Final Project

Erik Van Cruyningen

Introduction.

Our data consists of League of Legends statistics at the end of the game, such as the number of assists a player got throughout the course of the game. We sourced the game IDs from Kaggle and then used the Riot API to pull the full information about every game. We have a large amount of data, and our models were trained on about 25,000 rows with about 900 features. One notable feature about this data is that it is only games from the Western European server from the ranks emerald to diamond, which makes it a very specific dataset that may not generalize to all League of Legends games.

Question #1: How well does our model do in games that are not emerald to diamond elo?

Methods

I trained a random forest model to classify whether or not the first player, and thus one of the teams, was going to win the game. I then found ten random players from each rank in the North American League of Legends server and then pulled each of their last fifty matches in order to be able to compare across the ranks. I then tested my model on each of the games and analyzed the results.

Results

The results of this model and analysis was that the model did not have a significant change of performance when the rank of the players was changed. The model performed well at every single rank, despite what we thought was going to happen. We hypothesized that the model would perform better at higher ranks because a more skilled player or team would be able to take advantage of minute differences in statistics and subsequently win the game even if the game was close.

Discussion

I began by grabbing the data we pulled from the Riot API. I then used my domain knowledge to drop all the variables I determined as useless, such as their riotID and the matchID. I also took out variables that instantly determined a win, such as NexusDestroyed, or win1-9, which were the original variables to say who wins. I then found my categorical variables, such as which items the players built and which champion each player was playing. I trained a random forest model with a 80-20 train-test split and printed my results in a confusion matrix.

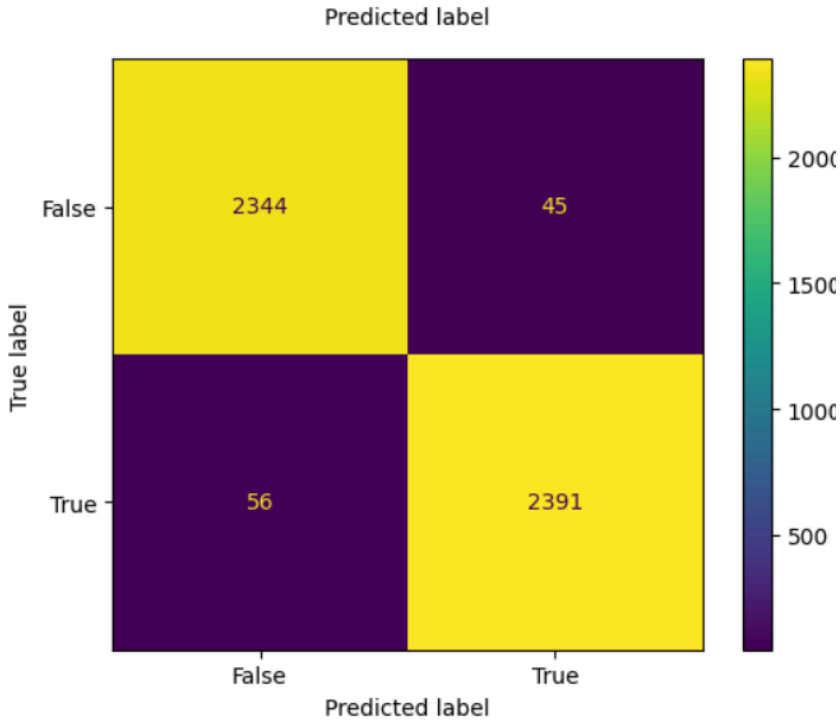

Figure 1: The Test Confusion Matrix

As we can see in Figure 1, the model is extremely accurate on this dataset. We also obtained an accuracy of 97.9 percent as well as an ROC AUC score of 0.9985. It appears that we can accurately predict who won a game with the statistics of after the game quite well, which we found quite unsurprising. We determined that if there was a person with domain knowledge and they looked at the statistics of a game after a match, they would also likely be able to get the correct answer 98% of the time.

I then pulled ten random summonerIDs from the Riot API. I grabbed one of each rank, and then pulled the fifty most recent games from each person, resulting in 500 games spread between all the ranks of League of Legends. We had some formatting errors when pulling the games, so we had to format it back to fit our original format. I then used my random forest model to predict the results for all 500 games, and then obtained the accuracy of our model on this data.

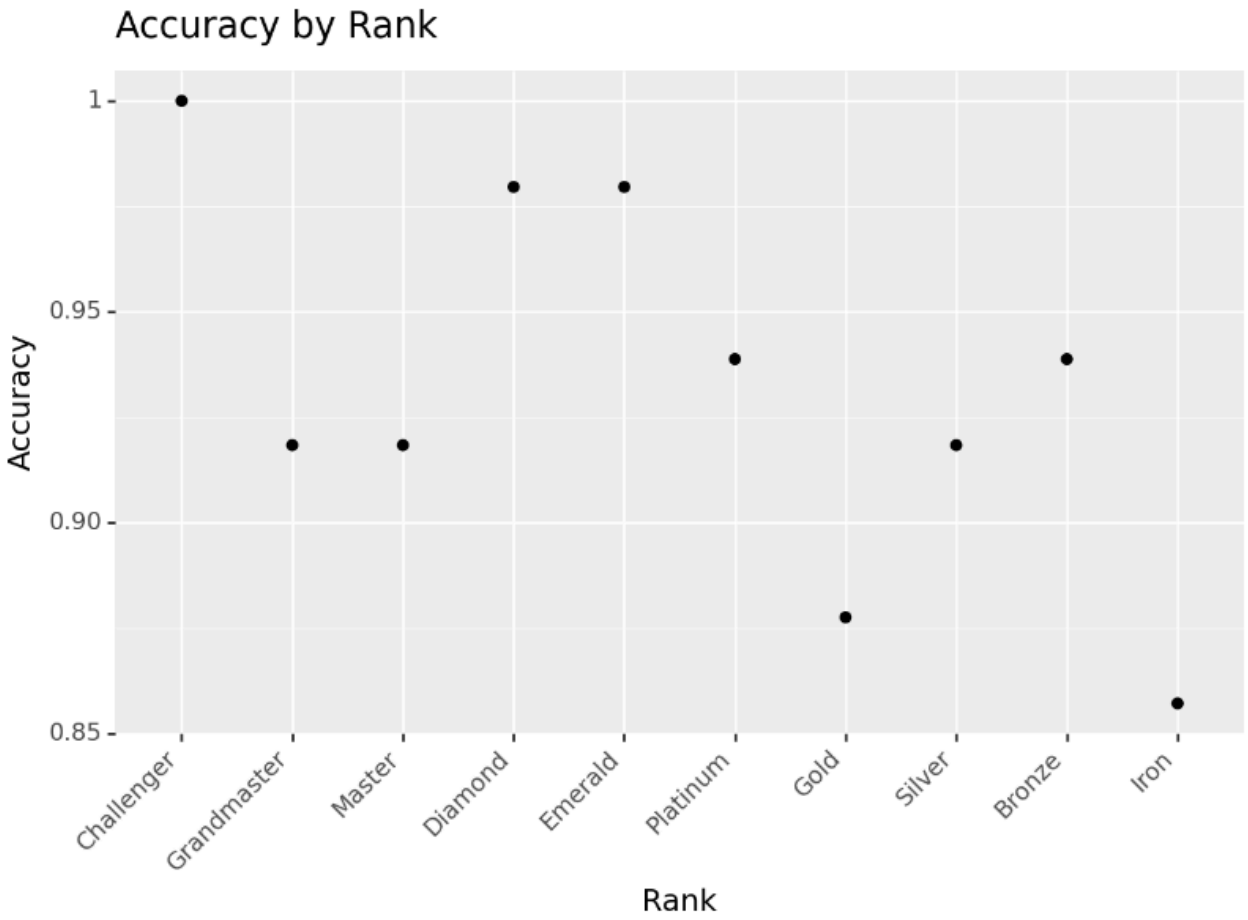

Figure 2: Precision for each Rank

As we can see in Figure 2, we can see that the accuracy of our model generally decreases as the rank decreases, but the correlation is not very strong. We figure that this is true because players who are better, the higher rank players, will be able to take advantage of any statistical lead they have early in the game. As expected, the accuracy of the diamond and emerald games were higher than most of the others because we trained our model on this data. One anomaly is the Challenger rank, which ended up being really accurate, despite the ranks Master and Grandmaster being poor. We have yet to figure out why this is the case.

Question #2: If we train a LASSO model, which variables are not very important, and can someone with domain knowledge interpret these variables?

Methods

I trained a LASSO model on the complete dataset and then used my domain knowledge to examine the top twenty five and bottom twenty five variables. This is applicable to the game so that a player can choose which objectives to focus, items to buy, and champions to play.

Results

The results of my model were quite surprising, but not too unbelievable. I found that the optimal team composition would consist of Rell, Brand, Sett, Fiddlesticks, and Ziggs, and that the best items to build are Demonic Embrace followed by Spectre's Cowl. Because this data is a year old, it is not applicable to an ever changing game like League of Legends, but if I pulled the most recent data from the after the most recent patch, I could find useful information. I also looked at the most important features from my random forest model, and all of the top features were how many towers your team took.

Discussion

We began by using all of the features we originally pulled from the Riot API and excluded all of the variables that instantly predicted a win. We then fit a LASSO model to this data and then matched the data with the LASSO predicted importance. We then used our domain knowledge to perform an analysis on the data. We can see in Figure 1 that our model pulled top features for a linear regression model predicting win outcome.

	lassoIMP	Features
13035	4.737559	championName3_Rell
2875	3.312723	item10_4637
12953	2.937537	championName3_Brand
1419	2.709889	item03_3211
13045	1.919577	championName3_Sett
4377	1.617588	item18_3047
9399	1.558280	item44_3123
13925	1.327848	championName9_FiddleSticks

Figure 1: The top eight features from the LASSO model

We then performed a similar analysis with our random forest importances. We pulled all the feature importances and then their names and matched them. We then displayed our data and surprisingly got very different results.

	FeatureNames	TreeImportance
607	standardscaler__turretsLost8	0.030630
631	standardscaler__inhibitorsLost9	0.030438
63	standardscaler__turretsLost0	0.024588
291	standardscaler__inhibitorsLost4	0.022680
335	standardscaler__turretsLost4	0.022158
548	standardscaler__damageDealtToBuildings8	0.020298
675	standardscaler__turretsLost9	0.020288
471	standardscaler__turretsLost6	0.020135
427	standardscaler__inhibitorsLost6	0.019052

Figure 2: The top nine features from the Random Forest model