

Predicting Match Outcome in League of Legends 5v5 Online Games

Pavel Petrukhin
petrukhp@tcd.ie
19328624

Cian Jinks
jinksc@tcd.ie
19334781

Ajchan Mamedov
mamedova@tcd.ie
19335023

Introduction

League of Legends ('LoL') is an online game where 2 teams of 5 players battle it out with the goal of destroying the other team's nexus first to win. Games last between 20-45 mins, with players choosing their preferred champion and position in each game. This project explores whether it is possible to predict the outcome of a 5v5 online competitive game of League of Legends based on prior known features/statistics about the players on each team. For example, the player's chosen champion, chosen position, competitive ranking, overall win rate, average kills in recent games, average deaths in recent games, etc.

This problem is particularly interesting because LoL is one of the largest competitive online multiplayer and esports games of all time [1]. Top players have salaries reaching millions per year [2], and there are millions in prizes given out for the top tournaments [3]. If it were possible to accurately predict the outcome of games purely based on player statistics, it would allow players and teams to draw statistical inferences such as what team champion compositions are good, how individual players are performing, whether a game is worth playing, etc.

We make use of several Machine Learning techniques and models for this project. These include Logistic Regression, Principal Component Analysis (PCA) + k-Nearest-Neighbours and Deep Learning. We also present our evaluation of the results and discuss our interpretation.

Data Gathering

To gather data for this project, we made use of the public developer API provided by Riot Games [4], the creator of League of Legends. This API contains endpoints that are specifically designed for retrieving data about matches and players called "MATCH-V5" and "SUMMONER-V4", respectively. We chose to only collect data based on matches which were played as part of the "Solo/Duo Ranked 5v5 Queue". This is the most popular "game mode" in the game and is also the most competitive, as players are playing on their own or with a maximum of one friend to try and improve their rank. The queue also implements banning measures to ensure players don't deliberately lose games or leave midway through. For these reasons, we believed matches from this queue would provide the best quality data for our models.

Given an initial player's summoner name (username), we could retrieve their most recent 10 matches data from the API and then recursively retrieve the same data for each of the 10 players in each match. To ensure that our match data encompassed games played at as wide a range of ranks as possible, we ran our data gathering script multiple times with initial players of all possible ranks. An example of the match data: returned by the API in JSON format can be found in Appendix A. Furthermore, to acquire the data we needed for each player in each match, we had to send 2 additional requests per player to other endpoints.

Data gathering was by far the most time-consuming part of our project. This was due to the rate limiting Riot Games imposes on basic development API keys. These API keys are only allowed to send 20 requests every 1 second and 100 requests every 2 minutes. They also expire every 24 hours, and the data retrieved is tied to the API key. This makes it impossible to start back up where you left off, gathering data with a new API key. The only way to increase these limits is to apply for a production API key, which requires a lengthy correspondence with Riot Games themselves. Based on these limitations, we calculated a rough estimate for how many data points we could gather in 24 hours with one API key:

For a given match we retrieve the match data and the player data for each of the 10 players in the match. This gives a total of $1 + 10 + 10 = 21$ requests per match. As we can only send 100 requests every 2 minutes, we can send 72000 requests per 24 hours which gives an upper bound of 3428 matches within 24 hours.

In the end, we gathered approximately 20845 matches, including the data for every player in each match. We gathered this data over the span of 48 hours by deploying our data gathering script to 8 Google Cloud Platform Compute Engine VMs [5] and giving each a unique initial player and unique API key generated by different developer accounts. We generated new API keys for the latter 24 hours and ran the script on the VMs again to ensure we covered initial players with all possible ranks.

Data Filtering

To ensure our models would be as accurate as possible we needed to filter out games which contained broken/incorrect data and games that were liable to skew our models from our dataset. In the end we filtered out the following:

- Games that are not on the current patch of LoL
 - New patches release every 2 weeks and affect the “meta” i.e different champions are better or worse from patch to patch
 - The patch a match was played on is contained in the API data
 - The current patch version at the time of this project was “12.22.479.5277”
- Games where a player leaves midway through
 - This majorly skews the result of the game and is not possible to represent in our models as we cannot predict if a player will leave or not
 - The API determines this for us and sets a flag in the match data
- Games where a player has deliberately attempted to lose the game
 - Similar to players leaving, the result of such a game will be skewed and we cannot predict when this could happen
 - To identify such games we set a death threshold of 20 as it is exceedingly rare to reach this number of deaths in one game unless a player is trying to lose

We also considered attempting to filter out games where a player who is very good makes a new account and plays at a lower rank (commonly referred to as a “smurf”). However, if a player is a smurf, it is likely to be reflected in their statistics as they will show up as a very good player with a high win rate and good average kills, deaths and assists. Therefore, they won’t skew the model, as better stats should mean they are more likely to win.

After data filtering, we were left with 18067 matches. The majority of the 2808 filtered-out games were played on an older patch (2213), with the remainder containing players with > 20 deaths (101), players leaving midway through (451) or other invalid data of any kind relating to our chosen features (43).

Feature Selection

To select features from our match data, we made use of our own prior knowledge about the game from many hundreds of hours of playtime. For each player in a match, we considered the following 10 features as having the most impact on the outcome:

- | | |
|--|--|
| • Champion | • Losses (this year in solo/duo ranked queue) |
| • Tier | • Average Kills (for their past 10 games) |
| • Rank | • Average Deaths (for their past 10 games) |
| • LP | • Average Assists (for their past 10 games) |
| • Wins (this year in solo/duo ranked queue) | • Champion Mastery |

Feature Mapping & Explanation

Some of the above features are not raw number data and so we needed to map them to numbers.

Tier, Rank and LP - all combine to represent a player’s competitive ranking. Their tier is one of Iron, Bronze, Silver, Gold, Platinum, Diamond, Master, Grandmaster or Challenger, their rank is one of 1, 2, 3 or 4, and their LP is a number from 0 to 100. For example, a player might have the competitive ranking of Diamond 1 - 32 LP. To encode this as numeric features, we mapped the tier to a number going from 1 to 9 for Iron to Challenger and 0 for unranked players. If a player has not received a rank yet, they are Unranked 0 - 0 LP.

The player’s chosen champion is already returned as a number by the API. There are 162 champions in the league and so the number goes from 1 to 162. Champion Mastery is a number used frequently throughout the game to represent how proficient a player is at a given champion and so we believed it a good choice of a feature on top of the other more obvious ones.

Representing Teams

In each match there are 2 teams of 5 players, blue and red. The difference between the blue and red teams is important to represent as they play on different sides of the map which may affect the game outcome as the map is not identical on both sides. To represent this difference we considered our features array to consist of 100 features, the first 50 features for the players on the blue team and the latter 50 for the players on the red team.

Each player also has a chosen position which is one of “TOP”, “JUNGLE”, “MID”, “ADC” and “SUPPORT”. Each team must have one player in each position, meaning no two players can play the same position on a team. The position is important as it influences a player’s chosen champion and also their kills, deaths and assists. For example, a support player usually has a low number of kills but a high number of assists.

To encode position in the features array, we divided the 50 features for the blue team into sets of 10 for each player. We used the first 10 features as the blue team’s top player, the second 10 for the blue team’s jungle player, the third 10 for the blue team’s mid player, etc. We did the same for the red team’s 50 features.

Models

Please note that we did not plot predictions vs actual data because we have a large number of features in each of the models. Therefore, there is no intuitive way in which we can represent predictions vs test labels that would enable us to do useful qualitative analysis.

Baseline

A baseline model is essential for evaluating the performance of other machine learning models. We implemented two baseline models - “most frequent” and “total champion mastery”.

Most frequent simply predicts the most frequent label for every data point. In our data, the red team had 9051 wins vs the blue team’s 9016. This means our most frequent model had an accuracy of around 50%. This is exactly as we expected as it is commonly believed by players that both red and blue teams are balanced in LoL (they have equal chances of winning). Therefore, the most frequent baseline is equivalent to the random baseline for our model in terms of accuracy.

Total champion mastery is a custom baseline we created that sums the champion mastery of both teams and compares them. Whichever team has the larger total champion mastery is labeled the winner. This model produced an accuracy of 54% on the data. As we discovered in the next section after fitting the logistic regression model, champion mastery appears to have no effect on the outcome of the match. This implies that this baseline is quite poor, even though one would expect it to perform reasonably well.

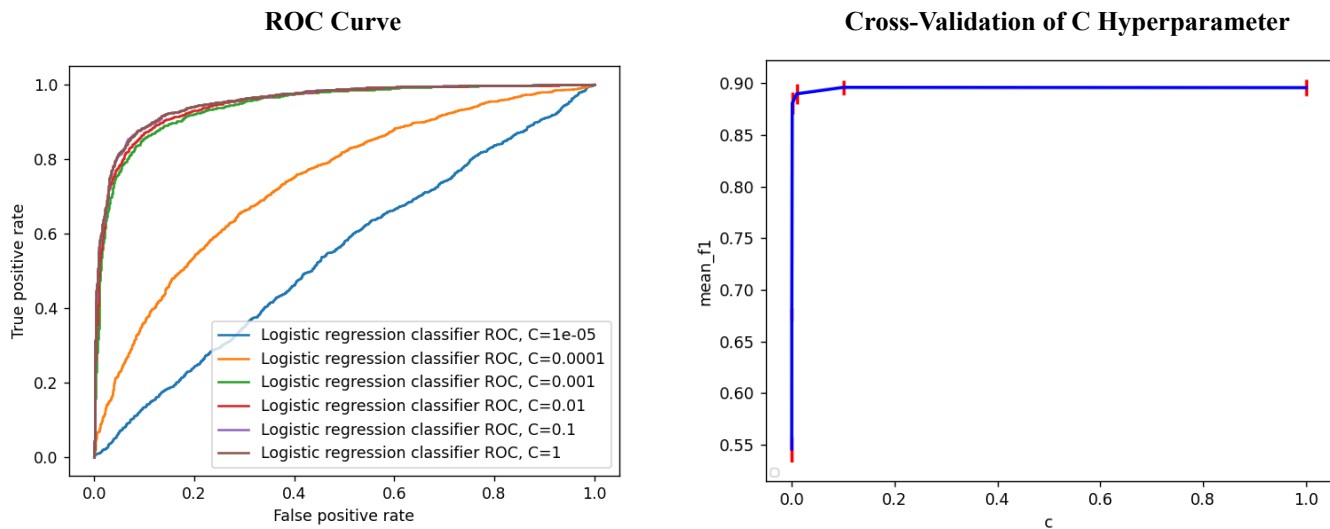
Logistic Regression

We applied a logistic regression with an L1 penalty and all of the 100 original features to our match data. Logistic regression is a simple linear model which is used for binary classification. Logistic regression assumes that the data is linearly separable and hence we can use linear regression to model the decision boundary. In order to compute the classification, we need to apply a sigmoid function on the linear regression result to get the logistic probability score of the observations belonging to the positive class.

We decided not to use polynomial features with our model because even if the maximum features power is 2, we get 5050 extra features in addition to the 100 we have already. This makes the model incredibly time-consuming to train. Moreover, it will likely lead to overfitting because we only have about ~20000 data points. We used an L1 penalty to encourage finding a sparse solution for our model because we have 100 features and it is likely some features may carry little to no weight. To choose a C value for the L1 penalty, we used 5-fold cross-validation. We used a split of 20% for the testing dataset and 80% for the training dataset.

The first step was finding the C value (L1 penalty factor) such that the model turns into a baseline predictor. We identified that $C=1e-5$ makes the model’s ROC curve mimic the $tpr = fpr$ line, which corresponds to the random baseline classifier. We considered this reasonable because all of the model coefficients were zero apart from the “Champion Mastery Weight” which had an extremely small non-zero value. This roughly corresponds to the most frequent classifier, but since we have found that the classes are of nearly equal size it also corresponds to the random classifier.

Below you can see the ROC curves for different values of C and the cross-validation of C values and f1 scores:



It seems that $C = 0.1$ yields the best model, because its ROC curve (purple colour) resembles the perfect square ROC curve the most. Additionally, we can see in the cross-validation plot that the same model achieves around 0.9 mean f1 score with a very small standard error.

Evaluation and Discussion

Below you can see the confusion matrix and computed precision, recall, f1-scores and accuracy for one of the 5-fold executions of the $C = 0.1$ model (20% unseen test data, approx. 3600 data points):

	Predicted negative	Predicted positive
Known to be negative	1609	179
Known to be positive	209	1617

Please note that in the below image 100 and 200 are the labels that identify the 2 opposing teams (blue and red respectively).

	precision	recall	f1-score	support
100	0.89	0.90	0.89	1788
200	0.90	0.89	0.89	1826
accuracy			0.89	3614
macro avg	0.89	0.89	0.89	3614
weighted avg	0.89	0.89	0.89	3614

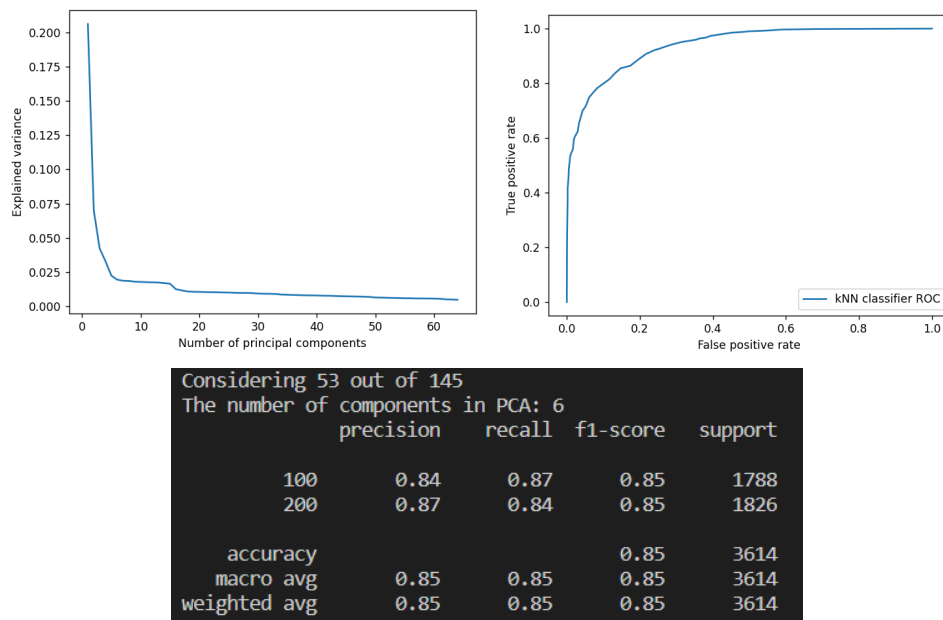
We are not going to list all of the coefficients of the model here, but it is interesting to discuss which features turned out to have more weighting than others. We found that average kills, deaths and assists have a large impact on the prediction. Additionally, the coefficients for the blue and red teams have different signs for these features. This is reasonable because we are trying to predict whether one of the teams beats the other, so naturally, the kills for one team should have an opposite impact on the prediction compared to the kills for the other team. Another observation we made is that “Tier” has the largest weight. Finally, “Rank” is quite important too, while “Wins”/“Losses” have low impact and “Champion Mastery Weight”/“Champion Weight” have practically no impact. An example of the coefficients for one of the players can be found in Appendix B.

PCA + kNN

We used principal component analysis (PCA) to decrease the dimensionality of the data and then perform k-Nearest-Neighbors (kNN) classification over the PCA space. PCA is a technique that computes eigenvectors of the covariance matrix, which are called principal components. Each eigenvector represents a general pattern (linear combination of features) in the data and the larger its corresponding eigenvalue, the more information it carries. PCA can be used for reducing dimensionality if only the first few vectors are taken into account (in decreasing order of their eigenvalue). The hyperparameter of PCA is the number of principal components we should take which corresponds to the proportion of variance we are going to account for. We decided to use kNN in the PCA space because it is a

combination of techniques used across various fields. kNN is a classification technique which labels the new observation based on the labels that its k closest data points have in the training set.

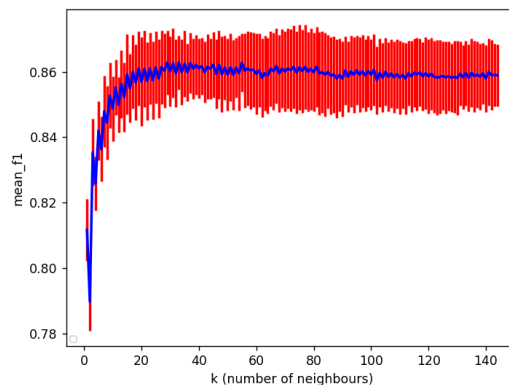
Below you can see a graph representing how the extra percentage of the explained variance changes as the number of components grows. It is a good idea to look for an “elbow” in such a plot, so we chose 6 as the number of principal components. Once we have projected the datapoints into 6-dimensional PCA space, we performed cross-validation for the k (number of neighbours) parameter in kNN. The range for cross-validation was chosen as $[1, \sqrt{\text{number of datapoints}} + 1]$, which is a common practice. Using 5-fold cross-validation with an 80% to 20% split between training and testing data, we found that $k = 53$ had the best average f1 score. You can see the ROC curve for $k = 53$ below. It is quite close to a square, which is a perfect classifier, so we can conclude that the model performs well. Additionally, you can see the classification report for the same model showing the precision, recall, f1-score and accuracy. We can see that it has an accuracy of 0.85 on the 3614 previously unseen data points (the testing data).



Here is the confusion matrix for the model described above:

	Predicted negative	Predicted positive
Known to be negative	1553	235
Known to be positive	299	1527

Here is the cross-validation plot which shows different values of k plotted against the mean f1 score (blue) and its standard error (red):



We can see that all values of k above 20 have a relatively similar f1-score, so we could have chosen a value of around 25 instead of going for the maximum at 53. We would still get very good results while keeping the model simpler.

We compare this model’s results against the previous Logistic Regression results in the “Summary” section of the report.

Deep Learning

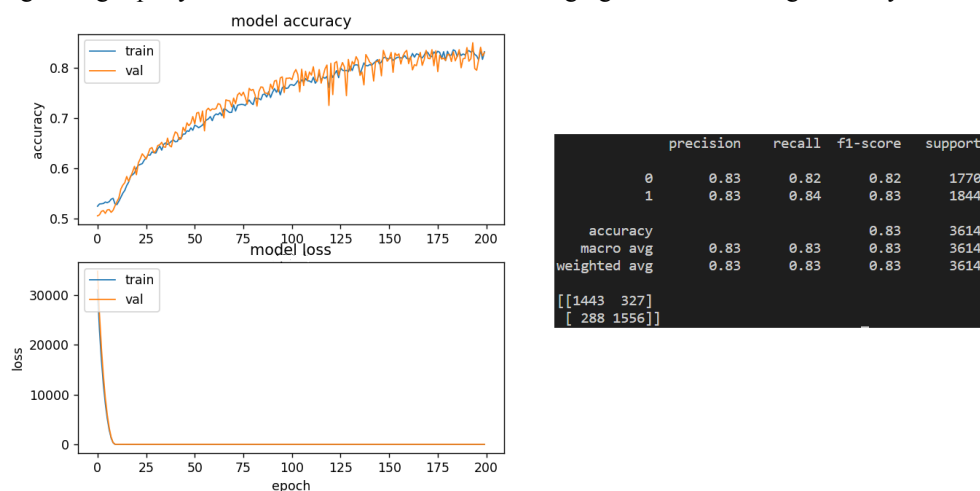
The last model we used was deep learning. The architecture of our model is detailed in Appendix C. The idea behind the architecture is that we first consider players' features individually and come up with 2 new values for each player based on the original 10. After that, the 20 features (2 features * 10 players) are grouped together and sent to a fully connected layer with a sigmoid activation function which performs the classification.

We chose the batch size of 128 so that the gradient descent algorithm converges more smoothly and we are not overfitting the data due to an excessively large batch size. We are using a small number of neurons to make sure that we have enough data to train the model properly. Additionally, we normalize the data before feeding it into the neural network in order to eliminate the issues coming from features of different scales such as certain weights being updated / penalized by a larger amount than others. We use L2 regularization on the last sigmoid layer to ensure that we do not overfit the data. Unfortunately, we could not perform cross-validation for the model as it takes a significantly long time to train. As a result, we chose a value of 0.001 for the penalty coefficient because it yielded reasonable results in our testing. We used a standard 80% to 20% split without 5-fold, because once again it was too computationally intensive to run cross-validation for the parameters. The adam optimizer was used and we tried different learning parameters to see which one gave the best results in terms of stability and convergence to a useful model. We chose 0.00001 as the learning rate. We used the sigmoid activation function in the last dense layer because our problem is binary classification. We use ReLU as the activation function for all other neurons. The final layer produces a probability of belonging to the positive class, so our final prediction is determined by the condition (probability > 0.5).

We used 10000 epochs because the learning rate is quite small and because a smaller number of epochs did not lead to a situation where the performance on the held-out validation data stopped improving significantly or started to deteriorate.

We also tried using other architectures for neural networks such as single fully connected layers or several fully connected layers with a decreasing number of neurons in each one of them. However, most of the time, the optimization algorithm converged on local minima which did not yield any good results in terms of accuracy.

Below you can see the history plots for the model, please note that each 50th epoch was subsampled, so that the graphs are more readable, hence there are $10000 / 50 = 200$ data points shown on each of the graphs. Please note that the confusion matrix is shown below the classification report in the image on the right hand side, it has the same structure as the matrices mentioned before. We can see that the model has an accuracy of 0.83 on 3614 previously unseen data points. The model needed around 10000 epochs for the accuracy to stop growing. In the model accuracy graph below, we can see that we reach a point of optimal fit because the validation accuracy is no longer growing rapidly and at the same time it is not diverging from the training accuracy.




Summary

All of the proposed models significantly outperformed the baseline models (around 0.5 accuracy) described above. Please note that we considered it valid to compare accuracies because the classes are balanced and hence accuracy is not a misleading metric. The logistic regression had the best performance on the test data (0.89 accuracy), while kNN had a slightly smaller accuracy of 0.85, but still performed very well. We also have to consider that the logistic regression uses 100 features, while kNN only uses 6 features which come from the principal component analysis. Hence, PCA + kNN is a more efficient approach in terms of compute power. Finally, the deep neural network also achieved good results (0.83 accuracy), even though its performance was slightly weaker than that of the logistic regression and PCA + kNN. We hypothesize that we did not have enough data and computing power to train a large enough

neural network that is able to beat the simpler models. Overall, the project was a great learning experience for each of us and we are very happy with how it turned out.

Contributions

- Pavel Petrukhin
 - Implemented rate limiter for gathering api data
 - Wrote code during pair programming sessions for data scraping
 - Set up a Google Cloud cluster of 8 virtual machines to fetch data from the Riot API
 - Primarily worked on the deep learning model, but also made contributions to logistic regression and PCA + kNN such as setting up 5-fold and cross validation
 - Wrote most of the “Models” section of the report
 - Signed *Pavel Petrukhin* (02/12/2022)
- Cian Jinks
 - Implemented initial data scraping/gathering script and pair programmed to finish it
 - Implemented reading in the processed and filtered data into feature arrays
 - Implemented the baseline models and logistic regression model
 - Wrote “Introduction”, “Data Gathering”, “Data Filtering” and “Feature Selection” sections of the report
 - Signed *Cian Jinks* (02/12/2022)
- Ajchan Mamedov
 - Primarily worked of the PCA + kNN model, but also made contributions to deep learning model
 - Implemented filtering for the matches
 - Helped with data scraping during pair programming
 - Helped with setting up cluster on Google Cloud
 - Contributed to project report
 - Signed  (02/12/2022)

Github

- <https://github.com/cianjinks/ML-Project>

References

1. [Riot Games Fact Sheets](#)
2. [Top Esports Players by Earnings, 2022](#)
3. [LoL World Championships Prize Pools, 2022](#)
4. [Riot Games Developer API](#)
5. [Google Cloud Platform - Compute Engine](#)

Appendix

Appendix A

An example of the JSON data returned by the Riot API for a single match. This example only shows the data for the first player, as otherwise, it would have taken 40 pages of this project report. Note the player IDs are based on the API key used to retrieve this data and are, therefore, no longer valid.

```
{
  "metadata":{
    "dataVersion":"2",
    "matchId":"EUW1_5460125905",
    "participants":[
      "L_wBk5aKdFIjn7MR2jppq6-86IFIoJwQ5Kbw5yv2E1eJzJhAlp0ZE13pk1MTmny08Q0798wN9FhFiGQ",
      "Tsc9F_AVNJ9aHK3P00B89fibyVmg_A7_mipzbFYejpUTnmF2S-Z-w_h36nDKCxeKa6pHi78aBDY2A",
      "3ppqqEki700-245KNkzext38in3l5zHE2p8dbIir9rCjq4Jp051rq-ZZbzotgShY97XF-luXSQ_Eg",
      "1BF7QSDlQNGe1D-MJNeIQAkou466rkOMXNsvteyThG0VQ1NfiRSSDY0qhzcyXhr-0iJ2fX0zPQD8yg",
      "ZksB4uQcKJHgqiXmYU3k1FaRu2YiXcSrW7Qe7vtONiyM342PLQIQJm5Api-jP4gF-QvdaIaCeqD5w",
      "I_X8Kwai76X9vOKlkf1V2Z0jUgU3dbnFg3YJfsHtUxZ60aLbCbFumx0ItB2jL5k09ZFpmEhsII5HQw",
      "5HDr8PtZq_EM3wAj3bAvL-i-m1sw67bG2CqIBOVjQvEf7syEJ_Q5bhqTWRuKeZ0GjeoQ0fb-AWrUnQ",
      "OYgh5CNm2HtjQP102p56DeS_c1m7W--edIAObdvyVbprcrk-s60dvsK4e40CC6GPcDHVzJ3NBmQomg",
      "5-WfDK9PC6S0qST-0wQ6TqtXXittddmmJgckXyX936InMhsNaG0PEbgTMpTid-UL8N7pLw7sxUe3yig",
      "IHJWoi2ihFBnztDpjc_nReDSCX-0imVzemxEr9puCl03QF0l6sEhML__Qaq4DLzE8-4TL5vvpEVWfg"
    ]
  },
  "info":{
    "gameCreation":1631592396000,
    "gameDuration":1907499,
    "gameId":5460125905,
    "gameMode":"CLASSIC",
    "gameName":"teambuilder-match-5460125905",
    "gameStartTimestamp":1631592441869,
    "gameType":"MATCHED_GAME",
    "gameVersion":"11.18.395.7538",
    "mapId":11,
    "participants":[
      {
        "assists":6,
        "baronKills":0,
        "bountyLevel":0,
        "champExperience":15836,
        "champlLevel":16,
        "championId":80,
        "championName":"Pantheon",
        "championTransform":0,
        "consumablesPurchased":0,
        "damageDealtToBuildings":2977,
        "damageDealtToObjectives":7237,
        "damageDealtToTurrets":2977,
        "damageSelfMitigated":21750,
        "deaths":7,
        "detectorWardsPlaced":0,
        "doubleKills":1,
        "dragonKills":0,
        "firstBloodAssist":false,
        "firstBloodKill":false,
        "firstTowerAssist":false,
        "firstTowerKill":false,
        "gameEndedInEarlySurrender":false,
        "gameEndedInSurrender":false,

```



```
"goldEarned":12582,
"goldSpent":12400,
"individualPosition":"TOP",
"inhibitorKills":0,
"inhibitorTakedowns":0,
"inhibitorsLost":1,
"item0":6692,
"item1":1028,
"item2":6676,
"item3":3047,
"item4":6333,
"item5":3133,
"item6":3340,
"itemsPurchased":18,
"killingSprees":3,
"kills":9,
"lane":"TOP",
"largestCriticalStrike":493,
"largestKillingSpree":3,
"largestMultiKill":2,
"longestTimeSpentLiving":543,
"magicDamageDealt":3514,
"magicDamageDealtToChampions":615,
"magicDamageTaken":5034,
"neutralMinionsKilled":5,
"nexusKills":0,
"nexusLost":1,
"nexusTakedowns":0,
"objectivesStolen":1,
"objectivesStolenAssists":0,
"participantId":1,
"pentaKills":0,
"perks":{
  "statPerks":{
    "defense":5001,
    "flex":5008,
    "offense":5005
  },
  "styles":[
    {
      "description":"primaryStyle",
      "selections":[
        {
          "perk":8010,
          "var1":576,
          "var2":0,
          "var3":0
        },
        {
          "perk":9111,
          "var1":1519,
          "var2":300,
          "var3":0
        },
        {
          "perk":9105,
          "var1":15,
          "var2":30,
          "var3":0
        }
      ]
    }
  ]
}
```

```

        },
        {
            "perk":8014,
            "var1":654,
            "var2":0,
            "var3":0
        }
    ],
    "style":8000
},
{
    "description":"subStyle",
    "selections":[
        {
            "perk":8139,
            "var1":1105,
            "var2":0,
            "var3":0
        },
        {
            "perk":8135,
            "var1":1302,
            "var2":3,
            "var3":0
        }
    ],
    "style":8100
}
]
},
"physicalDamageDealt":115657,
"physicalDamageDealtToChampions":24333,
"physicalDamageTaken":15317,
"profileIcon":4923,
"puuid":"L_wBk5aKdFIjn7MR2jpbq6-86IfIoJwQ5Kbw5yv2E1eJzJhAlpOZE13pk1MTmny08QO798wN9FhFiGQ",
"quadraKills":0,
"riotIdName":"",
"riotIdTagline":"",
"role":"SOLO",
"sightWardsBoughtInGame":0,
"spell1Casts":116,
"spell2Casts":25,
"spell3Casts":21,
"spell4Casts":5,
"summoner1Casts":3,
"summoner1Id":4,
"summoner2Casts":2,
"summoner2Id":12,
"summonerId":"ptr9_DiP6KCchkJse_qhY663kICc3V5o1frNadxMfpCIDE_b",
"summonerLevel":123,
"summonerName":"MafDelous",
"teamEarlySurrendered":false,
"teamId":100,
"teamPosition":"TOP",
"timeCCingOthers":25,
"timePlayed":1899,
"totalDamageDealt":137523,
"totalDamageDealtToChampions":25088,
"totalDamageShieldedOnTeammates":0,

```

```
    "totalDamageTaken":24832,  
    "totalHeal":4064,  
    "totalHealsOnTeammates":0,  
    "totalMinionsKilled":155,  
    "totalTimeCCDealt":60,  
    "totalTimeSpentDead":220,  
    "totalUnitsHealed":1,  
    "tripleKills":0,  
    "trueDamageDealt":18351,  
    "trueDamageDealtToChampions":140,  
    "trueDamageTaken":4480,  
    "turretKills":2,  
    "turretTakedowns":2,  
    "turretsLost":10,  
    "unrealKills":0,  
    "visionScore":12,  
    "visionWardsBoughtInGame":0,  
    "wardsKilled":0,  
    "wardsPlaced":7,  
    "win":false  
  },  
  ...
```

Appendix B

An example of the coefficients for a single player's features produced from training our Logistic Regression model.

```
Blue Team - Top Player:  
Champion Weight: -9.541276368742259e-05  
Tier Weight: -0.5709467984179142  
Rank Weight: 0.23668144221632453  
LP Weight: -0.0033255246710575358  
Wins Weight: -0.0030453355511289005  
Losses Weight: 0.003330085376173341  
Average Kills Weight: -0.14452519121058174  
Average Deaths Weight: 0.16336429489150733  
Average Assists Weight: -0.07562999294693391  
Champion Mastery Weight: -3.4809346481437244e-08
```

Appendix C

The complete architecture of our Deep Learning model (sideways).

