Elite Summoners Assemble: Predicting The Game Outcome In League Championship Series Of The League Of Legends

Amel Docena, Raghavprasanna Rajagopalan
Northeastern Univerity
360 Huntington Avenue
Boston, MA 02115
docena.a@husky.neu.edu, rajagopalan.rag@husky.neu.edu

Abstract

League of Legends is a popular multiplayer online battle arena (MOBA) video game, and has potentially the most well known championship series (known as the LCS). The complexity of the game make any attempts to accurately predict the winner of a match somewhat difficult. In this paper, we construct and compare several machine-learning models for the best performance in making predictions on competitive matches. We train these models on curated features from 2017 and 2018 data, show that the Random Forest model performs best, and we then test how well this model predicts the outcomes of matches in 2019. We find that the accuracy of the model drops, and we draw the conclusion that the game is constantly evolving as a likely explanation.

Introduction

League of Legends (LoL), and other MOBA, belong to a subcategory of strategy video games. Depending on the game mode, the strategy is different, and the goals change as well, but the overarching goal for any game is to defeat and take down the enemy teams Nexus. Before going into what that means, we provide some context.

LoL offers a variety of game types: 3v3, 5v5, Featured, and Custom. The featured game mode is a temporary game mode, with the rules changed or where players have different priorities. The 3v3 and 5v5 are unchanging - this is where ranked play occurs. The ranked 5v5 is the most popular game mode, and is the mode that the Riot Games, the company that owns *LoL*, uses when hosting the League Championship Series (LCS). In this game mode, there are two teams of five (each on opposite corners of the map (see Figure 1)). Each player, called a summoner, summons a champion (picks the champion they will play as), and each champion has its own unique abilities. In this game mode, players play one of five roles: Top laner, Mid laner, Bot laner (Attack Damage Carry), Support and Jungler. The champion a player picks also influences what role they will be playing as - certain champions are best fitted for certain roles, and a balanced team will have a composition such that each role is filled. Players might fight their opposing laner (the player on the other team who is also playing the same lane as them), cooperate with the rest of their team to destroy enemy towers, take objectives, and ultimately destroy the enemy teams Nexus (located in their base).

Any *LoL* match has two components - the Draft phase, where players of each team take turns to pick what champion each player will play as, but also attempt to ban the enemy team from playing certain champions. By the end of the Draft phase, the team composition will be fixed. Then, when the game begins, there are certain stages as well. The early game is where players are likely to be playing safe, gaining experience and killing enemy minions (amassing gold), which they can use to buy items that provide certain buffs. Any opportunity to get a kill on your counterpart on the enemy team is welcome, and the jungler is encouraged to gank, or help plan a surprise attack to take down or help the laner take down the enemy. By the mid game, the jungler and laners play a more active role around the whole map, focusing now on objectives, like monsters in the jungle, or participating in team fights, trying to minimize losses and maximize damage to the enemy team. Losses include losing towers, losing team fights around an objective (like the dragon or the baron). Killing an enemy champion has a certain reward, but it isnt always the best action to take. It becomes difficult to gauge just how a team is doing because different teams may prioritize different objectives, i.e., failing to take a dragon (letting the enemy team take it) but instead taking down a tower. Also, even if teams have clear priorities, players can make mistakes. All of these make predicting the outcome of a match even harder.

It is interesting to focus on the LCS (and the other *LoL* tournaments across the world) because certain features are bound to have more importance in pro-play that may not matter as much in lower ranked matches.

Problem Statement

Under this context, we wish to predict the outcome of a match in LCS pro tournaments using in-game features by devising several classification methods. Alongside our analysis, we wish to draw game insights, such as which of those in-game features are more likely to contribute to the chances of a team winning.

Figure 1: Summoner's Rift, the traditional MOBA map in League of Legends

Related Work

There has been some research on this relatively new game type. Silva, et. al. [1], Yu et al. [3], and Lin [4] have done work to predict match outcomes using different features and different levels of granularity. [1] utilized a dataset that gave them information about a match on a minute-by-minute basis, giving them higher granularity. Their work compares three different recurrent neural network (they had used RNNs because of the sequential nature of the data[1]) models, which they call SimpleRNN, Long Term Short Memory, and Gated Recurrent Unit, and showed that SimpleRNN actually had the best performance for predicting the outcome of a match in the 10-15 minute range. They then demonstrated, using various intervals from 0-25 min, that accuracy of the SimpleRNN model increased as the game progressed. All three of their models had the same topology, which was relatively simple (1 hidden layer, 8 recurrent ReLU nodes, with dropout of rate 0.25, and output layer with 2 nodes using the softmax activation function), which led us to believe our neural networks would not have to be extremely complex.

[3] takes a different approach and introduces a Slice Evaluation model, which takes into account the current state of the game, along with the future trend, and for each, looks at both the global state of the game as well as how each individual is performing at the current time to ultimately make a decision. In the future, we too wish to take into account individual gameplay, and [3]s work provides some groundwork that we could expound upon.

Some work has been done by Sapienza et al. [2] where, rather than predict the outcome of a match, they instead generate the optimal team composition - that is, they construct a network based on cooperation, which they call a co-play network. An edge between two players exists if the two players have played a match together on the same team, and the weight of that edge depends on the skill increase or decline as a result of playing with that player. Their work looks to quantify skill transfer to create the optimal team. The work done by [1] [2] [3] involved another popular MOBA, called *Defense of the Ancients 2 (Dota 2)*, not *LoL*, so while there are a lot of commonalities, we felt our domain knowledge on *LoL* could help us in developing performant models. [4], however, does focus on *LoL* - they showed that pre-game features were not as good predictors as in-game features. They used Gradient Boosted trees and Logistic Regression and Gradient-boosted variants that also use Logistic Regression to demonstrate this. A technique they employ for feature selection is one-hot encoding, which allows them to transform categorical variables as a series of binary indicator values. [2] makes use of autoencoders for feature selection, which we did not use, but we aim to use the simpler principal component analysis to discover hidden features in our future work.

Methods

We first inspected our data sets and studied the features included. We then processed them for meaningful use, partitioning into training, development, and test sets. Afterwards, we trained our classification models.

Data Processing

About the data set. Because we wanted to specifically look at match outcome prediction at the professional level, we used the 2017, 2018, and early 2019 *LoL* championship series data, curated and made publicly available by Oracles Elixir - this dataset contains professional tournament data from each of the following *LoL* tournaments across the world: NA LCS, EU LCS, LCK, LMS, CBLoL, TCL.

Processing the data set. Some of the game matches have missing values. We opted not to impute them and excluded those matches that do not have complete cases from our analysis. Such instances were not stored by the books, and so we presumed they are missing at random. Afterwards, we transformed the variables into meaningful ones, and applied one-hot encoding to categorical variables. We then normalized the predictors according to the model we used.

Resample the data into partitions. We set aside the 2019 Spring LCS match dataset as our test set. The 2017 and 2018 LCS data sets we resampled into training (80%) and development (20%) sets. We trained our models using the training set, compared their accuracies using the development set and selected the best one, then tested its accuracy on the test set.

Modeling

A careful treatment of variables. In this game where the objective is to destroy the enemys Nexus, the team who destroys this structure is rewarded by some gold amount. But since destroying the nexus determines who wins the game, we deducted 250 gold from the variable gold earned by the winning team to avoid this predetermination, because this reward determines who won the game. This deduction preserves our response variablewhich team wins the matchfrom potential bias by our predictors.

Proxy variables. Although there may be intricate in-game features that are not captured by the data set, we can still use proxy variables that capture them. For example: how well a team is able to control the lane, clear out creep waves and monsters, push and destroy the towers, clash with enemy team successfully, and eventually overthrow enemys nexus can be filled with unique game scenarios that have not been included in the data set. But the team who is able to win these game scenarios certainly earn gold. So even though these game-specific intricacies are absent from the data set, a proxy variable, say gold, can still measure team performance and can be a good predictor.

Classification methods. For our classification, we used logistic regression as baseline, a regularized logistic regression to include more predictors, random forest as our ensemble method, and neural networks. Alongside our predictions, we tried to draw insights from them.

Logistic Regression

We wish to predict the probability whether a team would win a match given some set of in-game features. That is, $P(y = 1|x) = G(B0 + \beta 1$ gameFeatures),where $0 < G(x) < 1$. For our baseline model, we collected some features from the early to mid game phase: side of the map, who drew first blood, XP lead at 10 minutes, and gold lead at 15 mins; for the mid to late game phase, we included which team took down towers 1 to 3 first, plus baron and dragon kills, (which are the stronger monsters of the game and both give buffs to champions who kill these beasts).

Being on the red side of the map posits a slight disadvantage (see Figure 2). (This resonates the clamor of players in forums for a well-balanced map [1].) Drawing first blood is not statistically significant; the chances of winning are not related to this feature. In-game features become more significant as we move on to the mid and late game, wherein the stronger predictors are percent gold lead at 15 mins, first team to destroy first to third towers, and baron and dragon kills. This suggests that which team controls the mid to late game is more likely to win it.

Cross-validated Lasso Logistic Regression

We included a larger set of predictors for our logistic regression. But we now regularized the parameters to avoid the risk of multicollinearity and overfitting. We opted for Lasso regularization (or the absolute magnitude error) as our shrinkage method. We first standardized the predictors prior to running the logit. To pick the tuned parameter (or the penalty), we performed a 10-fold cross-validation and determined which has the lowest misspecification error. The optimal penalty found for this model is =0.0008 with a cross-validated MSE of 0.0290.

The results of the cross-validated Lasso logistic regression are congruent to what we found earlier even after adding more predictors and shrinking some of the coefficients close to and even zero. In addition to those predictors that contribute to winning a game, greater number of champion (or enemy) kills and assists, creep score per minute, and percent gold lead post 15mins (which is a mid to late game feature) contribute to the likelihood of winning, wherein the last one is the strongest predictor. But the converse is true for the frequency of deaths, which is expected. Notice how the coefficients for which team drew the first blood and which team destroyed towers 1 to 3 first have been shrunk to zero. The former result attests to our previous result that who drew the first blood does not necessarily add up to the chances of winning. What is curious is the latter predictor. A possible explanation could be collinearity among our added predictors. For instance, being able to take three towers down implies a successful siege of enemy base, which would then entail killing enemy champions who are out there to defend it, thereby, rewarded with higher gold. And so, this predictor which is already related to these latter features must have been shrunk to zero.

Random Forest

Another method we employed for classification is Random Forest. We grew multiple subtrees from a subset of our predictors; in our case, the size of the subset is approximately square root the number of our predictors. We sampled the subset randomly with replacement as candidate splits, and grew 500 trees. The metric we used in determining the split for each subtree is the reduction in Gini Index, which is a measurement for node purity, which in turn indicates how well the observations are classified. We then averaged the decision by majority vote. Unlike a single decision tree where the splits are interpretable, in random forest this has been compromised because of the ensemble. But we could still identify which predictors have contributed most in the splits by the mean reduction in the Gini index.

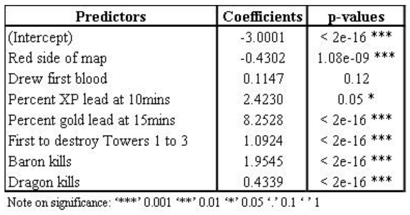
The top five features that contribute most in the classification are percent gold lead post 15 mins, earned gold per min, baron kills, deaths and assists. While, the lower five features are gold difference at 10 mins, wards (used for map vision) per min, being on the red side of the map, and who drew the first blood. This suggests that in pro tournaments where the battles can be too competitive, the stronger features that decide the outcome of the game lie in the mid to late game phase. Disadvantages during the early to mid game can be won over as the game progresses through the late game.

Neural Networks

Feature Extraction. In addition to the methods mentioned above, we were interested in building neural networks-based classifiers for predicting the winning team. When we first set out to build this model, we already had an idea of which features were of particular interest to us - these were:

- Which side of the map the team was on (this was a categorical variable that was one-hot encoded)
- The gold difference between the two teams 10 minutes into the game,
- The gold difference between the two teams 15 minutes into the game ,
- The XP difference between the two teams 10 minutes into the game,

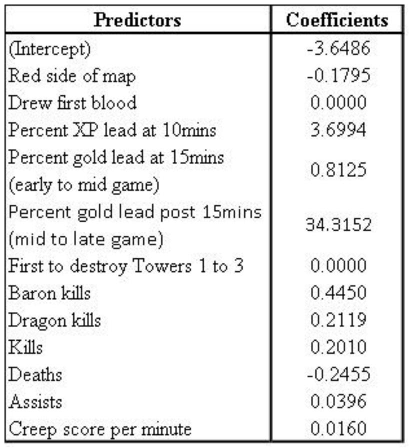
[1]Red is at a disadvantage. A LOL community forum. Retrieved online: https://boards.na.leagueoflegends.com/en/c/gameplay-balance/KEf5VH6q-red-is-at-a-disadvantage



Predictors	Coefficients	p-values
(Intercept)	-3.0001	< 2e-16 ***
Red side of map	-0.4302	1.08e-09 ***
Drew first blood	0.1147	0.12
Percent XP lead at 10mins	2.4230	0.05 *
Percent gold lead at 15mins	8.2528	< 2e-16 ***
First to destroy Towers 1 to 3	1.0924	< 2e-16 ***
Baron kills	1.9545	< 2e-16 ***
Dragon kills	0.4339	< 2e-16 ***

Note on significance: '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Figure 2: Logistic regression results



Predictors	Coefficients
(Intercept)	-3.6486
Red side of map	-0.1795
Drew first blood	0.0000
Percent XP lead at 10mins	3.6994
Percent gold lead at 15mins (early to mid game)	0.8125
Percent gold lead post 15mins (mid to late game)	34.3152
First to destroy Towers 1 to 3	0.0000
Baron kills	0.4450
Dragon kills	0.2119
Kills	0.2010
Deaths	-0.2455
Assists	0.0396
Creep score per minute	0.0160

Figure 3: Cross-validated Lasso logistic regression sparse matrix (= 0.0008, MSE= 0.0290)

- Which team got first blood (the first kill of the game),
- Which team took down three enemy towers first

We identified these as *basic features*. Subsequently we considered certain later-game features, like the amount of gold earned by a team after the 15 minute mark, using the gold at 15 statistic and the total gold statistic. We also ensured we normalized all our feature vectors before training.

Building Models. Prior to the construction of different neural network models, it was not yet clear what characteristics a neural network suited to our problem would need, so it was of foremost importance that we be able to configure many models without having to change the code to build these networks. Once we achieved this, training and testing was simply a task of iterating through these configurations.

From [1] we knew that it was likely that the best model would be a simpler one, with fewer layers. In our initial experimentation, we varied our models - it was necessary to vary number of nodes per hidden layer (either 2 or 4), number of layers (1, 2, 3, 10), which optimization function to use (ADAM or NADAM) [2]. We also created exact copies of existing models with dropout layers interleaved to prevent overfitting. Across all our models, the following remained the same:

- The ReLU (rectified linear unit) activation function was used for each node in a hidden layer (it is a popular activation function for hidden layers, and outputs values from 0 to infinity)
- The sigmoid activation function was used for the output layer (it is a popular activation function for output layers in multiclass classification problems as it outputs values between 0 and 1, and so can easily be partitioned.
- Binary cross entropy was used as our loss function, as it seems to be the preferred logarithmic loss function for binary classification problems

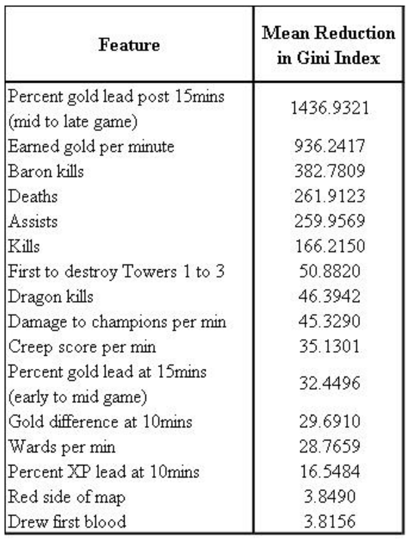
After our initial experimentation, we found that 10 layered neurals networks were too complex for the weights to converge, as we predicted. We thus removed these models from our configuration.

Results. Unlike the previous methods where we were able to interpret the results surrounding the the parameters of each feature, in neural networks we lose interpret-ability. Instead, we draw our attention solely to the accuracy of our neural models. When considering the basic features, these were our results (see Figure 5).

Our best performing model had the following structure:

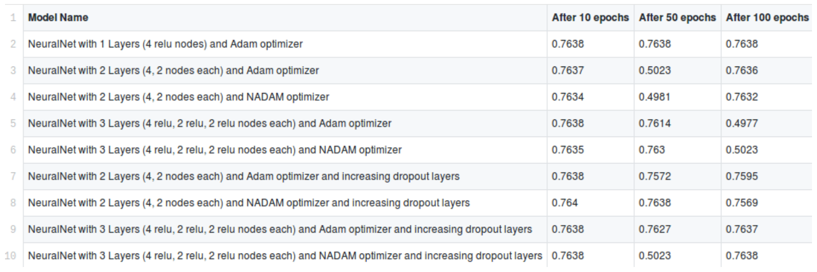
- 2 Hidden Layers
  - Hidden Layer 1: 4 Nodes with ReLU activation function
    - ∗ Regularizer - Dropout rate 0.1
  - Hidden Layer 2: 2 Nodes with ReLU activation function

[2]Adam is a relatively new (it was introduced in 2014) optimization algorithm for gradient descent, which is how we update our feature weights. Nesterov adam is also a gradient descent optimizer that uses a mechanism called Nesterov momentum as part of optimization. We do not use RMSprop, which [1] used, since that is better suited to RNNs which we did not use



Feature	Mean Reduction in Gini Index
Percent gold lead post 15mins (mid to late game)	1436.9321
Earned gold per minute	936.2417
Baron kills	382.7809
Deaths	261.9123
Assists	259.9569
Kills	166.2150
First to destroy Towers 1 to 3	50.8820
Dragon kills	46.3942
Damage to champions per min	45.3290
Creep score per min	35.1301
Percent gold lead at 15mins (early to mid game)	32.4496
Gold difference at 10mins	29.6910
Wards per min	28.7659
Percent XP lead at 10mins	16.5484
Red side of map	3.8490
Drew first blood	3.8156

Figure 4: Random forest results feature importance



Model Name	After 10 epochs	After 50 epochs	After 100 epochs
NeuralNet with 1 Layers (4 relu nodes) and Adam optimizer	0.7638	0.7638	0.7638
NeuralNet with 2 Layers (4, 2 nodes each) and Adam optimizer	0.7637	0.5023	0.7636
NeuralNet with 2 Layers (4, 2 nodes each) and NADAM optimizer	0.7634	0.4981	0.7632
NeuralNet with 3 Layers (4 relu, 2 relu, 2 relu nodes each) and Adam optimizer	0.7638	0.7614	0.4977
NeuralNet with 3 Layers (4 relu, 2 relu, 2 relu nodes each) and NADAM optimizer	0.7635	0.763	0.5023
NeuralNet with 2 Layers (4, 2 nodes each) and Adam optimizer and increasing dropout layers	0.7638	0.7572	0.7595
NeuralNet with 2 Layers (4, 2 nodes each) and NADAM optimizer and increasing dropout layers	0.764	0.7638	0.7569
NeuralNet with 3 Layers (4 relu, 2 relu, 2 relu nodes each) and Adam optimizer and increasing dropout layers	0.7638	0.7627	0.7637
NeuralNet with 3 Layers (4 relu, 2 relu, 2 relu nodes each) and NADAM optimizer and increasing dropout layers	0.7638	0.5023	0.7638

Figure 5: Neural Network model performance on basic features

    * Regularizer - Dropout rate 0.2

- Output layer: 1 Node with Sigmoid activation function
- Loss function: Binary Cross entropy
- Optimizer: ADAM

This model achieved an accuracy of 0.7640, with only 10 epochs of training! When considering the late game gold as well, we achieved these results (see Figure 6):

Again, the same model (let us call it NN4D2D]) outperformed the others, this time reaching an accuracy of 0.7673, after 50 epochs of training. As we see, 2 layer model performed better than 3, and using dropout also seems to successfully mitigate overfitting.

Model Accuracies

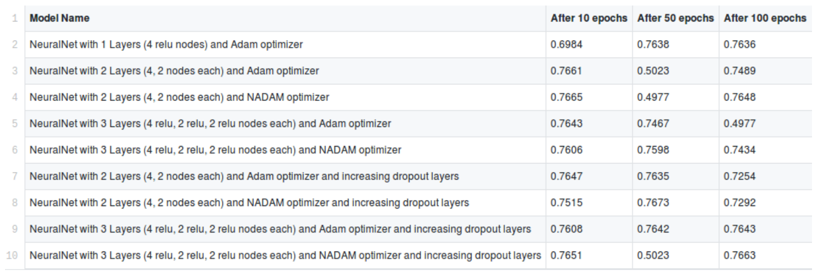
After training our models, we evaluated their performance on the development set. The model that performed best was the Random Forest, with accuracy of 0.9706 (see Figure 7 for other models' performance).

With such high accuracy on the development set, it would be exciting to find out how the Random Forest would fare when tested using the 2019 data set. Note that we trained the RF using 2017 and 2018 match data. But surprisingly after testing for the 2019 data, the RF was able to correctly predict only about 0.5491 of the game result, which is a huge drop from its optimistic accuracy using the development set that consisted of 2017 and 2018 match data. There are two suggestions we can think of for this drop. One possibility is that we might have overfitted the training data. But this cannot be so since we used a separate partition for our development set when we measured its accuracy. Perhaps this suggests that the model predicts outcomes well for 2017 and 2018 matches, but if introduced to match data that has had several patch updates, we cannot expect the model to perform relatively as well. Especially if these updates have tweaked the workings of the game, which they have [3] - this could mean that features that we found were important in 2017 and 2018 were not as important in 2019.

Conclusion

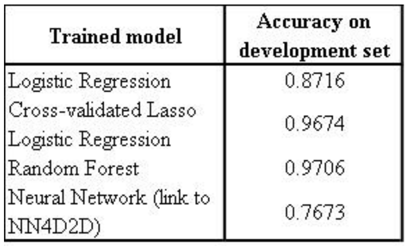
In this work we compared how well several different machine-learning models predict the outcome of professional matches in *League of Legends*, using a selected set of in-game features. The Random Forest classifier was found to be the best method for predicting outcomes of 2017 and 2018 data, but when we used this model to predict outcomes

[3]https://dotesports.com/news/2019-league-of-legends-preseason-changes



Model Name	After 10 epochs	After 50 epochs	After 100 epochs
NeuralNet with 1 Layers (4 relu nodes) and Adam optimizer	0.6984	0.7638	0.7636
NeuralNet with 2 Layers (4, 2 nodes each) and Adam optimizer	0.7661	0.5023	0.7489
NeuralNet with 2 Layers (4, 2 nodes each) and NADAM optimizer	0.7665	0.4977	0.7648
NeuralNet with 3 Layers (4 relu, 2 relu, 2 relu nodes each) and Adam optimizer	0.7643	0.7467	0.4977
NeuralNet with 3 Layers (4 relu, 2 relu, 2 relu nodes each) and NADAM optimizer	0.7606	0.7598	0.7434
NeuralNet with 2 Layers (4, 2 nodes each) and Adam optimizer and increasing dropout layers	0.7647	0.7635	0.7254
NeuralNet with 2 Layers (4, 2 nodes each) and NADAM optimizer and increasing dropout layers	0.7515	0.7673	0.7292
NeuralNet with 3 Layers (4 relu, 2 relu, 2 relu nodes each) and Adam optimizer and increasing dropout layers	0.7608	0.7642	0.7643
NeuralNet with 3 Layers (4 relu, 2 relu, 2 relu nodes each) and NADAM optimizer and increasing dropout layers	0.7651	0.5023	0.7663

Figure 6: Neural Network model performance on later in-game features



Trained model	Accuracy on development set
Logistic Regression	0.8716
Cross-validated Lasso Logistic Regression	0.9674
Random Forest	0.9706
Neural Network (link to NN4D2D)	0.7673

Figure 7: Accuracies of the trained models on the development set. (For reference, NN4D2D refers to the best neural network model)

on 2019 data, we saw a huge drop in accuracy. But we attribute this phenomenon to the game constantly evolving. Patch updates have since changed the game; professional teams have since been shuffled and so have adjusted their gameplay; new game strategies have been tinkered with; all of these factors make the new ranked season different from previous seasons.

Limitation and Future Work

For future work, we have a notion of what areas we want to explore. We first aim to evaluate how our neural networks perform against the 2019 dataset and take note of how the accuracy fluctuates, as an attempt to make a statement about the resilience or lack thereof of the model. At a high level, we would like to look into predicting optimal champion drafts, where a model might assist with giving advice about what champions to pick and which to ban. We would also like to take into account features related to the player/summoner, and the champions as well. Using one-hot encoding and further data processing methods would allow us to represent team compositions as a vector. At a slightly more granular level, with respect to neural networks, we hope to build more complex models that integrate classifiers for players with this global classifier, similar to the TSE model developed by [3]. We also hope to use principal component analysis to automate the feature engineering process.

Acknowledgments

We extend our gratitude to Prof. Chris Amato for his valuable guidance during our modeling period. This gave us insight as to what direction to proceed in when performing our experimentation.

Appendix

Our work is publicly available on Github at https://github.com/raghavp96/league-ai.

Bibliography

[1] A.L.C Silva, G.L. Pappa, L. Chaimowicz. Continuous Outcome Prediction of League of Legends Competitive Matches Using Recurrent Neural Networks. In Proceedings of SBGames 2018, Computing Track - Short Papers, October 29 - November 1 2018, 2018, pp 639-642. [Online]. Available: www.sbgames.org/sbgames2018/files/papers/ComputacaoShort/188226.pdf.

[2] A. Sapienza, P. Goyal, and E. Ferrara. Deep Neural Networks for Optimal Team Composition. 8 May 2018. [Online]. Available: https://arxiv.org/abs/1805.03285#

[3] L. Yu, D. Zhang, X. Chen, X. Xie. MOBA-Slice: A Time Slice Based Evaluation Framework of Relative Advantage between Teams in MOBA Games. July 22 2018, 2018, pp. 1-17. [Online]. Available: https://arxiv.org/abs/1807.08360

[4] L. Lin. League of Legends Match Outcome Prediction [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=2ahUKEwjm6eiNlejhAhVkUN8KHU05BVIQFjAAegQIABAC&url=http%3A%2F%2Fcs229.stanford.edu%2Fproj2016%2Freport%2FLin-LeagueOfLegendsMatchOutcomePrediction-report.pdf&usg=AOvVaw1BZOmqVnBUonSEU-DyxXw-

Additional References

An Introduction to Statistical Learning with Applications in R. James, G., Witten, D., Hastie, T., Tibshirani, R., Springer Series in Statistics.

Early-game Rating and Mid-late Rating: New Team Ratings. Oracless Elixir, League of Legends e-Sports Statistics. Retrieved online: http://oracleselixir.com/2015/10/egr-and-mlr-new-team-ratings/

Match Data Downloads (Beta), Oracless Elixir, League of Legends e-Sports Statistics. Retrieved online: http://oracleselixir.com/match-data/

The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Hastie, T., Tibshirani, R., Friedman, J., Springer Series in Statistics.

Helpful Documentation

- Tutorial: https://www.tensorflow.org/tutorials/keras/basic_classification
- PCA: https://www.tensorflow.org/guide/embedding, https://www.tensorflow.org/tfx/transform/api_docs/python/tft/pca
- Dropout Layers: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5, https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout
- Optimizers: https://keras.io/optimizers/