minutes and an hour–controls a unique champion chosen from a pool of more than a hundred with differing characteristics and abilities. The game boasts 100 million monthly players and a flourishing competitive scene with millions in tournament prize pools as well as online viewers. With such a large community behind *League of Legends*, predicting match outcomes for casual players and tournament games alike would be interesting and valuable to players and fans. Such a system would also provide insight into particular features that are influential in swaying win probability.

### A. Game Overview

In a standard *League of Legends* match, two teams of five players each face off on a map called the Summoner's Rift. Each player has a choice of playing one of 133 champions for the duration of a single match as well as customizing their initial stats and abilities by choosing two summoner spells (out of ten) and selecting sets of "masteries" and runes which effectively cause various increases to stats like more health regeneration or bonus characteristics such as moving faster when attacking an enemy. A typical game also sees the division of players on single team into distinct roles that they play throughout the game–for example, we would typically see a player be a "support" who spends the game trying to keep their teammates alive or trap enemies for their allies to attack. Other roles include the "carries"–players who are supposed to do the most damage and "jungle"–players who spend most of the beginning of the game leveling up by being in the jungle area of the map and also periodically helps the other teammates.

### B. Related Work

As *League of Legends* is a popular game, there are already a couple of applications that exist in the data analysis sphere. Websites like League of Graphs and MetaSrc collect basic stats and information from the *League of Legends* API and use it to construct simple analyses like what the most popular champion and what the win rate of two champions when played together are. What these existing applications do not currently do is prediction of a currently ongoing game with all of the features of that game in mind, and that is the hole that this project is attempting to fill.

## II. DATA COLLECTION

Riot Games provides a public API endpoint to access nearly all kinds of data that would be available to see in the official game client including match history, champion mastery levels, and more. We use Cassiopeia, a Python wrapper for the Riot Games API, to access the data.

### A. Match Data

The API endpoint gives access to match data which includes the champions selected and the roles of the players playing in the match. To obtain our set of matches that will become the training and test sets, we first seed an initial queue with players from a random

[1]L. Lin is an undergraduate in the Computer Science Department, Stanford University, 450 Serra Mall, Stanford, California lucaslin at stanford.edu

Gold league (referring to the skill rating of Gold). Then, we iterate through every player in the queue, storing matches that we have not seen yet that occurred after the most recent game update as well as adding players in that match to the queue that we have not seen before.

### B. Player Data

In addition to the data about matches, the Riot Games API also exposes the "mastery points" of players with regard to their champions. This value more or less corresponds to the amount of time that a player has played a certain champion weighted by how well the player does on that champion. We store the mastery points of each player in a given match with regards to the champion they play during that match to be used as a potential feature later.

## III. METHODOLOGY

### A. Feature Selection

Currently, the pre-match information that we store about consists of

- Champion picked for a given role on each team
- Summoner spells picked for a given role on each team
- Mastery points of each player for the champion they are playing
- The winner of the match

Using one-hot encoding, we can transform the first two items above–which are categorical variables–as a series of binary indicator values. For example, to represent the champion that is picked for support as a series of indicator values, we have 133 variables corresponding to the support role being a different champion. There are five roles per team and two teams, so in total there will be $133 * 5 * 2 = 1330$ features stemming from the first item. Similarly, for the summoner spells picked, we know there are 10 spells and players may not pick the same spell, meaning there are 90 possible summoner spell combinations. This would mean that the number of features stemming from this would be $90 * 10 = 900$. In total, we have a feature vector of length $1330 + 900 + 1 = 2231$. For the in-game knowledge, can extract and calculate from each match statistics like

- Total damage dealt per player
- Difference in gold between two players in a single lane
- Gold spread within a team through mean and standard deviations
- Kills to deaths ratio
- Obtained first kill of the game or not
- Other similar statistics

Since these statistics are all numerical and/or boolean, we can construct the feature vector for this set of information in a straightfoward manner, with the boolean statistics being represented as a 0 or 1. The template for this feature vector in Python code is as follows

```
{
        # Individual Statistics
        "[stat_name]_[side]_[LANE]_[ROLE]": [Value]
        "assists_red_TOP_SOLO": 10
        ...
        # Group Statistics
        "[stat_name]_[side]": [Value]
        "gold_standard_deviation_blue": 1434
}
```

Lastly, we label each match as follows

$$y^{(i)} = \begin{cases} 1, & \text{if the blue team won} \\ 0, & \text{if the red team won} \end{cases}$$

### B. Models

Using the feature vectors and labels extracted from the matches, we then use various supervised learning models to see if we can predict the outcome of the matches

*1) Gradient Boosted Trees:* We first look at gradient boosted trees because of several characteristics that make it suitable for the type of data that we are using. First, tree ensembles do not expect linear features and handle binary classification well because the method is using combinations of decision trees. Furthermore, boosting makes the algorithm a good fit for handling the high dimensional space we have as well as the expected large number of training examples.

The gradient boosting method attempts to an approximation to a function minimize the expected value of some loss function

$$\arg\min_F E_{x,y}[L(y, F(x))]$$

where $L(y, F(x))$ is the loss function and $F(x)$ is the prediction of the model that is in the form

$$F(x) = \sum_{i=1}^{m} \gamma^{(i)} h^{(i)}(x) + c$$

and $h^{(i)}(x)$ are functions in some hypothesis class of weak learners [1].

We use the tree boosting algorithm that Jerome H. Friedman derived from gradient boosting. In this method, at the $i$th iteration step, the hypothesis functions $h^{(i)}(x)$ are decision trees and with $J$ being the number of leaves, the $h^{(i)}(x)$ tree splits the input space

into $J$ regions that we can identify as $R_j^{(i)}$ where $j$ identifies the $j$th region [1]. The update rule for Friedman's gradient boosted trees is

$$F^{(i)}(x) = F^{(i-1)}(x) + \sum_{j=1}^{J} \gamma_j^{(i)} h^{(i)}(x)[x \in R_j^{(i)}]$$

$$\gamma_j^{(i)} = \arg\min_{\gamma} \sum_{x_k \in R_j^{(i)}} L(y_k, F^{(i-1)}(x_k) + \gamma h^{(i)}(x_k))$$

which we use in Python to generate the model.

*2) Gradient Boosted Trees with Logistic Regression:* We also attempt to see whether or not our data can be predicted through a linear model after it has been transformed by the ensemble of trees from gradient boosting. We fit the trees on the training set and then assign each leaf an arbitrary feature index in a new feature space which are encoded using one-hot encoding. The transformed data is then run through logistic regression.

## IV. RESULTS

We obtained the training and test data sets by scraping the match lists of Gold league (middle-ranked) *League of Legends* players and then using the Riot API to gather the information required. Since the pre-match data includes mastery level of champions which needs to be looked up for each unique player/champion pair, the number of samples in the pre-match data sets were comparatively less than that of the in-game set due to Riot API rate limiting.

Table 1 below shows the results of training and testing the gradient boosted trees (GBT) and gradient boosted trees with logistic regression (GBT+LR) algorithms on both the pre-match and in-game data sets.

TABLE I
TRAIN AND TEST ERRORS OF THE MODELS

| Model | Train Error | Test Error |
|---|---|---|
| Pre-Match GBT | 0.0 | 0.433 |
| Pre-Match GBT+LR | 0.0153 | 0.448 |
| In-Game GBT | 0.0 | 0.0608 |
| In-Game GBT+LR | 0.0 | 0.0642 |

As we can see, there does not seem to be a comparable difference in train/test error between the gradient boosted trees algorithm and the gradient boosted trees with logistic regression algorithm for either the pre-match or the in-game data sets. However, comparing between the pre-match and in-game feature representations, we see that the predictive ability of the in-game model is much higher than that of the pre-match model.

This is further supplemented with Figures 1-3 below which show the receiver operating curves of the models.

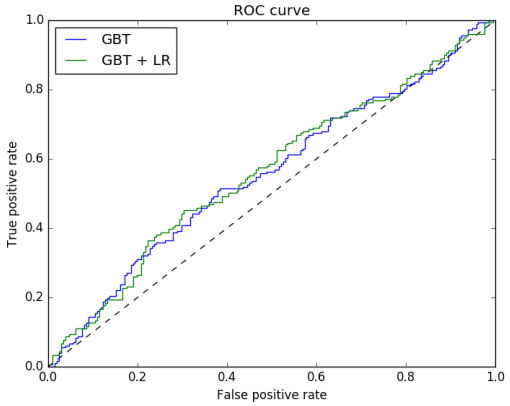


Fig. 1. Receiver operating characteristic curves for two models using pre-match knowledge–gradient boosted trees (GBT) and gradient boosted trees with logistic regression (GBT+LR)–using a training sample size of 196 and a test sample size of the 392.

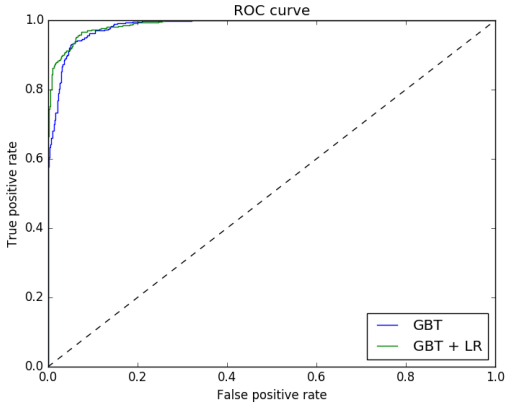


Fig. 2. Receiver operating characteristic curves for two models using in-game knowledge–gradient boosted trees (GBT) and gradient boosted trees with logistic regression (GBT+LR)–using a training sample size of 1000 and a test sample size of the 2000.

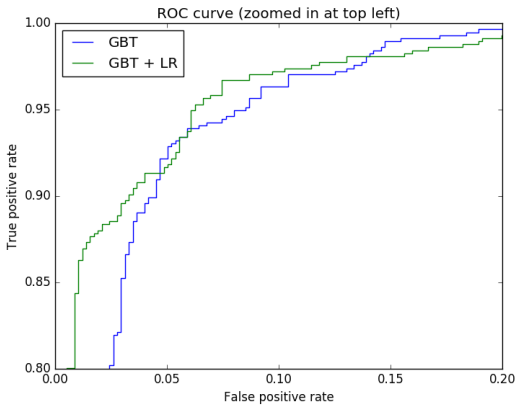


Fig. 3. A zoomed in version of Figure 3.

## V. DISCUSSION

We can see from the results that the algorithms being used with the pre-match information feature representation is only giving a prediction that is slightly better than random guessing. Furthermore the large discrepancy in the train and test errors of this specific model is indicative of over-fitting of the data. Of course, since boosting is generally more effective with a large sample size, it may be possible that the over-fitting is reduced and the test error reduced when more data is available.

On the other hand, the in-game feature representation model showed high predictive ability compared to the pre-match features. This is somewhat expected because the game itself contains information that should be correlating with who wins or loses the match.

## VI. FUTURE WORK

In the future, it would be interesting to develop a finer sense of how accurate the prediction system becomes given the amount of data at a certain point in a match. In other words, how much more accurate would the prediction becomes if data extracted from the first X minutes of each match is used as part of the feature vector as compared to the first Y minutes as well as how the accuracy changes with respect to X.

Furthermore, the overfitting in the pre-match data could potentially be dealt with by modifying the way the champion selection is used as a feature as it currently accounts for 1330 of the features in the pre-match feature vector.

## VII. CONCLUSION

This project demonstrates that using pre-match and in-game data, it is possible to predict the outcome of a *League of Legends* match with varying levels of success. We find that using the gradient boosted trees (with or without logistic regression) and a feature vector based on in-game knowledge gave us around a 95% success rate on prediction.

## ACKNOWLEDGMENT

I would like to thank Professors Andrew Ng and John Duchi for teaching the foundational material that allowed me to work on this project as well as my TA, Michael Zhu, for giving me feedback and tips for improving my model/project.

## REFERENCES

[1] J. H. Friedman, "Greedy function approximation: a gradient boosting machine." *Annals of statistics*, 2001, pp. 1189-1232.