**Building a League of Legends Champion Recommender**

*Christopher Gardner*

**Problem:**

Users are playing League of Legends less. Riot Games has identified that this problem stems from being bored with the champions (characters) they play. The proposed solution to this is a champion recommender system which will show users other champions they will enjoy playing. This will revitalize the user engagement by adding variety to their gaming experience.

**Client:**

The client is Riot Games the developer of Legends of Legends. League of legends is an online multiplayer game which derives profits from microtransactions typically for in-game cosmetic items. The recommender system serves to improve user engagement with the game by introducing users to champions they have not played much that they will enjoy. A side effect of the recommender is increasing revenue by having users play more champions and thus be interested in more different cosmetic items. Information can also be derived from the recommender system which can inform Riot Games’ future decisions. For example by examining preferences combine with user behavior, Riot Games could run promotions for champions that are not being monetized effectively by the microtransaction scheme.

**Dataset: (Explanation)**

To create a recommendation system we need items (Champions) and users. In this specific project the items we are using are known as Champions. These are characters that users play when competing against other users. Two main types of recommendation systems are collaborative filtering and content-based filtering. Collaborative filtering takes preference data from many users to make predictions about the interests of another user. It is based on the premise that similar users are more likely to share opinions than randomly paired users. Content-based filtering uses only the description and features of the items to create predictions based on what a user has liked in the past. For this project we chose collaborative filtering since there are a large amount of users (67 million monthly in 2014). Using a collaborative method means we need to obtain a data table filled with preference scores.

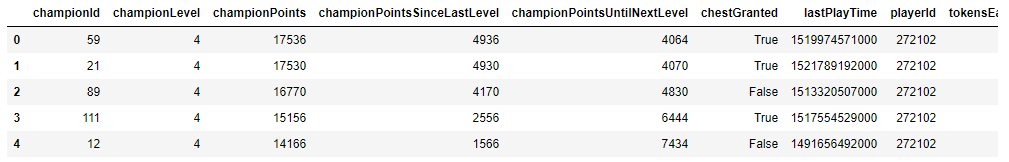
The dataset we will use is called a Champion Mastery score which is a metric created Riot Games to show how proficient a user is with said champion. Taken from Riot Games:

“The mastery system is designed to recognize a player's respective investment in a champion and uses skill-based progression to measure advancement... Points are based on your performance, your team’s performance, the length of the game, win vs. loss and a few other factors.”

Champion Mastery is a system which rewards skillful play. This recommender will rely on the assumption that users/players will be proficient with the champions that they enjoy. Thus the mastery system can be used as a measure of preference for our collaborative filtering model..

**Data Scraping**

This Dataframe of users and champion masteries was created by combining several pulls from the League of Legends API. A json of champion masteries can be pulled from the API if a summonerId (number to identify a user) is presented. Thus the first step is to create a list of unique users. This is done using matches. In League of Legends a match consists of 10 players so if we use one of them to find the match we can get 9 new users to our list. By providing a summonerId one can collect a list of their last 20 matches and then add users to our list and collect the recent matches from the new users we have added. Using my personal account as a seed a unique list of 10,000 users was built with the method explained above.

  
**Figure 1.** Sample of scraping mastery data.

*Sample output of champion mastery for a single user. This is mapped into a single row with ChampionPoints as the column and the playerId as the index.*

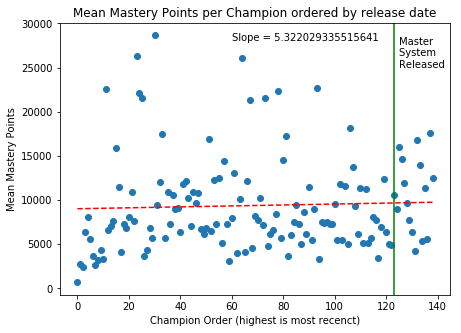
**Initial Data Exploration:**

|  |  |
| --- | --- |
| Champion (Id) | Total Summed Champion Points |
| Thresh (412) | 289064356 |
| Yasuo (157) | 265167147 |
| Vayne (67) | 263557123 |
| Ezreal (81) | 228536943 |
| Jhin (202) | 227652027 |

**Table 1.** Top 5 Champions by Total Champion Points across all users.

*Top champions by total champion points. Knowing the top champions by mastery point gives a good indicator of popularity. Champion points are roughly correlated play time/skill.*

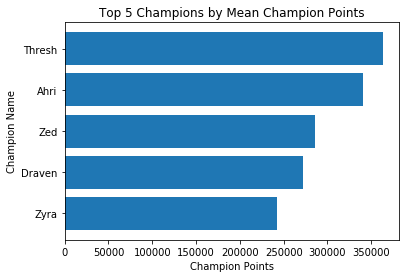
The 5 champions (championId) with the highest total mastery are interesting when you consider that higher champion Ids are typically associated with recently released champions (This is not always the case as Ids are given based on when the concept was created). We might expect that a feature like champion mastery would favor older champions since they’ve been around longer and thus users would have a longer time to get better with them. To explore this we used a scatter plot of mean champion/mastery points and ordered the champions by release date (Figure 1).



**Figure 1.** Scatter of Mean Champion Points by Champion.

*Ordering the champion data points by release date shows how the ‘age’ affects Mastery score. A 1d fit line of slope ~ 5 shows us that there is a small upward trend. Overall newer champions have a higher amount of champion points.*

This could be explained by the relative recency of the champion mastery system which may have a similar effect on champions released close to the launch of the champion mastery feature. The champion mastery system was implemented 3 years ago compared to the game’s open beta start 9 years again on October 22, 2009. A similar statistic mean mastery shows the top 5 champions are Thresh, Ahri, Zed, Draven, Zyra. Interestingly Thresh is the highest total and highest mean while the rest of the top scores are not shared. It would be interesting to look at these specific champions for item similarity to see if the most popular champions are all similar.

**

**Figure 2.** Top 5 Champions by Mean Champion Points.

*Mean champion points over all users gives the ranking in average skill/playtime. The mean is also interesting because it shows which champions players most frequently dedicate themselves too.*

The last interesting find from an initial exploration of the data is the standard deviation (std dev) of scores by champion. Standing out is that Thresh is again on top of the chart followed by Ahri, Zed and Draven. This is extremely similar to the info we got from the highest mean data. This shows me that these champions are not only popular due to the high mean but most of the players who have played the champion chose to play them to a high proficiency. There is a caveat though, we know conditions such as game length and win rate affect champion mastery points. A champion with typically long/short games or a high/low win rate would have their average number of points skewed and so it may not always be an equal rating of proficiency. This is an idea I’d like to explore in the future.

|  |  |
| --- | --- |
| Champion (Id) | Standard Deviation |
| Thresh (412) | 969689 |
| Ahri (103) | 924257 |
| Zed (238) | 767840 |
| Draven (119) | 740387 |
| Aatrox (266) | 661385 |
| ... | ... |
| Ornn (516) | 60886 |
| Kai’Sa (145) | 13676 |

**Table 3.** Standard Deviation of Champion Points.

*Standard Deviation of champion points gives useful insights into the player base’s interactions with a specific champion. Low std dev shows me that most users have a similar mastery of a champion either high skilled or never played. High std dev shows that there are a lot of players mastering said champion and a lot who have never played or barely played them.*

Another champion of note is champion 145, which has a std dev of raw values of 13676 where the next closest is 60886. Further exploration shows us that 145 also has the lowest maximum and mean mastery of all the champions. Not only is this champion the lowest in popularity by a long shot it is also played least by a long shot. After looking up the champion it becomes obvious why this is the case. ChampionId 145 is associated with the newest champion release as of this report, Kai’Sa. Having been released on 3/7/2018 Kai’Sa has only had a month to be played. We may want to exclude the most recent champion release in future iterations of the recommender as it is similar to an outlier.

**Building the recommender System:**

To build a recommender system we used the Surprise library for Python. The Surprise library doesn’t take a dataframe like we have as an input so we have to modify the data into a user, item, value format. To build our recommender we need to look at how to normalize the data, and which model to use and if applicable how to compute similarities. Normalization is the most important to discuss because when comparing users there can be a large variation in how long/how much a user plays League of Legends. A player who has put in more hours should have higher mastery overall and so a veteran user must be normalized with a new users to keep predictions accurate. To do the normalization we chose a Standard Scaler this will allow us to avoid high variance features from dominating the recommender. A champion with high mastery variance among users could prevent the recommender from learning from champions that aren’t pushed to high master consistently. This would be bad because it may miss valuable recommendations. We will use a standard scaler on the features, users and combinations of both (features -> users, users -> features) in order to determine the most effective method. Now that we have a normalization method to test we will build recommendation models such as SVDs, NMFs and similarity based nearest neighbors.

**Evaluating the recommender System:**

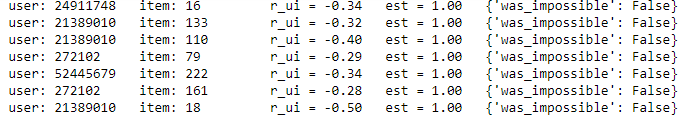
To evaluate the recommendation systems and normalization combinations we use a cross validated RMSE score. For the cross validation (cv) we use a built in method to perform a 5-fold cv. Since RMSE scores are respective to other RMSE scores we used a baseline model which uses the deviation of the user and champion to find a recommendation[1].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Column Scaled | Row Scaled | Row -> Column | Column -> Row |
| Base | 1.4142 | 1.4142 | 1.4142 | 1.4142 |
| Slope One | 1.4143 | 1.4142 | 1.4142 | 1.4142 |
| NMF | 1.4195 | 1.4246 | 1.4259 | 1.4272 |
| SVD | 1.4127 | 1.4100 | 1.4142 | 1.4132 |
| Item-item Cosine K-nn | 1.4125 | 1.4139 | 1.4142 | 1.4140 |
| Item-item MSD K-nn | 1.4156 | 1.4142 | 1.4142 | 1.4142 |
| Item-item Pearson K-nn | 1.4126 | 1.4127 | 1.4145 | 1.4138 |

**Table 4.** RMSE Comparison between algorithm (row) normalization (column) combinations.

*Root Mean Squared Error is a metric for relative comparison of different algorithms. SVD, Cosine K-nn and Pearson K-nn perform better than the baseline.*

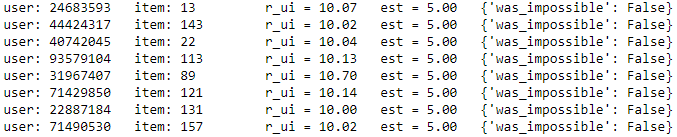
As you can see from Table 4 using the Standard scaler has some issues as most of the scores are at 1.4142. This is a point of concern to me because having the recommender systems perform exactly the same across multiple algorithms and normalization strategies should not be happening. Even among the recommenders that outperform the baseline it is typically by a low margin. SVD performs the best on Row scaled data and only out performs the baseline by 0.0042. This may be due to the standard scaler normalizing the data to match a normal distribution. I checked the prediction scores to see if this may be the case. See est (estimate) in the figure below. Interestingly the majority of the predictions were 1.



**Figure 2.** Predictions from the baseline.

*The minimum estimation score is 1 for all the scores from the test and the maximum is 5. This should not be the case sine the baseline model is Mean overall score + user deviation + item deviation. Standard scaler moves to a mean 0. The baseline method on normalized data should not predict 1 for all user-item pairs.*

This is a large issue that I hypothesize has to do with the surprise library not being able to produce prediction scores less than 1. To account for this I switched to a min-max scaler to scale the data into a range of my choosing. Before deciding on a range it was important to test for a maximum estimation value. After hand testing values I determined the max prediction score was 5.



**Figure 3.** Prediction Maximums

*Prediction maximum found by scaling the data to a range between 10-20 and putting it through the baseline estimator. There are no prediction values above 5 which shows me that it is the max prediction score.*

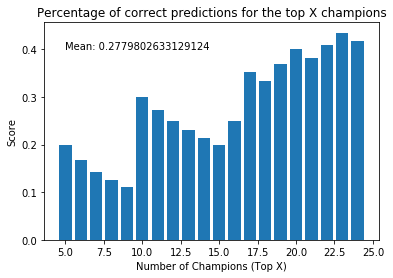
After determining the min and max prediction range, I chose to scale the data to a range of 1-5 to encompass all that the surprise library is capable of. Increasing the range of the data also has the benefit of causing the error to be larger for incorrect predictions and thus a bad model will be more obvious. With the standard scaler the values were so close together that the best models were indistinguishable for 3 significant figures. The change to a min-max scaler effectively corrected the issue with estimations (see below) so now model evaluation can be done effectively.

|  |  |  |
| --- | --- | --- |
| Model | Row Scaled | Column Scaled |
| Base | 0.5431 | 0.1295 |
| Slope One | 0.5462 | 0.1319 |
| NMF | 0.5259 | 0.1354 |
| SVD | 0.5109 | 0.1296 |
| Item-item Cosine K-nn | 0.5714 | 0.1341 |
| Item-item MSD K-nn | 0.5710 | 0.1341 |
| Item-item Pearson K-nn | 0.5203 | 0.1420 |

**Table 5.** RMSE Comparison of algorithms after data is normalized by Min Max Scaler.

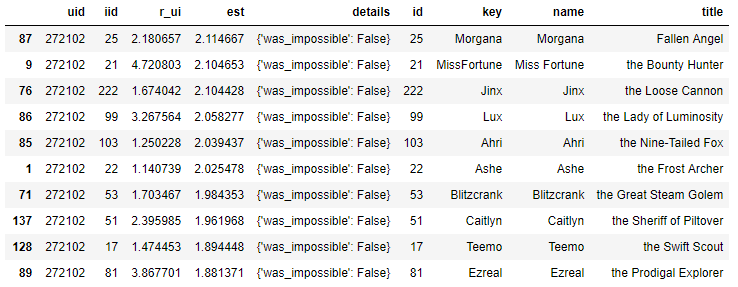
*Using a min max scaler with a range from 1-5 to encompass the prediction range that surprise is capable of. The RMSE values are more distinct than when a standard scaler was used. Itme-itme cosing k-nn and SVD work the best. We will focus on the row scaled data since normalizing the users is a more valuable application.*

Using the min-max scaler the difference between column and row normalization has become even more distinct. The first thing I note is that the overall RMSE score is much less than when the standard scaler was used, this can be attributed to the algorithms being able to make predictions rather than being forced to estimate as 1. The column scaled results are now very poor and none of the algorithms are able to outperform the baseline. Row scaled results are much more clear. Avoiding the estimation limitations allows the RMSE values to spread out. In this case similar to when the standard scaler was used, SVD is the highest performing algorithm with a score ~0.03 points below the baseline. Following the SVD, Pearson K-nn and NMF are the next best algorithms both around 0.02 points below the baseline. Not only has the min-max scaler improved the RMSE score it allowed the algorithms to have more distinct predictions. Having Distinct predictions make determining the best algorithm easier. As a sanity check I have used the svd model to output predictions of my own account to ensure that the predictions match up to my actual preferences.



**Figure 3.** Percentage of correct predictions

*A 3 fold cross validation was used to ensure that the test results were not trained on. The downward trends in the 5-10 and 10-15 range are interesting because it shows that correct predictions in the top 15 are hard. The correctness percent increases as we move towards the top 25.*

**

**Figure 4.** Comparison of In-game mastery score vs. predictions.

*A quick comparison shows that of the top 10 predictions 3 of them fall into the actual top 10, Miss Fortune, Ezreal and Lux. Looking at the other predictions and the top 28 (shown in the game screencap) a few more champions pop out, Blitzcrank, Jinx, Morgana.*

To obtain this data I performed a 3-fold cross validation and took the prediction results for my account from the parts of the cv. This was to ensure that the data I was looking to test on was not included when training the model. From this prediction data we can see some promise but a lot of optimization needs to be done. Comparing the top 10 champions to the screen cap of the game (top 28 by champion master) we notice that only 3/10 predictions (Ezreal, Lux and MissFortune) are in the actual top ten. Optimizations will need to be done but the current results will make for good recommendations.

**Conclusion + Next Steps:**

During this project we have scraped data from the League of Legends API to create our own database of champion mastery by user. Using this user-mastery database we built a recommendation system using the singular value decomposition algorithm [2] and normalized the data using a min-max scaler. Confirming that the actual predictions are relevant by comparing my overall predictions to my actual champion mastery and preferences shows us that an effective recommender system can be built off of the champion mastery scores.

For the next steps to create an api for this project and work with others for a web app I will want to use the flask module as well as modify the prediction outputs so that we do not recommend champions to users that they play frequently. Apart from moving the project forward into a web app I’d like to add various features to improve the recommender as I’ve mentioned throughout this report. One main things I would like to do to optimize the recommender is adding features. We have seen that cosine similarities between items may be beneficial for predictions so using static champion data such as their difficulty, attack, defense or magic rating could be very useful. Another potentially powerful idea to implement is additional user features for user-user comparison. Specifically some players are more competitive or have a high skill level than others. We could leverage ranked placement data to pre-screen the recommendation training set for players of a similar skill level to those of the input user.

Some other datasets/features we could add are the last time a champion was played, static champion data and ranked matchmaking info. First, last champion playtime is an interesting statistic which would help us understand which champions are most enjoyed by a user. This is under the assumption that the player plays his/her favorite often, sometimes this isn’t the case as a recent champion release will cause players to play the new character more or a user’s favorite character may not be viable in the competitive meta causing them to have not played them in a while. To help take into the account how competitive a player is and also add features for computing user similarity we can consider their ranked play information. This includes data such as win/loss ratio, overall rank, what tier they are in and how many league points they have. All of the ranked data can help us group players of similar skill rating and should help cover for some of the downfalls of only using champion mastery. The extra data could also be used in recommendation algorithms that can make use of data outside of the limited user-item matrix paradigm such as a Factorization Machine.

References:

[1] Yehuda Koren. Factor in the neighbors: scalable and accurate collaborative filtering. 2010. URL: <http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1-koren.pdf>.

[2] Daniel Lemire and Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. 2007. URL: <http://arxiv.org/abs/cs/0702144>.