# 1. Introduction

**T**he goal of our analysis is to predict the winning team given data about matches in the video game League of Legends and determine which factors are most important to winning.

League of Legends is an online multiplayer video game released in 2009, currently with over 150 million registered players. Players participate in 5 versus 5 matches where each play picks a “champion” to play and take on a specialized role in the game. The main objective of each match is for a team to destroy the other team’s “nexus” building, but there are a variety of side objectives that give teams various advantages. The majority of these side objectives are measured in our dataset.

Analyzing how to win in League of Legends is important because of the millions of players competing against each other to prove their skill level and achieve high leaderboard rankings. There are also countless tournaments held worldwide, multi million dollars prizes, and professional players who have built their careers on League of Legends. The sheer number of players result in huge repercussions for gameplay when the developers of League of Legends make gameplay changes - our utilization of recent data makes our analysis novel and far more relevant than any previous attempts to analyze the game.

# 2. Related Work

A game as large and competitive as LoL attracts many academic studies. Every study focuses on a slightly different factor contributing to the only outcome that matters: winning or losing. One paper, “Profiling Successful Team Behaviors in League of Legends”[7] leverages M/L to analyze player behaviour. Another, “A Confidence-Calibrated MOBA Game Winner Predictor,”[8] delves into the methods for analysis themselves. Most published work is quite scientific, but we wanted to explore on a tactical level, what teams should be focusing on accomplishing on Summoners Rift. That’s why our niche is looking at the game-level data and picking features that reflect actions and objectives during actual gameplay.

# 3. Methodology

### 3.1 Data Exploration

1. Missing value

There are 10+ columns with 10%-20% data missing values. They are all categorical columns, with 6-152 unique values in each column. Since each game is independent from others, we cannot fill values with nearby values.By multiple imputation, a strategy for imputing missing values by modeling each feature with missing values as a function of other features in a round-robin fashion. Multiple imputation offers great advantage over other missing data techniques in terms of its flexibility.[1] By choosing estimators as a decision tree, initial strategy as most frequent, finishes the imputation.

1. Histogram

Histograms help us check the distribution of data points, any variation can quickly be detected by this graph. We used histograms to check if there is sparse data.

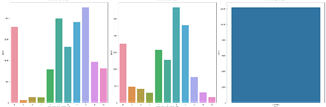


Fig. 1. histogram for categorical variable [Appendix 1]

From the histograms for each categorical column, it looks like the type column is sparse data, delete this column, because it has 100% the same value.

1. Correlation:

Correlation map: [Appendix 2]

Since there are too many attributes, use python to find pairs of correlations > 0.95 to reduce multicollinearity. Multicollinearity refers to a situation in which more than two explanatory variables in a multiple regression model are highly correlated.

|  |  |  |
| --- | --- | --- |
| team\_one\_level | team\_one\_gold | 0.964718 |
| team\_two\_level | team\_two\_gold | 0.965946 |
| team\_two\_kills | team\_one\_deaths | 0.999336 |
| team\_one\_kills | team\_two\_deaths | 0.999521 |

Table 1

High correlated variables

Here is the result of pairs. Delete one of the features for each pair to reduce multicollinearity.

### 

### 3.2 Data Preprocessing

1. **Type of columns**

We have string type, numerical type and float type data. For discrete numeric variables, when unique values > 50, consider them as continuous variables. Such as duration is the total time for the game, it is numeric type in our dataset but should be considered as a continuous numeric variable.

1. **Encoding**

According to the types of columns, using different methods to encoding our dataset. I’m using onehog encoding for smaller than 20 unique values, label encoder for 20-50 unique value, grouping when greater than 50 unique values in each column,

Onehog works well with columns for less unique values, so I’m using onehog encoding for smaller than 20 unique values for each column . However, onehog will generate too many columns when the number of unique values in each column is large. So, use a label encoder for 20-50 unique values to generate less columns. However, when unique values become larger in label encoder, it may have some bias, since numbers are considered as ranking. Hence, use grouping since the column has too many unique values. I choose to group by frequency.

1. **Binning**

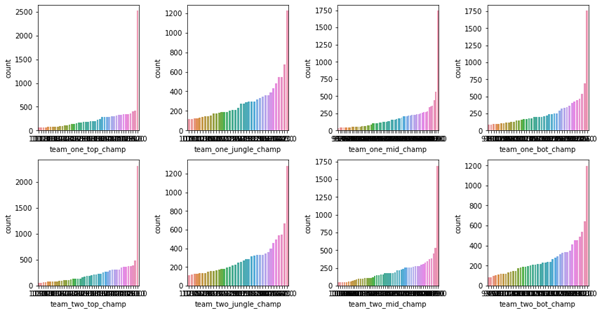


Fig. 2. Sorted frequency histogram

Visualize the columns with unique values greater than 50, by counting frequency because they are all categorical variables.

I choose to group with 7 groups, then use onehog encoder again to convert to 0s and 1s.

1. **Scaler**

Our data has continuous numeric columns with different scales. So, I’m using the minmax scaler to scale continuous numeric columns to 0-1.

### 4.3 Clustering

Temporarily dimensional reduction to 2 by PCA for the visualization purpose.

1. **K means**

K means clustering aims to partition data points to k clusters, with each observation belongs to the cluster with the nearest mean. It assigns data points to a cluster by minimizing the sum of the squared distance between the data points and the cluster’s centroid. I’m choosing the best k for minimum intra cluster distance and maximum inter cluster distance. The result is k = 5.

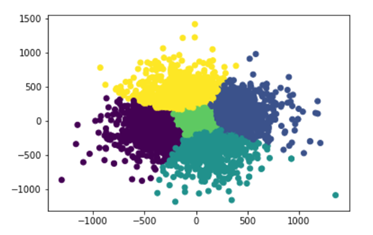


Fig. 3. K-means Clustering

From the graph, it has a clear boundary among each group.

1. **Hierarchical clustering**

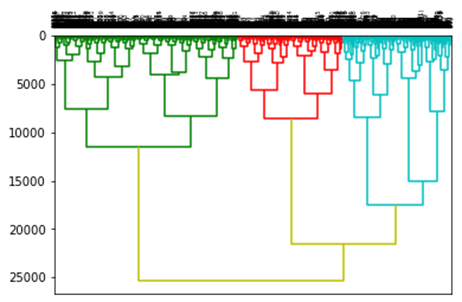
****

Fig. 4. Hierarchical Clustering

Hierarchical clustering deals with data in the form of trees. The process uses measures such as distance to neighbours to decide which objects might be in the same cluster.

I’m using agglomerative hierarchical clustering. Agglomerative hierarchical clustering analysis is performed by calculating a pairwise dissimilarity matrix and then clustering similar observations until all observations are grouped within a cluster.[2] In this algorithm, each object is initially considered as a single leaf. At each step of the algorithm, the two clusters that are the most similar are combined into a new bigger cluster (nodes). This step is iterated until all points are in this cluster.

I’m using the bottom-up approach, since it’s simpler and faster. The bottom-up method assumes each object is its cluster, then iteratively joins the two most similar clusters.

I choose Ward’s distance measure, because it’s usually the best measurement, but the most expensive one. Ward’s distance measure is the sum of within-cluster sum of squares for both clusters.

Hierarchical clustering provides a clear relationship between groups of data, but it has a high computational cost.

By considering the previous result, our group can choose to use the grouping from K means, append the grouping as a new column in our original dataframe.

### 4.4 Random Forests & Feature Selection

Our group uses the python sklearn, Select From Model package with decision trees, random forests, ada boosting, and select k best by f\_classif score function.

We believed, accurately, that random forest would perform well for this binary classification problem. Due to it’s innate cross-validation where training and testing partitions are shuffled across the entire dataset, we wouldn’t be overly concerned with overfitting. The power to handle our relatively large dataset with high dimensionality meant fast initial accuracy testing (96-97%). Increasing the number of trees had large diminishing returns, and the class 1 & 0 target records were already quite balanced, so not much tuning was needed for the sklearn classifier.

Decision trees and random forests can calculate variable importance. The decision tree classifies a population into branch-like segments that construct an inverted tree with a root node, internal nodes, and leaf nodes.[3] Decision trees consider all features and create a split that best separates class labels. The RF consists of hundreds of decision trees, each built over a random extraction of the observations from the dataset. When we train a tree, it is possible to compute how much each feature decreases the impurity, such as Gini. All Gini importances sum to 1, so with 171 features, we can say with confidence our multiple features over 0.05 were quite impactful. This was proven in our feature selected random forest models as well.

As well, AdaBoost algorithm embeds a feature selection mechanism based on training a classifier for each feature. [4] F-classif is ANOVA F value. We also use this method to select features, determining whether the means from two or more samples of data (often three or more) come from the same distribution or not.

### 4.5 XGboost

Xgboost is a method as a gradient boosting tree. Tree boosting is a highly effective and widely used machine learning method. Other than normal boosting method, XGboost include a novel tree learning algorithm is for handling sparse data; a theoretically justified weighted quantile sketch procedure enables handling instance weights in approximate tree learning.[5]

By cross validation score with 5 folders, my result for XGboost is 0.971. After parameter pruning with learning rate, max depth, min child weight and other parameters, my final score is 0.972.

**4.51 XGboost + Bagging**

Bagging is an ensemble method for improving unstable estimation or classiﬁcation schemes. Bagging

is also a smoothing operation which turns out to be advantageous when aiming to improve the predictive performance of classification trees. [6]

The cross validation score for XGboost + Bagging is 0.975, which makes the improvement.

### 4.6 SVM

Support vector machines are great for two-class classification problems and work extremely well for problems that require a higher dimension. It uses SVC classifiers to help build the model.In our case, the Support vector classifiers aim to find the best hyperplane that best separates a dataset into two groups. SVM constructs a hyperplane in multidimensional space to separate different classes and generates optimal hyperplane in an iterative manner, which is used to minimize an error.

We first tried the SVM model with the original cleaned dataset and linear kernel. However, it took extremely long execution time as the dataset included a lot of attributes, which made it hard for SVC to find a maximum marginal hyperplane. Then we fine tuned the model with a feature selected dataset and tried a polynomial kernel, gaussian kernel, and sigmoid kernel to see if they can improve the accuracy. Our final choice is linear kernel as it provided the highest accuracy at 97% and took less execution time.

Figure

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | f1-score | support |
| 0 | 0.98 | 0.98 | 0.98 | 1220 |
| 1 | 0.98 | 0.98 | 0.98 | 1221 |
| accuracy |  |  | 0.98 | 2441 |
| macro avg | 0.98 | 0.98 | 0.98 | 2441 |
| weighted avg | 0.98 | 0.98 | 0.98 | 2441 |

### 4.7 MLP

We chose the Multilayer Precipitation Classifier as our neural network predictor because it is a general network suited for feature data we selected. MLPClassifier under sklearn has a few parameters with the main tunable parameter being the number of hidden layers and nodes. We experimented with different numbers of layers to test out the effect of hidden layers and nodes per layer.

According to Table ##, we found the model with the highest prediction accuracy is one hidden layer with 4 nodes with an accuracy of 96.64%, a recall of 95% and precision of 98%. We concluded that giving our data, more hidden layers and nodes do not correlate with high accuracy. This is likely a result from our selected data have clear boundaries and strong and direct links to the prediction outcomes. The setting with 4 nodes and 1 layer will be used in our final ensemble model.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layers | Nodes | Accuracy | Recall | Precision |
| 1 | 4 | 0.9664 | 0.95 | 0.98 |
| 3 | 32,16,8 | 0.9570 | 0.96 | 0.96 |
| 7 | 512,256,128,64,16,8 | 0.9611 | 0.95 | 0.97 |

### 4.8 Ensemble

To finalize our model, we tested all the aforementioned methods under the same settings to rank their accuracy. We also incorporated other commonly used predictors such as logistic regression, naive bayes, and KNN to find a potential baseline model for our final model. For our ensemble model, we built 3 different stacked classifiers from the pool of predictors. The results of our ensemble process will be reported under the experiments and results section.

# 4. Dataset

Source and Method of Collection

Our dataset is sourced from the Riot Games, the developer of League of Legends, through the use of an API. The Riot Games API contains detailed information on almost every game of League of Legends played including each player’s activities during the game and at what time they were performed; the features that were extracted for our dataset represents a relatively small subset of all features available.

To extract data from the Riot Games API, we wrote a script that utilized the Javascript libraries ReactJS for a basic user interface and Axios for generating XMLHttpRequests. The script looped API requests, preprocessed raw data, and aggregated it into a downloadable csv file. After deploying the script to the Amazon Web Services cloud, each member of our group was able to extract roughly 3000 records from the Riot Games API.

Preprocessing

The preprocessing required mostly involved converting Riot Games’ format for its data into one that would be relatively easy to work with in Python. There was an allowance for missing values and redundant features that would be appropriately handled in later steps.

An example of preprocessing that was done was converting a feature extracted from the Riot Games API, championId (Integer), into champion names and types that were used in our dataset. We did this by parsing through a JSON file sourced from Riot Games that contained the appropriate Id-to-information mappings.

Summary Statistics

* There are 12,201 records, each representing a game of League of Legends, all of which were played by players of similar skill level.
* Of the 79 features, 17 of them are categorical and 62 are numerical.
* The dataset is almost perfectly balanced - “team one” wins 49.16% of the time and “team two” wins 50.84% of the time.

# 5. Experiments and Results

### 

### 5.1 Experiments

During the process of examining different models, we found some models that can potentially improve the prediction results but ended up being inappropriate due to the nature of the dataset. The first one is convolutional neural networks. CNN is a deep learning method that is commonly used in picture classification as it can operate a volume of inputs and each layer of CNN tries to find a pattern or useful information of the data. It is difficult to apply this model to our project since our dataset includes a lot of features and applying the filters over inputs may incur high computational cost. Another method we considered is transfer learning which is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. This method is usually paired with CNN and intends to improve deep learning results. As our dataset did not include any text or image data, this method is not applicable to our projects.

### 5.2 Results

Under 5-fold validation, our individual models achieve the following accuracies;

**Accuracy: 0.9735 (+/- 0.0020) [XGBoost] [time execution: 54.0256]**

**Accuracy: 0.9731 (+/- 0.0031) [SVM] [time execution: 753.6416]**

**Accuracy: 0.9693 (+/- 0.0029) [Random Forest] [time execution: 7.8166]**

Accuracy: 0.9625 (+/- 0.0023) [MLP] [time execution: 23.7477]

Accuracy: 0.9485 (+/- 0.0023) [Naive Bayes] [time execution: 0.1037]

Accuracy: 0.9438 (+/- 0.0074) [Logistic] [time execution: 1.1695]

Accuracy: 0.7953 (+/- 0.0104) [KNN] [time execution: 3.0558]

Because KNN had the lowest accuracy out of all the individual models, we chose KNN as our baseline approach. Based on the results of our individual model, the three stacks we built contained the following predictors:

StackingClassifier(Random Forest, xgboost, SVM, MLP), StackingClassifier2(All models), StackingClassifier3(Random Forest, xgboost, SVM). When tested again under 5-fold cross validation, the stacked models achieved the following results:

Accuracy: 0.9693 (+/- 0.0029) [StackingClassifier]

Accuracy: 0.9714 (+/- 0.0036) [StackingClassifier\_2]

**Accuracy: 0.9749 (+/- 0.0033) [StackingClassifier\_3]**

StackingClassifier3 which contained our top 3 predictors ended up yielded the highest accuracy, a 17.61% increase from the baseline KNN model. The 3rd ensemble classifier is our final model for the classification task and it is reported to have a 97.49% accuracy.

# 6. Group Member Contribution

Tao Shan - assisted in the data collection process and collected 1500+ data from riot API. Programmed data cleaning, data exploration, data preprocessing, clustering, XGboost, Bagging. Also, I make presentations and write reports for these parts.

Yao - assisted in the data collection process and collected 1500+ data. Programmed and tested the feasibility of SVM and CNN models and fine tuned the model with different kernels. Prepared slides and writing about SVM and CNN for both the report and presentation.

Hansen - Riot API data pull. Programmed “pure” random forest classifier, random forest feature selection and related visualization. Related slides. “Related Work” and RF segment in report.

Apollo - Helped with Riot API and 3000+ data entries retrieval. Programmed MLP models and completed related slides and report sections. Concluded and reported experiment results in presentation and report.

Adam - Wrote the data collection script in its entirety and collected ~5000 points of data. Performed some data preprocessing and wrote the introduction and data section in the report and presentation.

# 7. Replication Package

Github Repo: https://github.com/AWQian/League-Data

# 

**References**

[1] Melissa J. Azur, Elizabeth A. Stuart, Constantine Frangakis, Philip J. *“*[Multiple imputation by chained equations: what is it and how does it work?](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3074241/)*” Leaf Int J Methods Psychiatr Res.*, pp. 40–49, Feb. 2011.

[2] Winder Z, Sudduth TL, Fardo D, Cheng Q, Goldstein LB, Nelson PT, Schmitt FA, Jicha GA and Wilcock DM, “Hierarchical Clustering Analyses of Plasma Proteins in Subjects With Cardiovascular Risk Factors Identify Informative Subsets Based on Differential Levels of Angiogenic and Inflammatory Biomarkers,” *Front. Neurosci*, pp. 14–84, 2020.

[3] Song, Y. Y., & Lu, Y, “Decision tree methods: applications for classification and prediction,” *Shanghai archives of psychiatry*, pp. 130–135.

[4] Chensheng Sun, Jiwei Hu and Kin-Man Lam, “Feature subset selection for efficient AdaBoost training,” *2011 IEEE International Conference on Multimedia and Expo, Barcelona, Spain, 2011*, pp. 1–6.

[5] Chen, Tianqi, and Carlos Guestrin, *“XGBoost.” Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

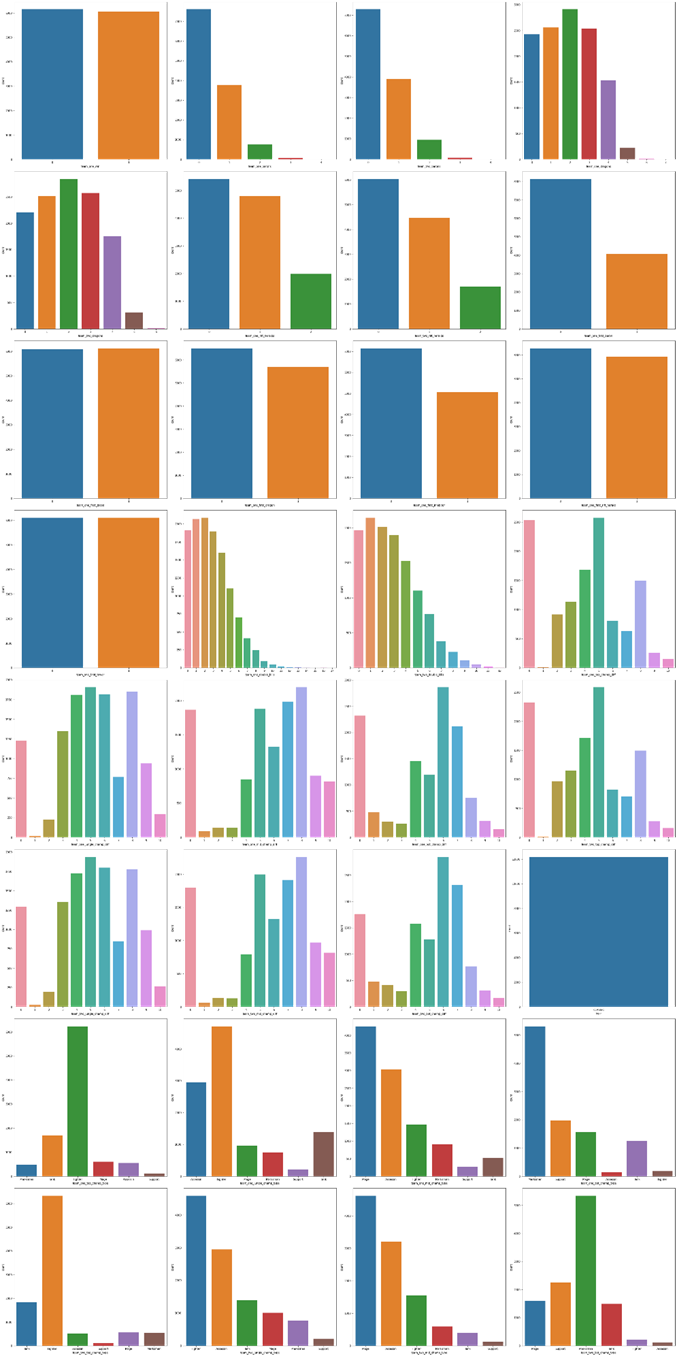
[6] Bühlmann, Peter, “Bagging, Boosting and Ensemble Methods,” *Handbook of Computational Statistics*, 2012.

[7] Fernando Felix do Nascimento Junior, Allan Sales da Costa Melo, Igor Barbosa da Costa, and Leandro Balby Marinho, “Profiling Successful Team Behaviors in League of Legends. In Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web,” *Association for Computing Machinery*, pp. 261–268, 2017.

[8] D. -H. Kim, C. Lee and K. -S. Chung, “A Confidence-Calibrated MOBA Game Winner Predictor,” *2020 IEEE Conference on Games (CoG)*, 2020.

**Appendix**

**1.**

****

**2.**

