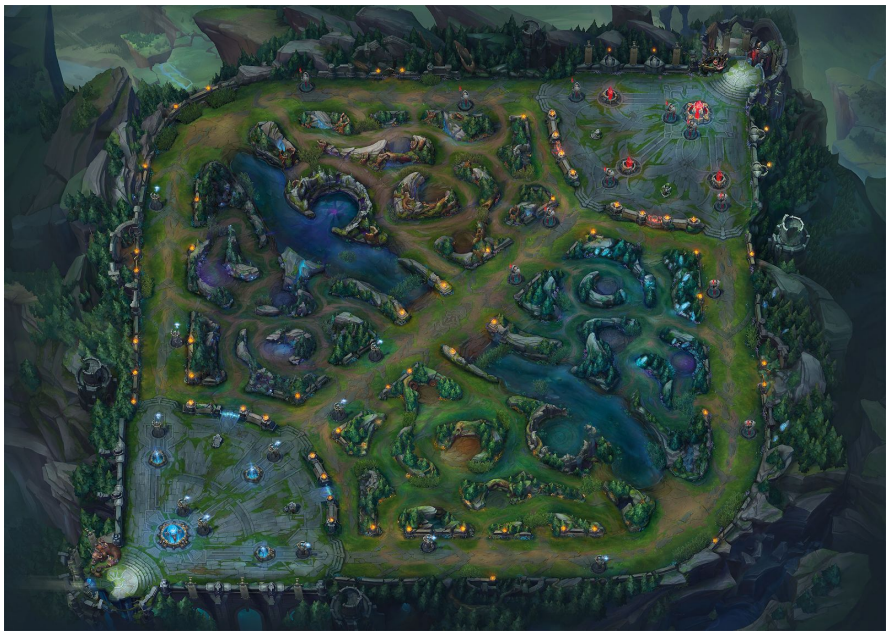
League of Legends: Item Balancing

Capstone Project 1 Final Report
Sebastian Alvis

Problem Statement

League of Legends is a Multiplayer Online Battle Arena (MOBA) game. It is played by two five-player teams of champions (the in-game characters) racing to destroy the other team's primary building: the nexus. Each nexus is protected by a few sets of towers and its team's players. The three lanes (known as top, middle, and bottom lane) from each base connect together to form the map. The area between the lanes is known as the jungle, and contains neutral (not on either team) monsters.



Throughout the game, players make their champions more powerful in two ways: leveling up and buying items. Experience and gold are generally earned by defeating various enemies, be they lane minions, enemy champions, towers, or neutral jungle monsters. Gold is used to purchase items to give the player's champion various stat boosts and abilities. Items form one of the pillars of League of Legends gameplay; deciding which items to build is a constantly changing topic as champions and items can be tweaked on a weekly basis.

For the developers of Riot Games (the studio who created and maintains League of Legends), one problem is **what items should be changed and why?** This is a task of game balance. They have a strong incentive to keep their game as fair as possible. Fairness in champions and items fosters diversity in which of these are picked in games, and this is where the game is supposed to shine. A lack of diversity can make the game feel stale, and cause players to stop

playing and spending money on the game. Making sure items are balanced is probably an ongoing task at Riot Games.

I attempted to do a reasonable balance analysis using ONLY the purchased items and match result (win/loss) of each game.

Dataset Description

There are three tables I acquired for this project. The first two are the official lists of champions and items, and were acquired from the Riot Static API / Data Dragon as JSON files, then normalized into tables. These mostly serve to increase readability, and can be joined to the main table. The information in these tables may change from patch to patch (~1-2 weeks), so it may need to be updated if I pick a different set of games.

The JSON format required that I flatten with only the values of the dictionary, and I tacked on the keys afterwards. Additionally, the items' tag column is preserved in the csv format as a unicoded string that looks like a list. I have a line of regex to make it a usable list of strings, but it needs to be reapplied every time the file is read into Pandas.

The champion table contains 141 champions, and I used the same regex fix as above for the champion tags, but did not require any further cleaning.

The item table contains 237 items, of which 80 will be in the final analysis. This involved a boolean column made to filter for items that did not build into anything else (full items being the focus of this project), some edge cases for the column, and filtering out other sets of items that are very different from the main purchasable items (Boots, Trinkets, Consumables). This helps reduce how many trends I will be trying to track, since this isn't meant to be an exhaustive analysis.

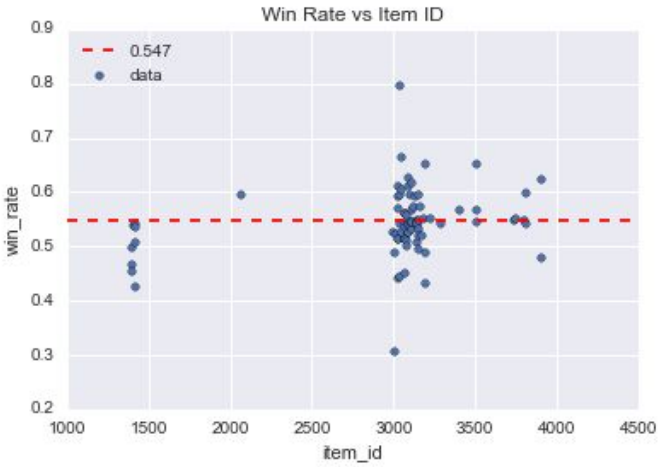
The last table is composed of match-level data. Finding specific matches was fairly complicated. I used 5 requests from the API, going from league IDs to summoner IDs to account IDs to match histories to actual games. I used a random subset of summoner IDs and a random subset of games from the available match histories to preserve the independence of the data. Acquiring match data had several problems, including the join complexity for the timeline data (which ended up not being used), missing match data, edge case items, and exceeded request rates.

The game data is information on the state of the end of the game, including players' champions, final item builds, kill/death/assist (KDA) ratios, etc. The player data from these requests forms most of the final table I use. This data was cleaned in creation, and the cleaning was mostly logic for items that wouldn't show up as "purchased" in the timeline table. My workaround was essentially a blacklist.

The match table has information for 10 players in each of 925 matches, making 9250 rows.

Exploratory Data Analysis

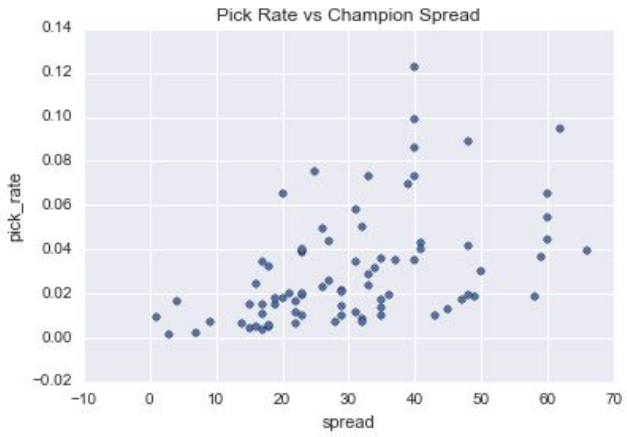
In the match data, each player builds up to 7 items (of which up to 5 will be relevant to the final analysis, accounting for Boots and Trinkets). From this, data was aggregated on an item-indexed basis to get descriptive statistics on how often items were picked (pick rate) and how often players won games when they picked the item (win rate). The lengths of the sets of unique champions who built each item (champion spread) were also recorded, but not used. The pick rate and win rate formed the main metrics by which I measure item performance and popularity.



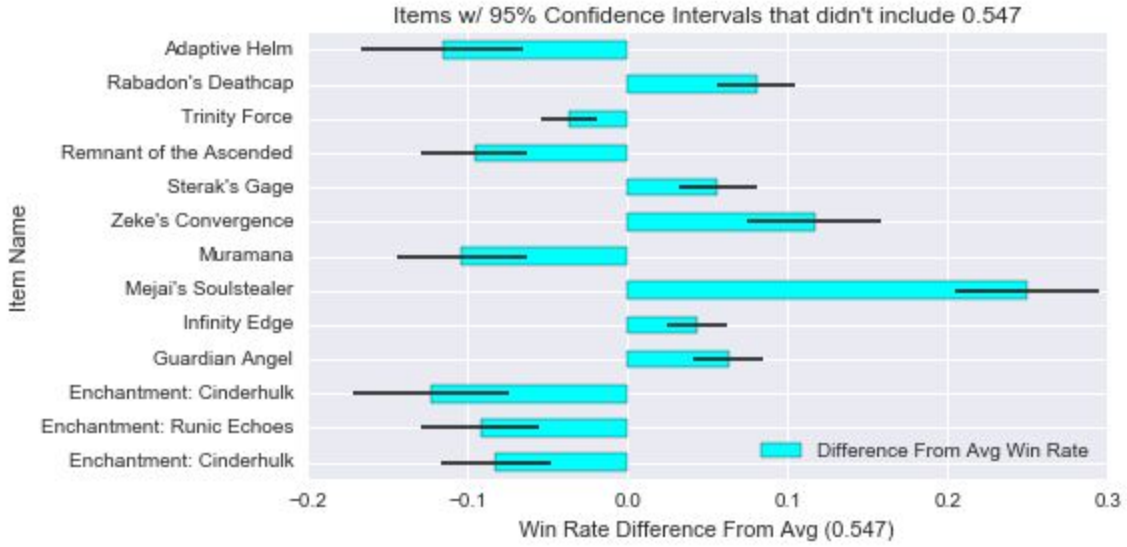
The most relevant variable for telling whether an item is too strong or too weak is the win rate. If an item has an abnormally high win rate, then it is too strong and should be weakened. Conversely, if an item has an abnormally low win rate, then it is too weak and should be strengthened. Both of these can be tested with a simple z-test against the average win rate of all full items and a 95% confidence interval on each item's win rate. This is especially easy because the win rate is a Bernoulli variable, as the proportion of games won.

Pick rate, the next most relevant variable, explains how often items are purchased. Pick rate is the proportion of item slots that have a given item (7 item slots per player). An item's popularity can indicate strength as well, as often-built items may be more viable than anything else.

Pick rate also indirectly shows up in the variance of the win rate, since it is proportional to the number of times an item is built.



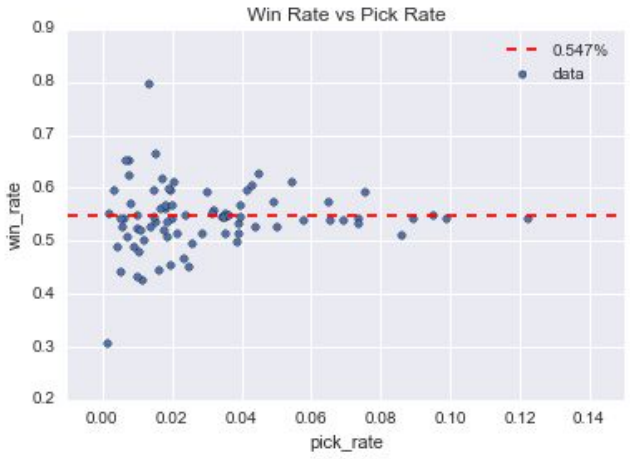
Pick rate and champion spread are expected to correlate. It is important to find correlations to pick rate and win rate, as the two primary metrics for answering the initial question. The highest correlation to win rate is through champion spread, and the highest to pick rate is through champion spread, and then total item cost. However, since they correlate rather well, I wouldn't want to use both in a machine learning analysis.



The primary test done here was the z-test for the proportion of wins. My null hypothesis is that If the game is perfectly balanced, then each items' win rate should be the same. For the data I have, the relevant full items have an average win rate of 0.547. The test is done to see if this value falls within the 95% confidence interval (alpha = 0.05) of each item's individual win rate. 13 of the 80 relevant items had win rates outside their confidence interval, 7 of them too low, and 6 too high. Most notable were items with a win rate 0.80 and a win rate of 0.31.

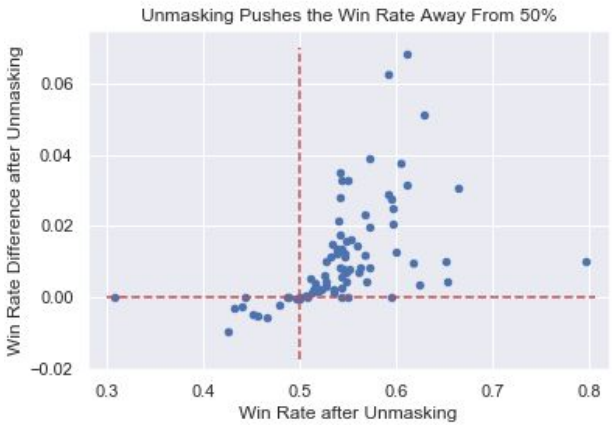
In-Depth Analysis

However, I am realizing that I should probably normalize win rate with pick rate. If one player on each team builds the same item, regardless of the game's outcome, one player will win and one will lose. The data I have would get two games with an average win rate of 50%, but it is artificial, since there is no way for the win rate to be anything else. This is a form of noise in the data. I should remove games like this, so I can see how well an item performs when it isn't on both teams.



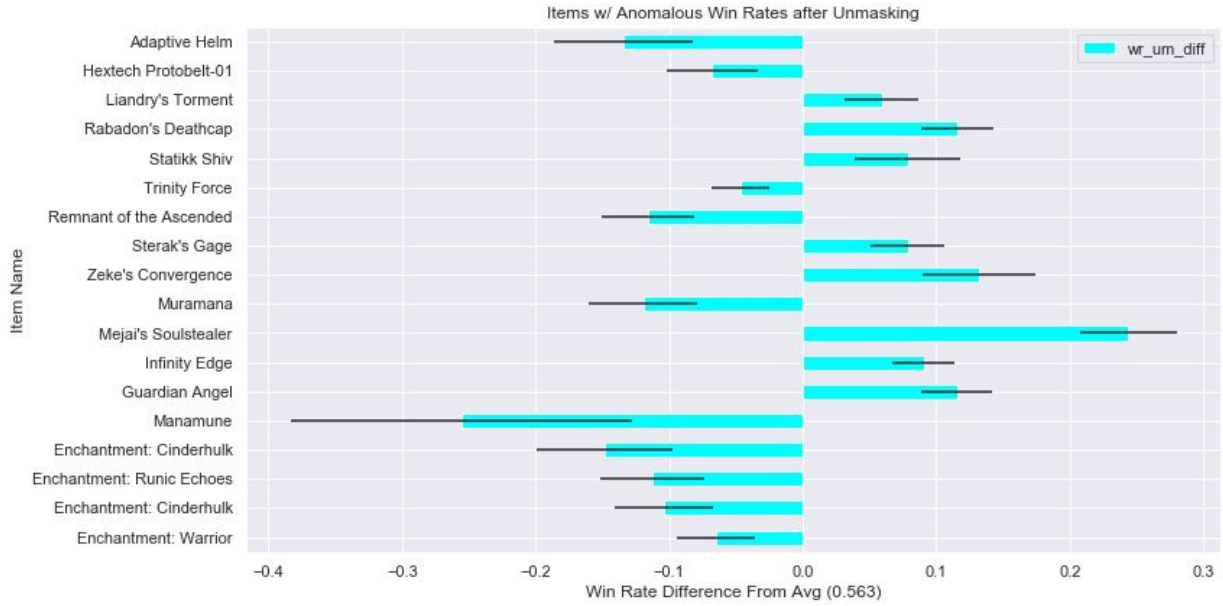
It is immediately obvious that items with higher pick rates tend to have win rates closer to the average. However, part of this is the aforementioned noise. I will games where an item is built the same number of times by each team. I made a loop to only consider games where this doesn't happen, and recomputed the same statistics as before. This noise "masks" the win rate I want to look at, so I proceeded to unmask it.

Here are the unmasked win rates vs pick rate. The red line is the old average win rate, and the dark yellow line is the new average win rate. There are some points that have risen considerably, mostly in the high pick-rate region, where I expected the masking to take effect more often.



This figure shows the effect of unmasking. Every item with a win rate over 0.50 had an equal or greater win rate after unmasking, and every item with a win rate under 0.50 had an equal or lesser win rate after unmasking. This is what I expected, and is a good sanity check.

After seeing how the win rates changed, I used the new counts and win rates to find a more accurate set of anomalous items, pictured below.



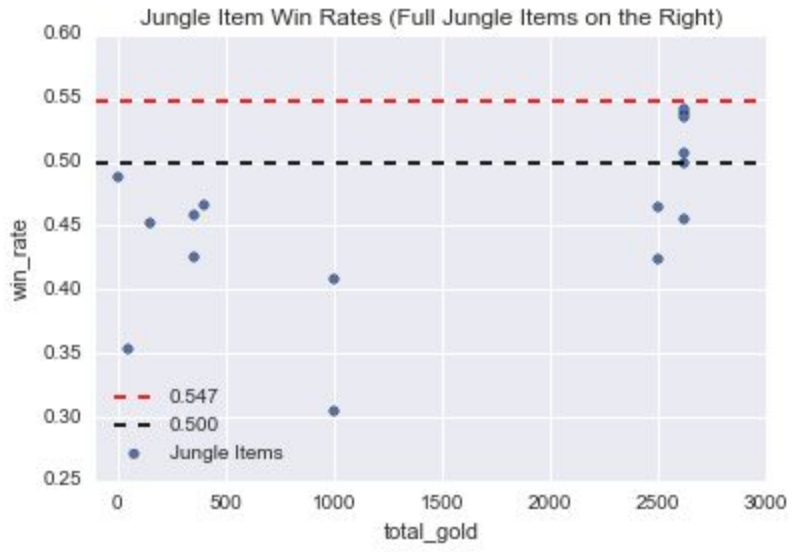
This plot has 5 more items than the previous one of its kind, and every item present in the last plot is also here. There is a notably low item present (Manamune), with a huge error bar. This item's win rate was probably close to being 1.96 standard deviations away from the mean before hand, but didn't show up because the win rate wasn't quite low enough.

What caught my eye in both of the anomalous win rate plots were the "Enchantment" items at the bottom. These are Jungle items (where your jungle item is "enchanted" based on the particular stats you want from it: attack speed, attack damage, health, or ability power). I decided to look at the jungle items to see why they seemed to be underperforming.

The win rate of the entire set of jungle items sits below the average full-item win rate. That struck me as very odd, until I realized an assumption I had made earlier was only partially correct. Originally, I had assumed the average win rate for an item would be 50%, since half of the players win and half lose in any given game (this turns out to only be true if I call the empty item slot an “item”). Below, the plot shows win rate as a function of item cost. The main 80 items I am analyzing mostly fall into the cluster on the right side of the plot, which has a higher average win rate than the left cluster.

The right cluster mostly houses the full items that I am focusing on. Since many games end before all players can buy 5 full items, it can be easy to see which team is ahead in gold at the end of the game. Teams that have fully upgraded all items in their inventories are stronger than teams that have not, and are thus more likely to win the game. This is the main driver of the shift in win rate between the two clusters.

However, jungle items are a little different. Players generally aim to build 5 full items and may or may not get them all before the game ends. Players who jungle will have one of those full items be a jungle item, and the jungle item is almost always built first. Games basically never end before players build one item each, so the shift in win rate for incomplete and complete items doesn’t affect the set of jungle items nearly as much. Jungle items center around 50% (see below), as per my original assumption.



So, when all that is said and done, I have 8 items with statistically high win rates and 10 with statistically low win rates. 4 of those 10 weak items are jungle items, and so I need to look more closely at which of those items are actually underperforming. The one closest to the average win rate is about at 50%, which is fine for a jungle item, going off of the plot above. The other three, however, seem to be doing much worse according to both plots.

In-Depth Analysis: Machine Learning

For machine learning, the obvious thing to do was to build models to predict the outcome of a game based on some aspect of the end-game state. Some questions I had to start were: Which models should I use? What data do I feed the model? What is my interpretation of the results?

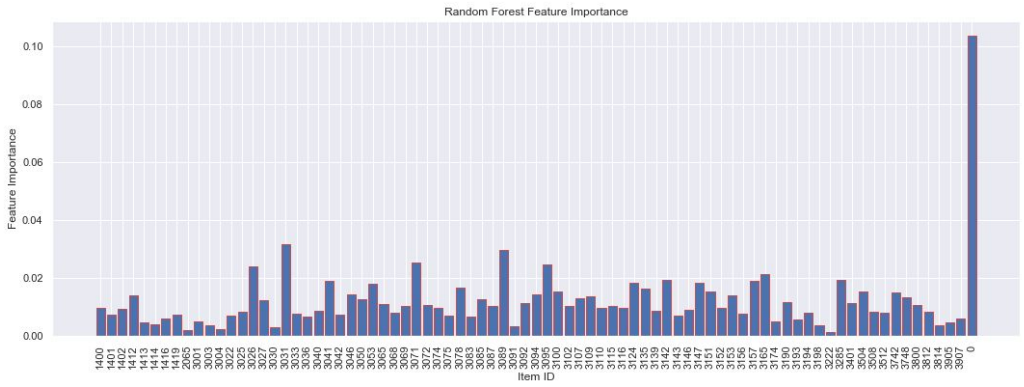
Initially, I used a random forest classifier. These perform well right out of the box and are resilient to a lot of data problems that affect other models. My data for the first random forest started with 2 columns: a column with the item id and a binary win/loss column. I then called the item id column a categorical variable, so that the random forest would try to use the value of the item id, since the value is meaningless. Thus, the first input data was a very sparse matrix of single items, with a binary win/loss column as the predictor variable.

This performed poorly. Accuracy sat around 0.53, or roughly the average win rate (pre-unmasking). A grid search for hyperparameter tuning did nothing for the accuracy, and I concluded that this data wasn't good enough for making predictions.

I needed to give the model more data for a predictor. I had a few options: each player's inventory, each team's inventory, the difference between each laner's inventory, or the difference between each team's inventory. This last option I realized would be the best. It has a similar noise-cancelling property to the unmasking procedure because it takes the difference between inventories. Also, it encapsulates the team-centric style of League of Legends by

adding together each team-member's inventory. This way, I counted the number of each item on a team into the same categorical style of data I used the first time, adding the results of the first team's total and subtracting the results of the second team's total.

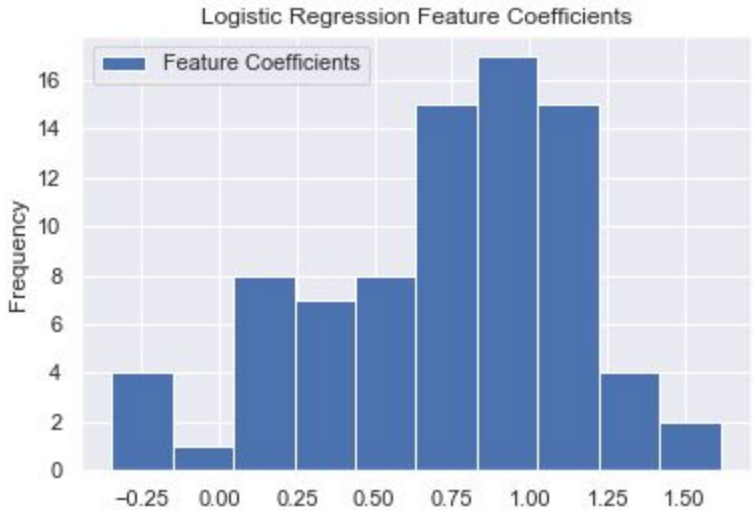
The second random forest performed much better. It had a test accuracy of 0.69. A grid search cross-validation brought that up to 0.73. The most interesting hyperparameter was the number of features for the forest: 81. That is one feature per item, including the empty item slot. This allows me to track the feature importance for each item and do other correlations.



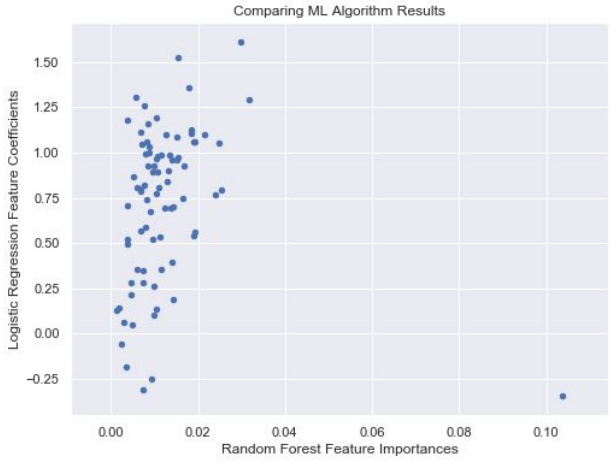
As we can see in the bar chart of feature importances, the most important feature by far is the empty item slot. Games can be predicted most accurately based on how many empty item slots each team has. This can encapsulate which team has more gold and is stronger. However, this lacks interpretability. I would think that the taller the bar, the more likely the item is to make you win the game. However, the tallest bar is the empty item slot, and having more empty item slots definitely indicates you are losing. So very poorly performing items could have small bars indicating minimal success or large bars indicating catastrophic failure, and I can't distinguish between them easily. These results lack interpretability.

Random forests generally perform well right out of the box, but I figured that another model could do better. I chose logistic regression. The predictor variable of win/loss is naturally binary, and the data I wrangled for the random forest has the same scale across all columns, centered at 0. The feature coefficients for logistic regression should also be much easier to interpret!

The out-of-the-box logistic regression performed beautifully, with a training accuracy of 0.92 and a test accuracy of 0.89! Attempting to tune the hyperparameters just showed me that the default ones were best, and no other model needed to be run.



Looking at the feature coefficients for this logistic regression, a few things stand out immediately. First is that there are several below 0. I would expect the empty item slot to be here, but there are also very poorly performing items there as well. Secondly, there are a few very high feature coefficients. This is not a normal distribution, but more of a bimodal distribution with some outliers, so I expect the few above 1.25 to be noteworthy.



Comparing the two machine learning models, it seems that my interpretability question for the random forest was irrelevant. The random forest generally found items to be important if they were going to help you win the game, with the one exception of the empty item slot.

Now, there are two parts of my logistic regression that I don't like. One is that the empty item column, while good for predictive power, is not actually an item and cannot be balanced. As the purpose of this analysis is to advise on what items need to be changed, I figure that I should remove this column so that the feature coefficients can more accurately reflect the effect of each item. The other part of my logistic regression that I want to change is the intercept. The

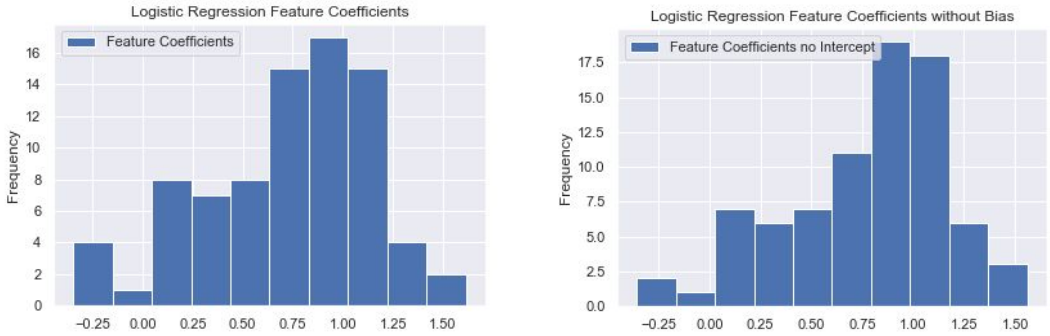
intercept, by default, is set to whatever value that most accurately helps predict the data (in this case, it was about 0.23). The intercept (β) can readily give you the predictions for feeding the model empty data from the equation:

$$P = \frac{1}{e^{\beta}}$$

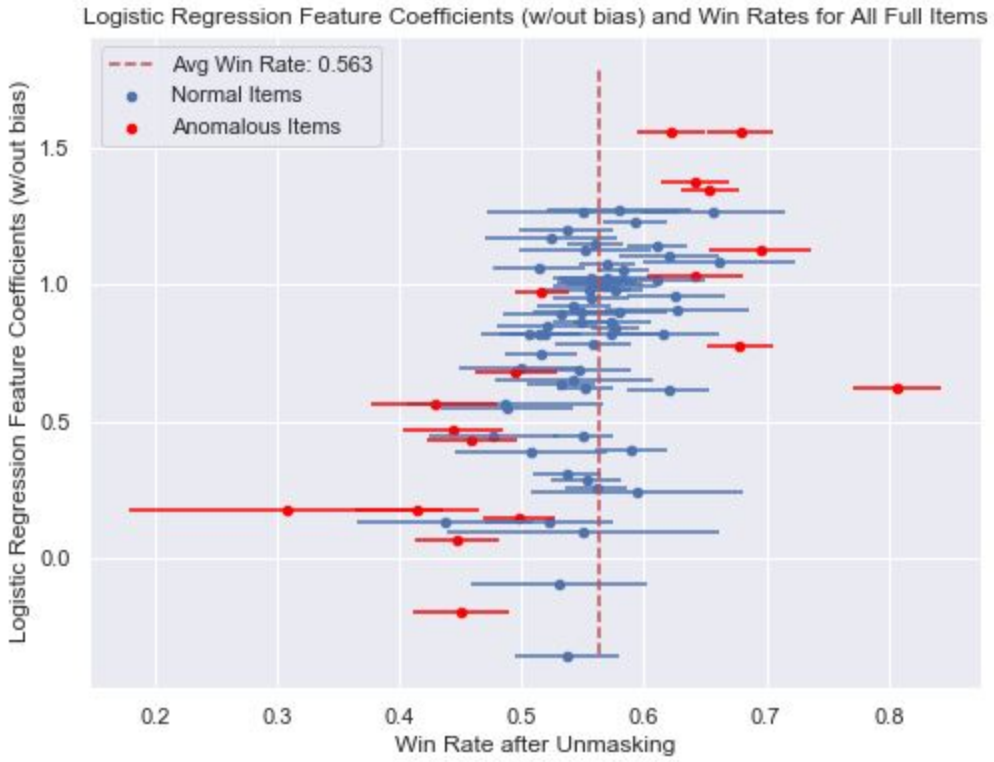
Thus, an intercept of 0.23 says that, given empty data, the model predicts a win with a p-value of 0.56. This shouldn't be the case, realistically. If both teams have the same items, I should equally expect either team to win, a p-value of 0.50.

So, I construct another logistic regression without an intercept and feed it data without the empty item column. Its training accuracy is 0.898, and its testing accuracy is 0.881. These numbers are slightly lower than before, but much better than I expected. Now, feeding the model empty data returns a p-value of 0.50, as desired. Hyperparameter tuning again returns the default parameters as best.

Most of the feature coefficients are about what they were before, but they are slightly different. Below, the left figure is the feature coefficients from the logistic regression with the intercept and 0-column, and the right is without both of these. There are now fewer outliers on the low end and more on the high end, and the distribution seems smoother, at least for this binning.



Now, since the logistic regressions also assigned one feature to each item column, I can compare the feature coefficients to the win rates from my statistical analyses.



The y-axis has the feature coefficients and the x-axis has the post-unmasking win rates, with error bars. The abnormal win-rate items are colored in red. Immediately, we see that the four items in the top-right of the plot have been flagged by both analyses as very strong. These are good candidates for balancing. There are also several low win-rate anomalies with very low feature coefficients, which are also good candidates for balancing.

In the end, I made four sets of items I could recommend for balancing. There were items flagged by my statistical analysis as winning more often than normal, those flagged as winning less often than normal, items flagged by the logistic regression as very impactful (top 10 feature coefficients), and items flagged as very low-impact (bottom 10 feature coefficients).

Items flagged as too strong by both analyses: Rabadon's Deathcap, Liandry's Torment, Sterak's Gage, and Infinity Edge.

Items flagged as too strong by one analysis: Mortal Reminder, Edge of Night, Gargoyle Stoneplate, Guinsoo's Rageblade, Thornmail, Frozen Mallet, Mejai's Soulstealer, Zeke's Convergence, Guardian Angel, and Statikk Shiv.

Items flagged as too weak by both analyses: Enchantment Runic Echoes (blue), Remnant of the Ascended, Enchantment Warrior (blue), Manamune, Enchantment Cinderhulk (red).

Items flagged as too weak by one analysis: Enchantment Bloodrazor (red), Wit's End, Mikael's Crucible, Hextech GLP-800, Athene's Unholy Grail, Trinity Force, Hextech Protobelt-01, Enchantment Cinderhulk (blue), Muramana, and Adaptive Helm.

Now, recall there is a difference between the average jungle (enchantment) item and other items. Most of the jungle items flagged by the statistical analysis are fine. Enchantment Warrior (blue) does not perform poorly (win rate is 0.498), so I don't think it deserves to be in that category. Runic Echoes (blue) and Cinderhulk (red) do actually perform poorly (0.450 and 0.415, respectively), and should be rebalanced.

Also, I'm not sure why Bloodrazor (red) was assigned a feature coefficient of -0.357. It has a win rate of 0.537 (well above normal jungle items). It's blue counterpart (Bloodrazor (blue)) has a win rate of 0.508 and a feature coefficient of 0.391. I would expect these two numbers to rise and fall together. The pair of Runic Echoes items have win rates of 0.451 and 0.543 and feature coefficients of -0.193 and 0.655, respectively. This is more of what I would expect.

The bottom two of the top 10 feature coefficients - Thornmail and Frozen Mallet - have feature coefficients that are solidly below the rest of the top 10 and more in-line with the rest of the items, so they probably don't need rebalancing.

Thus, the final item rebalancing recommendations are:

Enchantment Runic Echoes (blue), Remnant of the Ascended, Manamune, and Enchantment Cinderhulk (red) all should be made stronger.

Wit's End, Mikael's Crucible, Hextech GLP-800, Athene's Unholy Grail, Adaptive helm, Hextech Protobelt-01, Trinity Force, and Muramana could be made stronger.

Mortal Reminder, Edge of Night, Gargoyle Stoneplate, Guinsoo's Rageblade, Statikk Shiv, Zeke's Convergence, Mejai's Soulstealer, and Guardian Angel could be made weaker.

Rabadon's Deathcap, Liandry's Torment, Sterak's Gage, and Infinity Edge should be made weaker.