
Machine Learning Methods for Predicting League of Legends Game Outcome

Damita Gomez
damitago@usc.edu

Loring Scott Hoag
lhoag@usc.edu

Robert Pabo
pabo@usc.edu

Abstract

This paper attempts to reproduce the results and methodology via the research paper "Machine Learning Methods for Predicting League of Legends Game Outcome" by Hitar-Garcia et al. While the paper explores various machine learning models, our focus is replicating their best performing model "*xgbStack*", which consists of an ensemble of Naive Bayes and a neural network as base models, and extreme gradient boost (xgBoost) as the meta-model, using the features engineered in the original paper.

1 Introduction

Esports (electronic sports) can be defined as the broadcasting of professional video game matches. Compared to traditional sports, esports can be considered a relatively new phenomenon. League of Legends is considered one of the most popular competitive games in esports, whose regional and international tournaments generate some of the largest prize pools.

Riot Games, the developers of League of Legends [1], created an application programming interface (API) that allows access to vast statistical data, including pre-game and post-game information. This has enabled numerous studies in machine learning. At the time of the publication of the research paper [2], they noted that there was little research on professional levels of the game. Since then, a number of research papers have been published that shed additional light in this area and have proposed improved models for predicting the winning team.

1.1 Game description

League of Legends (LoL) is a competitive 5v5 team game that takes place primarily on a nearly symmetrical map called "Summoner's Rift," divided by a river and featuring three primary lanes (top, middle, and bottom) with neutral jungle zones in between (see Figure 1). Each team operates from their base and assigns players to roles (top, jungle, middle, ADC, support) that correspond to specific regions or functions on the map.

Before the match begins, players select champions from a pool of over 160 options, each with unique abilities. Teams can then "ban" a certain number of champions during a match, preventing either side from using them. Team strategies heavily depend on the composition of champions and their roles. Gameplay involves destroying enemy towers, structures (known as "inhibitors") and eventually the "nexus", the main structure in a base in order to secure victory. Throughout the match, players control champions that accumulate experience and gold through in-game actions and completing map objectives. Player coordination and role execution are pivotal for winning, especially in later phases of the game, where team battles become prevalent and typically are the deciding factor in which team wins the match.



Figure 1: Map of "summoner's rift". Image from research paper, [Hitar-Garcia et al].

2 Methodology

Our goal was to attempt to replicate the methods and results of the original paper. One note is that the original paper focused on comparing eight different models and two meta-models. With approval from the class TAs, we decided to focus on just replicating some of the paper's best performing models (Naive Bayes classifier, and neural network) and the best performing meta-model (xgbStack) in order to complete the project within the class semester time frame.

While the paper explored model performance using the original features from the dataset, in the interest of time we opted to forego this testing and focus strictly on reproducing the results based the engineered features cited in the paper.

3 Implementation details

Work for replicating the models was split up evenly between the three teammates. Damita Gomez wrote the Naive Bayes classifier implementation, Loring Hoag wrote the neural network implementation, and Robert Pabo wrote the xgbStack meta-model. Damita and Loring wrote scripts for matching the paper's data pre-processing and feature creation steps. Robert combined the model work from Damita and Loring and verified the meta-model results. Robert also organized group meetings and handled many of the project management elements.

3.1 Pre-processing

Pre-processing consisted strictly of reusing the engineered features proposed in the original paper. No additional engineered features were created. Figure 2 shows the engineered features used in reproducing the model.

Features were created by running the same dataset used in the paper through our own pre-processing steps. Each feature ratio in the original paper is recreated using the methods described by the authors, including a random 90%/10% split for training and testing sets, and Z-score normalization. To ensure correctness, we did the pre-processing twice using different methods to create two distinct feature sets: One set was created by manually counting all of the win/loss ratios for each player, champion, team, etc, and sorting those into individual tables similar to how the authors described performing it themselves. The other set was made by using the pandas library to automate the counting and ratio calculation without performing intermediate matrix creation for each generated feature. In the code portion of the project, we have included both the manual counting python script as *makedata.py* (which can be run as part of a data-creation pipeline with Z-score normalization using the *makedata.bat* batch script) and the single-step pandas notebook version in *preprocess_data.ipynb*.

Feature	Description
<i>btPlayerRole</i>	w.r.* player in role top blue
<i>bjPlayerRole</i>	w.r. player in role jungle blue
<i>bmPlayerRole</i>	w.r. player in role middle blue
<i>baPlayerRole</i>	w.r. player in role ADC blue
<i>bsPlayerRole</i>	w.r. player in role support blue
<i>rtPlayerRole</i>	w.r. player in role top red
<i>rjPlayerRole</i>	w.r. player in role jungle red
<i>rmPlayerRole</i>	w.r. player in role middle red
<i>raPlayerRole</i>	w.r. player in role ADC red
<i>rsPlayerRole</i>	w.r. player in role support red
<i>btPlayerChampion</i>	w.r. player with champion in role top blue
<i>bjPlayerChampion</i>	w.r. player with champion in role jungle blue
<i>bmPlayerChampion</i>	w.r. player with champion in role middle blue
<i>baPlayerChampion</i>	w.r. player with champion in role ADC blue
<i>bsPlayerChampion</i>	w.r. player with champion in role support blue
<i>rtPlayerChampion</i>	w.r. player with champion in role top red
<i>rjPlayerChampion</i>	w.r. player with champion in role jungle red
<i>rmPlayerChampion</i>	w.r. player with champion in role middle red
<i>raPlayerChampion</i>	w.r. player with champion in role ADC red
<i>rsPlayerChampion</i>	w.r. player with champion in role support red
<i>bCoopPlayer</i>	\sum w.r. pairs blue player with blue player
<i>rCoopPlayer</i>	\sum w.r. pairs red player with red player
<i>bCoopChampion</i>	\sum w.r. pairs blue champion with blue champion
<i>rCoopChampion</i>	\sum w.r. pairs red champion with red champion
<i>vsPlayer</i>	\sum w.r. pairs blue player vs red player
<i>vsChampion</i>	\sum w.r. pairs blue champion vs red champion
<i>bTeamColor</i>	w.r. team in blue side
<i>rTeamColor</i>	w.r. team in red side

* w.r. = win ratio

Figure 2: Engineered features from original paper, [Hitar-Garcia et al].

3.2 Model implementation

The ensemble model was written in Python, primarily using the scikit-learn (sklearn) and pandas libraries. The Naive Bayes implementation was also implemented using the sklearn library. Tensorflow and Keras were used for the implementation of the neural network, while scikeras was used to interface the neural network with the ensemble model. The xgboost library was used in the implementation of the meta-model.

The original paper did not explicitly mention the variant of Naive Bayes utilized. Based on the nature of the features in the dataset, which heavily relies on the use of ratios, we settled on using a Gaussian Naive Bayes implementation.

3.3 Hyperparameters

There were some differences between the hyperparameters used in our implementation versus those cited in the research paper, due to the differences in programming languages used. In the sections below, we describe the values used for the base models and the meta-model.

3.3.1 Naive Bayes

The original paper disclosed a number of hyperparameter settings that could not be correlated to any parameters defined in sklearn’s GaussianNB implementation [3]. The original paper mentioned that they built their models using R, but they did not specify whether they used R libraries or built everything from scratch. Nonetheless, there was not a clear way to apply the kernel density estimation (KDE) function hyperparameter and its adjustment value to the model when using sklearn. In hindsight, the Naive Bayes model could have included KDE if implemented from scratch. Since the original paper used a Laplace smoothing value of zero, it was not included in the final model. We found that the default values defined in GaussianNB to be optimal in our model.

3.3.2 Neural Network

The neural network was built with TensorFlow. Initially, we used the same structure and hyperparameters as in the original paper. However, through tuning using the RMSProp optimizer, we found different hyperparameter values which returned slightly more accurate results. The paper’s hyperparameters compared with our own along with the difference in testing accuracy are listed in table 1. In either case, we found that our testing accuracy was significantly higher than the reported values from the paper. We will attempt to explain the higher accuracy later in the Results section.

Also, as the assignment goal was to try to replicate the original paper’s methods and results, we used the original paper’s hyperparameters to train the neural network model used for our xgbStack meta model.

Table 1: Neural Network Hyperparameter Comparison

Hyperparameter	Original Values [Hitar-Garcia et al]	Tuned Values
Hidden Layers	1	1
Neurons Per Layer	256	135
Dropout Rate	0.4	0.26
Learning Rate	3e-4	3e-3
Gradient Decay rho	0.9	0.74
Activation Function	tanh	tanh

3.3.3 Extreme Gradient Boost (xgboost)

The hyperparameter values mostly followed those cited in the research paper, with the exception of "learning_rate," which uses the default value provided. Some of the property names cited in the paper did not match the names that were used in the xgboost library, and is noted in the table below.

Table 2: xgboost Hyperparameter Comparison

Hyperparameter Property Name [Hitar-Garcia et al]	Original Values [Hitar-Garcia et al]	xgboost Property Name	Reproduced Values
nrounds	14000	n_estimators	14000
maxdepth	2	max_depth	2
eta	1e-6	learning_rate	0.3
gamma	0	gamma	0
colsample_bytrees	1e-9	colsample_bytree	1e-9
minchildweight	1	min_child_weight	1
subsample	0.15	subsample	0.15

4 Experiments

Our experiment setup closely followed the methodology documented in the research paper, including the ten-fold cross-validation across five repetitions. The main dataset used was identical to the one cited in the original paper, which consists of approximately 7700 match data entries from 2014 to 2018. For additional testing, we obtained datasets of matches generated between 2019 and 2024 ¹. However, the structure of the newer datasets required some additional preparation to convert them into a format similar to the original dataset used. After conversion, this allowed approximately an additional 61,000 entries for use.

Initial performance testing was done separately on the base models and on the combined ensemble model. The accuracy values were obtained using a 90%/10% train-test-split followed by a fit and predict operation on the models.

All training and testing was done on personal machines. None of the workload was conducted on purpose-built AI-compute resources.

Because of the high prediction values in relation to those in the original research, we used the mean and standard deviation of the cross-validation folds to get a better idea of model performance. For evaluating the importance of the engineered features, we used permutation feature importance to determine what features were contributing most to the overall performance of the model. This was also used to check for potential overfitting related to certain features.

¹OE Public Match Data. [Online] Available: <https://drive.google.com/drive/folders/1gLSw0RLjBbtaNy0dgnGQDAZOHIgCe-HH>

4.1 Results

When testing via the base models and with the ensemble, we observed accuracy values that vastly outperformed the values cited in the research paper. We achieved similar results when using the newer datasets as well. Table 3 shows a comparison of our results versus those produced in the original paper using the cited dataset. The values shown in "Reproduced Accuracy (fit, predict)" were obtained through performing a fit on the training set and then a predict operation on the test set. The "Accuracy, mean" and "Standard Deviation" fields were results produced from running k-fold cross-validation with k=10.

Table 3: Accuracy Comparison: Original Dataset (approx. 7700 entries)

Model	Accuracy [Hitar-Garcia et al]	Reproduced Accuracy (fit, predict)	Accuracy, mean (cv, k=10)	Standard Deviation (cv, k=10)
nb	0.6896	0.996	0.8789	0.0132
nnet	0.6407	0.9895	0.9482	0.008
xgbStack	0.7041	0.9998	0.9350	0.005

Given each of the models exhibited high accuracy values, we initially believed the main issue must lie with the pre-processing of the dataset and not with how the models were implemented. To test this hypothesis, we trained the models on the newer data containing the 61000 entries. Table 4 below shows the results.

Table 4: Accuracy Comparison: New Dataset (approx. 61000 entries)

Model	Reproduced Accuracy (fit, predict)	Accuracy, mean (cv, k=10)	Standard Deviation (cv, k=10)
nb	0.996	0.8921	0.0036
nnet	0.997	0.9431	0.0290
xgbStack	1.000	0.9411	0.0024

Note that the neural network accuracy appears to be comparable in performance to the full ensemble stack. The hyperparameter values used in the paper for the xgb meta-model did not appear to be optimal for our implementation. Additional hyperparameter tuning could have improved the mean accuracy of the ensemble such that it is not outperformed by the neural network.

While training the model on the newer dataset resulted in slight improvements, it also reinforced our assumption that the accuracy values observed may be due to how the engineered features were implemented. Figure 3 below illustrates which features the model prioritized when making predictions.

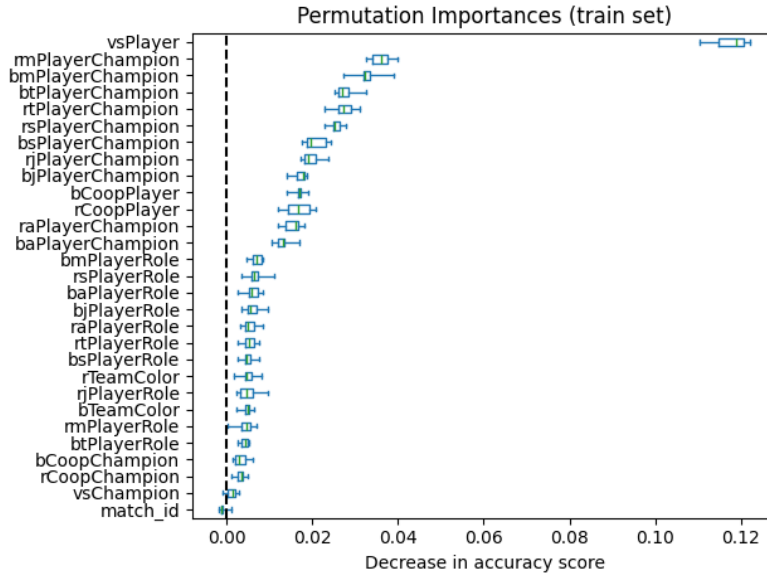


Figure 3: Permutation feature importance using original dataset

It was noted that the "vsPlayer" feature contributed heavily to the overall performance of the model, and was a good candidate to look into for potential data leakage. The original paper describes vsPlayer as a feature that "calculated as the sum of the winning ratios of the 25 possible combinations of each player of the blue team, when facing each of the players of the red team." In other words, this feature was a cumulative value that encapsulated each respective blue player's performance against all red players. Therefore, it would not be surprising that this feature would contribute significantly to the model. However, the feature appeared to dominate all others in determining the performance of the model, and warranted further investigation for potential leakage.

After analyzing the implementation of the vsPlayer feature along with the other features, we were unable to rule out data leakage within our implemented features.

Additional research within this area has reported results with accuracy results in the low 90% range, thus it is not unreasonable for a given classification model such as our own to produce high results. However, given our efforts to follow the steps of the original paper and the data they used, we expected our accuracy results to be much lower. The table below shows the accuracy of various other research papers within the area, in addition to papers referenced by Hitar-Garcia et al [4][5].

Table 5: Comparison of performance

Authors	Model Type	Accuracy
Bahrololloomi et al., 2023	CatB/VotR	0.9298
White et al., 2020	RNN + LR	0.72
Hitar et al., 2022	xgbStack	0.70
Makine et al., 2023	LightGBM	0.968

5 Conclusion

We set out to replicate the models of the original paper and accomplished that, although we did not achieve the same results. Our higher testing accuracy may be due to data leakage or other implementation issues. With more information regarding the original source code, we would have a better understanding of where our model differs from what the researchers intended.

We performed the pre-processing step multiple times with different counting and processing methods to isolate and remedy the issue without success.

It was noted that the research paper was published early in 2022, and that it could be possible that version differences between libraries used around the time of the original publication and now may have netted some performance improvement. However, that alone does not explain the massive jump in performance, and it is likely that we did not correctly implement the engineered features. In retrospect, it might have been beneficial for us to train and test the model using the original features in the dataset to get an accurate baseline of the model's performance.

6 Source Code Repository

All source code for this project was written entirely by the project teammates Gomez, Hoag, and Pabo, and can be found at the following public GitHub repository.

<https://github.com/scotty-hoag/csci-567>

References

- [1] Riot Games, *League of Legends*, Accessed: Dec. 2024. [Online]. Available: <https://www.leagueoflegends.com>
- [2] Hitar-Garica et al. *Machine Learning Methods for Predicting League of Legends Game Outcome*, IEEE Transactions on Games, Vol. 15, No. 2, June 2023. pp. 171-181.
- [3] Scikit Learn. *GaussianNB*. [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.naive_bayes.GaussianNB.html
- [4] Makine et al. *Machine Learning Models on MOBA Gaming: League of Legends Winner Prediction*, Acta Infologica, 2023, pp. 139-151
- [5] Bahrololloomi1 et al. *E-Sports Player Performance Metrics for Predicting the Outcome of League of Legends Matches Considering Player Roles*, SN Computer Science, Vol 4, article number 238, 2023