# Ranked: a study of Machine Learning and League

By Lamonte Brooks

In this project, I attempt to use machine learning algorithms to predict the outcome of a ranked match before it begins. My dataset consisted of 70 features that I deemed likely to have an effect on the match's result, collected as public information available to the player in champion select. With this information, I was able to predict a ranked games' result with an accuracy of 60%. I then tested my model's accuracy on subsets of features to discover important factors impacting the game. I found that the ad carry role has the least impact on determining the winner of a match, with the top role being the most inconsistent. I also found that the champion's prowess in the meta does not have as important a role as a player's effectiveness on the champion.

First some background: League of Legends is a multiplayer online battle arena (MOBA) video game developed and released by Riot Games in 2009. In the standard mode, ten players queue up to play a 5v5 match, with five roles per team and each player selecting their own character (champion). Since its inception, League of Legends has grown into a massive international eSport, with competitive leagues in multiple continents and tens of millions of players and viewers across the world. As hundreds of millions of dollars being invested into the North American League this year, all eyes are on League of Legends, representing the future of eSports. It is the most played PC game in the world. I play it quite heavily and enjoy it a lot, so I felt that results from this project would be more meaningful to me.

So how does League work? In League of Legends, players can queue up to play five versus five competitive ranked games. Each player is assigned one of five roles, and can choose their own champion. After every champion has been selected, there is a brief thirty second delay before the game begins, where one can leave the game to avoid playing. Although the system tries its best to create even games, once champions have been selected by all ten players, a common strategy is to look up statistical data on the players and champions to determine how winnable a game is. Thus, applying machine learning models on statistical features can help us discover patterns between such data and ranked game outcomes, allowing one to avoid playing a heavily unfavored match and loss of rank.

To get started, I generated a data from the public APIs of Riot Games and Champion.gg, a website for champion statistics. The data consists of 14 features corresponding to each of the 5 roles in League of Legends, for a total of 70 features. These features are derived from the specific champion choice of each role and the player statistics in relation to that champion, which I considered impactful on game results. My data was gathered through crawling through recent ranked matches and gathering data on each match deemed recent enough, generating one sample for each team. The reasoning behind this is that some features I seek are not shown in the match data, and must be found through lookup of player data. Thus, to stay accurate, I only look for

very recent matches so my player data is synchronized with the match data. I generated a total of 3400 working training samples to work with.

There were potential problems with some samples, as Riot's algorithm for determining the role of a player in their API is not very accurate. To account for this, matches that did not contain all five standard roles were discarded. Furthermore, in some cases, a champion would be played in a role where not enough total games exist to accurately determine its statistics. The data values for the features of this champion would then be NaN in the data, which I accounted for by setting all NaN values to the mean value of their respective feature. I also tested considering only those samples which contained no NaN values in my model selection.

Here are the Role-specific Features and their descriptions:

| Features (5 per sample) | Descriptions |
|---|---|
| **champion_winrate*** | Average win rate of the champion |
| **champion_playrate*** | Play rate of the champion |
| **champion_percentRolePlayed*** | Play rate of the champion specific to its role |
| **champion_overallPerformanceScore*** | Champion.gg's champion performance score |
| **champion_matchupWinrate*** | Average win rate of the champion in role matchup |
| **wins_last2** | Player win count in the two most recent games |
| **wins_last15** | Player win count in the 15 last ranked games |
| **gamesPlayed** | Player game count on champion in last 300 games |
| **gamesPlayedRanked** | Player's champ. game count last 300 rank. games |
| **championPoints** | Player champion mastery points |
| **summonerLevel** | Player summoner level |
| **rank** | Player rank, calculated numerically |
| **lanePlayed** | Player game count on lane in last 300 games |
| **lanePlayedRanked** | Player lane game count in last 300 ranked games |

(*Tier and role specific)

Besides modeling ranked match outcomes, I was also interested in determining the importance of certain sets of features and their impact on the game. As such, I would train my models on only the set of features I was interested in, compare their performances. My motivation is to find out which roles have the most impact on the game, the importance of choosing high win rate and popular champions, and the significance of the player's skill in deciding the outcome of a game.

I leveraged my data on support vector machines, gradient boosting, deep neural networks, random forests, and logistic regression algorithms for binary classification of ranked game outcomes (win/loss). For each model, I performed grid search to find the optimal

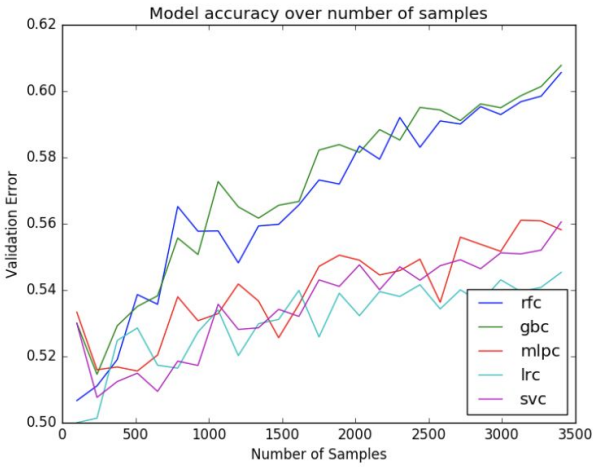
hyper-parameters, using a randomized 70% of my data as training, with 30% for validation. These are the results after averaging out NaN values.

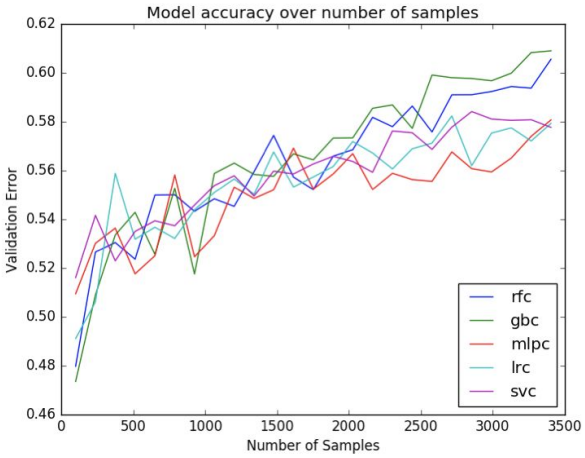
| | SVM | Neural Net | Random Forests | Gradient Boosting | Logistic Regression |
|---|---|---|---|---|---|
| Sensitivity | 56.75% | 56.04% | 62.86% | 63.90% | 55.21% |
| Specificity | 53.57% | 55.76% | 57.70% | 57.81% | 52.46% |
| Validation | 55.14% | 55.91% | 60.24% | 60.83% | 53.80% |

These are the results after removing samples which contained NaN values (very big loss of data and resulting likely overfitting)

| | SVM | Neural Net | Random Forests | Gradient Boosting | Logistic Regression |
|---|---|---|---|---|---|
| Sensitivity | 60.56% | 58.26% | 62.45% | 63.28% | 59.95% |
| Specificity | 56.91% | 56.73% | 57.94% | 58.63% | 56.52% |
| Validation | 58.72% | 57.50% | 60.14% | 60.93% | 58.19% |

In order to determine if I have enough data to avoid underfitting, I plotted each models' average validation accuracy over random subsamples of my data of increasing size. These are the resulting plots:

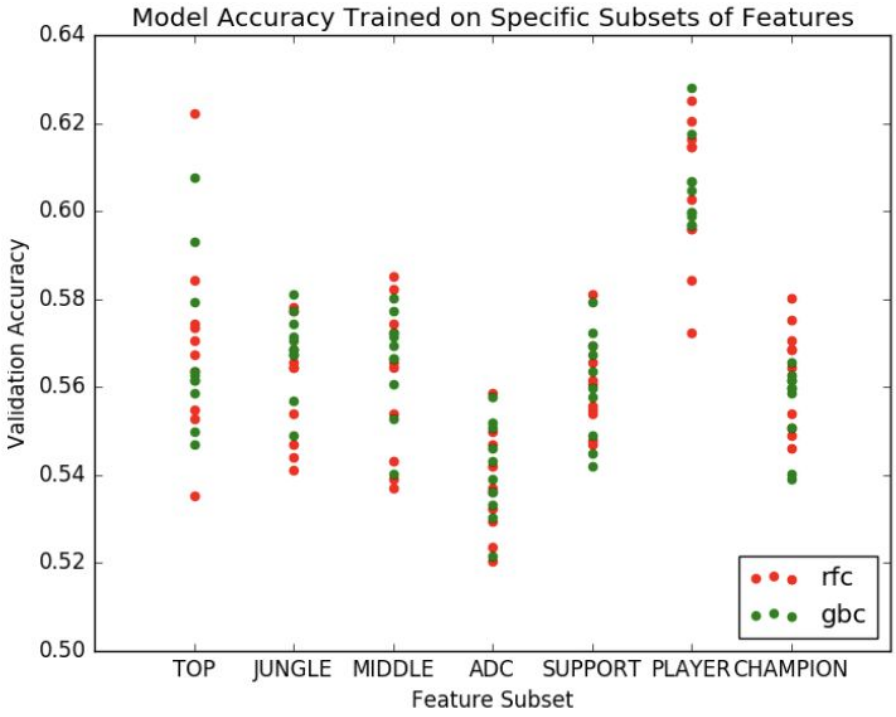



While SVM, neural nets, and logistic regression improve massively from training on non-NaN samples, gradient boosting and random forests see almost no difference in effectiveness. Even with the increase in accuracy, the other models' performance does not match that of gradient boosting and random forests. It is also interesting to note that each model performed better on identifying wins rather than losses. I think this is due to my receiving a dataset with around 50% wins and 50% losses (The ideal League average), my models may have a higher tolerance to predicting wins. From the tables, I saw that all of my models were

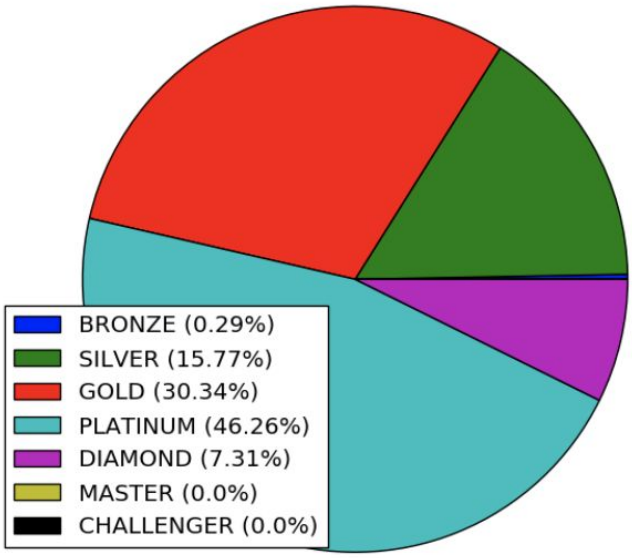
plateauing in performance around 2000 samples, so my dataset size of 3400 samples is sizable enough to get an accurate representation of the overall sample space. Additionally, the results show that random forests and gradient boosting models outperform the other models for both NaN replacement schemes. Note the performance improvement of neural net, SVM, and logistics regression models on samples not containing NaN. All NaN values pertained to champion-specific features, so perhaps the three models with improvement are much more sensitive for those features than random forests and gradient boosting. This is highly indicative of higher overfitting risk upon removing NaN data. Random forests and gradient boosting performed around the same for both schemes, however. Due to this, I decided to only use those two algorithms to continue and observe feature importance; this allows me to retain the most data while not costing as much accuracy.

To continue, I trained the random forests and gradient boosting classifier on specific subsets of features. I ran twenty samples for each subset, with ten for each model. Here are the results:




As I can see from my table, my models seem to work best when trained only on specific player information, doing significantly better than if trained only on champion specific information. This implies that overall champion strength in the meta is not as important to a match as a player's experience playing on said champion. Furthermore, I can see that training on the top role only has the most variation in performance, whereas training on the ad carry role only has significantly worse performance. This implies that the ad carry role has much less impact on a game as the other roles, with top lane having the most variation in effectiveness.

These results are surprising but do appear consistent with how League games typically go. However this could still have bias. As I crawled matches through a platinum-ranked account as a seed, my data may have a bias for ranks near platinum. Additionally, from this chart summarizing the data I received:




It can be seen that the majority of my data is from platinum and gold, with almost no data on bronze, master, and challenger. Because of this, the conclusions of this study are limited to the aforementioned ranks. All of the data for this study was gathered from matches near the end of the preseason, which is a time before big changes happen to the game; due to this, this also is not representative of the current state of League as I'm sure if these models were run again in a month once changes for the new season were, biases would be revealed in the classifications. Furthermore, as I sampled unevenly from different ranked tiers, I could get a more holistic view by sampling evenly from each tier.

I enjoyed this project and was pleasantly surprised by the results as I found out that a) you can classify a likelihood of winning from data, b) skill is shown to be more important than the game's meta, and c) certain roles are far more important than others in winning. With future updates to Riot's API, one could gather more accurate data for each match to generate samples with less noise. Currently, one must manually calculate a player's win rate on a champion, by requesting information for each match. As this would take too long for this study, I only found the win rate for the last 15 games played. Furthermore, more features could be generated for more accurate predictions. Some that I could consider would be features such as finding each player's experience playing with and against certain champions by looking in their past games, as well as their average match stats, such as KDA, damage dealt, and wards placed. I can also condition my champion features on how many games the player has on the champion, which

would take more request time. A cool further use of this data I intend to look into is possibly using this data and these models to create an application where the user can input the details of their current match, such as their teammates' champion picks. The application would then use this input to create a sample point, which my models can use to predict a probability of the match being a win or a loss. With a high enough probability of a loss, the user can either change characters, or change strategies to preserve their ranking.