League of Legends Ranked Games Win Analysis

Zijie (Eric) Mei

Background Context and Key Objective

League of Legends" (LoL) is a popular multiplayer online battle arena (MOBA) game developed and published by Riot Games. First released in 2009, it has since become one of the most played and widely recognized games in the world, particularly known for its competitive E-Sports scene.

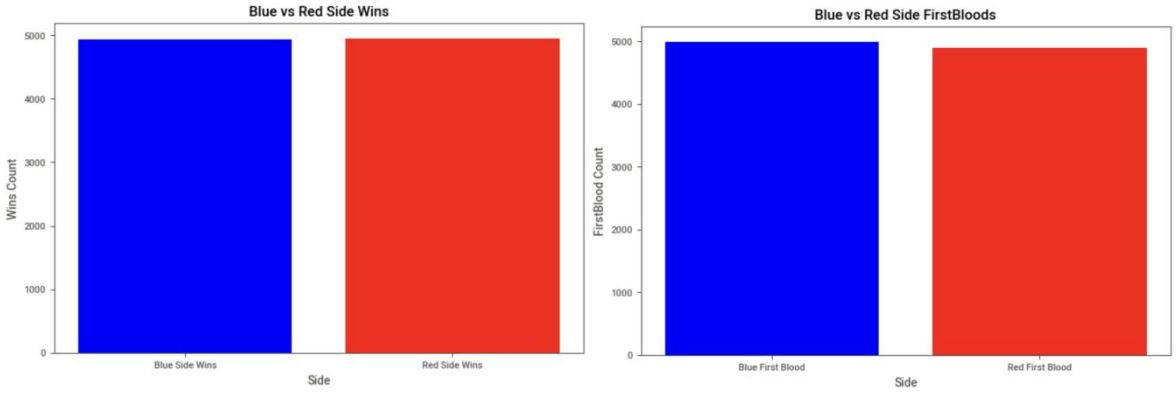
My key objective here to is use deep learning methods to build models that can predict winning chances for ranked players using in game stats as the game goes on. Furthermore, I can use results of the model to generate insights for players to improve their winning chances.

Data Understanding

My data is from Kaggle. It contains 9879 ranked games (Diamond 1 to Master Level) of League of Legend in game stats at the 10 minute time mark. There are 40 columns, one for game id, one for blue wins (This is our target), and the other 38 columns are divided into 19 columns for different in game stats for both blue and red teams.

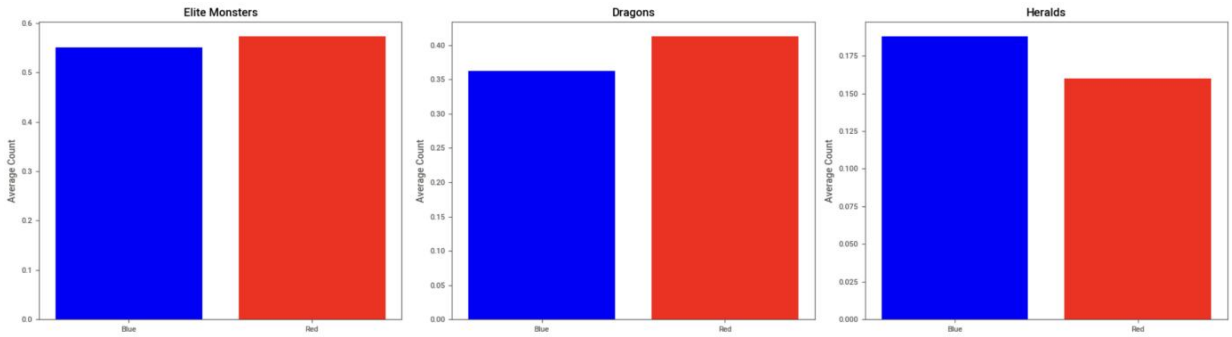
I did lots of exploratory data analysis (EDA) for the data, including comparing every single feature across both teams, unsupervised learning methods such as factor analysis and clustering, and I also came up with a correlation matrix to find interesting connections through columns.

For comparing features across teams, I found that most of the features are pretty even between both teams, such as win/loss, kills, deaths, assists, gold, experience, first bloods, etc. Here are some examples:



As you can see from the plots, among our data, blue and red side had pretty similar data across almost all features, which means our data set is pretty evenly distributed without any major biases.

However, there is one feature that is not evenly distributed between both teams, which is neutral objectives (dragons and heralds):

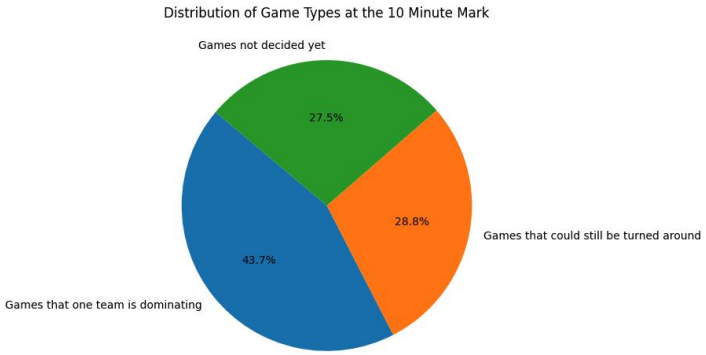


Red team had significant more dragons than blue team while blue team had significant more heralds than red team. Overall, red team had slightly more elite monsters than blue team.

This is a very interesting finding, and I suspect it is because of the map design, where the position of dragon is open and closer to red side while the position of herald is closer to blue side. And because dragons are easier to kill resulting what I

am seeing here. I am definitely going to pay more attention on these features when I am interpreting results from my models as I move forward.

Then, I used unsupervised techniques like factor analysis (PCA) and cluster analysis (K-means) to further explore our data trying to identify interesting trends. I compress the data set into 6 factors and 8 clusters, and discovered 3 main categories of games at the 10 minute mark:



Based on my discovery, there are 56.3% of the games that still could be won through the right effort at the 10 minute mark.

After that, I made a correlation matrix trying to discover more interesting correlations between columns, however, I did not discover anything too interesting, since I already have significant amount of domain knowledge, there is nothing new here in the correlation matrix.

Data Preparation

My ultimate goal here is to design a model that can predict winning chances of a team as the game goes one, and for that, I need sequence data, which means what I have from Kaggle is not enough. Before data preparation, I first generated sequence data for each game from 0 to 9 minute mark, combined it with the 10 minute mark data from Kaggle, and formed my sequence data. I used domain

knowledge I have from playing the game, and applied appropriate logic to each column when generating data. So now I have sequence data of 9879 ranked games from 0 to 10 minutes to work with.

Then I did basic data splitting, loading, and batching for both the original data and the newly generated sequence data. For data splitting, I divided both data sets into 80% training data, 10% validation data, and 10% testing data.

Modeling

My task is just a very simple binary classification task here, so I decided to use logistic regression here as my benchmark. Then I designed both a MLP model and a LSTM model for my original and sequence data.

- Logistic Regression:

I got 73.4% accuracy on the validation data and 72.9% on the testing data.

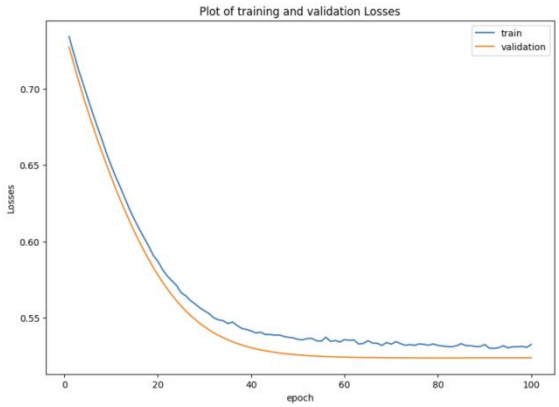
- MLP Model:

My approach here is to train the model with the 80% training data, while validating it on the 10% validation data, and in the end, test the performance on the testing data to make sure it does not over-fit.

I designed a class to build, train, and validate the model. I used BCE loss due to the fact that my task is binary classification, and used dropout and l2 regularization together to make sure my model does not over-fit. I also implemented the logic of saving the model whenever validation loss decreased to have the best performance model at hand while I train the mode.

For optimization, I first manually tuned hyper-parameters trying to figure out a general direction of the range of hyper-parameters. Then I used Bayesian

Optimization to run 50 trials on the range I decided earlier, trying to find the best hyper-parameter combination. And lastly, I adapted the hyper-parameters from Bayesian Optimization and tuned it further to achieve the best result possible. Here is a plot for the final training losses and validation losses over epochs:

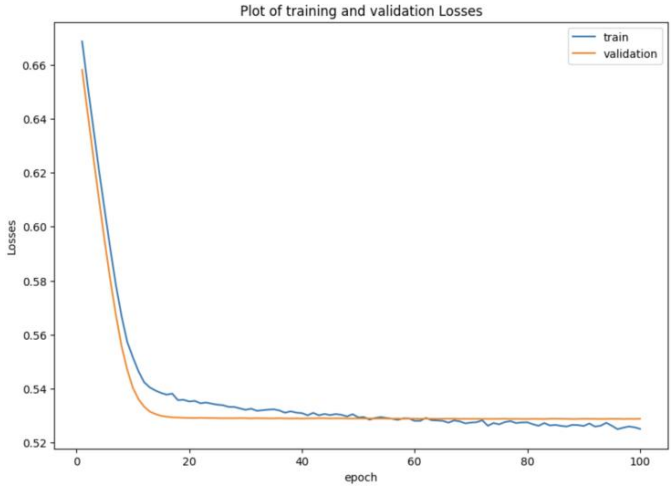


Validation loss stopped decreasing after 76th epoch and I saved the model at that epoch. I got a validation accuracy of 73.3% on the validation data and a testing accuracy of 73.1%, which is a 0.2% improvement from the benchmark of Logistic Regression.

- LSTM Model:

I used the same approach as what I did with the MLP model. I also used BCE loss as our loss function, and also implemented l2 regularization and dropout as techniques to prevent over-fitting, and also saved my model whenever there is a decrease in validation loss.

However, this time I did not use Bayesian Optimization due to the fact that I do not have enough time or computational power to do so. I ended up tuning hyper-parameters for my LSTM model manually to get the best performance out of it. Here is a plot for the final training losses and validation losses over epochs:



Validation loss stopped decreasing at 87th epoch and I saved our model at that epoch. I got a validation accuracy of 74% on the validation data and a testing accuracy of 73.3%, which is a 0.6% improvement on validation accuracy and a 0.5% improvement on testing accuracy from the benchmark of Logistic Regression.

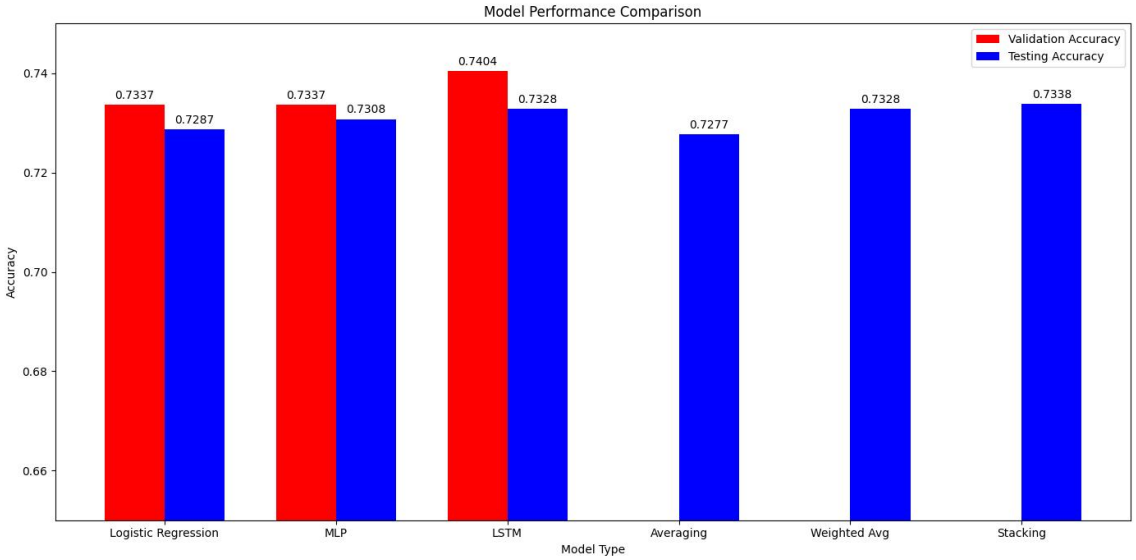
- Combining Models to Optimize Performance

I tried different methods to combine the results of my MLP model and LSTM model, including averaging the results, weighted averages, stacking (run a logistic regression on the results of the 2 models) and model blending.

Stacking really helped on improving performance and I got a 73.4% testing accuracy from it, which is a 0.1% increase. However, the sad part is I could not get model blending to work properly, the code was too complex for me. I am definitely going to try more in the future to figure this out.

- Model Performance Conclusion

After I finished training both models, I made a visualization to compare performance between my MLP and LSTM model with the benchmark Logistic Regression model:



As you can see from the plot, although both of my deep learning methods beat the benchmark, it was only by a tiny little bit. I can't say that I am not disappointed. I think that the reason behind is that I do not have enough data. Deep learning models are way too data hungry, and my data does not even contain 10 thousand rows, which could be the main reason of deep learning methods not performing significantly better. Another reason could be because that my task is a very simple binary classification task, and Logistic Regression really excels at this task.

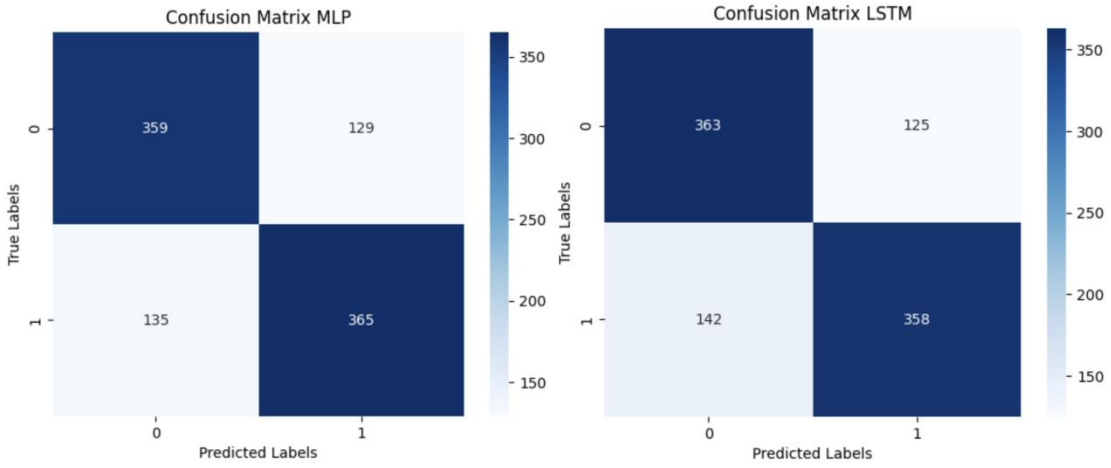
One other interesting finding I have here is that my LSTM model performed the best out of the 3 models, with the data I generated! This is very surprising to me, and I suspect the main reason behind is that I really applied my deep domain knowledge here to generate data so that the data I generated are not garbage. They could mimic what real data looks like!

Overall, combining results from the 2 models worked, but not that much, the reason behind could be they are based on very similar data.

Model Interpretation and Evaluation

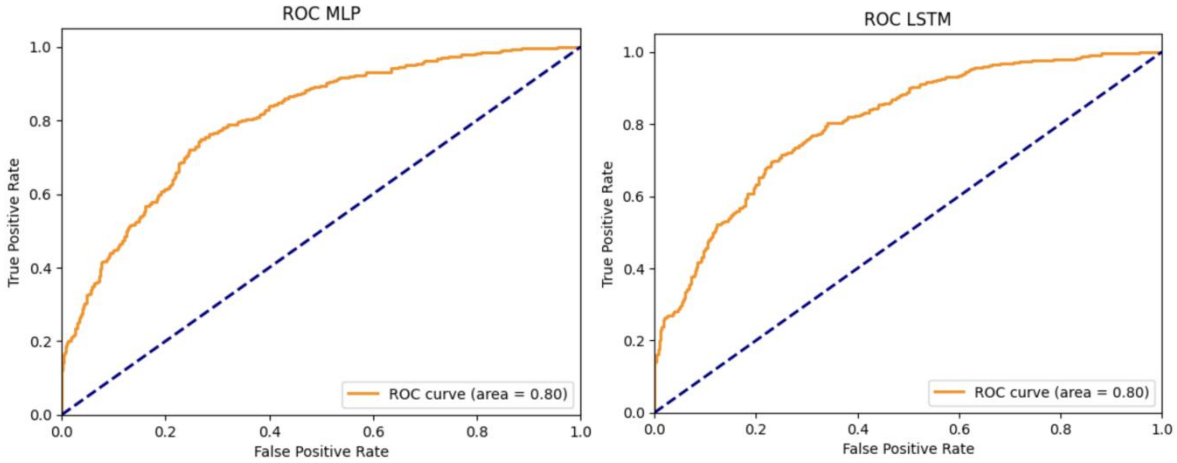
For this part, I came up with confusion matrix, ROC curves, and interpreted feature importance through permutation importance for both of our deep learning models.

- Confusion Matrix: I used testing data to calculate confusion matrix



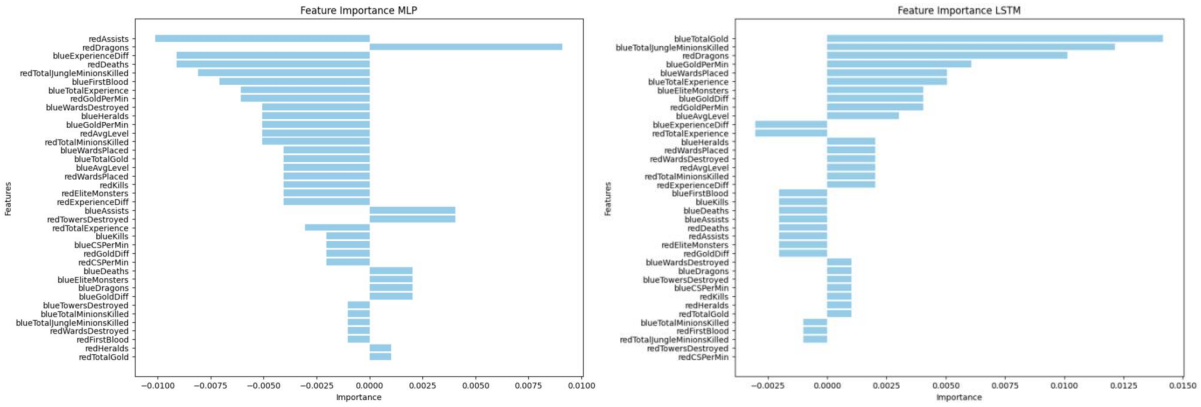
For my specific task, the cost of false positive and false negative is the same, so I do not need to come up with a cost/profit matrix here. As you can see from the plot, both models have pretty similar performance here.

- ROC Curve: I used testing data to come up with ROC curves as well



As you can see from the plots, both models have a 0.8 area under curve, which is not bad for the task.

- Feature Importance: It is really hard to interpret results from deep learning models, this is one of the major disadvantages for deep learning models. However, I managed to figure out a method called Permutation Importance to interpret feature importance for my models.



As you can see from the plots, the two models have very different feature importance. Other than the common ones such as gold, experience and kills. It is very interesting to see that for both models, dragons are on top of the list, which is a very surprising and interesting finding for me.

Recall the exploratory data analysis part in the beginning where I discovered elite monsters (includes dragons) kills are not the same for the 2 teams, and given my domain knowledge that players generally do not care about dragons and heralds that much early in game, this could be a huge discovery. This means that the elite monsters has a significant impact on the game winning even in the early stages of the game!

Deployment

My main objective of this project is to come up with a model that can predict winning chances for a ranked League of Legends player based on in game stats

when the game is on going, and with the help of generative AI, give live feedback to the player in game to help make better decisions.

I already trained a LSTM model here that can predict winning chances of a player based on sequence data, which means it can give live winning chance predictions when the game goes on. Unfortunately, I do not have enough knowledge right now on how to train a generative model that gives live feedback for the player.

With more knowledge and more data, I could eventually build a in-game plug in software that does the job. And I can charge a monthly premium fee for League of Legends players who uses my services, or Riot Games could form a department that develop a similar software that does the same thing.

And furthermore, with more training data and knowledge, I could eventually build a model that provides item suggestions, lane setup suggestions, ability suggestions, detailed death summaries, champion pick suggestions, and even post-game analysis.

With the right data, a model like this could act as a live commentary assistance for E-sport events as well, where there are already some AI use applications.

The only risk associated with the deployment of this model could be privacy. However, I do not see it as a major concern since all ranked and E-sport games data are public, and even there are potential privacy issues, company could make players sign a statement of acknowledgement of sharing stats.