SENG 550: League of Legends Game Predictor

Justin Flores
University of Calgary
justin.flores@ucalagry.ca

Jen-Wei Huang
University of Calgary
jenwei.huang@ucalgary.ca

Touseef Hossain
University of Calgary
touseef.hossain@ucalgary.ca

***Abstract*—League of Legends is a very unpredictable game affected by many various factors. Encouraged by this challenge, we created multiple machine learning models that can predict the outcome of a game with high accuracy. Each model uses a different binary classification algorithm, since the goal of the model is to predict which team will win the match. Each model was given the same features that describe the state of the game during a certain time of the match. The models were trained on a large data set containing over 7000 records of matches. To deal with a large data set, we used Spark, which is a big data platform to help us clean, pre-process, and train the models. The resulting models each had at least 98% accuracy, with the best one having over 99% accuracy.**

I. Introduction

A. Game Overview

A game developed by Riot Games, *League of Legends*(LoL) is one of the most popular multiplayer online arena (MOBA) game in the world. A standard game of *League of Legends* consist of two teams, each with five players, compete to destroy each other's Nexus. However, before a team can destroy the other team's Nexus, they must first destroy one of the three enemy inhibitors, followed by two towers guarding the Nexus. Before a team can reach one of the inhibitors, they must destroy the three towers guarding each one of them. Teams must achieve this all while defending their own structures from the enemy team. Players start out relatively weak and they get stronger throughout the game. This can be achieved by gaining experience to level up and buying items using gold. Experience points and gold can be obtained from killing enemy minions, champions, and structures. There are also neutral objectives around the map that can give teams buffs, gold, and experience.

B. Problem

Due to the complexity of the game, we want to know if the outcome of the game can be predicted simply by using the current state of the game. Such a system would be helpful in providing insights as to which factors are influential in helping a team win a game. This is will be valuable to both casual players and professional players since they can use to make in-game decisions, or utilize it to help them plan strategies before games start. It will also be useful for broadcasting matches as it can provide real-time predictions as a game progresses, making it more interactive for viewers.

C. Related Work

Given that *League of Legends* is the most popular multi-player online arena game, there are a few applications that exist to collect game data and analyze the game. Websites like *OP.GG* and *League of Graph* are collecting the stats in-game and the user rankings using *Riot Games* API to analyze the game in a large scope. However, these systems do not provide game predictions; only analysis on which champion has the highest win-rate, or which combination of champions has the highest win-rate.

There are systems out there the use machine learning to predict outcome of games, but they are not widely available for players like the ones mentioned above. In [2], describes a machine learning model that uses deep neural nets to predict the outcome of the game based on a a player's previous performance. The model described will use individual data of players such as kills, assists, deaths, etc.. taken from previous matches, and use those as features for the model. This means that the model's prediction will be made before the game starts and not during the game. Also, the prediction does not change as the game progresses. Another prediction model described in [3], uses machine learning algorithms like gradient boosted trees and gradient boosted trees with logistic regression to predict the winner of a match. This model uses more features than the one described in [2]. Furthermore, the features for the model in [3] require data at the end of the game , which means the prediction can only be done at the end of the game.

D. Our Solution

As mentioned earlier, the existing applications were merely focused on analyzing the data collected prior to the start or end of a game or predicting a game's outcome right before a game starts. The goal behind this project is to take this analysis further by attempting to predict the outcome of an ongoing game. The aim is to develop multiple models that can predict the outcome of a LoL match at a given state during the game. For the scope of this project, we will attempt to predict the outcome from analyzing the state of the game when it is halfway done. Thus, the main findings of this project will focus on if it is possible to accurately predict the winner of a match near the midway point of a match in LoL.

The large amount of data used for models will be retrieved from a Kaggle dataset [1]. The models created in this project utilize the concept of machine learning in order to predict matches at a given state. These models will be driven by user

supervised learning algorithms for binary classification such as logistic regression, decision trees, and random forest.

II. Data Collection

The model described in the introduction requires match data which we obtained from Kaggle [1]. The dataset contains over 7000 professional competitive match records between 2015-2018. The reason why we decided to use professional matches is because we do not want the player skill gap to be an unaccounted factor. This is because skill gaps in players can have a huge influence on who will win the game. However, when using professional matches, we can minimize the influence of skill gaps since all the players relatively have the same skill level.

III. Methodology

A. Development Setup

In the project, we are using *pySpark* in *Google Colab* environment to preprocess the data and train the models. Since the data used to train the model is fairly small (137.65MB), it is stored in a shared the Google Drive folder for simplicity. To set up environment, we first need to install *pySpark* in the *Google Colab* notebook, and mount *Google Drive* to the *Google Colab* notebook so we have access to the data set.

B. Feature Selection

The Kaggle data used for this project contains 57 columns worth of data variables that are used to identify the key aspects of a LoL match. For the development of the project models, some of these columns need to be filtered out in order to produce effective outcome results. The project will make use of the team stats: *bResult, rResult, golddiff, bKills, bTowers, bInhibs, bDragons, bBarons, bHeralds, goldred, rKills, rTowers, rInhibs, rDragons, rBarons* and *rHeralds*. These columns will stored in an initial stage resilient distributed dataset (RDD).The usage of the player-related variables from the dataset will not be used in this project because of the assumption made earlier that all players' performance are relatively at the same level of skill.

The letters, *r* and *b*, found at the beginning of some of those variables refer to the two names used to identify the teams playing in the match. The *result* variables contain a 0 or 1 value, which indicates whether the team won the game or not (1 = win, 0 = loss). The *golddiff* stat containts a list of the difference in gold between the two teams at every minute of the game (calculated by the subtracting the gold value of red team from blue team). *Kills* contains the records of kills recorded in the match; each element consisted of the minute stamp, who was killed, who performed the kill, along with the players who have assisted with the kill. Finally, the *Towers, Inhibs, Dragons, Barons,* and *Heralds* parameters contain a list of minute time stamps of every time those things were destroyed/killed.

This project will focus on trying to predict a LoL match outcome by analyzing the data near the middle of a match. In order to retrieve that specific data, the mid-game timestamp will need to be determined. The next step consists of adding new columns to the RDD that will help with creating the desired models for mid-game analysis.

1) The first new column is called *winner*, which will contain a string indicating which team won the game (Red or Blue). This column is calculated by using the *rResult* and *bResult* columns. This column will be used as the label for each record.
2) The second new column is called *mid_golddiff*,which is the difference in gold between the two teams (blue - red) at around the middle of the match. The dataset has a *golddiff* column that stores an array of gold differences of the two teams every five minutes of the game. To get this new column, we simple take the $\lfloor gameDuration/2 \rfloor$th element from
3) The next two new columns, *bKillCount* and *rKillCount*, will be integer values that represent the total number of kills each respective team yielded by the end of the middle timestamp.
4) *first_blood* is a column which will store string value which will indicate which team obtained the first kill in the game (Red or Blue).
5) The next 10 new columns being added to the initial RDD are named *bTowerCount, rTowerCount, bInhibCount, rInhibCount, bDragonCount, rDragonCount, bHeraldCount, rHeraldCount, bBaronCount* and *rBaronCount*. As their names suggest, these columns contain integer values of the number of towers,inhibitors, dragons, heralds, and barons destroyed/killed by each team respectively by the certain middle-point of the game.
6) The last set of columns added are: *first_tower, first_inhib, first_dragon, first_herald,* and *first_baron*. These columns will contain a string value indicating which team destroyed/killed a respective aspect first within the game. If both the team did not take the objective, it will have a 'None' instead of 'Red' or 'Blue'.

The finalized dataframe to be used for outcome prediction will consist of the columns: *winner, mid_golddiff, bKillCount, rKillCount, first_blood, bTowerCount, rTowerCount, bInhibCount, rInhibCount, bDragonCount, rDragonCount, bHeraldCount, rHeraldCount, bBaronCount, rBaronCount, first_tower, first_inhib, first_dragon, first_herald,* and *first_baron*.

Before we can use these new columns to build a models, we first need to hot-encode the categorical columns as machine learning models work better with numbers. Also, we are going to normalize the numerical columns as they will be used as features. This is a necessary step since the ranges of the numerical columns are very different. We do not want values in larger ranges to influence the prediction more than values in smaller ranges, since that might not be the case.

C. Models

Using the feature vector and the labels from the matches record, we then create three different supervised machine learning algorithms that solve binary classification problems.

We will be tuning multiple parameters for each model, and use 3-fold cross validation to select the best performing model and their paramaters. The cross-validation will also help assess how well our models generalize. For performance metrics, we will be using the area under the ROC curve to assess the performance each of our model and compare which one is the best performer.

1) Logistic Regression: The first model we looked at is logistic regression as it is the most simple and common binary classification algorithm. For this model, we are tuning the number of epochs and the depth of aggregation. By increasing the number of epochs and depth of aggregation, the model can learn more features, which results in a reduction in underfitting. However, this can also lead to overfitting to the train data which is why we also decided to tune the L2 regularizer λ. By changing and tuning our L2 regularizer, we can control and constrain our model's complexity, which can help with overfitting.

Table 1 below shows the parameters for the best performing logistic regression model.

TABLE I
HYPERPARAMETERS FOR THE BEST PERFORMING LOGISTIC REGRESSION MODEL.

Parameter	Result
Max Iterations	50
L2 Regularizer	0.0
Aggregation Depth	10

2) Decision Trees: The second model we looked at is the decision tree model. The decision tree algorithm works really well with the data we are using as it can both handle categorical and numerical features, which is what we have in our feature vector. Another advantage of decision trees is that they very fast at classifying unknown records which makes it suitable for predicting real-time data. For this model, we are tuning the max depth of the tree and the max number of bins in the tree. By playing around with the max depth of the tree we can control how much features the mode can learn, the more deep a tree is, the better it is at learning more features. However, having a tree that is too deep can lead to overfitting. Another parameter we are tuning is the maximum number of bins. The number of bins tells us how well we are representing our numerical features, the more bins, the better the representation. However, there is a trade-off as the number of bins can lead to slower training time.

Table 2 below shows the parameters for the best performing logistic regression model.

TABLE II
HYPERPARAMETERS FOR THE BEST PERFORMING DECISION TREE MODEL.

Parameter	Result
Max Depth	21
Max Bins	12

3) Random Forest Model: Simple decision trees often overfit and have poor prediction performance. However, tree performance can be improved using random forest which is why we chose to investigate this model. In this particular model, we used the maxDepth, maxBins, minInstancesPerNode, and bootstrap as the hyperparameters to tune our model. Having multiple parameters can help us to train our model to become more accurate in predicting the results. We are tuning the maximum depth of each decision tree to prevent overfitting in each tree. We are also tuning the maximum number of bins. The more bins we have, the better classification. However, having too many bins will also lead us to overfitting. Another parameter used in this model is the minimum number of instance per node. In the construction of a decision tree, there needs to be a strict minimum of child nodes. If a node cannot split more than this number, the tree is forced to stop growing at that node. This parameter also helps us to control the size of each decision tree, to prevent both bias and variance problem. The last parameter we plan on tuning is the bootstrap. Bootstrapping our data might help us to get a better trained model, since it randomizes the training dataset more. Random forest algorithm is very precise with regards to classification tasks, and we do not really need to worry about the overfitting problem since the random forest classifier will automatically detect if there is enough trees in the model, and will prevent us from overfitting.

Table 3 below shows the parameters for the best performing logistic regression model.

TABLE III
HYPERPARAMETERS FOR THE BEST PERFORMING RANDOM FOREST MODEL.

Parameter	Result
Max Depth	18
Max Bins	18
Min Instances Per Node	3
Bootstrap	False

IV. RESULTS

Table 4 below shows area under the ROC curve after during training and testing of the three model described using the dataset of matches that we sourced from Kaggle. All three models were tuned using a 3-fold cross validation.

TABLE IV
TRAIN AND TEST AREA UNDER ROC OF THE MODELS

Model	Train	Test
Logistic Regression	0.98922	0.98393
Decision Tree	0.99893	0.99808
Random Forest	0.99689	0.99570

As we can see, there are no significant differences between each models' performance. All of them have high AUC(area under the ROC curve); over 0.98. This tells us that 98% of the time, the models were able to correctly predict which team will win during training and testing. Furthermore, we can see from

table 4 that the decision tree model performed better than the random forest model. Although the difference in performance was marginal, it was an unexpected result as random forest is known to be more suitable for large datasets [4].

V. Conclusions and future work

From our results, we can see that it is possible to predict the outcome of a *League of Legends* match during the halfway point of the game. This also tells us that the feature we have chosen have a big influence on the outcome of a match and that teams should focus on objectives related to those factors. For future work, we would like to add more feature by looking into each team's champion composition. This includes which champion chosen for each role, the items they have, masteries, summoner spells, etc... Such a system can help players what items to buy for there champion, and what masteries they can take so that they can increase their chance of winning. Another improvement we want to add is to add data even earlier than the halfway point of the game. This can help our models become better real-time predictors as they are exposed to earlier game state data.

References

[1] C. Ephron, *League of Legends: Competitive Matches, 2015 to 2018*, Jan. 29, 2018. Accessed on: Nov. 10, 2020.[Online]. Available:https://www.kaggle.com/chuckephron/leagueoflegends

[2] K.T. Hall, *Deep Learning for League for League of Legends Match Prediction*, Dec. 10, 2017. Accessed on: Dec. 15, 2020.[Online]. Available:https://github.com/minihat/LoL-Match-Prediction/blob/master/FINAL%20REPORT.pdf

[3] L. Lin, *League of Legends Match Outcome Prediction*, Sep. 09, 2017. Accessed on: Dec. 15, 2020.[Online]. Available:http://cs229.stanford.edu/proj2016/report/Lin-LeagueOfLegendsMatchOutcomePrediction-report.pdf

[4] A. Sharam, *Decision Tree vs. Random Forest – Which Algorithm Should you Use?*, May. 12, 2020. Accessed on: Dec. 16, 2020.[Online]. Available:https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/