INFSCI2595 Machine Learning – Final Report
League of Legends: Match Prediction
Jordan Widjaja (jow105)

**Abstract:**
League of Legends is a competitive game in which players on two teams must team up in order to destroy the opposing team's base. Each match consists of many different features which affect the outcome of a game, such as gold differentials, damage dealt to champions, or champion level. These features can be divided into two different categories, based on the stage of the game. We analyze the features with respect to two machine learning models, namely, logistic regression and random forests, and in doing so, we are able to discover that there is a greater effect on the outcome of a game from near-end-game features than from features in the beginning and middle of the game. Both machine learning models used had relatively equal accuracies on most feature selection combinations. Furthermore, the middle lane has the greatest variation in impact on a match's success while the bot lane carry role has the greatest correlation between in-lane success and match victory.

**Introduction:**
League of Legends is a multiplayer online battle arena video game in which players assume the role of an unseen "summoner" who controls a champion with various abilities, and in a team with other players, must battle against other (AI- or player-controlled) champions to destroy the enemy base, known as the "nexus". Each League of Legends match can be subdivided into three parts: the pre-game (also known as champion selection), in-game, and post-game. The game mode we primarily focus on in this project is known as Summoner's Rift, in which each of the five players on both teams (blue team and red team) are assigned one of five roles: top lane, middle lane, jungle, bottom lane carry, or bottom lane support. These roles are selected in the pre-game champion selection phase. Features that exist in this pre-game phase, but were not analyzed in this project, are as follows: runes (points in a chart selected for in-game champion buffing), summoner spells (selected utility abilities aside from original champion abilities), and masteries (an unseen score from 0-7 denoting a player's experience in playing the selected champion). Throughout the course of the game, each player's champion can slay minions to gain small gold values or other enemy champions for higher gold values, which can be used to buy items for permanent buffs, and doing so will give players in-game experience to be used to level up, increasing champion ability impact. Note that all in-game features that were analyzed were conducted with respect to the blue side. As such, for a particular outcome of blue team's match (victory or defeat), we utilize the creep score (number of minions slain) *differential* per minute, damage taken *differential* per minute, and experience *differential* per minute as indicators during the match, and damage dealt to champion *differential*, gold earned *differential*, average champion level *differential*, and average KDA (sum of kills and assists divided by deaths) ratio *differential* as indicators nearing the end of the match. Utilizing these features, we should be able to trace, for the most part, how the course of a game flows, as well as most of the important features in determining its outcome (there do exist features such as selecting emotes or selecting champion skins which should not have any significant impact on game outcomes, and because of this, we will not consider these features).

**Relevant Work:**

*Predicting the Outcomes of MLB Games with a Machine Learning Approach*

This research paper focuses on the sport of baseball, which draws many similarities to analysis of matches in League of Legends. The article claims that even the best teams in baseball win only about 60% of their games in a season, while the worst team wins about 40%, and because these statistics hover around 50%, it can be argued that baseball games are sufficiently random due to their complexity. However, using historical data and machine learning algorithms, this paper challenges this claim. In doing so, it was able to focus on certain features that it thought were particularly important to determining the outcome of a baseball game, namely, offense, defense, pitching, and general statistics, and in addition to this, some additional features such as time of day, whether it was a home game or away game, location, opposing team, and day of the week were added. Four machine learning algorithms were used: generalized linear regression, random forest, XGBoost, and boosted logistic regression (used for binary problems only). According to the results, the accuracies of the models are only about a little more than 50% accurate, with XGBoost producing the most accurate results. The conclusion is that despite the random aspect of baseball games, these generated results produced a higher prediction accuracy than random generation. An important notion to take away from this paper is that despite the complexity of baseball games, one can always select features which are easier to analyze. Though the selected features that one can notice may not entirely encompass the true breadth of features that affect the games' outcomes, using machine learning algorithms, they should be able to predict with more accuracy than considering games to be a coin-toss probability of who the winner is. Because these games are also matches with the same output variable as League of Legends matches, either victory or defeat, using the suggested logistic regression model as the predictor was shown to be very beneficial to our goals, with either continuous or discrete input variables. Decision trees would have also been another model that could be used in this setting, but using the random forests model described in this paper, it can overcome the shortcomings of decision trees, which are namely the reduction of overfitting by the averaging of several trees and a lower variance. Additionally, in practice, it is difficult for one to completely assume independence of features, but by introducing the notion of bagging, independence can be achieved to a certain degree. While this paper does indeed boast effective methods, limitations do exist including only considering the aggregate of individual player performance and environmental factors. If one were to assume an equal skill level on both teams, the machine learning algorithms used in this paper would provide more information on the effects of the environmental factors of a match.

*Using Machine Learning to Predict the Outcome of English County twenty over Cricket Matches*

Similar to the sport described in the previous research paper, the game of Cricket is one which has been deemed as a multi-billion-dollar betting facet. Because of this, prediction techniques have been implemented in order to guarantee the best results. The features of games are divided into two categories: team data only, and team and player data. The algorithms that were used were Naïve Bayes, logistic regression, random forests, and gradient boosted trees, where the previous year's data is used as a training data set and the current year's data is used as the testing data set. Within each feature category, the features are divided into different

levels, and each successive level is more combinations of features. According to the results, we notice that there is a higher mutual information score for higher levels than lower levels, and some features did appear to have greater impact on the outcome than others. All in all, Naïve Bayes classifier produced the highest accuracies. The models that were utilized in this research paper, as they were similar to those used in the previous paper, were all strongly considered in the League of Legends prediction. However, the notion that seemed to be particularly enlightening was the idea that we can take features in terms of phases. While this article analyzes features with respect to teams and then with respect to players, we likewise used this notion into considering features at different game stages. In addition, as another takeaway from this research paper, we can analyze combinations of features so that we can have a stronger predictor, and through this, consider which particular combinations of features (in this article, either the only team data or the team and player data) is a more accurate predictor. This paper does indeed divide up their features, but better relationship results can be derived from an analysis of exclusively player features. Additionally, when we consider each feature themselves, there exists a lack of independence of features because teams comprise of the players, so features will not be as independent as they could be. This, however, using random forests, could be remedied.

**Problem Description:**

The important aspects of this project that will be analyzed in terms of League of Legends matches are, in a general sense, the ability for machine learning algorithms to sufficiently and effectively predict the outcome of matches. In doing so, there are several factors that we need to consider. As we look into the algorithms that can be used in this task, it is also important to be able to analyze which essential features of a given League of Legends match have the most impact on the game's outcome (victory or defeat). As complex of a game as League of Legends is, the organization of the game allows us to narrow down our search of essential features to only a few that are particularly relevant. We will only consider those features that each individual player has full control over, and any extraneous features that are not controlled by the players are assumed to have equal impact on both teams such that it does not bias the outcome of a match by randomness. Once our features are selected, we can analyze which machine learning algorithms are most relevant to analyze, and once we have our set of learning models to be considered, we fit our model on the training data set and observe the model accuracy in predicting match outcomes in our testing data set. If there are any noticeable differences in model fitting, the accuracies will be able to determine which of the models are the best fit and which has the worst fit, either supporting or contrasting our theoretical notions. In addition, within each model, we can also consider parameters that do not simply rely on respect to the features themselves, such as regularization factor in logistic regression or number of trees in random forests. As such, we can execute the same notion of accuracy analysis on a set of parameter possibilities to analyze which parameters best fit the particular model. Additionally, another goal that is important to consider, similar to what was described by the second research paper above, is the idea of considering phases of the game and the various features associated with each phase in order to observe which phase and which features have the greatest impact on the outcome of the game. Lastly, given that there are unique and specific roles played by each player, we can also observe every subset of features

from each of the roles and draw conclusions from these results, such as which role has the biggest impact on the success of the game or which role has the greatest variation in the outcome of games.

**Data Set:**

The data set used in this project is sourced from Riot Games' (the company that made League of Legends) API. All API data requests are returned in the form of valid JSON files. The seed data that was extracted for the purposes of this project is a set of 1000 *ranked Summoner's Rift solo queue games* (i.e. matches in the Summoner's Rift game mode played by ten real, non-AI players in teams of five versus five with roughly equal skill level on both teams and in such a way that each player queued into the game must queue either alone or with only one other person as to avoid any major communication advantages). The JSON files can be easily read as dictionaries in which the available features are the keys and their values are the feature values. The following is the format of such JSON files:

```
{matches:
      [{"gameId":2585563902,…,
      "teams":[{"teamID":100,"win":"Win",…}],
      "participants":
            [{"participantId":1,…,
            "runes":…,
            "masteries":,…,
            …,
            "stats":{…,"kills":17,"deaths":5,…,
                        "totalDamageDealttoChampions":42926,…,
                        "goldEarned":26248,…},…,
            },
            "timeline":{…,
                  "creepsPerMinDeltas":{
                        "10-20":7.7,"0-10":5.7,…},…
            }
            ],…
      ]
}
```

We describe a few of the entries as follows:

- `matches` – the entire set of matches whose statistics are described in detail by additional dictionaries. Each match has its own match identification number and consists of each team's overall statistics as well as the performances of each individual player.
- `teams` – in-depth statistics of each team as a whole, such as number of epic monsters taken, number of towers taken, etc. Blue team has a team ID number of 100 and red team has a team ID number of 200.

- `participants` – in-depth statistics of each player (whose identification number is an integer from 0-9), containing pre-game features such as runes and mastery selection as well as in-game features (see `stats` and `timeline` below).
- `stats` – complete set of player-by-player performance during the given match, containing statistics such as number of kills, number of deaths, total damage dealt to champions, and total amount of gold earned. This dictionary will be used to simulate the near-end-of-game in-game features.
- `timeline` – player-by-player performance during particular time intervals of the game (such as between 0 minutes and 10 minutes or between 10 minutes and 20 minutes) in features such as experience difference, creep score difference, and damage taken difference, all between player and opposing player with the same role.

We can parse through this dataset using the following lines of code in python, and using standard dictionary entry lookup, we can analyze the values of each desired feature:

```
import json
with open('matches_all.json') as f:
     data = json.load(f)
```

Above is the main routine for our simulation, where data is the variable that corresponds to the entire data set of matches.

**Problem Formulation and Feature Engineering:**
Our method for formulating the analysis of our match predictions will come in several parts. In each part, we will consider some combination of the following features that will then be used in generating our machine learning models:

*Overtime*
1. creep score differential per minute
2. damage taken differential per minute
3. experience differential per minute

*Totals (Near-end)*
1. damage dealt to champions
2. gold earned
3. champion levels
4. kills/deaths/assists ratio ((kills + assists)/deaths)

Before utilizing any of our algorithms, it is important to first scale our features such that one feature does not have higher precedent than any other feature. As such, we utilize python's StandardScaler package for feature scaling. Our first model to be considered will utilize the standard logistic regression algorithms as a means of classification. Using logistic regression, it is a relatively simple model to analyze in terms of relationship between our predictors and the outcome label, and in particular, fits our setting of utilizing it for binary classification.

Additionally, because logistic regression deals with maximum likelihood estimation, the assignment of classification based on the highest probability of a certain class occurring given its predictors, it allows for the reduction of noise and allows us to look at the bigger picture of correlation. For both the 'overtime' features and the 'totals' features, we run logistic regression on subsets of features as well as on the entire set of features. In doing so, we also aim to analyze the particular regularization parameter that best suits the set of features in question. In order to do this, we must execute k-fold cross validation in order to select the proper model to be used for predictions in our testing data set. For our purposes, we will utilize 5-fold cross validation, and split our data 75-25 as is default in the train-test-split subroutine used in the sklearn package, and the parameter that we vary each time is the regularization parameter, in which we test the best among the list [0.01,0.1,1,10,100].

The second model that we will be considering is random forests as our means of classification. While one may argue that the utilization of decision trees may allow for a simpler model, the advantages of utilizing random forests is also well worth the complexity cost. By nature of the algorithm, random forests can actually run efficiently even on large data sets. Furthermore, because we have that we are averaging several trees together, there is actually a significantly smaller risk of overfitting. By utilizing multiple trees, we will also have a lowered variance because there is a reduced chance of coming across a classifier that does not perform well. Same as for logistic regression, we will run our random forest algorithm for both the 'overtime' features and the 'totals' features. In doing so, we will be able to fit our model on training data and test accuracy of our test data to be used as a comparison to our logistic regression model. The parameter that we will vary each time is the number of trees used in our random forest, in which we test the best among the list [1,2,5,10,20,50].

As our last step, recall that each player on both teams has their own unique and individual role in team, namely, top lane, middle lane, jungle, bot lane carry, or bot lane support. Using all of the features described above, we can utilize the model of highest validation accuracy in order to analyze all subsets of features, and using our selected model, draw conclusions from each of these roles. For instance, the sparsity in accuracy in a particular role or the noticeably higher average accuracy will tell us what impacts each role has on the outcome of a match.

Our training data set will comprise of a list of lists in which each inner list represents a new training data point. Each inner list will contain the features that are considered and will be indexed by the enumeration of the feature described above (for instance, if we want to only consider the features damage dealt to champions, gold earned, and KDA ratio, then our feature vector will be [(1),(2),(4)]). For our output, recall that we are considering all of our data with respect to the blue side in each match. If we have that the blue team won a match, then the value representing this in our output vector will be 1, and if blue team lost a match, then the value representing this in our output vector will be 0. So, after all matches are accounted for, our output vector with size of the total number of matches (in this case, 1000) will contain only 0's and 1's.

Upon observing our available data set, we notice that while the entire set of 'totals' features exists for every data point, not all of the features in the 'overtime' features set exists for every data point. In order to accommodate for this, we devised two methods, as follows:

- Method 1 – for any data points with *any* missing features, delete such data points.
- Method 2 – for each missing feature values, use the average of such a feature from among all of the remaining available data points with that feature value available.

Once we have successfully executed these methods, we observe the accuracies of our models using each method and analyze which method is the more effective one.

**Results:**

We fit our logistic regression model and our random forests model on our training data set, utilizing, for each category of features ('overtime' and 'totals'), each one individually as well as combinations of features and the entire set of features. We have the following results for model accuracy, for both models, on each individual feature and for all features considered (note that all numbered features refer to the feature associated with the enumeration in the features listing in *Problem Formulation and Feature Engineering* above):

**Table 1:** Individual Features and All Features Accuracies for 'Overtime' Data

| **Feature** | **Logistic Regression Test Accuracy (Optimal C value) Method 1** | **Logistic Regression Test Accuracy (Optimal C value) Method 2** | **Random Forests (Optimal n_estimators value) Method 1** | **Random Forests (Optimal n_estimators value) Method 2** |
|---|---|---|---|---|
| CS Differential Per Minute | 0.676 (1) | 0.64 (0.01) | 0.671 (50) | 0.592 (5) |
| Damage Taken Differential Per Minute | 0.653 (0.01) | 0.576 (0.1) | 0.658 (20) | 0.64 (5) |
| Experience Differential Per Minute | 0.876 (1) | 0.788 (0.01) | 0.862 (10) | 0.82 (1) |
| All | 0.893 (0.1) | 0.788 (10) | 0.871 (50) | 0.796 (1) |

**Table 2:** Individual Features and All Features Accuracies for 'Totals' Data

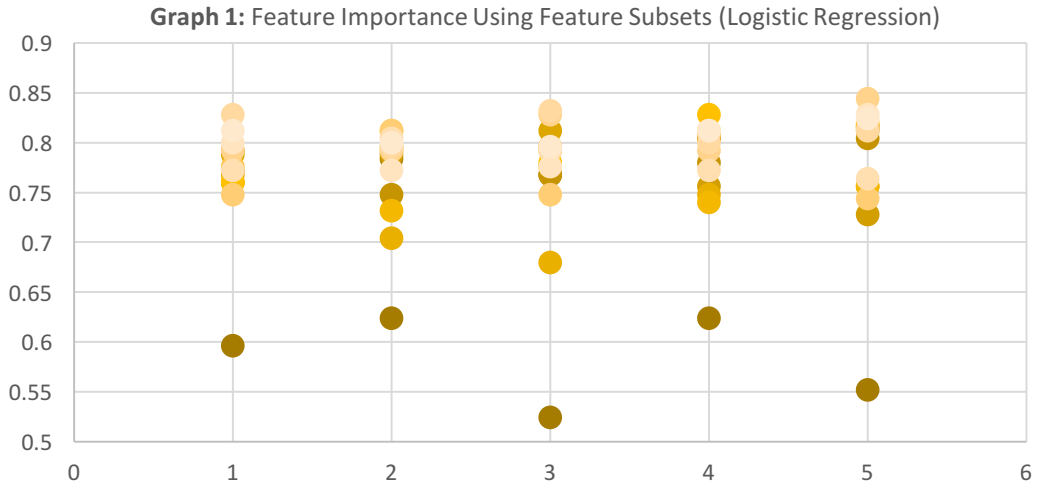
| Feature | Logistic Regression Test Accuracy (Optimal C value) | Random Forests Test Accuracy (Optimal n_estimators value) |
|---|---|---|
| Total Damage Difference | 0.856 (0.01) | 0.816 (5) |
| Total Gold Difference | 0.968 (1) | 0.96 (1) |
| Average Champion Level | 0.952 (0.01) | 0.952 (1) |
| Average KDA | 0.892 (0.01) | 0.88 (10) |
| All | 0.972 (1) | 0.964 (50) |

**Table 3:** Sets of Two Features Accuracies for 'Totals' Data

| Features | Logistic Regression Test Accuracy (Optimal C value) | Random Forests Test Accuracy (Optimal n_estimators value) |
|---|---|---|
| (1) + (2) | 0.836 (1) | 0.84 (5) |
| (1) + (3) | 0.812 (0.1) | 0.808 (5) |
| (1) + (4) | 0.784 (0.1) | 0.788 (5) |
| (2) + (3) | 0.832 (0.01) | 0.824 (5) |
| (2) + (4) | 0.808 (0.01) | 0.824 (1) |
| (3) + (4) | 0.792 (0.01) | 0.768 (5) |

**Table 4:** Sets of Three Features Accuracies for 'Totals' Data

| Features | Logistic Regression Test Accuracy (Optimal C value) | Random Forests Test Accuracy (Optimal n_estimators value) |
|---|---|---|
| (1) + (2) + (3) | 0.86 (0.01) | 0.852 (5) |
| (1) + (3) + (4) | 0.792 (0.01) | 0.808 (5) |
| (1) + (2) + (4) | 0.812 (0.01) | 0.848 (5) |
| (2) + (3) + (4) | 0.808 (0.01) | 0.82 (1) |




In Graph 1, we have that the accuracies are shown on the y-axis, and the roles of each player are on the x-axis (1 = top lane, 2 = middle lane, 3 = jungle, 4 = bot lane carry, 5 = bot lane support).

**Analysis and Conclusions:**

We notice that from our results, our accuracy for our 'totals' features had all in general been greater for logistic regression than for random forests for each individual feature and also when considering all features at the same time. However, this difference is not by much. We have that there is only about a 0.008 difference in accuracy when considering all 'totals' features and a difference of only a maximum of 0.04 when considering individual features, occurring at the total damage dealt to champions feature. When we analyze sets of two features, we notice that depending on the features selected, there are some cases where logistic regression out-performs random forests and some cases where random forests out-perform logistic regression. As such, we can generalize that the performances of the two models are relatively equal for two features. Similarly, when we consider three features, both models perform relatively equally, where for some combinations either one can out-perform the other. This implies that due to the greater simplicity in the logistic regression model than the random forests model, there could have been noise-fitting of training data in the random forest model that causes inaccuracy in testing data. This can also imply that the true model for match prediction is a simpler model. The greatest accuracy in feature combinations occurs when we utilize all four features at the same time, where doing so can produce accuracy improvements of up to about 0.2. When we analyze the results from 'overtime' features, we notice that when we consider all features, there is actually lower accuracy than considering all of the 'totals' features, by as much as about 0.2. Additionally, each individual feature in the 'overtime' features is outperformed in both models by the individual features in the 'totals' features, by as much as almost 0.4, with the exception of one feature in the 'overtime' features, experience differential per minute, which is relatively consistent with the accuracies of all of the 'totals' features. From this, we conclude that, in general, League of Legends matches are much more dependent on the features near the end of the game rather than the features that occur in the beginning and middle of the game. Additionally, for our overtime data, we notice that there is

greater accuracy, in both models, when we delete data points with missing feature values than when we fill in missing feature values with average of respective features, when considering all features in the 'overtime' feature set. Individual features in the 'overtime' feature set have relatively equal accuracies for both methods and using both models. Our last piece of data to analyze is our feature importance with respect to each individual role. We notice that there is the greatest variation in accuracy for middle lane players, which implies that middle lane has the greatest variation in impact towards a team's victory or defeat in a match. Furthermore, we have that as bot lane carry has the greatest average accuracy, we can conclude that success in the bot lane carry role has a greater tendency to result in victory in a match.

**Future Research:**
Riot Game's API does indeed possess a plethora of information regarding many League of Legends matches and statistics of each player. However, in future projects, should this research continue, we hope to be able to incorporate more complicated features such as those in the pre-game champion selection, including runes selected and summoner spells selected. Doing so will most likely involve a 1-hot encoding of our categorical data and the runes will have to be specific to the type of champion role (tank, mage, marksman, etc.). Furthermore, we would also incorporate API's from additional sources such as champion.gg. Because League of Legends is not a static game, in the sense that updates occur every few weeks, each champion and roles in one patch will not necessarily have the same impact on games as in another patch. Thus, it is important to be able to consider additional features such as champion win rate (which can indeed be sourced from champion.gg) when making match predictions.

**References:**
[1] Tim Elfrink. *Predicting the Outcomes of MLB Games with a Machine Learning Approach*. 2018. https://beta.vu.nl/nl/Images/werkstuk-elfrink_tcm235-888205.pdf

[2] Stylianos Kampakis and William Thomas. *Using Machine Learning to Predict the Outcome of English County twenty over Cricket Matches*. https://arxiv.org/pdf/1511.05837.pdf

[3] Avinash Navlani. *Understanding Random Forests Classifiers in Python*. 2018.

*All aspects and responsibilities of this project are completed by Jordan Widjaja.*