League of Legends
Winning Prediction and Hero Recommendation

ZENG Cheng *20645703*
CHEN Chen *20619546*
LIU Xueling *20643822*
SONG Huancheng *20635473*

Abstract

League of Legends is a multiplayer online battle arena video game developed and published by Riot Games. Players are concerned with lineup of their team, which may influence the game result. In this article, we built several winning prediction models based on several different algorithms and a Hero2Vec model to achieve hero recommendation.

1 Introduction

1.1 Project Background

League of Legends is a multiplayer online battle arena video game developed and published by Riot Games. In each game, players control characters called heroes, chosen or assigned every match, who each have a set of unique abilities. Five players work together to achieve a victory condition, typically destroy the core building in the enemy team's base.

There are five positions in each game:

TOP: A position where heroes are always hard to kill and soak up damage for their team.

JUNGLE: A position where heroes gain experience and gold by killing wild monsters and help their teammates to kill the enemy and often cooperate with MID.

MID: Heroes in this position often enjoy more team resources and have powerful damage skills.

DUOCARRY: Heroes in this position deal sustained damage over time rather than in a short burst, but they tend to have weak defense.

DUOSUPPORT: A position where heroes' skills are meant to directly aid the rest of the team like providing healing.

Players begin each match with a low amount of gold, and they can earn additional gold throughout the match in a variety of ways. This gold can then be spent throughout the match to buy in-game items that further augment each hero's abilities and gameplay in a variety of ways [1].

1.2 Project Description

Our project is going to predict the game win rate by considering different features for both a single hero and hero combination and finally achieve the hero recommendation.

In section 1, we introduce the background and basic idea of this project and list our individual contributions.

In section 2, we do the data visualization, extract meaningful features and give out intuitive results to see the win rate in some cases.

In section 3, we introduced data sources, as well as data pre-processing, feature engineering and feature ranking.

In section 4, we perform the winning prediction training procedure and analysis the models according to the training results.

In section 5, we discuss the evaluation results of winning prediction model, as well as possible improvement of this model.

In section 6, we discuss the Hero2Vec model from several aspects, including the motivation of establishing this model, the introduction of Continuous Bag of Words/Heroes (CBOW/CBOH) model and the CBOH model architecture used for exploring hero embeddings and hero recommendation.

The individual contribution of the project is shown as

Table 1.

Table 1: Individual Contribution

Name	Contribution
ZENG Cheng	Data visualization and Feature engineering
CHEN Chen	Feature engineering, Winning prediction modeling and Hero2Vec modeling
LIU Xueling	Data pre-processing, Feature engineering, Feature ranking, and Winning Prediction Model Evaluation
SONG Huancheng	Feature engineering, Model exploring, parameters finetuning

2 Data Visualization

We use a dataset which has a large number of detailed data in each game match. In this case, we are going to extract features based on heroes rather than for a match. First of all, we implement some data visualization to find out the relationship between multiple features and win rate.

2.1 Data Visualization for Single Hero

In this game, each hero has features based on their own position and attributes. Firstly, we are going to print violin plot in order to consider the vision score, and the result is shown in Figure 1.

As we can see, from season three to season eight, game designers have increased the importance of vision. In this case, we should consider that in the future, the vision may still be a important feature contributed to the game result.

Secondly, we tend to consider the number of kills for a hero in games with an upper and lower limit, the result can be seen in Figure 2.

It is obvious that no matter in which season, the number of kills have a strong influence on the win rate. The win team's heroes may always achieve a higher number of kills.

Then, we turn to draw a heatmap to find out correlation between features and win rate. Firstly, we draw a figure based on all the matches in Figure 3.

Then we find that if we put a time limit to the data, we could get more clear result, which can be seen in Figure 4 and Figure 5.

We can see that for games that end soon, KDA and the amount of damage to turrets may be quite important features to consider, while for long-lasting game, all the features make little sense.

2.2 Data Visualization for Hero Combination

As League of Legends is a team game, picking a hero who can have a good cooperation with teammates is a very important feature leading to a result of victory. Meanwhile, different hero may have different restraint relationship, so players should also consider which heroes competitive team choose.

In this case, we are going to analyze the relationship between heroes and give out some intuitive results for players to see the restraint relationship or the good cooperation between different heroes. As a result, players can choose heroes more reasonable.

Firstly, we pick out two positions where heroes always have the greatest impact on the game result and choose heroes who are popular to be used to draw the heatmap to show that when two heroes meet each other in the same position, how the average win rate will change. Results can be seen in Figure 6 and Figure 7 for MID position and DUOCARRY position respectively.

Then, we turn to show different hero combination in a team will lead to different win rate. As we can see, MID and JUNGLE, DUOCARRY and DUOSUPPORT always have more interaction in the game. In this case, we choose these two combinations to analyze in Figure 8 and Figure 9.

In conclusion, choosing a reasonable hero with considering both teammates and enemies has a strong influence on the result of victory.

3 Winning Prediction Feature

3.1 Data pre-processing

In this project,we use League of Legends Ranked Matches data from Kaggle website, which is data about 184070 League of Legends ranked solo games, spanning across several years [2].

This data set contains 7 CSV files. *Champs.csv* store id and name of champions. Matches.csv store information about matches, including *id, gameid, platformid, queueid, Seasonid, duration, creation, version*. *Teambans.csv* store

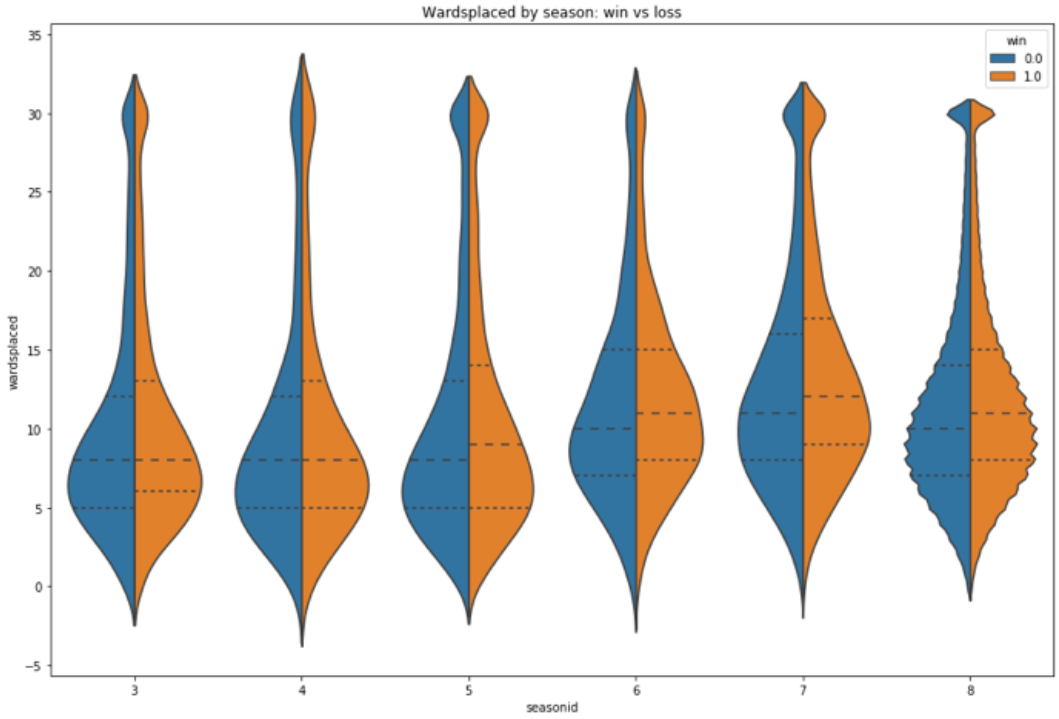

Figure 1: Wardsplaced by season: win vs loss

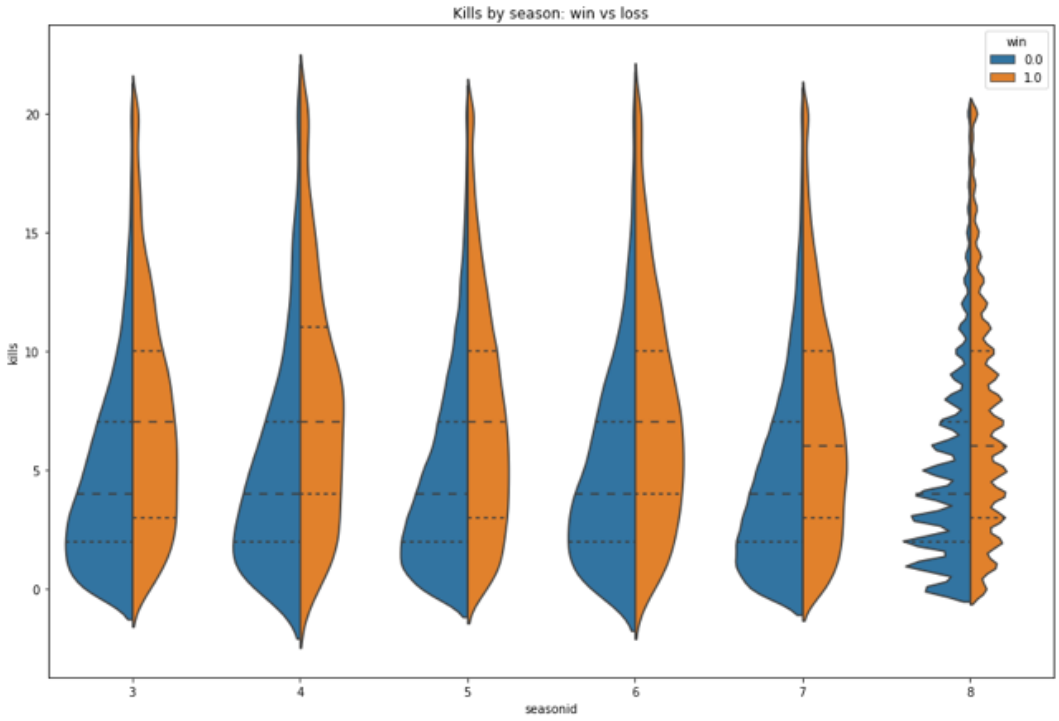

Figure 2: Kills by season: win vs loss

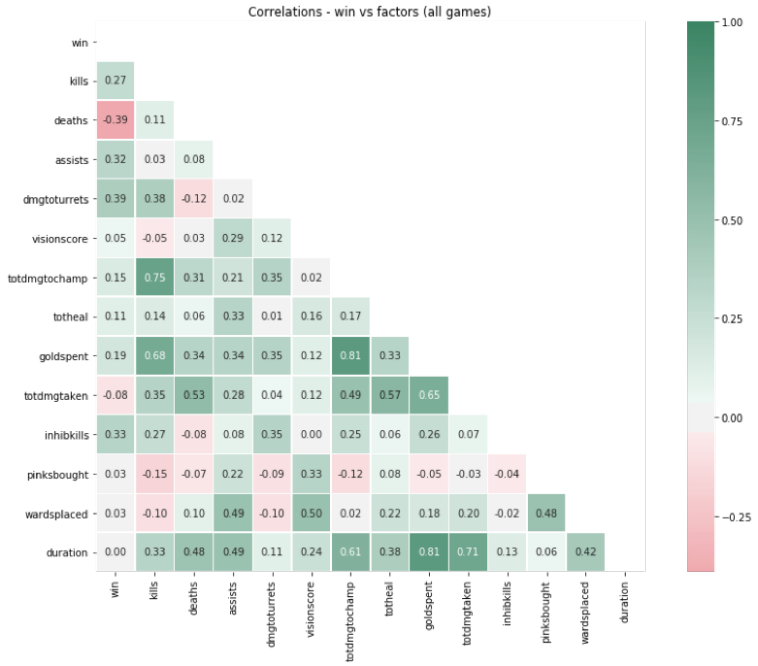

Figure 3: Correlations - win vs factors (all games)

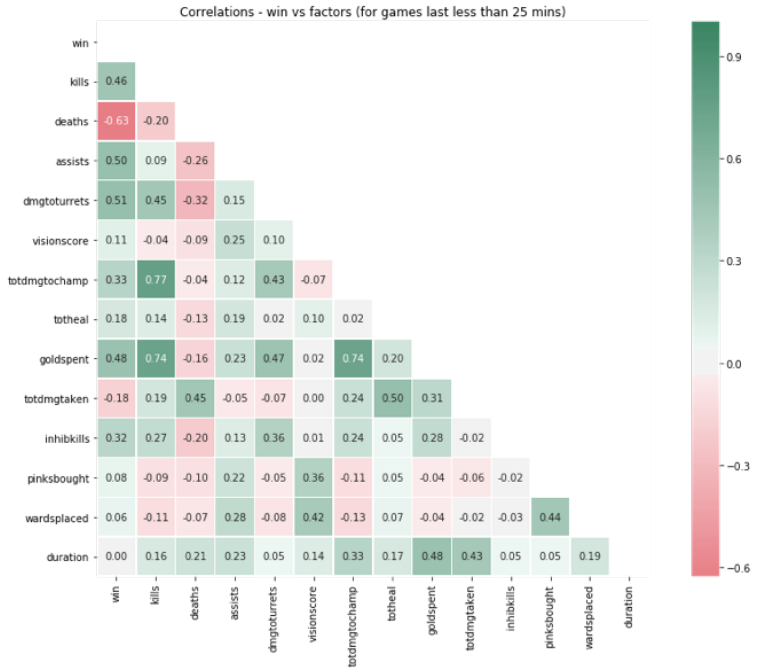

Figure 4: Correlations - win vs factors (for games last less than 25 mins

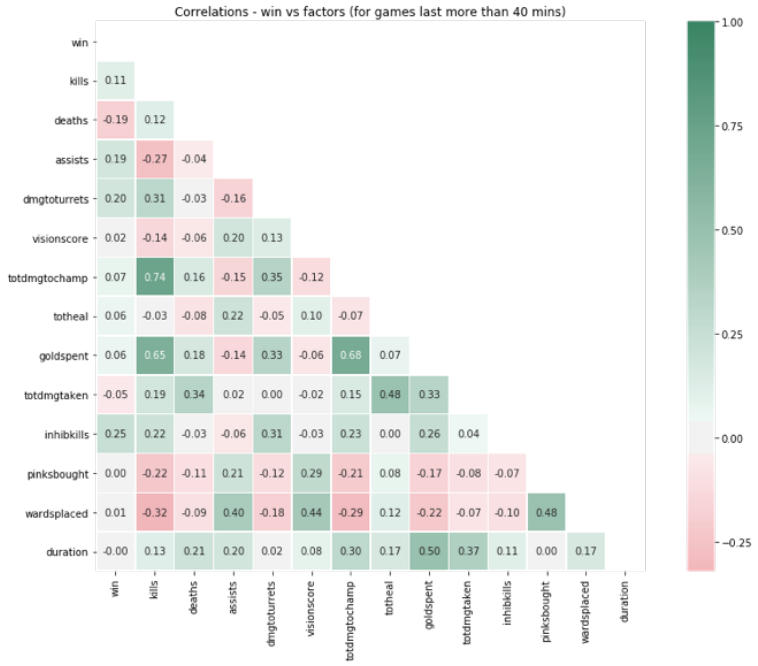

Figure 5: Correlations - win vs factors (for games last more than 40 mins

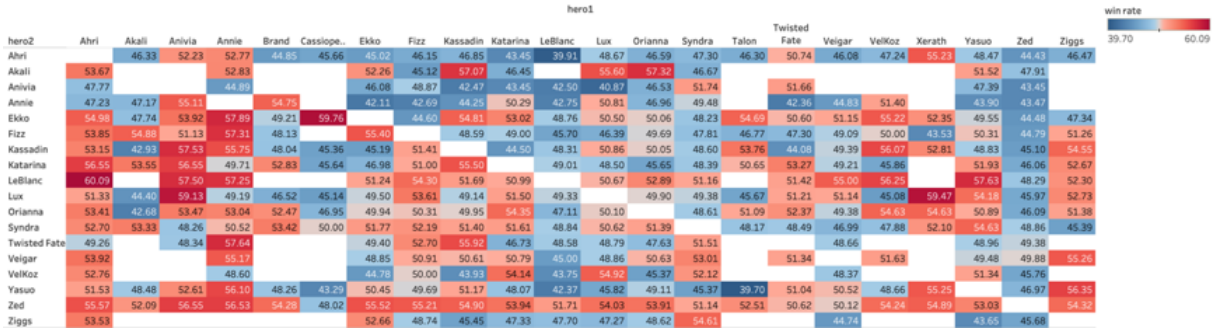

Figure 6: Win rate between MID

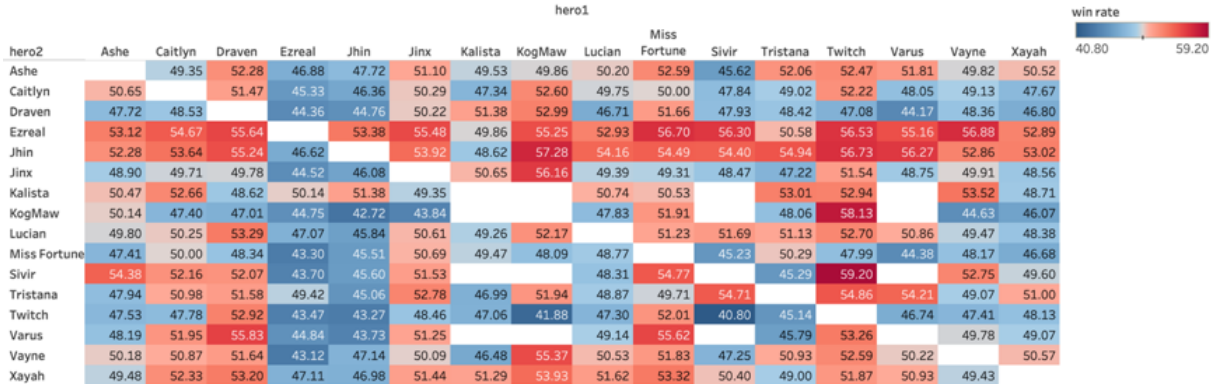

Figure 7: Win rate between DUOCARRY

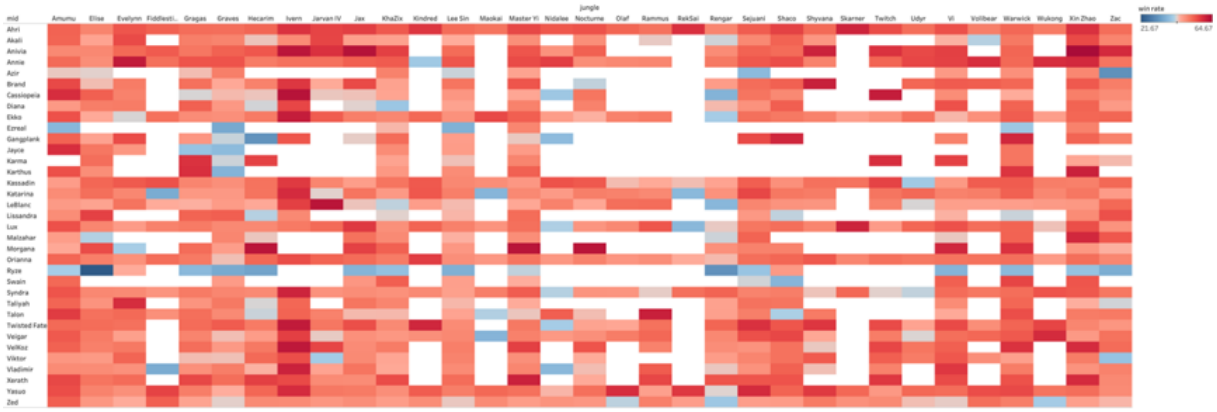

Figure 8: Win rate for JUNGLE and MID

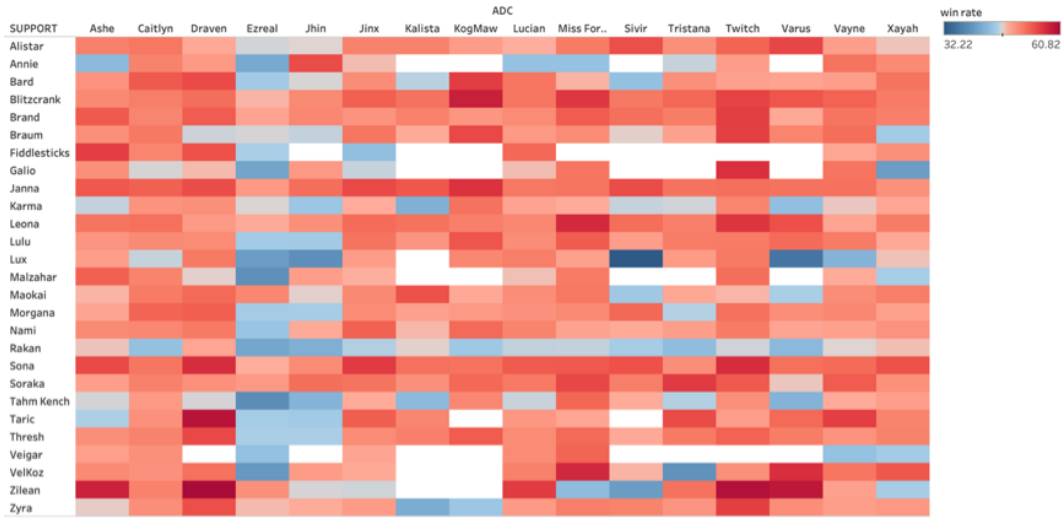

Figure 9: Win rate for DUOCARRY and DUOSUPPORT

information about champions banned in matches: *Matchid, teamid, championid, banturn*. *Participants.csv* store information about participants in matches, including *id, matchid, player, championid, ss1, ss2, role, position*. *Stats1.csv* and *Stats2.csv* store statistical data of each participant in each match, including *Win, kills, deaths, assists, largestkillingspree*, ect. *TeamStats.csv* store statistical data of each team in each match, including *Firstblood, firsttower, firstinhib, firstbaron, firstdragon*, ect.

As described above, in our raw data there exists 7 tables, and the data structure is complex. For the convenience of feature engineering, we need to transform the structure of data, and screen out information we need in the following processes. In addition, there are some missing value in the data, since our data set is large, and missing data are not many, we decided to remove missing data directly.

Firstly, we use "*participant id*" as primary key to merge those 7 CSV files in raw data. Next, we remove matches that have duplicate roles, unclassified position "BOT" or missing value.

3.2 Feature engineering

Using pre-processed data, we create a pivot with index 'matchid' and column 'team_role'. And then we merge this pivot and pre-processed data to get all the champions participated in each match and whether team 1 win or not.

Then we create features in each match. Winning score is relatively statistical win rate of champions, combination of mid and jungle, and combination of carry and support in two teams. The formula of winning score is:

$$Score = \frac{winmatchnumber}{totalmatchnumber} - 50.$$

Also, we compute mean kills, deaths, assists, damage to turrets, visionscore, rate of participation, KDA of each champion. Rate of participation and KDA are common indexes used to measure a champion's contribution in a team fight. Their formulas are:

$$RateOfParticipation = \frac{kills + assists}{teamkills}$$

,

$$KDA = \frac{kills + deaths}{assists}.$$

And we use total kills of each team as a feature of each team. For carry and mid, damage conversion ratio is a common index, too. Its formula is:

$$DamageConversionRatio = \frac{totdmgtochamp}{goldspent}.$$

For top, dmgtaken of every death is also a common index. Its formula is:

$$DmgtakenOfEveryDeath = \frac{totdmgtaken}{death}.$$

3.3 Feature ranking

It is unknown which feature is valid for a machine learning project. There are many different methods to rank features by its importance. Stability selection method is extremely general and has a very wide range of applicability, and it is based on subsampling in combination with (high-dimensional) selection algorithms [3]. The selection algorithm can be regression, SVM, or other similar methods.Recursive feature elimination (RFE) is based on the idea to repeatedly construct a model (for example an SVM or a regression model) and choose either the best or worst performing feature (for example based on coefficients), setting the feature aside and then repeating the process with the rest of the features. This process is applied until all features in the dataset are exhausted. Features are then ranked according to when they were eliminated. As such, it is a greedy optimization for finding the best performing subset of features [4].

In this project, we use stability selection via randomized Lasso, recursive feature elimination (RFE) via linear regression, several linear model feature ranking, and several ensemble model feature ranking to get the ranking score of features. Finally, we compute mean of these ranking scores, and use mean ranking to sort these features. Figure 10 shows top 20 features in mean ranking. We can find that score, kills, KDA are most important features.

4 Winning Prediction Model Training

4.1 Training Procedure

We use ensemble learning to perform the winning prediction and XGBoost, Light GBM and CatBoost models are used as Classifiers to do the training. Figure 11 shows the flow diagram of the training procedure. At the beginning, the processed data is read and split into train/test parts with the test_size of 0.3. After that the GridSearch function is used

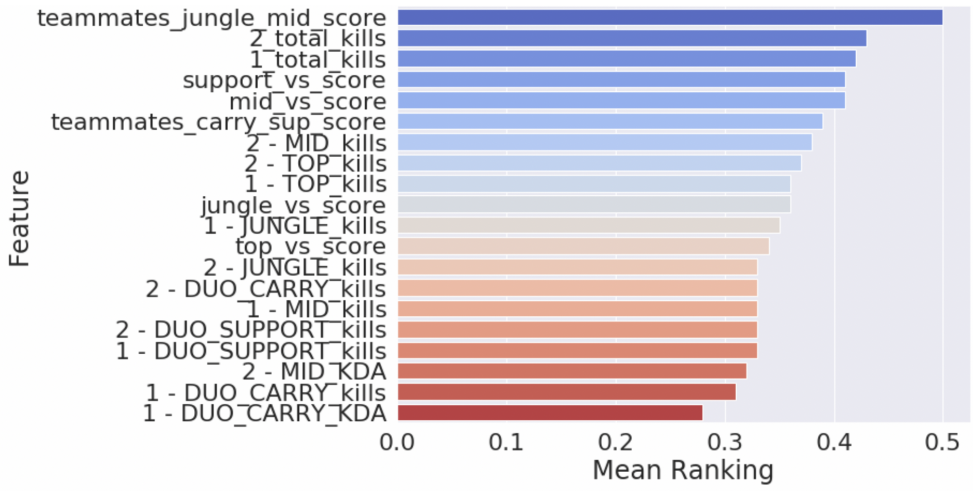

Figure 10: Feature Ranking

to finetune the parameters during which the best parameters combination and the finetuning time of each model will be recorded every calling. Then the training data is fitted into models with finetuned parameters to perform the evaluation.

4.2 Model Parameters

eXtreme Gradient Boosting (XGBoost)

XGBoost was designed by Tianqi Chen in 2014, and had become widely used since 2016. It's an integration of CART regression trees and is based on boosting ensemble learning.

The below parameters are set to be finetuned:

* learning_rate
* max_depth: the maximum depth of the decision tree XGBoost build. The splitting is stopped when the depth of the tree reached max_depth.
* min_child_weight: the minimum weight of subtrees when a tree is being split. If subtrees weight of a tree is less than min_child_weight, it'll not be split.

Light Gradient Boosting Machine (Light GBM)

Light GBM is a gradient boosting framework released by Microsoft in 2017. Compared with XGBoosting, Light GBM is more efficient with the implementation of Gradient-Based Onde-Side Sampling (GOSS) and Exclusive Feature Bunding (EFB) Algorithm. Unlike XGBoosting, which scans all samples for every feature to select the best split point, Light GBM uses GOSS to randomly choose a subset of samples to compute the gradient of features and uses EFB to reduce the dimension of features.

The below parameters are set to be finetuned:

* learning_rate
* num_leaves: Light GBM uses leaf-wise method to growth the tree. Therefore, an additional parameter num_leaves is set as supplement of max_depth. when growing the tree, the number of leaf nodes will not exceed
* num_leaves.
* min_data_in_leaf: It is used to control the data size of leaf nodes to prevent the level of tree from being too deep.

Categorical Boost (CatBoost)

CatBoost is developed by Yandex and open sourced in 2017. One of the advantages of CatBoost is that it can preprocessing categorical features itself. It uses symmetric trees as the structure and replace the traditional boosting algorithm with an ordered boosting to avoid overfitting and improve the robustness of the model.

Since CatBoost has already implemented very strong parameters, there is no need to modify them too much. And the below parameters are only used to test the finetuning speed of CatBoost for further comparison.

The below parameters are set to be finetuned:

* learning_rate
* depth
* l2_leaf_reg: L2 regularization factor

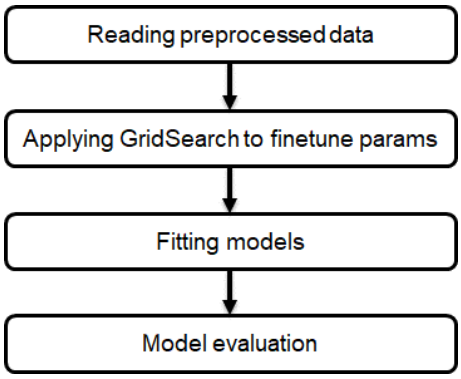

Figure 11: Flow Diagram of Training Procedure

Table 2: Training Result

	XGBoost	Light GBM	CatBoost
Parameters	max_depth=3 learning_rate=0.07 min_child_weight=10 n_estimators=100	num_leaves=12 learning_rate=0.05 min_child_samples=15 n_estimators=100	max_depth=6 learning_rate=0.03 l2_leaf_reg=3 iterations=1000
Training AUC score	0.6014	0.6048	0.7469
Training f1 score	0.6049	0.6047	0.6919
Finetuning time (for 81 fits, 100 iterations)	41.8min	3.2min	7.5min

4.3 Training Result Analysis

Table 2 records the training result with the parameters from finetuning procedure. From the table we can find that in the same finetuning iterations and parameters, the training speed of XGBoost is relatively slow comparing with the other two models and Light GBM is the fastest without any descending of training result. It's noted that the training result of CatBoost is high because we use the default parameters with the iterations of 1000 (There is no need to worry much about overfitting in CatBoost). The test result analysis of each models is left to Model Evaluation Section.
Another phenomenon of the training result is that, although training with a very high iterations, the measure scores of each model is still low. The reason for it, in our analysis, is that, the winning data distribution of the lol games is mainly depending on the match information. However, when performing winning predictions, there is no way to get the match information in advance and the model can only use statistic information from past games as input features. Therefore, it is unreasonable to expect a result which is very high. Also, the low result is associate with the balance of games. There is no such a lineup that can ensure the victory of every game.

5 Winning Prediction Model Evaluation

In this model, there are 104043 records in the training dataset and 44591 records in the test dataset. In this section, we apply winning prediction models on test data to perform model evaluation. In test data, there are approximately equal numbers of records of class 0 and class 1, which is 21632 and 22959.

From the confusion matrixes in Fifgure 12, we can find that there is only a marginal performance difference between these 3 models we built. CatBoost Classifier perform a little bit better on class 0, while XGBoost Classifier perform a little bit better on class 1.

In addition, the precision, recall and f1-score of these 3 models are showed in the tables. All the scores are distributed around 0.55, differences between these models are very small. And in all these models, the performance on class 1 is better than class 0.

Accuracy of these models are showed in table 6. From the view of accuracy, XGBoost classifier is the best model to predict winning result.

Table 3: Evaluation Indexes of XGBClassifier

	precision	recall	f1-score	support
class0	0.5506	0.4852	0.5158	21632
class1	0.5638	0.6269	0.5937	22959
micro avg	0.5581	0.5581	0.5581	44591
macro avg	0.5572	0.556	0.5547	44591
weighted avg	0.5574	0.5581	0.5559	0.44591

Table 4: Evaluation Indexes of CatBoostClassifier

	precision	recall	f1-score	support
class0	0.5444	0.5086	0.5259	21632
class1	0.5640	0.5989	0.5809	22959
micro avg	0.5551	0.5551	0.5551	44591
macro avg	0.5542	0.5538	0.5534	44591
weighted avg	0.5545	0.5581	0.5551	0.5551

Table 5: Evaluation Indexes of LightGBM Classifier

	precision	recall	f1-score	support
class0	0.5478	0.4867	0.5154	21632
class1	0.5624	0.6215	0.5905	22959
micro avg	0.5561	0.5561	0.5561	44591
macro avg	0.5551	0.5541	0.553	44591
weighted avg	0.5553	0.5561	0.5541	44591

Table 6: Accuracy of Models

	XGBClassifier	CatBoostClassifier	LightGBM Classifier
Accuracy	0.5581	0.5551	0.5561

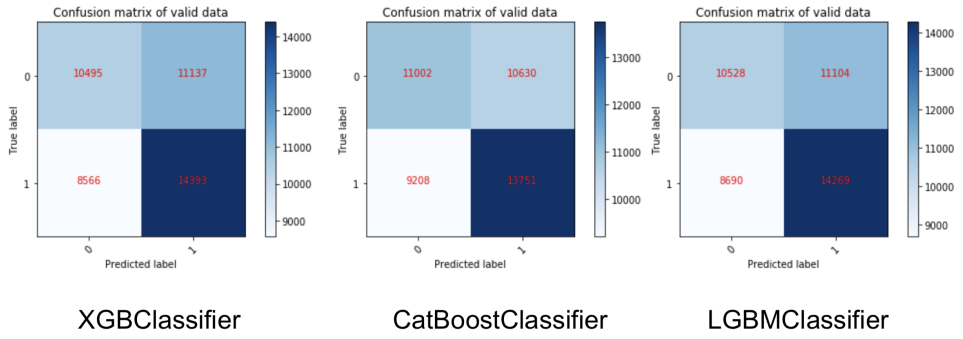

Figure 12: Confusion Matrixes

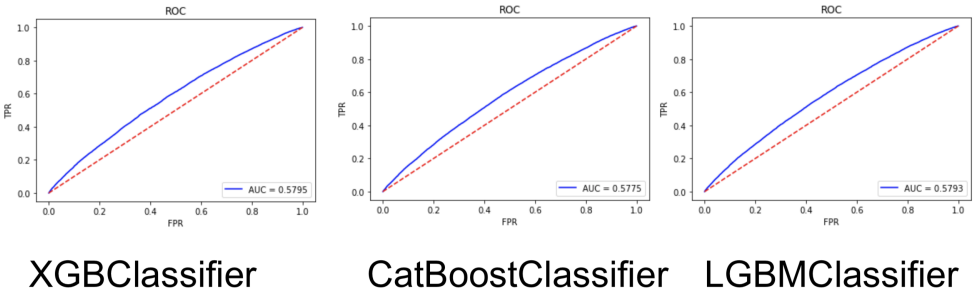

Figure 13: ROC Curves

As showed in Figure 13, ROC curves of these model still look very similar. From the view of AUC value, XGBoost classifier is also the best model to predict winning result with AUC value 0.5795.

In conclusion, the performance of 3 models we built are close to each other. Accuracy of our models on test data are close to 0.55, which is not very high. But this is reasonable because in a real match, except champions line-up, other factors such as ability of players are also of great importance on winning result. To improve accuracy, a real-time prediction model may be an approach worth exploring.

6 Hero2Vec Model

6.1 Motivation

Teamwork is one of the most significant concepts of LoL. Just as different people have different specialties in the real world, each hero in LoL has his/her strengths and weaknesses, and they play different roles including ADC, Top, Mid, Jungler, and Support in a match, which is similar to a football team that relies on the collaboration of Back, Midfielder, Forward and Striker. To some extent, a team's cooperation ability determines its winning rate in a match, so a successful team should be well balanced and collaborative. Based on this idea, we can try to find some patterns from the team composition and hero similarity. For example, which heroes are more likely to cooperate in a team, and which heroes have similar characteristics.

6.2 Continuous Bag of Words

As we know, words have semantic meaning, and there is a certain similarity between them. In 2013, a deep-learning-based model "word2vec" was created by Google [5], which could capture contextual and semantic similarity. The Continuous Bag of Words (CBOW) is a typical model of word2vec, which can be used to predict the current target word (the center word) based on the source context words (surrounding words), i.e. the probability P(center word | context words).

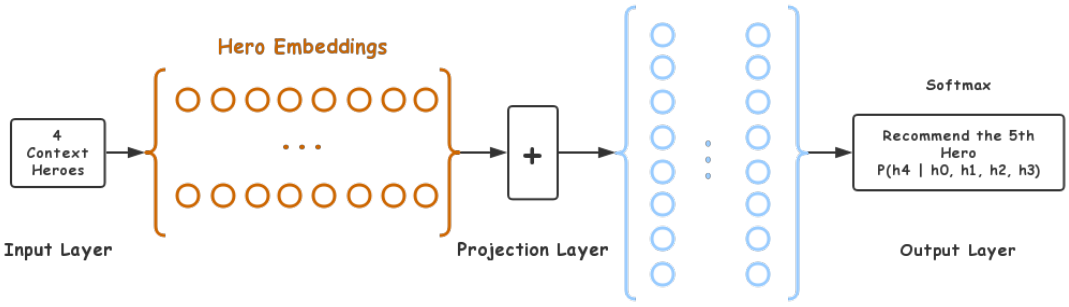

Figure 14: CBOH model architecture

6.3 Continuous Bag of Heroes

As mentioned above, words contain meanings and a sentence formed with words are more meaningful. Similarly, Heroes also have "meanings" (characteristics) behind them, e.g., some heroes are strong in attacking and some are good at defending and heroes' roles become more complicated if they form a team. Based on word2vec idea, consider a hero2vec model - Continuous bag of heroes (CBOH), which can find the similarity between heroes and predict the 5th hero after inputting other 4 heroes in a team, i.e. the probability P(h4 | h0, h1, h2, h3). [6]

Besides, permutation of (h0, h1, h2, h3) does not affect the probability, so the sum of the embeddings for (h0, h1, h2, h3) is exactly a good summary of the input. A small trick here is in addition to P(h4 | h0, h1, h2, h3), we can also model P(h3 | h0, h1, h2, h4), etc. so the dataset effectively expands 5 times.

6.4 CBOH Model Architecture

Word2hero model is a fully connected neural network with a single hidden layer (i.e. the projection layer) and the neurons in the hidden layer are all linear neurons. The number of neurons in the input layer is equal to the number of heroes in LoL. The hidden layer size is set to the dimensionality of the hero embedding and weight of the hidden layer is the hero embedding. The output layer is a softmax layer whose size is same as the input layer.

Figure 14 illustrates the architecture of CBOH model. Input 4 context heroes, and the projection layer will compute the average of hero embeddings corresponding to the 4 context heroes. Finally, the 5th hero will be predicted in the output layer, which could also be used for hero teammate recommendation.

6.5 Model Utilization

Figure 15 visualizes the hero embeddings we get from the network. It's obvious that heroes tend to cluster based on their characteristics.

As mentioned earlier, this model could also predict or recommend the 5th hero after imputing other 4 heroes in a team. Figure 16 shows some examples.

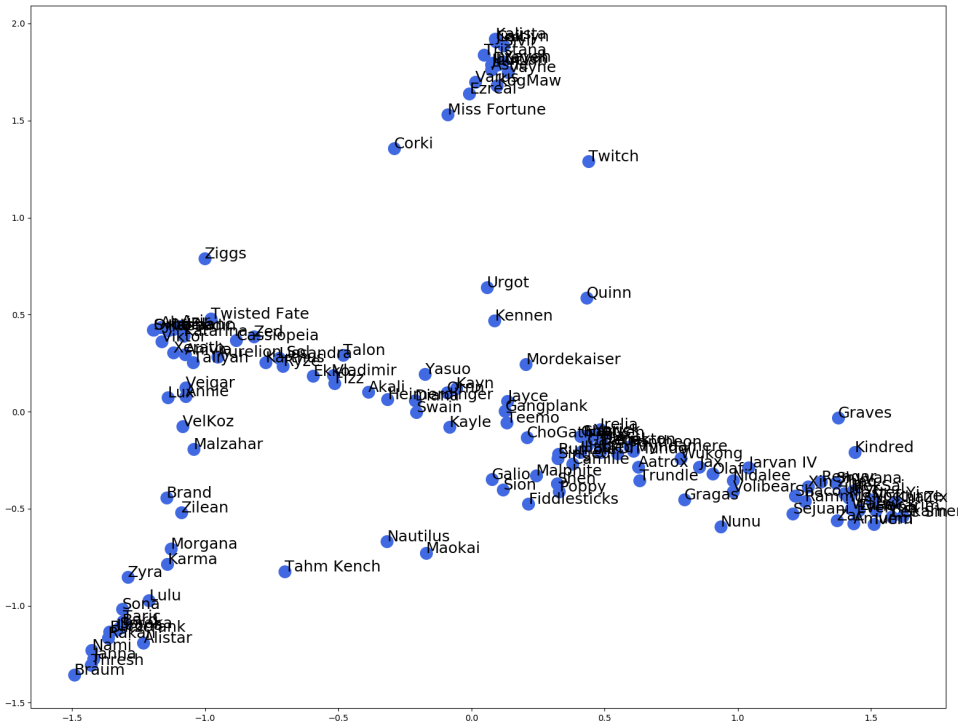

Figure 15: Hero embeddings visualization

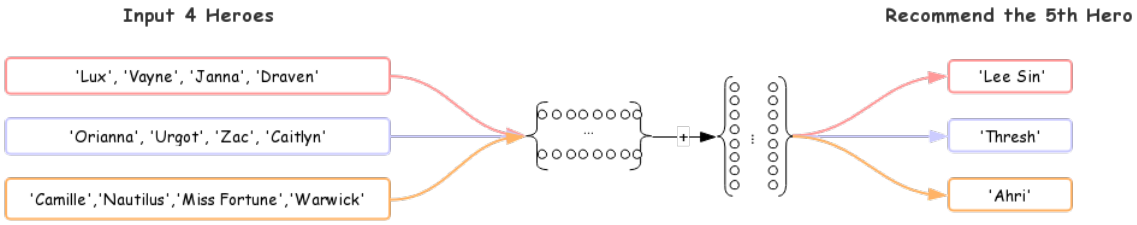

Figure 16: Hero recommendation

Acknowledgments

First of all, we would like to express our gratitude to professor Nevin L. Zhang, who have taught us machine learning algorithms and gave us this opportunity to learn machine learning in a more hands-on way.

Secondly, we owe a special debt of gratitude to Kaggle website who provided us valuable data set and Bowen Yang who inspired us with a creative analysis thought and method.

References

[1] Wikipedia. League of legends, 2019. https://en.wikipedia.org/wiki/League_of_Legends.

[2] Paolo Campanelli. League of legends ranked matches, 2017. https://www.kaggle.com/paololol/league-of-legends-ranked-matches.

[3] Peter Bühlmann Nicolai Meinshausen. Stability selection, 2009. https://stat.ethz.ch/~nicolai/stability.pdf.

[4] Ando Saabas. Selecting good features – part iv: stability selection, rfe and everything side by side, 2014. http://blog.datadive.net/selecting-good-features-part-iv-stability-selection-rfe-and-everything-side-by-side/.

[5] Tomas Mikolov. Distributed representations of words and phrases and their compositionality, 2013. https://arxiv.org/pdf/1310.4546.pdf.

[6] Bowen Yang. Predicting e-sports winners with machine learning, 2018. https://blog.insightdatascience.com/hero2vec-d42d6838c941.