League of Legends: The Meta as a Function of Time and Rank

Bryant Chen, Ted Li, and Sean Rafferty

I. Introduction

League of Legends is one of the most popular online games, attracting over 100 million monthly players. League of Legends is classified as a Multiplayer Online Battle Arena (MOBA) game, where teams of 5 players will compete against each other to complete their objective of destroying the opponent's base. Each player controls one of 133 champions as their avatar. The game is broken up into two major phases, drafting and gameplay. Drafting is the process by which each team assembles their 5 champions.

For our project, we are focusing on investigating the relatively unexplored champion selection networks that are found in League of Legends. The problems we will address are of two types. The first will be to perform various classification tasks in order to get an objective evaluation metric. These tasks include prediction of player skill and classification of champion roles.The second problem will be one of exploration - League of Legends is unique in that every two weeks, the game is updated in order to add variety while trying to fix balance issues. Because of this, champion selection networks are constantly changing, and we want to see if we can identify these changes through our network analysis.

II. Related Work

One resource we used was the paper "A tutorial on spectral clustering" by Von Luxburg. As the title implies, it provides a detailed explanation on the implementation of spectral clustering. It presents three different types of similarity graphs and two classes of the Laplacian matrix for similarity graphs. The Laplacian matrix approach we used for the League of Legends match data is described in detail later in this paper. This paper also provides three different approaches to clustering: graph cut, random walks, and perturbation. This paper does not address overlapping communities, this algorithm results into partitioning of the graph nodes into distinct sets.

Another helpful resource was the paper "Clustering in weighted networks" by Opsahl and Panzarasa. This paper explored various methods for calculating the clustering coefficient of weighted graphs. The bulk of the paper is focused on their new model, which involves a new scoring mechanism for triplets. Instead of only counting the number of triplets, each triplet, defined by a central node connected by edges to two other nodes, is given a score. From here, the weights of the two edges connected to the central node are combined to get a triplet score using either the arithmetic mean, geometric mean, maximum, or minimum edge weights. To calculate the global clustering coefficient, we use a similar formula as above. However, instead of using 3 times the the triangle count, we use the scores of all 3 triples in each triangle; and instead of dividing by the number of triples, we divide by the sum of all triples. They then demonstrate that many of the properties of the traditional clustering coefficient are consistent. This algorithm was very useful to this project, as we used a weighted graph and it is a large improvement over the naive alternative of binarizing the edges into low and high buckets.

III. Data Collection

We collected data from League of Legends matches from April to July 2016. There is a patch (a change to the game) every two weeks. Thus, our dataset spans 6 patches. We downloaded around 30,000 matches for every patch which resulted in about 180,000 total matches. We did not discriminate on the type of matches during our collections so our dataset contains matches from a variety of game modes and skill levels. This allows us to filter our dataset and analyze how game mode, skill level, or other attributes affect the network structure of champion picks.

There are two major types of matches when it comes champion selection order: draft and non-draft. Normal, All-Random-All-Mid, and AI matches are all non-draft matches. Ranked matches are draft matches. In non-draft matches there is no explicit champion selection order, and you do not get to see what champions your opponents picked. Hence, non-draft matches yield undirected edges between the champions on each team but no edges between the teams. This results in two disjoint complete components. In draft matches there is an explicit champion selection order that switches off between the two teams. Only one player is choosing a

champion at a time, and that players gets to see which allied and enemy champions were previously chosen. Thus, draft matches produce directed edges within teams and between teams in both directions. We choose the convention that an edge from champion a points to champion b if a was chosen before b (a could have influenced b).

We will also provide a mathematical definition of a general champion selection network G given a set of matches. G will consist of n fully-connected nodes. Each node is associated with a particular champion. Let A be the symmetric adjecency matrix for G. Define A_{ij} to be the number of times that champion i and champion j were selected on the same team in the set of matches. Since you can not select the same champion as a teammate (except in a special game mode), the diagonal is zero in A.

IV. Initial exploration

A. Spectral Analysis

Overview: We hypothesize that there are multiple prominent champion clusters. We believe the most distinct cluster will be the "role" cluster. In League of Legends there are five roles: Top, Mid, Jungle, AD Carry, and Support. These roles are heavily ingrained in the game and are even enforced in ranked (draft) matches. Each player fulfills exactly one role, and all roles must be filled on each team. Note that this does not imply that a champion can only be used in one role for every game. However, our experience playing the game tells us that champions are very frequently used in only one role, rarely in two, and almost never in three or more. Thus, we believe it is possible to separate the champions into five distinct clusters associated with the five roles. If we can find the role clusters using spectral clustering, we will conclude that the dataset is not just noise and that spectral clustering is an effective tool that may be able to find other clusters if we change our similarity matrix. In the rest of this section we provide a technical overview of spectral clustering and show that it is a useful technique in finding champion roles.

Similarity Matrix: Spectral clustering requires a similarity matrix. There are many choices one can make for their similarity matrix and no one choice is best (Luxburg). We will use a post-processed version of the adjacency matrix, A. First we note that A is a co-occurence matrix: it counts the number of times that pairs of nodes occur on the same team in the same game. However, more popular champions are overrepresented in this matrix. We want to model interactions between champions, not champion popularity. Thus, we will normalize A using association strength, which yields the best results for normalizing co-occurence matrices (Eck). Next, we notice that some entries in A are sparse. This may produce some noisy clusters so we will use Laplacian smoothing by pretending that each champion combination appeared at least once before normalizing our co-occurence matrix. Finally, we will normalize the resulting matrix to live in the range $[0, 1]$ by subtracting the minimum entry and multiplying by the new maximum entry. We will call this final matrix W.

Algorithm: Given a similarity matrix W we compute the graph Laplacian L as follows:

$$L = D - W \tag{1}$$

Where $D \in \mathcal{R}^{n \times n}$ is a diagonal matrix defined as:

$$D_{ii} = \sum_{j=1}^{n} W_{ij} = \sum_{j=1}^{n} W_{ji} \tag{2}$$

Then, we will solve the generalized eigenvalue problem below for λ and u (this implemented *normalized* spectral clustering as recommended by Luxburg):

$$Lu = \lambda Du \tag{3}$$

Let $U \in \mathcal{R}^{n \times k}$ be the matrix containing the first k eigenvectors as columns. The rows of this matrix $y_1, y_2, \ldots y_n \in \mathcal{R}^k$ are used as the features for the n champions.

Finally, we cluster the rows $y_1, y_2, \ldots y_n$ into k clusters using K-means clustering.

Comparison: We will first look at some similarity graphs (**Fig. 1**). The top row are adjacency matrices (unnormalized) and the bottom row are post-processed matrices found using the process detailed earlier. The left column were formed by looking at all matches in our data set. The right column were formed by looking only at ranked (draft) matches. Notice that some champions are drastically overrepresented in the left column which caused issues even in our post-processed matrix. In contrast, the right post-processed matrix for ranked games looks much nicer. This will help explain our results.

We first ran unnormalized spectral clustering on the post-processed similarity matrix generated by looking at our entire dataset (the bottom left matrix in **Fig. 1**). This produces most of the role clusters we expected but it had one double cluster and one cluster with just a few champions. This can be explained by the few outliers remaining in the normalized similarity matrix (the rows and columns with many bright spots). This lead us to use normalized spectral clustering on the same dataset which produced the five clusters we expected with no errors.

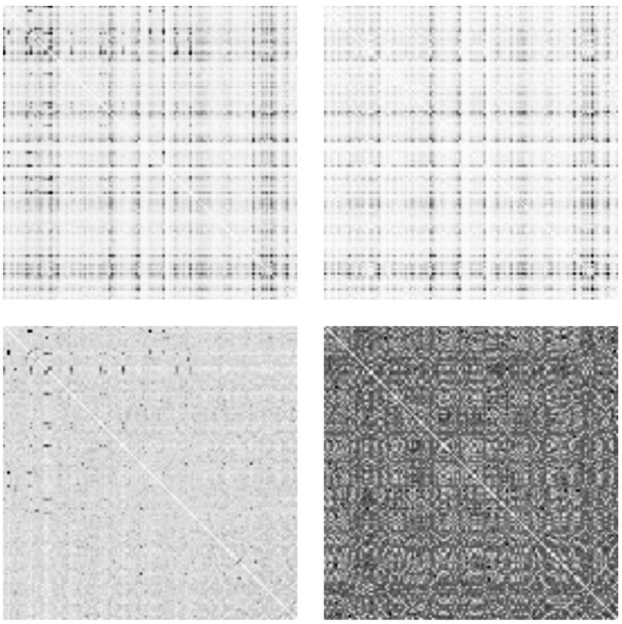

Fig. 1 Similarity Matrices. The top row is unnormalized and the bottom row is normalized. The left column has data from all matches and the left column has data only from ranked matches

We then ran unnormalized spectral clustering on the post-processed similarity matrix generated by looking at only ranked (draft) games (the bottom right matrix in **Fig. 1**). This produced the same clusters as normalized spectral clustering on the entire dataset. We also ran normalized spectral clustering on the ranked dataset and achieved the same results. We believe that running normalized spectral clustering may be redundant when the similarity matrix was correctly normalized.

Results: We will focus our attention on the results of normalized spectral clustering on the post-processed similarity matrices associated with ranked games across multiple patches. We will approach this in several ways. First we compare the role that a champion is picked for most often with the role that spectral clustering assigns it. Second we assign scores to roles for each champion by computing Euclidean distance between the champions and each cluster in the reduced five-dimensional eigenspace used during clustering and compare these to the frequencies that champions are chosen to fill those roles. Finally, we plot our champions in two-dimensional eigenspace associated with the normalized spectral clustering by choosing two of the top-five eigenvectors (eigenvectors having the largest associated eigenvalues). We conclude that spectral clustering does a fantastic job of clustering champions and aiding in visualization of the champion graph.

First we observe how often the anonymous cluster labels match the most-chosen role for each champion. Fortunately, our dataset contains information about which role a champion is filling in each ranked game. We will map anonymous cluster labels to actual roles by selecting a champion that is known to be associated with each role and mapping its assigned anonymous cluster to its known role. For completeness, we chose Irelia for Top, Shaco for Jungle, Syndra for Mid, Vayne for AD, and Bard for support. We use data from all six patches. Using this method, we assign 129 of the 131 champions to their correct cluster. The two champions we label incorrectly, Jax and Ryze, were both used to fill two roles about equally during these six patches. In **Fig. 2** we plot the empirical pick rate in green and our estimated pick rates in blue (the details of how we estimate pick rate is given in the next paragraph).

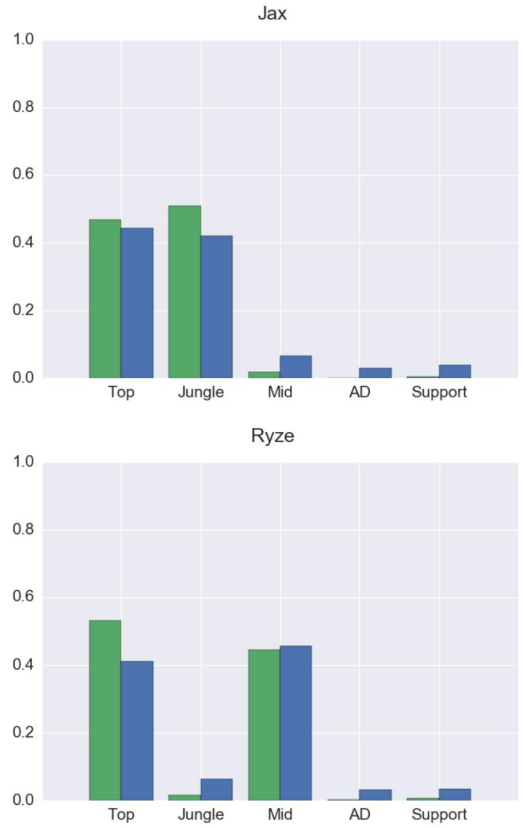

Fig. 2 Empirical role frequencies (green) and estimated role frequencies (blue) for Jax and Ryze using all data.

While spectral clustering does well when a champion fills just one role, it fails to model to ability for a champion to fill multiple roles. As shown in the previous paragraph, this causes issues when attempting to assign a particularly versatile champion to a single role. Instead of computing a single role for each champion, we now

compute the distance from each champion to each role. To make these more like pick rates, we pass the distance through a sigmoid function and normalize them so that all roles for a particular champion sum to one. These are the estimated pick rates that we showed in **Fig. 2**. As you can see, these scores to do a much better job of modeling the ability for a champion to be used in multiple roles. Furthermore, it is strictly more general than clustering because the role with highest estimated pick rate is also the role that clustering would assign. To get a sense for how well we do from patch to patch and how well we generalize, we will compute our accuracy in choosing the kth most common role for each champion. For instance, a high accuracy value for the 2nd most common role means that we are often correct in choosing a champions secondary role. These results are tabulated in Table 1. As expected, we see an extremely high accuracy for primary role. However, our accuracy for the secondary role is much lower than expected. This is explained by the fact that most champions do not have a secondary role but are still included in the calculation. To explore this further, we calculated the accuracy only for roles which were picked at least ten percent of the time. These are reported in Table 2 as fractions where the numerator is the number correct and the denominator is the number of trials. We see that only half of the champions are used to fill at least two roles, only one sixth of champions are used to fill three roles, and almost no champions are used to fill four or five roles.

Patch	1	2	3	4	5
6.07	0.99	0.59	0.38	0.41	0.52
6.08	0.98	0.58	0.47	0.45	0.70
6.09	0.96	0.52	0.44	0.37	0.54
6.10	0.98	0.55	0.50	0.45	0.73
6.11	0.98	0.54	0.49	0.47	0.67
6.12	0.99	0.61	0.49	0.34	0.53
All	0.98	0.60	0.59	0.56	0.85

TABLE I
ROLE ACCURACY FOR EACH PATCH

Patch	1	2	3	4	5
6.07	129/130	54/62	10/19	1/1	N/A
6.08	129/130	51/60	13/19	0/1	N/A
6.09	125/130	50/65	11/17	N/A	N/A
6.10	129/131	47/66	8/18	1/2	N/A
6.11	128/131	47/62	13/14	N/A	N/A
6.12	130/131	49/61	10/15	N/A	N/A
All	129/131	52/64	12/19	N/A	N/A

TABLE II
CONDITIONAL ROLE ACCURACY FOR EACH PATCH

Lastly we visualize our plots in various two-dimensional eigenspaces induced by the spectral clustering. Let a champion be associated with index i. Supposed we wish to visualize it in the eigenspace associated with the first and fourth eigenvectors (in descending order of associated eigenvalue). Them its x-coordinate would be the ith element in the first eigenvector and its y-coordinate would be the ith element in the fourth eigenvector. In **Fig. 3** we take the network created from all ranked games in our dataset, plot the champions in the eigenspace associated with the first and second eigenvector, color the champions according to their spectral cluster, and omit the edges (our network is fully connected). In **Fig. 4** we do the same with the first and fourth eigenvectors. As you can see, the gold and silver clusters are inseparable in the first figure but easily separable in the second figure.

We conclude that spectral analysis is extremely effective for clustering, visualizing, and scoring champion roles. While our dataset does provide roles which champions fulfill in each ranked game, it does not provide these roles for other games. Hence, spectral analysis could produce useful features for normal games and special game modes such as Dominion.

V. EXPERIMENTS OVER TIME

A. Weighted Cluster Analysis

Algorithm: The standard clustering coefficient algorithm uses unweighted edges between nodes and is defined as

$$C = \frac{\sum \tau_\Delta}{\sum \tau}$$

where a triangle is a fully connected set of three nodes, represented by τ_Δ and a triple is a set of three nodes that contains at least two edges, represented by τ. This method can also be applied to our weighted graphs by using a threshold to binarize our edges. If the weight is above the threshold, the edge exists, otherwise the edge does not. This threshold is typically set to 0, but we can vary it to remove low-frequency edges.

The weighted clustering coefficient algorithm given by Opsahl and Panzarasa is similar to the above algorithm, but instead of simply counting triples and triangles, it scores them. If we define $\omega(\tau)$ as our scoring method, then our weighted clustering coefficient algorithm is defined as

$$C = \frac{\sum \omega(\tau_\Delta)}{\sum \omega(\tau)}$$

For a given center node n in a triad or triple, we take the two edges connected to n calculate a score using the arithmetic mean, geometric mean, maximum, or minimum. In addition, we preprocess our edges by applying a threshold much like the simple clustering

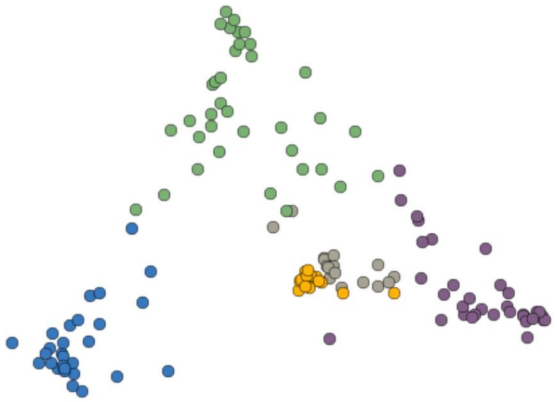

Fig. 3 Clusters plotted using the first and second eigenvectors using all patches. The silver and gold clusters are hard to separate.

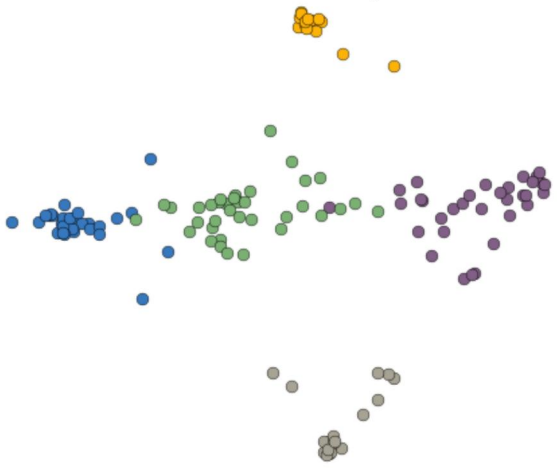

Fig. 4 Clusters plotted using the first and fourth eigenvectors using all patches. The silver and gold clusters are easily separated.

algorithm. However, instead of binarizing, we keep our values and set values below the threshold to 0.

Findings: For our investigation, we have applied four algorithms: the simple clustering coefficient algorithm, a thresholded algorithm, a weighted simple clustering algorithm, and a weighted thresholded algorithm. We began by testing how changing the threshold affects the clustering coefficient.

As threshold increases, all of the clustering coefficients decrease. However, at around a threshold of 0.005, we see that weighted blind and weighted ARAM start to deviate from the other curves. Upon further analysis of the clusters, it seems that in both of these graphs, as threshold increases, we're left with a cluster of highly connected champions as well as less connected peripheral champions. This explains why we still see

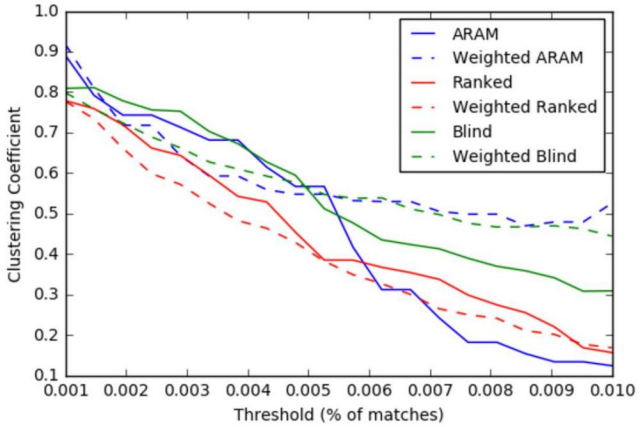

Fig. 5 Threshold vs Clustering Coef for Patch 6.10

the unweighted clustering coefficients decrease rapidly. In ranked games, we don't see this pattern at all. This is likely due to the ban process. Many of the highly clustered champions in blind pick, such as Elise, Braum, and Cassiopeia, are mostly banned in ranked games, preventing them from being in games and becoming more connected on the network.

We also examined changes in the clustering coefficient of individual champions between patches. For our timeframe, we chose our initial point as patch 6.07 and our end point as patch 6.10, and used weighted simple clustering on ranked games. To evaluate how much they changed, we ranked the clustering coefficients and then found the change in clustering coefficient between the patches. Over these three patches, many champions were buffed or reworked. Of the champions that were buffed, most saw an increase in their clustering coefficient, as they were picked more often and used in more team compositions. As for reworks, they also generally had large variance in clustering coefficient, though they were not necessarily positive, as some champion identities changed and players could not find a niche for the reworked champions. Table 1 has a few notable examples.

Champion	Changes	Δ Rank
Anivia	Rework	69
Fiddlesticks	Rework	63
Vladimir	Rework	49
Zyra	Rework	-13
Rumble	Buffed	-32
Malzahar	Buffed	-63
Syndra	Rework	-79
Zac	Buffed	-93
Vel'Koz	Rework	-97

TABLE III
CHANGE IN CLUSTERING COEFFICIENT BETWEEN PATCHES

VI. Skill classifier

Another goal of this project is to be able to leverage our champion select networks in order to predict the skill tiers that games are played in. To do this, we model our question as a regression, using various features that we can extract from champion select to predict the average skill tier of all ten players in the game.

A. Features and Response Variables

Riot stores the peak rank that each player in a League of Legends game has achieved in the current season. In order of increasing skill level, the tiers are: [UNRANKED, BRONZE, SILVER, GOLD, PLATINUM, DIAMOND, MASTER, CHALLENGER]. These tiers are then mapped to numbers, corresponding to their index in the ordered list of tiers, so UNRANKED would have a level of 0, while CHALLENGER would have a level of 7. To calculate the average skill level of each game, we then average the skill levels between all 10 people in the game.

To create our features, we began by using one-hot vectors for many of the simple properties of networks. We began with a one-hot vector for the presence of each of the 10 champion in a game. This is based on the idea that as player ranks change, their preferences for picking different champions also changes. In addition to that, we added a feature space of teammate-pairs, which are represented in a network as an each edge between any two champions that were selected on the same team. We also used a feature space of opponent-pairs, which are edges between any two champions selected on opposite teams. While this may not be as relevant in lower level matches, we believe that as skill level increases, these features may play a bigger role, since this helps capture the interactions between champion select picks.

We also leveraged some of our previous experiments to construct features. To use our spectral clustering results, we summed up the scores for each role for each champion in a game, and added this as a feature. Even though every team consists of 5 players in each of the designated roles, the champions they pick may be more versatile. It's possible that with different skill tiers, we may see a preference for picking more champions that serve certain niches in a team composition. In addition to role scores, we used the sum of the weighted thresholded clustering coefficients for each champion as a feature, as well as the total degree of each champion in the training network and summed the degrees of all champions found in a game. These features give us an idea of how central or clustered the champion picks in each game are.

B. Training and Testing Data

To train and test our features, we split up our data in a 4:1 test:train ratio. We randomly sampled our train and test sets and evaluated each new model 50 times. In order to maximize our training sample size, we also used both ranked and blind pick games. We also did analyses using data over multiple patches (from 6.07 to 6.12), as well as analyses using data from only a single patch.

C. Models

We trained our features and responses on two different models. The first was a linear model regressor that was fitted by minimizing l2 loss using stochastic gradient descent. Our second model was a Random Forest Regressor, which we hoped would help capture more nuanced interactions between champion picks. For both of these models, we leveraged the scikit-learn library. For all of these, we evaluated our models using mean squared error loss.

D. Results

Our initial testing was done on all of the patches combined, in order to have the largest dataset possible. Initial testing showed that our Stochastic Gradient Descent Regressor performs better than random Forest on our dataset. The baseline implementations was to pick the mean of the skill levels of the training data for all predictions in the test data. We then compared this to a model trained with various combinations of features from above, and found that we had the best performance when using the champion selection pairs, and when we used all of our features.

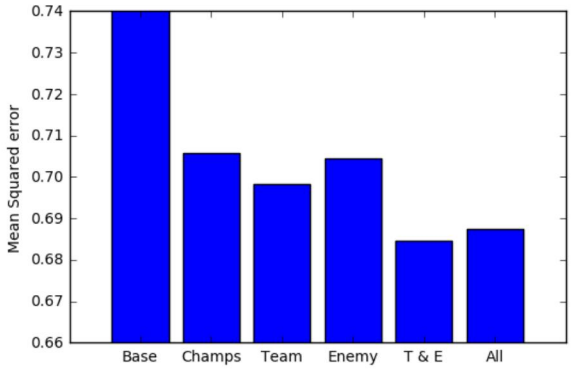

Fig. 5 Mean Squared Errors based on Features

The mean squared errors for many of our feature combinations are shown in Figure 5. Our baseline model's mean squared error goes off the chart, with an error of 1.078. Similarly our errors for just using Roles or

Clustering coefficients are very high, similar to the baseline. As shown by the graph, we have a decent improvement in performance when using direct champion picks. Furthermore, we see even more improvements when we add more features, but have the lowest loss when we use only pairs of champion picks, showing that the interactions between champion picks may offer more insight into the average skill level of a game.

To get a closer look at which features are indicative or higher or lower skill level games, we extracted all of the coefficients in our SGD Regressor and looked at the highest and lowest coefficients.

Type	Feature	Coefficient
Role	Support	0.327
Champ	Janna	0.292
Enemies	Caitlyn vs. Sivir	0.119
Teammates	Caitlyn and Blitzcrank	0.103
Enemies	Zed vs. Lissandra	0.091
Teammates	Xerath and Orianna	-0.83
Enemies	Fiddlesticks vs. Jhin	-0.091
Champ	Master Yi	-0.244
Champ	Teemo	-0.247

TABLE IV
HIGHEST AND LOWEST COEFFICIENTS FROM REGRESSOR

Many of these coefficients can be explained by the opinions of high level players and patterns found in professional gameplay. The high coefficient features generally correlate with patterns of professional play, while the low coefficient features are the opposite. For example, playing supportive champions, especially Janna, are often noted to be good ways to improve ones skill, but at the same time, they are also stereotyped as boring, and thus are usually associated with higher skill players. On the other hand, Master Yi and Teemo are often played for their simplicity or aesthetics, but are never selected in professional or high level play due to their one-dimensional playstyle, and they have some of the lowest coefficients in our regressor. The high coefficient enemy pairs show many of the traditional counter-picks found in the game; Lissandra can completely negate most of Zed's utility, while Sivir can block every skill from Caitlyn. On the other hand, Jhin is very immobile and does poorly versus champions like Fiddlesticks. Thus, we see a low coefficient associated with people who select Jhin into Fiddlesticks. We see similar patterns with teammate pairs, where Caitlyn and Blitzcrank synergize well and have a high coefficient, while Xerath and Orianna fill the same role, and thus do not synergize well on the same team, and have a low coefficient.

VII. CONCLUSIONS

Overall, this investigation consisted of three major sections. The first was an exploration of our dataset involving hypothesis testing; we did this mainly through spectral analysis. This allowed us to see that our data has informational data within it, as well as explore some well-known trends in League of Legends champion roles. The next part of our project involved looking at changes in weighted clustering coefficients over time. Once again, this was primarily fueled by exploration and hypothesis, and we were able to find many strong correlations between large champion changes and reworks, and changes in their clustering coefficients. Finally, we created a regression model to predict the average skill level of a match based on the champion selection by both teams. In here, we were able to leverage many of the simple network features, such as champion nodes present or teammate or enemy edges, as well as introduce features generated from our exploration. These additional features included champion selection roles from spectral analysis, champion degrees, and weighted clustering coefficients. With this final model, we were not only able to increase our performance in predicting skill level of games, but we were able to look at the coefficients to learn about what features are specifically indicative of higher or lower level games. In the end, not only were our experiments successful in finding trends and matching our expectations based on our experiences with the game, but we were able to learn more about how champion selections interact with each other and how they change over time and between different skill levels.

REFERENCES

[1] Von Luxburg, Ulrike. "A tutorial on spectral clustering." *Statistics and computing* 17.4 (2007): 395-416.
[2] Tore Opsahl, Pietro Panzarasa. "Clustering in weighted networks" *Social Networks* 31 (2009): 155-163
[3] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, Michael W. Mahoney. "Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters ."
[4] Eck, Nees Jan van, and Ludo Waltman. "How to normalize cooccurrence data? An analysis of some wellknown similarity measures." Journal of the American Society for Information Science and Technology 60.8 (2009): 1635-1651.

Ted: Weighted and Thresholded Clustering, Patch changes, Skill Classifier
Sean: Data Mining, Spectral Analysis
Bryant: Writing up related work, moral support
