League of Legends Match Outcome Prediction Using Machine Learning

Jacob M. Frank, *Member, IEEE*, Tigran V. Avetisov, *Member*, *IEEE*

***Abstract*—League of Legends is a Multiplayer Online Battle Arena (MOBA) game with two teams of five players. The game utilizes ELO based matchmaking to make games as fair as possible. A player's performance in their given role may vary depending on their selected champion, experience in the role, and recent performance in previous games. In this paper, we attempt to predict the outcome of a match during the character select lobby based on the team composition and statistics of our team and the team composition of the enemy team. Statistics from teammates are retrieved from the Riot Games API. We used machine learning and deep learning methods to make multiple models to predict the outcome. With deep learning and various data splits our model is able to get ~55% accuracy. With machine learning, ensembling, and various data splits, our models are able to achieve a score of 56.73% accuracy. These models show that predicting match outcomes can be difficult, but these results are fairly good for a simple dataset. Utilizing the predictions, games can be effectively left during character selection for a small penalty in visible rank. Overtime, the visible rank will even out with the hidden matchmaking rating.**

***Index Terms*--Neural networks, Machine learning, Games, Web sites, Data science, Data processing, Data mining, Python**

I. Introduction

League of Legends is a Multiplayer Online Battle Arena (MOBA) game with two teams of five players with the goal of destroying the enemy's nexus. This game is one of the most popular games in the world with most players playing in a ranked scenario. This game utilizes an ELO Ranked system that puts players into ranks (ex. iron, bronze, silver, gold etc.) to make sure that people are matched up with other players of about the same skill level. To increase your ELO you need to win games to gain Ladder Points (LP) and once you get to 100 you have the chance to move up a rank. This ranking system doesn't always work. With there being new players everyday, the system doesn't accurately judge the newer player's skill levels and might get put into a game with a very experienced player. This causes a large skill gap in some games, but this can be prevented by looking into each player's match history and see if they are on the same skill level as you or not. Knowing this information before the game even starts might save you time and LP, because you have the ability to leave the game in the character selection screen. Leaving does lose you LP but it is a lot less than if you were to play the full game, and saves a lot of time.

Using machine learning and deep learning methods, we made a web application that will output the likelihood of the outcome of the game before it starts. Our dataset was limited due to time constraints and limitations with the creators of the game, Riot Games, having a limit on how many API calls you are allowed to make per minute. Even with having up to four different API keys running at the same time we were only able to get around 15,000 matches in our dataset after running for a couple weeks. Because of this limitation, our models could only peak at a maximum accuracy score of ~57% with the average at ~55%.

II. Problem Statement

The problem that we are trying to help solve is knowing if the player has a good chance to win the current match. This information can give the player confidence and reassurance that they aren't fighting a losing battle. Once all champions have been selected, there is still a small time frame where one can leave and cancel the match, for a small penalty, known as dodging.

Current third party websites exist where a user can copy and paste their lobbies join text, which list all names of their team, and get access to their teams match histories, these sites display stats like total games played, win rates on specific champions, scores of recent games played during the current ranked season. These websites do not predict the outcome of the match however. They can only offer you the statistics of the players on your team then you would have to make the judgment call to dodge or to play out the game.

III. Related Works

There are many websites that utilize the Riot API to display player match histories, rank distributions, champion statistics, and more. None of these have an extension to predict the outcome of matches during champion select. There are multiple reports that claim to predict match outcomes but we believe that some of them are flawed or are not exactly doing it the way we believe it should be done.

Lin [1] uses statistics from both teams on unique games to predict the winner of the match, achieving 95% accuracy. Our problem statement is to predict the winner before knowing the enemy team's players the statistics of that unique match. His testing size is also twice the size of training.

Do, Tiffany, et al. [2] aims to predict the winner of a match with statistics from both teams. Using only four features per player, reduced to two later on and then adding extra features such as variance, standard deviation, skewness, etc, based on the chosen two. They achieve an accuracy of 75.1%. Once again, they used both teams and their sample size was 5,000 unique matches.

Lastly, Hall [3], was the closest project compared to ours. He used much more data than the other projects but still

utilized data from both teams. This time the data that was gathered was averaged over previous games in a player's match history instead of choosing only a couple stats like overall win rate of a player. This project accuracies seemed to vary between 59.8% and 84.4% based on features selected. None of these seem to only keep track of one team and predict their chance of winning like being in an actual champion select lobby, where there is a small time window to leave before the match begins.

IV. Solution

Our approach to this problem was to make a simple web application with a deep learning model as its backend. We found that, with our dataset, having a simple shallow model produced better results than when using a bigger more complex model. We also tested using machine learning algorithms and found them to be just as accurate if not more. We considered using an ensemble of the best machine learning algorithms, but they were very slow and didn't produce results that were measurably better than the deep learning model, which was minutes faster.

V. Dataset

A. Data Collection

Riot Games has a public API available to access game data given a user has an API key. This key can be accessed in a user's developer portal account. We used Cassiopeia to access the data. Cassiopeia is a framework/wrapper for the Riot Games API written in python.

To reiterate, during champion select, the information available to a player is their teams usernames, champions selected, and enemy teams champions selected. Using this information we want to create a dataset that can simulate being in champion select and know if the game was won or lost.

To collect data of champion select lobbies, the first thing that was done is to find a random player's account and get their account ID. Using this ID we can now call their match history and save up to one hundred games, we chose ten. For each game in this match history we will select one team at random to simulate being in a champion select lobby and store if the game was a win or loss and the side of the map they were on. Once a team has been selected we will get the stats of each of the five players. The ten champions selected will also be stored. To get stats for each player we will call the match history for each player and access their last twenty games played before this current selected match was played. In these twenty games, some general information about the player will be collected, such as the count of games on their currently selected champion, the count of games on their current role, and the count of their wins and loses. The count of their wins in the last two games and the last five games were also tracked. The other data collected from a player were their kills, deaths, damage dealt to enemy champions, creep score, vision score, first bloods, etc. Some stats were kept as a sum, others were averaged across the twenty games.

Once all ten games from the initial player account match history has been added to the dataset, a random player from their recent game would be selected and the same process would begin.

B. Description

In total, the original dataset has 18172 samples across four different ranks with each row having the stats of one hundred games called on with the API. Each sample has 149 columns.

VI. Preprocessing

To process the data each row with a duplicate match ID was dropped and then the column was removed. Rows that incorrectly had a "0" as a champion name were removed and rows that didn't have a total of 100 matches summed were removed. The "Win" and "Side" columns were turned into binary values. "Rank" and champion categorical columns were one-hot-encoded. After preprocessing the dataset, the shape of the dataset is 15,000 x 1,382, excluding the target column.

VII. Models

A. Neural Network

To create a neural network, we used Keras, a Python deep learning API. Our final model consisted of only one Dense hidden layer with 128 neurons and a sigmoid activation. Our optimizer is Stochastic Gradient Descent with a learning rate of 0.01. Many other models were tried, some with many hidden layers, dropout, varying neuron amounts, different optimizers such as Adam and Adagrad. Too many neurons caused low accuracy improvements per epoch, too few caused underfitting, other optimizers stagnated validation accuracy, or caused validation loss to increase after one or two epochs. Dropouts also stagnated validation accuracy.

Fig. 1 shows a run with our final model, achieving 55.90% accuracy with 80% training - 20% testing. The data was scaled using MinMax. Although this run was close to 56%, the neural network seems to be inconsistent and vary its scores even on the same random state for training and testing. 15 runs with the same random state had a mean accuracy of 54.72% with a standard deviation of 0.33. Randomizing the states had an accuracy of 54.49% with a standard deviation of 0.77.

Fig. 2 shows a run with our final model with a different data split. 80% training - 10% validation - 10% testing. The testing accuracy was 54.79%. For this data split doing 15 runs on the same random state had a mean accuracy of 54.05% and standard deviation of 0.78. Randomizing the states had a mean accuracy of 53.70% and a standard deviation of 0.72.

Changing the data split to 90% training - 10% testing had a much smaller change for the neural network compared to the machine learning algorithms, as the network still struggled to consistently get ~56%.

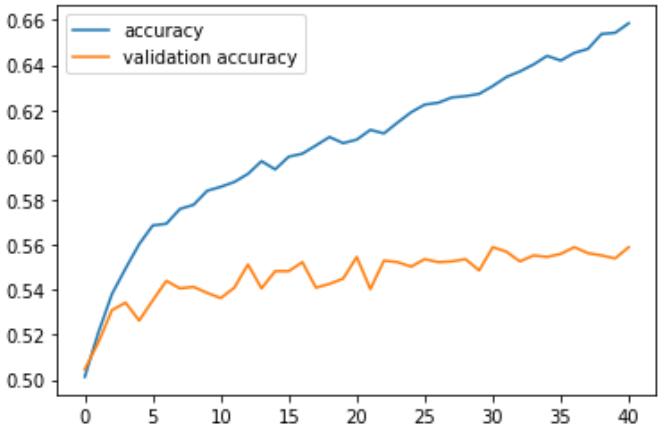

Fig. 1. Keras Neural Network 80/20 Data Split Accuracy vs Validation Accuracy.

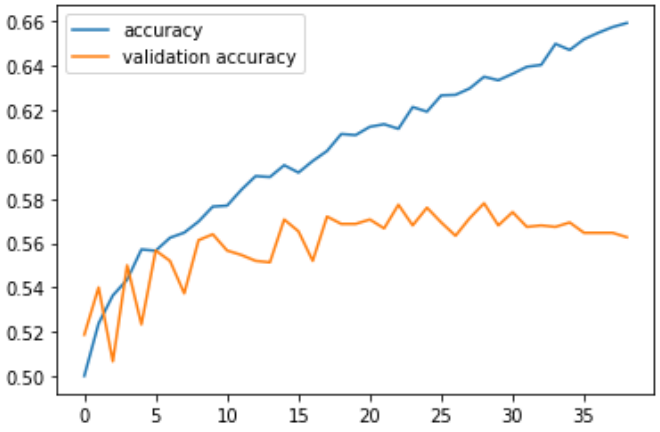

Fig. 2. Keras Neural Network 80/10/10 Data Split Accuracy vs Validation Accuracy.

B. Random Forest

A random forest is a meta estimator that uses averaging to increase predicted accuracy and control over-fitting by fitting a number of decision tree classifiers. Using the sklearn library, our model was fitted with these parameters, n_estimators=500, min_samples_leaf=100,min_samples_split=25,max_features= 500, bootstrap=False.

C. Extra Trees

Similar to random forest, extra trees fits many decision trees on subsamples of the dataset to control overfitting. Using the sklearn library, our model was fitted with these parameters, n_estimators=500,min_samples_leaf=100,min_samples_split= 25, max_features=500.

D. Ada Boost

Adaboost fits a classifier on the original dataset and then fits more copies on the same dataset where the weights of wrongly classified samples are adjusted. Using the sklearn library, our model was fitted with these parameters, n_estimators=500, learning_rate=0.01.

E. Gradient Boost

Gradient Boost optimizes arbitrary differentiable loss functions by building an additive model in a forward stage-wise approach. The negative gradient of the binomial or multinomial deviance loss function is used to fit n_classes_ regression trees in each stage. A specific example of binary classification is when just one regression tree is produced. Using the sklearn library, our model was fitted with these parameters,n_estimators=500,learning_rate=0.01,max_feature s=500,min_samples_split=25,min_samples_leaf=100.

F. Histogram Gradient Boost

Histogram based Gradient Boost. Using the sklearn library, our model was fitted with these parameters, loss='binary_crossentropy',learning_rate=0.01,max_iter=500

G. Logistic Regression

Logistic Regression (aka logit, MaxEnt) classifier. Data was scaled using MinMax scaler. Using the sklearn library, our model was fitted with these parameters, solver='sag', max_iter=1000, C=1

H. Linear Discriminant Analysis

A linear decision boundary classifier created by fitting class conditional densities to data and using Bayes' rule.Each class is given a Gaussian density by the model, which assumes that all classes have the same covariance matrix. Using the sklearn library, our model was fitted with all default parameters.

I. Multi-Layer Perceptron

This model uses LBFGS or stochastic gradient descent to optimize the log-loss function. Data was scaled using MinMax scaler. Using the sklearn library, our model was fitted with these parameters, hidden_layer_sizes=(16), max_iter=200,early_stopping=True.

J. Gaussian Naive Bayes

Using the sklearn library, our model was fitted with all default parameters.

K. Bernoulli Naive Bayes

Using the sklearn library, our model was fitted with all default parameters.

L. XGBoost

Utilizes the Gradient Boosting framework to provide a parallel tree boosting algorithm. This model was fitted with these parameters, use_label_encoder=False, eval_metric=’error’, n_estimators=500, learning_rate=0.01.

VIII. Machine Learning Ensembling

Machine learning had various performances based on data split ratio and ensembling. With 80% training - 20% testing:

	classifier	accuracy
9	ada	0.555333
7	gradient	0.555000
10	extra	0.552000
0	logistic	0.549333
6	random	0.549000
5	lda	0.543000
8	histogram	0.540667
1	mlp	0.537667
4	xgb	0.533667
3	gaus	0.532667
2	bern	0.528333

Fig.3. Accuracies of machine learning algorithms on a 80/20 data split

Ensembling the top five classifiers from Fig. 3 by counting the votes for each class raised the accuracy to 56.03%. With a 90% training - 10% testing split the results improve.

	classifier	accuracy
6	random	0.563333
10	extra	0.563333
9	ada	0.562000
7	gradient	0.559333
8	histogram	0.559333
0	logistic	0.551333
5	lda	0.546667
4	xgb	0.540667
1	mlp	0.536667
3	gaus	0.533333
2	bern	0.525333

Fig.4. Accuracies of machine learning algorithms on a 90/10 data split

Ensembling the votes from Fig 4. between the top three classifiers raises accuracy to 56.73%.

IX. Application

We made a web application that utilizes the flask library in python. This web application takes the side of the map you are on, your rank, the names of all the players on your team, the character they are playing, and the characters the other team's players are playing. Because of how the character selection screen works you cannot see the player names of the other team so this will not be a parameter.

The application takes these parameters and passes them into a python algorithm that will gather the data of the average statistics from each player's last 20 games and uses those values to pass into our model. The model will then output whether you will statistically win or lose the game.

We decided to not output the direct prediction because it isn't a true probability. It is a prediction on whether they will win or lose either way so we decided to simplify the output

X. Future Work

The data for this project was retrieved using a personal API given to anyone who logins into the developer portal. This key is limited to 100 requests every two minutes. Some of us had multiple accounts so we were able to use four API keys simultaneously. It is possible to get a production key given that the user submits an application with a working or close to working product. Our application was completed towards the end of the project and is still not publicly hosted and may not get approved. A production has increased rates, with up to 30,000 requests every ten minutes. Using this key, the amount of data our project can retrieve would increase dramatically. Getting a dataset with hundreds of thousands of samples, if not a few million, would allow the network to learn how champions synergize and counter each other, which can increase network performance by a few percent.

The dataset is also fairly simple. Which can lead to some area of exploration. Perhaps twenty games is not the best number of games to track. It could be five or ten. Match length was also not tracked in this dataset. Some players may have an average of 28 minutes per game in their last 20 games while another teammate had an average length of 22 minutes. Averaging stats without counting game length can cause some imbalance. Also, any game on any role is being tracked and being inserted into the same columns. Differentiating and adding columns to separate all roles could be a possibility.

With a production key it could also be possible to use all 100 matches that can be called with the Cassiopeia match history function. With all of this data, instead of having all roles go into the same ~25 columns, there could be ~25 columns for each role for each player.

The current prediction is Win or Loss only. In the future, the prediction could be a probability. The current probabilities output by Keras or sklearn are not real probabilities and require calibration to be accurate. Having an output be 49% instead of Loss can help a user's decision.

And lastly, instead of having just one output, another model can be made to predict the performance of each player on the team, so a user can use that information to more accurately decide whether to leave the lobby or start the game and play around a better performing teammate and assist them more.

XI. Conclusion

Although some of the machine learning algorithms had a slightly higher score than the neural network, we chose to use the neural network, as the training time for most machine learning models is much longer than the neural network. Currently, with 12000-135000 training samples, some classifiers take several minutes to train, with support vector

machine taking over eight minutes. If future work is to be performed, adding hundreds of thousands would increase training time for many classifiers, and training multiple classifiers to ensemble them would take a long time. Also, neural networks will at some point handle additional data when machine learning algorithms hit their limit when new data does not help as much. Achieving an accuracy of ~55-57% can be seen as a success for how limited the scope of the project is compared to how much potential it has.

XII. References

[1] Lin, Lucas. “League of Legends Match Outcome Prediction.” (2016). https://cs229.stanford.edu/proj2016/report/Lin-LeagueOfLegendsMatchOutcomePrediction-report.pdf

[2] Do, Tiffany D. (August, 2021). Using Machine Learning to Predict Game Outcomes Based on Player-Champion Experience in League of Legends. Presented at The 16th International Conference on the Foundations of Digital Games (FDG). [Research Document]. Available: https://arxiv.org/pdf/2108.02799.pdf

[3] Kenneth T. Hall, (2017), “Deep Learning for League of Legends Match Prediction”, https://github.com/minihat/LoL-Match-Prediction/blob/master/FINAL%20REPORT.pdf