Predicting Game Outcome of League of Legends

Junhan Liu
j896liu@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Abstract

In this short paper, we introduce a MOBA game AI agent based on logistic regression to predict the outcome of a game. The AI agent analyzes game events and uses the data to continuously make predictions while a player is playing his/her game. This project is inspired by the system called Colonel KI, which is a mature commercial system used to predict professional matches. The game we are interested in this paper is called League of Legends. We discover that it is possible to predict the outcome of a game using some game statistics. To build such a system, we collect the initial data from the game server maintained by RIOT company. Next, we build our predictive model based on logistic regression. Finally, we achieve a testing accuracy of 95%. The final goal of our project is that one could easily use the system as a tool to continuously predict the winning probabilities while one is playing a game on his/her local machine.

Introduction

- **Background of League of Legends**

  Multiplayer online battle arena(MOBA) game is currently one of the most popular genres of online games. Typically, in a MOBA game, there are two teams. One is the blue team and the other is the red team. Each side of a team contains exactly five players (champions), and players in a team compete against an opposing team. Each game is a larger battle composed of a series of combat events. The game we are focusing on in this report is called League of Legends.

- **Motivation, Importance and Impact**

  The project[1] is inspired by the idea of **Colonel KI** [2]. Colonel KI is a real-time game analyzing system that provides game predictions. Based on game events such as taking a dragon or a baron or a champion kill, the system continuously analyzes a game and returns predicted winning/losing probabilities of both teams. For professional tournaments, this Colonel KI system has been online for a couple of years, undoubtedly, this add-on has increased entertainment of the game and also is a significant symbol that LPL region is making a huge contribution to e-Sports. There are numerous players around the world and making contributions using their expertise. On GitHub, one could find a variety of applications and add-ons of this game. As a programmer and a player, making contributions to the game community motivates me to build something unusual.

  Numerous platforms provide real-time data checking services, for example, **OPGG.com** [3]. However, no platforms are providing online game analysis services. In this project, we would like to fill that gap and provide a reliable system to predict a game. The goal of our project is to take advantage of some game events to predict the outcome of a game. By analyzing the stats of 10 players and some game events, we would like to give a winning/losing probability for each team. The underlying implication of this project is, as well, noteworthy. Our project adds value from start to finish as players could track predictions before, during and after a match. Unlike a typical ball game, e-Sports games like LoL[4] have many more data such as gold earned, team kills, champions banned, champions chosen, total damage taken, etc. Before, players and tournament audiences often rely on gut feel and sheer excitement to predict the winner when all these numbers churned out in real-time. Instead, now, our system has the opportunity to explain a game in a more scientific way. Not only does the system make a game more enjoyable, but also users could track how game events are reflected in the predicted winning curve.

- **Problem**

  Technically, in this project, we aim to build a machine learning model to solve a binary classification problem of whether a game instance is 'win' or 'lose'. We would like to classify whether a game is winning or losing based on some game information such as 'a team's total gold earned', 'champions chosen', etc. The predicted result is given in probabilities(percentages) where a probability higher than 0.5 means win and lose otherwise. The research question our report will address is that will an

[1] GitHub: https://github.com/jimjimliu/LOL_Match_Prediction.git

[2] Colonel KI:https://www.mindshareworld.com/china/work/kfc-colonel-ki

[3] OPGG: https://www.op.gg

[4] LoL: League of Legends, https://na.leagueoflegends.com/en-us/

AI agent based on logistic regression achieves a higher classification accuracy than a baseline model based on feed forward neural network. The hypothesis is that the outcome of a game is affected by some in-game events such as 'kills', 'deaths', 'baron killed', 'gold earned', etc. For example, the team gets more gold, kills, dragons, and barons has a better chance to win. To answer the research question and test our hypothesis, we perform an experiment to compare our final model with the baseline model.

- **Methodologies & Performance Measurements**

  To perform such an experiment, we start with a very simple baseline model based on feed forward neural network. This baseline model is simple to set up and gives us a descent result. It aims to predict the outcome of a game using champions lineup as features such as champions chosen and banned before a game starts. We obtain the results generated and compare them with the final model. The main model is based on logistic regression using complex in-game data as features. These features are generated during an active game and they may include 'total kills', 'gold earned', 'champion levels', etc. When comparing the two models, if the final model achieves a higher F1 score than the baseline model does, then we conclude that the outcome of a game is affected by some in-game data.

  The quality of data collected is one of the foundations that needs to be treated with care. To train the final model, we need data sets that could live up to our expected standards. The data sets are extracted from RIOT using APIs provided by **RIOT**[5] company. The data is reorganized and transformed by following the ETL process. 7,220 accounts of LoL players are randomly selected out of 234,410 accounts retrieved. For each game rank (i.e. Diamond II), there are roughly 250 accounts selected so that the data are evenly distributed along 24 ranks. From each account, the most recent 20 matches are selected. There are 124,310 ranked matches stored in the database. To evaluate the predictive performance of our final model, the data set is split to three subsets: a training set (80%), a validation set (10%) and a test set (10%). In our work, we use classification accuracy, F1 score, ROC curve, and a confusion matrix to evaluate our final model. Finally, on top of these values, we perform practical experiments to test the performance of the model. We perform experiments by playing games and test whether game events are well reflected in the predicted results. For example, after the blue team takes all four dragons and the first baron, is the winning probability of blue team goes up? When repeatedly playing games, our algorithm captures game data continuously and feeds those data as features into our model to see whether the predicted results reflect the game situation.

- **Main Results**

  This project is based on the hypothesis that the outcome of a game is affected by some game events. There must be some indicators that we can look at to analyze a game. For this task, we have shown that it is possible to make a prediction based on game events such as total kills, gold earned, dragons taken, etc. The model was built based on logistics regression and the model has a testing accuracy of 94.8%. We compared the final model based on logistic regression with our baseline model based on feed forward neural network and showed that the performance of the final model was improved drastically. That means by using the features listed in Table 3, we can successfully predict the outcome of a game. We also achieved the ultimate goal discussed in the introduction section. After we built the model, the next step was to predict future instances. The system grabs game data from one's local machine if there is a game going on, and make predictions while one is playing the game. The system acts as a plug-in to prompt users their winning probabilities while they are playing. The practical experiment is also very satisfying. Every game event was well reflected in the predicted results.

- **Contributions**

  In the introduction section, we discussed a similar system called Colonel KI, which is a commercial system that is not accessible to the public. We did some researches on the Internet and it turned out there are no similar projects that have been built before. Our first goal is to find out if it is possible to predict the outcome of a game and the ultimate goal is to build a system that can be easily used by players to analyze their games. After doing researches, we found that if we achieved our goal, we would be the first to implement such a system successfully. There are no platforms providing online game analysis services. We have filled that gap and provided a reliable system to predict a game. This project adds value from start to finish as players could track predictions before, during, and after a game. Not only does the system make a game more enjoyable, but also users could track how game events are reflected in the predicted winning probabilities. The following are the contributions we have made:

  – We have shown that it is possible to predict the outcome of a game by using some game statistics. The idea is not restricted to a particular game.

  – Comparing to some similar existing works, we have achieved the highest model testing accuracy.

  – We are the first to build a system that predicts the game outcome of League of Legends.

  – Unlike many existing works, our system is complete. We have trained a model with a very high testing accuracy and F1 score, and also we have implemented a live prediction functionality which allows users to monitor the winning probabilities while they are playing.

In the following sections, we explain in more details of how data are collected, what results are generated, what goals are completed, what methods are employed, and how the model is evaluated.

[5]RIOT developer: https://developer.riotgames.com

Related Work

Trying to predict the outcome before a game is even started is not something new. Also, audiences of games are often trying to tell whether their favourite teams are winning or not based on some sheer knowledge about the teams, the scores so far, etc. Many works, programs, and researches of different types of game have been done in an exploring manner to try to discover some underlying conditions that make a team wins. In this section, some prior works related to our project are discussed.

As aforementioned, this project is inspired by the idea of **Colonel KI**[6], which is a mature product deployed in World Championship in LPL region. The success of the system "Colonel KI" tells people in the game community that it is possible to build such a system to analyze a game. To the best of our knowledge, there is not much precious work has been done on GitHub to predict results of MOBA games with neural networks. Especially, continuously predicting results while a game is still in progress has not been done by many people in the community. Programmers and researchers mostly focus on predicting results using information available before the beginning of a game. In (Kinkade, Jolla, and Lim 2015), the authors use the information of champions picked and the champions' win rates to predict the outcome using Logistic Regression and Random Forests. In (Wang and Shang 2017), the authors also try to predict the outcomes of DOTA 2 matches by using champions picked, but they are using different machine learning models such as Naive Bayes. In contrast, in this project, we are trying to address what specific in-game factors contribute to the success or failure of a team.

The Colonel KI system forementioned is a business system which is integrated with professional LPL World Championship. The underlying data are completely professional players oriented. Simply saying, it is not a system you can download and make a personal use. The system is using professional players' data to build models. The system does not apply to normal ranked games. That is, there are no open-source systems that people can use to predict their own casual games. Our project is based on a different data set. The data are normal players oriented. Any legal players' data could be stored in our database, and those data are provided by RIOT company's API. Therefore, our model aims to predict normal ranked games instead of professional Championship games. The reason behind this is very easy to understand. For casual players, a game's outcome is more and often determined by champions casts, damage made, barons taken, gold earned, and sometimes mistakes made by the opposing team, etc. For professional players, on top of all those dimensions, their IDs can also influence the direction of a game. Professional players are highly and well trained. The personal data of professional players and teams should be collected as well to better analyze a game. Therefore, it might not be accurate to apply our model on a professional Championship match. But there are some works done by peers to predict professional games. Their data are based on professional players and professional e-Sports teams. (Pobiedina et al. 2013) used game logs and player communities to analyze the factors of team success in a MOBA game. In (Hodge et al. 2019), the authors use conventional machine learning models, feature engineering and optimization to build a live predictive model which has up to 85% accuracy after 5 minutes of game play. Similarly, (Kang and Kim 2015) gathered game data from 2013 to 2014 in region LCK and use Poisson Model and Bradley Terry Model to predict professional LoL matches.

As an intermediate goal, our project aims to use different dimensions as features to perform predictions. The final goal is to continuously predict outcomes during a game. The first step of any model building projects should be baseline model building when they have data on hands. Our baseline model contains basic features such as champions chosen, champions banned, champions win rates, etc. like authors (Hanke and Chaimowicz 2017; Wang and Shang 2017; Kinkade, Jolla, and Lim 2015; Silva, Pappa, and Chaimowicz 2018) did. In (Hanke and Chaimowicz 2017), the system uses a neural network that obtained an accuracy of 88.63% in the test set. On top of those static features such as champion picked and champions' win rates, we use ELO score ranking as a new dimension in the feature set. The ELO score has been applied to many fields such as chess, NBA games, etc. In (Yan et al. 2019), authors proposed a method using ELO algorithms for prediction NBA playoffs.

Methodology

- **Data sets**

  Table 1 is a list of data sets[7] we collect from **RIOT**[8] using APIs provided by RIOT company, which is the company develops League of Legends. All the data provided by the APIs are recorded, organized, and maintained by RIOT. The goal of this work is to predict the winning probabilities of two teams in a game and to find whether game features affect a game's outcome. Game features include 'kills', 'gold earned', 'deaths', 'dragons killed', etc. Therefore, in the first place, different kinds of game data that describe a game are necessary for model training. After doing plenty of searches, there are no available data sets fit our needs. All existing sets are limited in either the size or the relevance of variables. Data features recorded in many existing sets are irrelevant. Those variables recorded lack of the abilities to describe a game well. For example, most of the existing data sets only contain the champions chosen and banned; however, this project seeks to find out whether in-game data would affect the outcome of a game. In-game data includes 'kills', 'dragons killed', 'barons killed', etc. Meanwhile, the number of instances of those existing sets are not large enough. Sets only have thousands of rows are not good enough. To better understand what factors contribute to the outcome of a game, we decide to collect data from the official game website. RIOT company

[6]Colonel KI:https://www.mindshareworld.com/china/work/kfc-colonel-ki

[7]data sets: https://github.com/jimjimliu/LOL_Match_Prediction/tree/master/DATA

[8]RIOT developer: https://developer.riotgames.com

keeps recording each game's information and detailed game data. These data sets describe a game very well in multiple dimensions, therefore, they are appropriate. In terms of limitations, for each game, there are more than 400 variables recorded. We have plenty of strategies to transform the sets to fit our needs. The features we choose are discussed in feature selection section. Using the APIs provided by RIOT, 235,487 player accounts are collected. 7,220 accounts are randomly selected. Roughly 250 accounts are selected for each game rank(i.e. Diamond II) so that the data are evenly distributed along 24 ranks. 20 most recent ranked games are collected from each player's account. The final data set contains 119,184 game instances. The training set contains 80% of the entire data. The test set and validation set contains 10% each. In the final data set, "Blue team win" is a binary data which 0 represents lose and 1 represents win. It is chosen to be the target variable (the label). The data set is collected manually using Python scripts. Therefore, the data set is balanced as Table 2 shows.

Set	Explanation
all champions.csv	game champions
champ winrate.csv	win rate of champions
match list.csv	match IDs
match stat.csv	game data of champions
matches.csv	game data of teams
summoners.csv	player accounts
game data.csv	final data set

Table 1: Data sets

Blue team win = 0	Blue team win = 1
47695	47652

Table 2: Number of instances of each target value

- **Feature Selection**

  Each data instance contains information of two teams and game data of 10 champions chosen. There are 30 columns for each team and more than 40 columns for each champion. Due to the reason that not all variables are necessary to be included as features, only some variables are kept for model training. For each game instance, only the features listed in Table 3 are kept. These features are chosen since each of them has a potential impact on a game's outcome. For example, the team gets all four dragons and the first baron has a better chance to win. All of the variables are integer, some of them are categorical or binary such as champions chosen and firstBlood. For example, 22 represents the champion 'Ashe'. Some of the features are continuous such as kills, deaths, assists, and totalMinionsKilled. The range of continuous values are not large, and normally they are within the range from 0 to 500.

- **Pre-Processing**

Team Data	Data Type
team champions chosen	categorical
team champions banned	categorical
firstBlood	binary
firstTower	binary
firstInhibitor	binary
firstBaron	binary
firstDragon	binary
firstRiftHerald	binary
towerKills	continuous
inhibitorKills	continuous
baronKills	continuous
dragonKills	continuous
Champion Data	**Data Type**
items [0-6]	categorical
kills	continuous
death	continuous
assists	continuous
double kills	continuous
triple kills	continuous
quadra kills	continuous
penta kills	continuous
totalMinionsKilled	continuous
champ level	ordinal

Table 3: Features of each game instance

From the APIs provided by RIOT, we collect a large number of player accounts of each rank. From those accounts, we retrieve 20 most recent ranked matches for each player. From each match, detailed game data are extracted. Table 1 is a list of data sets collected. The final data set is "game data.csv" which contains each team's game data and the game data of 10 champions chosen in a game. In summoners.csv, we collected a vast amount of active player accounts. From each player's account, we retrieve their most recent ranked games and store in match list.csv. From each game match, we retrieve game details and champions game data. Those are stored in matches.csv and match stat.csv. The final data set is formed by joining all the other sets together.

Unlike many existing data sets gathered by multiple third parties, the data collected are gathered and organized by RIOT company. Each data instance contains all the information of a past game. There are no missing values in the final data set. All the variables are integer values and they are well recorded. Therefore, no further actions are required to clean the data set.

- **Models & Algorithms**

  First of all, we define 'blue team win' as the target variable. The value is binary which 0 represents lose and 1 represents win. As Albert Einstein once said, everything should be made as simple as possible, but not simpler. When solving a machine learning problem, we exploit the exact same method by starting with a very simple

model. This very simple model is our baseline model. For this supervised learning problem, we first build a baseline model based on Feed Forward Neural Network(MLP) using champions chosen, champions banned, and champions' win rates as features. These features are variables one could get during the ban/pick phase of a game, and these variables will not change as a game goes on. Putting it in another way, if one of the two teams in a game is doing very well, it will not be reflected in those variables. Our baseline model is trying to tell whether the outcome of a game is determined by the champions lineup. Our baseline model is based on a feed forward neural network, which is one of the simplest types of neural network. Since it is one of the simplest neural networks, the nature of this model fits the needs of our baseline model. With previous model training experiences, a feed forward neural network is simple to set up and it provides decent results as a baseline model.

Feed forward neural networks are one type of artificial neural networks where the edges between nodes do not form a cycle. The computed information and knowledge are passed to the next level and only goes forward. A feed forward neural network contains several layers. The first layer is the input layer, which in our case, we have 30 input neurons: 10 champions chosen, 10 champions banned, and 10 champions' win rates. The input neurons represent information comes from the outside world. The input layer brings information to the hidden layers. The middle part of a feed forward neural network is called the hidden layer. Complex computations are performed. In our case, we only set one hidden layer using RELU as the activation function. The last part of a feed forward neural network is the output layer. This layer brings the computed and transferred information to the outside world. Briefly saying, the output layer returns the results of training and testing to us. Figure 1 is a simple representation of a feed forward neural network example.

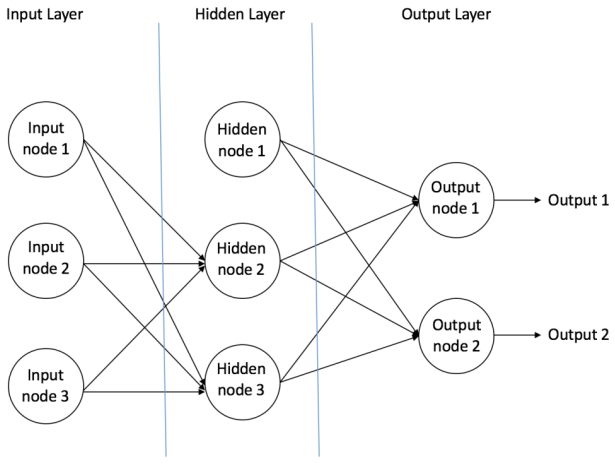

Figure 1: Feed Forward Neural Network (ujjwalkarn 2016)

The final model is built based on logistic regression using the features listed in Table 3. The goal is that we would like to discover whether the outcome of a game is affected by some in-game data and compare the result with our baseline model. On top of those features used in our baseline model, we add more in-game data. These in-game features reflect how a team is doing in a game. These features are generated and continuously updated during an active game. As forementioned, the predicted values we would like to obtain are probabilities(percentages). Linear models do not fit this criteria. For linear models, the target variable has to be continuous and unbounded. Percentage has a range within 0 and 100. Though the values are continuous and ratio scale, there are boundaries. A linear model does not output probabilities. Instead, logistic regression models for binary classification use 0.5 as a threshold. The output in logistic regression is a probability between 0 and 1, which is exactly the information we would like to obtain.

Logistic regression is an extension of linear regression for classification problems. The root difference between linear regression and logistic regression is that the activation function of logistic regression is sigmoid function. Logistic regression is a conventional machine learning technique to predict a binary outcome based on a set of features. The independent variables can be continuous, discrete, ordinal, and nominal. In our case, we use logistic regression to calculate the probability of a binary event (win or lose).

Results

- **Experimental Design**

  Our hypothesis is that the outcome of a game is affected by some in-game data such as 'kills', 'deaths', 'baron killed', 'gold earned', etc. In other words, we are able to predict the outcome of a game using some game information. For example, the team gets more gold, kills, and dragons has a better chance to win the game or the team gets the firstBlood and firstBaron has a better probability to win. To answer our research question, we compare our final model based on logistic regression with the baseline model. By comparing the two models' F1 scores, we would find out whether the information(variables) listed in table 3 would affect the outcome of a game. The final model takes features listed in Table 3, where the baseline model takes only naive features such as champions chosen, banned, and win rates of champions. On one hand, naive features such as champions chosen and banned are information one could get before a game is started. Those variables are settled during the ban/pick phase and will not change as a game goes on. On the other hand, the features used in our final model are not static. Those variables can change as a game goes on such as gold earned by a team and total kills of a team. The purpose of this comparison and the meaning of this approach is that we

are trying to conclude whether the features listed in table 3 add more values to the final model. If the final model performs much better and gives us a much better accuracy than the baseline model does, then we can conclude that the outcome of a game is affected by those game variables.

Our baseline model is based on feed forward neural network. Detailed configuration is listed in table 4. The baseline model is a simple 3 layer neural network takes only 30 input features. The final model is based on logistic regression and the parameter configuration is listed in table 5. The values are tested and altered. Values listed in the following two tables are the best ones that fit our models.

Parameters	Values
Input dimension	30
Drop out rate	0.1
Input activation function	relu
Loss function	cross entropy
Batch size	56
Epochs	30
Output activation function	softmax

Table 4: Keras neural network configuration (baseline)

Parameters	Values
penalty	L2
C	1.0
fit_intercept	True
intercept_scaling	1
random_state	0
max_iter	100

Table 5: Logistic regression configuration

To test our hypothesis, we need more meaningful features to train the logistic regression model to see if it performs differently. In match stat.csv, there are more than 400 columns recorded describing a game instance. In the first place, we do not know which ones are useful and which ones to discard. Therefore, we keep deleting features in our feature set to find out which variables are not that useful to obtain high accuracy. As a result, the final feature set is listed in table 3 and they are necessary for us to analyze a game. Deleting any one of them, the model's accuracy goes down. To train, tune, and test our model, we split our data set based on an 8-2 principle. The training set takes 80% of the entire data set. The validation set and test set take 10% each. By using sklearn.model_selection.train_test_split[9], data is split in a stratified fashion using the target value as label. The target value distribution in three data sets remains balanced. Table 6 shows the total number of instances included in three sets. The target variable distribution is shown in table 2.

Finally, if the performance of the final model is improved(higher F1 score and testing accuracy) when comparing with our baseline model, we accept the alternate hypothesis. This suggests that there is a significant difference in performance when adding more in-game data into the feature set. That is, the outcome of a game is indeed affected by in-game data listed in table 3.

[9]https://scikit-learn.org/

Data Sets	Counts
Training set	95347
Validation set	11918
Test set	11918

Table 6: Counts of each data set

- **Performance Measurements**

In this section, we describe how we evaluate the performance of our models. The original data set is split into three subsets: a training set(80%), a validation set (10%), and a test set(10%). After the final model is trained using the training set and tuned using the validation set, some results are returned, for example, the training accuracy. To better understand the model, to see if the model overfits the data and to use the model to predict future instances, we need to know its predictive performance. The predictive performance is tested using the test set. There are several metrics to look at. Before discussing those metrics, we introduce some important values. When solving classification problems, there are four types of outcomes that could occur. First, true positive and true negative mean the predicted class of an instance actually belongs to that class. If the predicted value is true, the true class of that instance is also true and vice versa. False negative means the predicted instance does not belong to class when it does. False positive means the predicted instance belongs to a class but actually it does not. In a binary classification problem, we often show those information in a confusion matrix. Figure 2 is a confusion matrix.

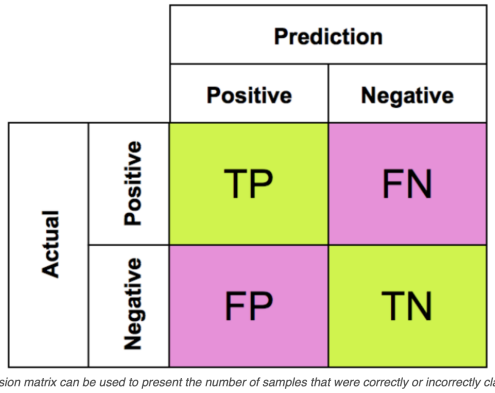

Figure 2: Confusion matrix (Dr. Shahin Rostami 2018)

From a confusion matrix, several common metrics used

to evaluate a model can be calculated. In our work, we choose to use classification accuracy, F1 score, ROC curve, and a confusion matrix to evaluate our final model. Accuracy is one of the most common measures to see how a model performs in predictions. In our case, we would like to know how many instances are correctly identified. The math calculation is shown by the following formula.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Sometimes, accuracy can be misleading. If the data set is imbalanced, accuracy is not adequate to conclude the predictive performance. In our case, the problem is solved since our initial data set is symmetric (50% - 50%). On top of that, F1 score is still an important value to look at. It is the weighted average of precision and recall. Both false positives and false negatives are taken into account. We would like to know how many true values are correctly identified and how many false values are correctly labelled (TP and TN). The calculation of F1 score follows.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

A high value for accuracy(symmetric data) and F1 score means the classification performance is good. AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve is one of the most important evaluation metrics to check how a model performs. It tells how much a model is able to distinguish between classes. A model with high AUC means the model performs well. A ROC curve calculates two parameters: TPR(true positive rate) and FPR(false positive rate). AUC is the area under a ROC curve and ranges in value from 0 to 1. For example, if the prediction is 100% correct then the AUC is 1.0. Figure 3 shows a ROC curve and AUC.

To generalize our model, we also need to study bias and variance. The analysis of those two measures is left for future studies. By far, to evaluate how good the final model performs, the metrics mentioned above are adequate.

- **Implementation & Performance**

  In this subsection, we briefly explain how our system works and how it is built. Also, we discuss what challenges we overcome to make the model and the system perform better. The system consists of two main parts: the first part is model building and the second part is instance predicting. After we build the model and analyze how the model performs, we automatically answer the research question listed in the very beginning of this paper. We say that game statistics such as gold earned, barons have taken, etc would affect the outcome of a game. The

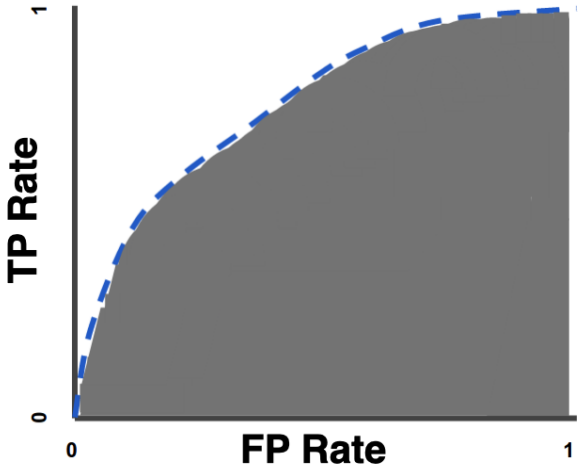

Figure 3: AUC-ROC (Google Developers Site 2018)

measurements are shown and explained in this subsection. The second part, which is the ultimate goal, aims to fetch an active game instance from one's local machine and feed the instance into our model to make a game prediction using the well-trained model.

First, for model building, we choose to use logistic regression and feed forward neural network together to make a prediction. When given a future instance, we feed it into an LR(logistic regression) model and feed it into an FNN(feed forward neural network) model as well. Then we obtain the final probabilities by averaging the predicted probabilities produced by two models. The math calculation is shown by the following formula.

$$Final\ Prediction = \frac{LR\ prediction + FNN\ prediction}{2}$$

The reason of taking the average of two models is that when using a single model (i.e. LR model) to make a prediction, the generated probabilities are sometimes not user friendly and the readability is quite poor. In terms of system performance, the predicted results are undoubtedly reliable and correct; however, the predicted probabilities lack of uncertainty and interpretability. For example, when a team takes the baron or takes a dragon or killed several opponents, the winning probability of that team might increase. At that moment, sometimes the predicted probabilities might be like this [0.05, 0.95]. It means the blue team has a winning probability of 95% and the red team has a winning probability of 5%. When players analyze a game that is still in progress, they are less likely to say that "the blue team is definitely winning" or "the red team has taken the baron and it is for sure they are winning". More often, we would like to say that "the blue team has taken the baron and the team has the advantage to win". Therefore, a 95% winning probability generated by a single model does not capture this human behaviour and thus decreases the readability. Hence, we overcome

this technical challenge(to some extent) by using a combined approach. After we average the probabilities generated by the LR model and the FNN model, we offset extreme values. After a game event happened, the new prediction probabilities become more readable and flatter than before. For example, taking the former case as an example, the predicted probabilities after a team takes the baron becomes to [0.21, 0.79] from [0.05, 0.95]. The new output becomes more user friendly than before. One could interpret the new output, [0.21, 0.79], as we add uncertainty into the prediction. Uncertainty refers to situations that no one could anticipate. For example, unexpected mistakes made by a player of a team and this kind of uncertainty would indeed shift the winning probability from one side to the other. Therefore, to some extent, our solution captures this kind of uncertainty and makes the predicted result more readable.

The model building is mainly about data preparation and model training/testing. The data preparation and how to configure the LR model are explained in the previous sections already. Table 7 is the detailed configuration of the 4-layer feed forward neural network used in our combined approach. The LR model and the FNN model take the same input data and their input dimensions are the same. Please visit the methodology section for how the data sets are collected, how the features are selected, and how the final data set is constructed. A more concrete procedure on how the data sets are collected is shown in figure 4.

Parameters	Values
Input dimension	96
Drop out rate	0.2
Input activation function	relu
Loss function	cross entropy
Batch size	56
Epochs	30
Output activation function	softmax

Table 7: feed forward neural network configuration

Before we explain how we implement the second main part of our system, we would like to share some interesting findings when exploring the input data set. Later on, we discuss the reasons that support us to accept the alternative hypothesis. We show that the outcome of a game is affected by some game statistics such as gold earned, barons have taken, etc. Additionally, the following figures provide a more concrete illustration on how the outcome of a game is related to the statistics.

Figure 5 shows the difference of total gold earned between two teams given the blue team has won. One could clearly see that the blue team's gold earned is more than the red team's at almost everywhere. This is consistent with what we presume. A team has more money can buy more items and a player with better items can kill his opponents easily. Finally, this leads to victory.

Figure 6 shows the number of dragons killed by each team given the blue team has won. This is also consistent with

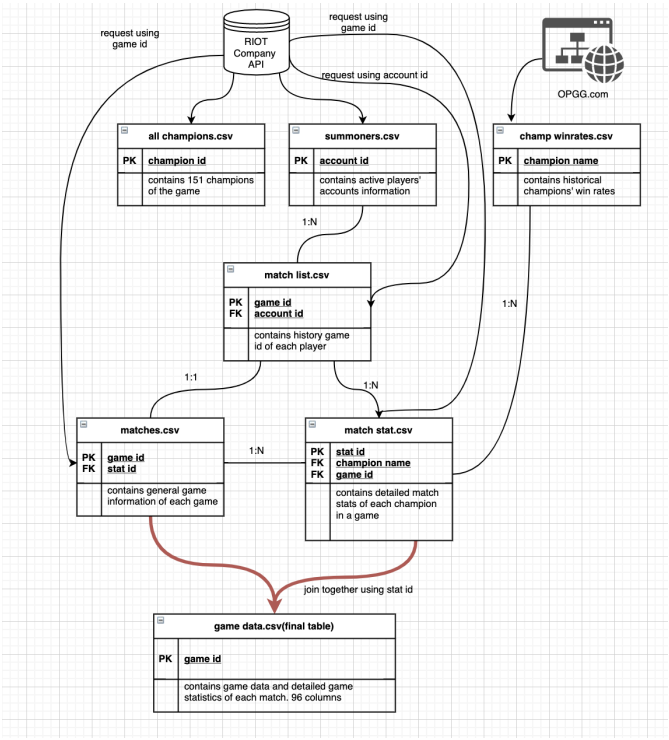

Figure 4: Data Preparation

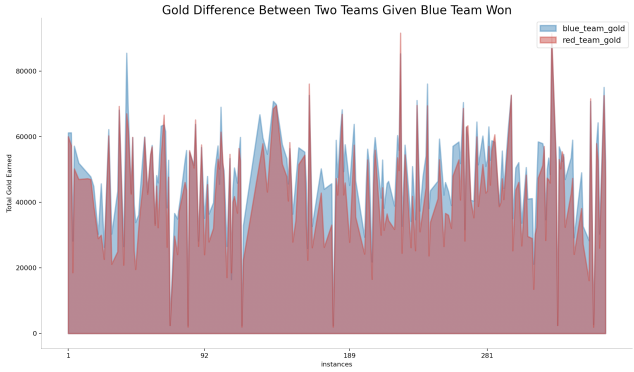

Figure 5: Gold difference between teams

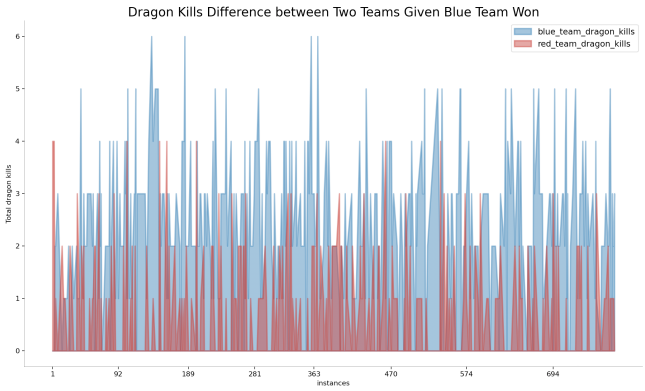

Figure 6: dragon kills of two teams

what we presume. The team(blue) has won took more dragons than the other team(red).

There is another challenge we have encountered. In table 3, we introduce the final feature set. Each game instance contains the listed information. In the methodology section, we explain how a rather large feature set is reduced to the features listed in table 3 and those are the final features. Taking one more feature away would decrease the accuracy. That is indeed true. However, the feature set is still large and it has more than 120 dimensions. In the feature set, each game instance contains the statistics of 10 individual champions. We could transform some information into a more compact representation. We reduce the size of the feature set without losing information. For example, we add up each champion's kills in a team and the summed result is considered as "total kills" of the team. We add up champion's deaths and assists as well. Doing so, we eliminate 27 features and bring the size of the feature set down to 96 from 123. Additionally, the models become more sensitive to the number of kills than before. When a player kills several his opponents, the gained advantage is well reflected on the predicted winning probabilities.

The second part of the system is to predict future instances. This part of the system grabs active game data every 5 minutes from one's local machine and transforms the game data to features. The features are then fed into our trained models for prediction. Riot company provides a very convenient game client API[10] for developers to test locally. We retrieve a full list of game data by simply requesting the endpoint: **https://127.0.0.1:2999/liveclientdata/allgamedata**. The data are given in JSON format and the size of the data is quite large. One can find a sample response here[11]. The data is transformed to features and the features are fed into the two trained models to obtain the final prediction. Figure 7 gives a concrete illustration of how the process works. One could find the complete project[12] implementation on GitHub.

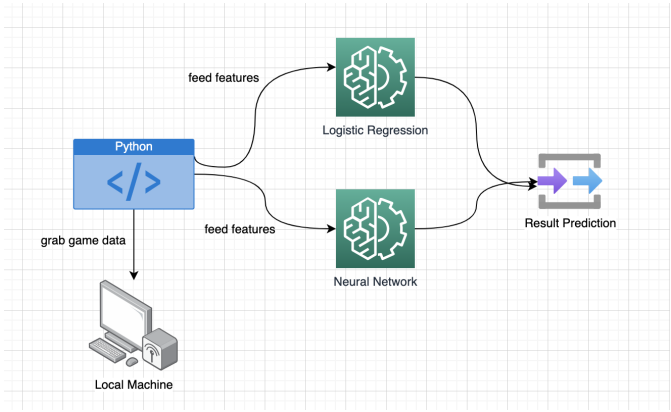

Figure 7: predicting future instances

In the experimental design section, we say that if the performance of the final model is improved (higher F1 score and testing accuracy) when comparing with our baseline model, we accept the alternative hypothesis. Our hypothesis is that the outcome of a game is affected by some in-game data such as 'kills', 'deaths', 'baron killed', 'gold earned', etc. In other words, we are able to predict the outcome of a game using some game information. We look at two main indicators: testing accuracy and F1 score. Table 8 shows the accuracy and F1 scores of models we build. The numbers related to our baseline model are not satisfying at all due to the fact that features used to train the model are less meaningful and not representative. It is quite clear that the accuracy and F1 score of the FNN model and the LR model are very satisfying and perform much better than the baseline model. Therefore, we can answer our research question such that our AI agent based on logistic regression achieves a higher classification accuracy than a baseline model based on feed forward neural network. There is a significant difference in performance when adding more in-game statistics into the feature set. That is, the outcome of a game is indeed affected by in-game data listed in table 3.

Model	**Accuracy**	**F1(wgtd)**	**F1(mac)**	**F1(mic)**
Baseline	0.516885	0.5102	0.510257	0.51688
FNN	**0.96723**	**0.96723**	0.967231	0.967235
LR	**0.948902**	**0.948902**	0.948902	0.948902
Combined	0.96215	0.9621	0.96213	0.95992

Table 8: Accuracy and F1 score comparison

So far, we have followed our experimental design to answer our research question by testing F1 scores and model accuracies. In performance measurements subsection, we introduced server metrics to check our model's predictive

[10]https://developer.riotgames.com/docs/lolgame-client-api

[11]https://static.developer.riotgames.com/docs/lol/liveclientdata_sample.json

[12]https://github.com/jimjimliu/LOL_Match_Prediction.git

performance. We choose to focus on a model's testing accuracy, F1 score, confusion matrix, and ROC curve. In table 8, the accuracy of our FNN and our LR model are both above 90%, which are quite satisfying. Given the fact that our data is perfectly balanced (as shown in table 2), the accuracy itself tells us that our models perform very well without checking F1 scores. As one can see the F1 scores do not deviate from the accuracies too much, since the data is balanced. Sometimes, accuracy can be misleading. If the data is imbalanced, accuracy is not adequate to conclude the predictive performance. Even though our data is perfectly balanced, to provide a rigorous assessment, we still check the confusion matrix and ROC of our models.

Table 9 and 10 show the confusion matrix of our two models. It is clear that the labelling of classes is very satisfying. One thing worth mentioning is that we do not care about the combined results. At the beginning of this subsection, we explained that the final prediction probabilities used in practical experiments are the average of the results of our FNN and LR model. The reason for doing this is to increase the readability of the results and add uncertainty to the results, instead of increasing the model's predictive performance. The predictive performance of either model is satisfying, therefore, there is no need to focus on the predictive performance of the combined results.

Finally, a ROC curve tells us how much a model is able to distinguish between classes. A model with high AUC means the model performs well. As figure 8 and 9 show, the performance is very satisfying. Both AUCs are above 95% meaning that our models are able to distinguish between classes very well.

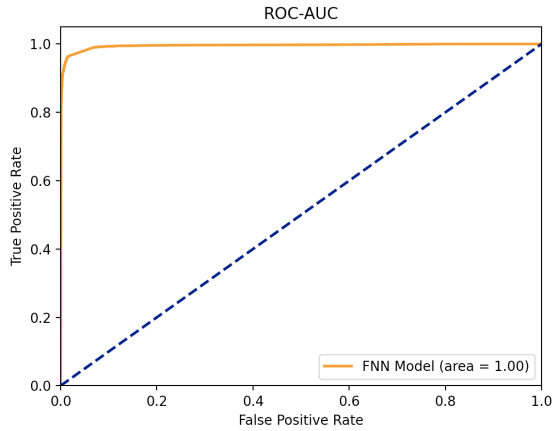

Figure 8: ROC-AUC of FNN model

	Red Team Win	**Blue Team Win**
Red Team Win	11283	641
Blue Team Win	577	11336

Table 9: Confusion matrix of LR model

Finally, based on the metrics we just explained, we say

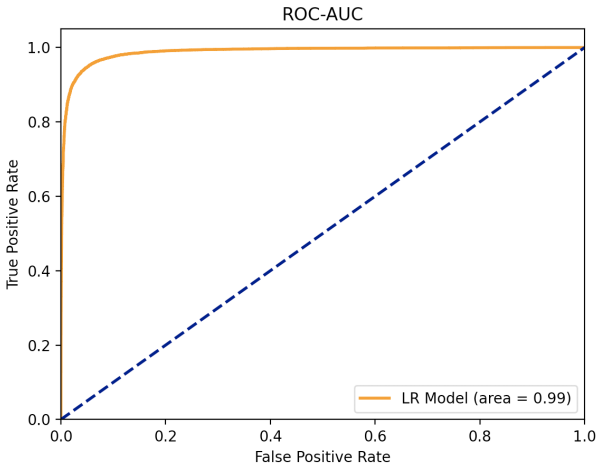

Figure 9: ROC-AUC of FNN model

	Red Team Win	**Blue Team Win**
Red Team Win	11670	254
Blue Team Win	527	11386

Table 10: Confusion matrix of FNN model

that the predictive performance of the models is reliable and satisfactory.

To get the best predictive models and to improve performance, some steps are required to make sure our data is internally consistent. That is, each data type has the same content and format. Data standardization is the process of rescaling the attributes so that they have mean value as 0 and variances as 1. This step is important when the data has different units. Some features are measured at different scales do not contribute equally to the analysis. When we collect our data sets, the units are mostly the same except one which is "gold earned". The range of "gold earned" outweighs other variables' ranges. Among 96 features, this single feature that has a wider range may have little negative effect to the overall performance. Indeed, table 11 compares the accuracies of models before and after data standardization and the accuracy after standardization increases a little but not too much.

Model	**normal**	**standardized**
FNN	0.94230	0.96723
LR	0.92385	0.948902

Table 11: Model accuracy before and after standardization

In table 5, we introduced the configuration of our LR model. The configuration parameters are mostly default values and we find that the initial parameters are efficient enough to train a satisfactory model. Therefore, we do not tune our LR model any further. On the other hand, our feed forward neural network model takes more time to figure out what the best hyperparameters are. Table 12 lists several hyperparameter configurations of the feed forward

neural network. The second column gives us the best accuracy.

	FNN	FNN	FNN	FNN
dropout	0.1	0.2	0.3	0.4
act.func.	relu	relu	relu	relu
batch size	28	56	128	256
epochs	100	80	60	50
hidden layers	1	2	2	2
accuracy	0.90385	**0.948902**	0.9415	0.9325

Table 12: FNN accuracy given different hyperparameters

- **Inspiration**

  In this section, we have conducted our experimental design, we have answered our initial research question and we have shown various metrics to demonstrate the predictive performance of the models. Unlike projects spending a significant amount of time to tune hyperparameters and transform data sets to reach the best accuracy, we spend less time to do so. A major factor is that our data quality is high and the hypothesis is feasible. The data is The accuracy of models are satisfactory at our first attempts. In terms of predictive model building, we always believe that high-quality data sets would make one's life a lot easier. We did spend a significant amount of time to collect our data sets, to transform the data sets to the format that we desire, and to make the data sets noise-free. We believe this step lays a solid foundation for model building. This might be case-oriented; however, making sure the data is clean and noise-free should always be the priority before going into the next step. Additionally, many projects do not end up with a satisfactory result. Occasionally, researchers may find the predictive performance is disappointing. A baseline model gives us an idea of where we are. By comparing the baseline model with a more complex model, it is clear that whether we have made any progress. Sometimes, a baseline model may suggest whether a topic is feasible or whether it is worth to go deeper.

Discussion

- **Result Analysis**

  In the result section, we introduced the experimental design to describe the experiments conducted to answer the research question, explained the models selected and their parameters, and discussed several performance measurements to evaluate the models and the actual performance of the models. Combining all these, the research question can be answered. In the introduction section, we discussed the research question and the hypothesis which is that the outcome of a game is affected by some in-game events such as 'kills', 'deaths', 'baron killed', 'gold earned', etc. As stated in experimental design subsection, if the performance of the final model is improved (higher F1 score and testing accuracy) when comparing to the baseline model, then the AI agent based on logistic regression achieves a higher classification accuracy than a baseline model. Table 8 shows that the F1 score of our baseline model is 0.51 and the testing accuracy is 52%. The F1 score of final model based on logistic regression is 0.95 and testing accuracy is 95%. The input data is perfectly balanced, thus, it is not surprising to see the F1 score and testing accuracy do not deviate from each other too much. The confusion matrixes shown in table 9 and table 10 are both nicely labelled. The baseline model takes naive and simple features and the baseline model does not suggest a feasible solution. The baseline model performs no better than a blind guess since the problem we focus on is a binary classification problem. On the other hand, the final model based on logistic regression takes more sophisticated game data as features. Even though our baseline model performs poorly, the predictive performance of our final model performs surprisingly well. It is safe to say that the outcome of a game can be predicted by some in-game events. By comparing the F1 scores and testing accuracies, we conclude that our AI agent based on logistic regression achieves a higher classification accuracy than a baseline model based on feed forward neural network.

  Before conducting any experiments and building models, the hypothesis states the outcome of a game is highly related to some game events. Figure 5 and 6 provide a very useful insight such that a team that wins has more gold earned and dragons taken. Our final feature selection was based on this fundamental idea, which the outcome of a game can be predicted if we gather enough data in terms of gold earned, dragons taken, barons taken, and opponents killed. Just like people would use these statistics to analyze a game while watching, we use scientific models that take these data to explain a game systematically. The initial result from data exploration looks promising and predictive evaluation on the models is very satisfying. In terms of F1 score, testing accuracy, ROC, and confusion matrix, we conclude that the system can predict the outcome of a game. The actual result agrees with our initial hypothesis. Therefore, we accept the alternative hypothesis. This suggests that there is a significant difference in performance when adding more in-game data into the feature set. The outcome of a game is indeed affected by in-game data listed in table 3.

  Finally, the size of our feature set is a surprise. First, at the beginning of building models, the size of the feature set is larger than the current one and the original feature set contains more than 400 features. We thought it would be more complicated to analyze a game and to get the best testing accuracy; however, after we tried to shrink the size of the feature set, the testing accuracy did not decrease as explained in the methodology section. The final size of the feature set was cut in half and contains about 200 features. The predictive performance was not affected by the shrunk size of the feature set. This unexpected result decreased the model training time and maintained the predictive performance.

- **Implications**

  In related work section, we talked about the commercial

system called **Colonel KI** [13]. This system is not open source; however, the behaviour of this system is the ultimate goal of our project. In terms of predictive performance and continuously predicting a game, we have shown that our system is doing a very good job and have achieved the ultimate goal. Previously, in the related work section, many similar projects and works were conducted to predict different kind of sports. Those projects include predicting NBA matches, professional MOBA games, using ELO to predict chess, etc. These existing works suggest that it is possible to analyze a game using some sophisticated features. Indeed, our results have shown that it is possible to predict and analyze a game well. When searching for similar existing works, there were few works done on analyzing this particular game, which is League of Legends. Comparing to previous existing works, our system describes a game more systematically and it is able to generalize on future instances. Many similar works stopped at the point of model building; whereas, our system is able to do practical experiments. That is, our system can be used as a tool to analyze any future game instances. On top of that, to some extents, there is a huge potential to put it into commercial use just like the system Colonel KI. On a website like OPGG.com, one could search any players by typing in their user names and check their active games. Although the website provides a panel shows active game information, the information is limited and not very dynamic. Imagine that if we added the functionality of our system to the existing website modules, the entertainment of the game would be increased and website searching volume would go up. Moreover, in professional World Championships, if this predicting functionality was integrated, people would find the game more interesting.

- **Limitations**

  As explained in the implementation section, the project contains two major components. The first one is model training and testing. The second major part is future instances predicting. The second part is also the ultimate goal of this project. We would like to use the system to continuously predict active games. One of the limitations of the project is that the system can only predict the active game running on one's local machine. The system grabs game data using a live client API from the address: **https://127.0.0.1:2999/liveclientdata/allgamedata**.
  RIOT company does not provide any other available APIs to retrieve other's active game data, instead, the existing APIs only provide methods to retrieve data of past games. Therefore, the general application of the system is limited and restricted to one's local machine.

  Meanwhile, the data we collected is based on the players located within the North America region. If one tends to use the system to analyze games outside of the NA region (i.e. LPL region), the predictive performance might be affected and might not be accurate.

[13] Colonel KI:https://www.mindshareworld.com/china/work/kfc-colonel-ki

Conclusion

The purpose of this project is to build a system that is able to predict the outcome of a MOBA game. The game is called League of Legends, which is one of the most popular online games in the world. The goal is to take advantage of some game events to predict each team's winning probability while a game is in progress. We would like to generate a winning probability for each team, and the project adds value from start to finish as players could track predictions before, during, and after a match. As stated in the introduction, our motivation of building such a system is to see if such a predictive model can be built and if it is possible to predict a game using game events. The project is also inspired by the idea of the system called Colonel KI. As a programmer and also a player, making contributions to the game community motivates us to build something different. With this project, there is a huge potential for community use and commercial use. The particular research question we try to answer in this paper is that "will an AI agent based on logistic regression achieves a higher classification accuracy than a baseline model based on feed forward neural network". Our hypothesis is that the outcome of a game is affected by some game events and the outcome can be predicted using these game events. To build such a system, we collected data using APIs provided by RIOT company and built the final model based on logistic regression and feed forward neural network. To predict future instances, the system grabs game data from one's local machine and transforms the data to a feature set to feed the models to make a prediction. Our results show that the final model has an F1 score of 0.948 and a testing accuracy of 95%. In the confusion matrix, 95% of the data is correctly labelled. In terms of practical experiments, we tested the system on our local machine, every game event, such as killing opponents, taking dragons or barons, are well reflected in the predicted winning probabilities. Our system predicts a game continuously while a game is in progress every 60 seconds. Overall, the predictive performance of our models is satisfactory and the system performance is also reliable and satisfying.

Throughout this short paper, we have shown that it is possible to take advantage of some game statistics to analyze a particular MOBA game. This entire process might be useful and inspiring when one would like to build a similar system to predict other online video games. The central and fundamental idea is very simple, which is the outcome of an online video game is related to some in-game statistics to some extents. As previously stated, there are limitations and experiments we could not do. In the future, we would like to try to gather data from different regions (currently North America). We would like to know if the merged input data based on all regions would change the predictive performance. Meanwhile, we could not build a public system that could be used for large communities or the Internet as a whole since we did not have a method to retrieve other players' active game data. We can only access one's game data from his/her local machine. Hopefully, in the future, there would be a way for us to do so. Finally, our final prediction takes the average of the predicted results of a logistic regression model and a neural network model. For the reasons to

do so, please visit implementation and performance subsection on page 7. We used the average to offset extreme values given by the predicted results. Sometimes, the predicted result generated by a single model gives a very high winning probability, i.e. [0.99, 0.01]. This kind of probability lacks uncertainty and interpretability. Therefore, in the future, we would like to do some minor model modifications to accommodate and to eliminate this issue.

To sum up, we have successfully answered the research question and accepted the alternative hypothesis based on our results. We reached a 95% testing accuracy and an F1 score of 0.948. Overall, the predictive performance and system performance is acceptable.

References

Dr. Shahin Rostami. 2018. Experimental design. [Online; accessed October 17, 2020].

Google Developers Site. 2018. Classification: Roc curve and auc. [Online; accessed October 17, 2020].

Hanke, L., and Chaimowicz, L. 2017. A recommender system for hero line-ups in moba games. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Hodge, V. J.; Devlin, S. M.; Sephton, N. J.; Block, F. O.; Cowling, P. I.; and Drachen, A. 2019. Win prediction in multi-player esports: Live professional match prediction. *IEEE Transactions on Games*.

Kang, D.-K., and Kim, M.-J. 2015. Poisson model and bradley terry model for predicting multiplayer online battle games. In *2015 Seventh International Conference on Ubiquitous and Future Networks*, 882–887. IEEE.

Kinkade, N.; Jolla, L.; and Lim, K. 2015. Dota 2 win prediction. *Univ Calif* 1:1–13.

Pobiedina, N.; Neidhardt, J.; Calatrava Moreno, M. d. C.; and Werthner, H. 2013. Ranking factors of team success. In *Proceedings of the 22nd International Conference on World Wide Web*, 1185–1194.

Silva, A. L. C.; Pappa, G. L.; and Chaimowicz, L. 2018. Continuous outcome prediction of league of legends competitive matches using recurrent neural networks. In *SBC-Proceedings of SBCGames*, 2179–2259.

ujjwalkarn. 2016. A quick introduction to neural networks. [Online; accessed October 17, 2020].

Wang, K., and Shang, W. 2017. Outcome prediction of dota2 based on naïve bayes classifier. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, 591–593. IEEE.

Yan, S.; Meng, S.; Liu, Q.; and Li, J. 2019. Design and implementation of nba playoff prediction method based on elo algorithm and graph database. *Journal of Computer and Communications* 7(11):54–64.