League of Legends Champion Recommender System

Zihan Yi, Langtian Qin, Haozhe An

December 4, 2018

1 Introduction

In this project we will explore various models to build a recommendation system for one of the most popular MOBA game *League of Legends (LOL)* . In this game, each player will pick a champion to represent itself the entire game, but League has been released for more than 8 years and is such a well-developed game that there are 141 champions in total for you to pick. Due to the fact that different team compositions and situations require different champions, players likely want to be proficient on a wide range of champions. However, it's often hard and time consuming to find a suitable champion to pick up and train with, which makes practicing a champion that one ends up not liking a pretty bad experience. Our main goal in this project is to solve this problem by building a personalized champion recommender system that would propose such champions for individual players.

2 Dataset

2.1 Source and Format

We obtain our dataset from RIOT API, which provides *LOL* developer community with access to game data. We first use "MATCH-V3" endpoint to crawl 1000 matches happened since November 26th, and select all those players in those 1000 matches as our dataset (active players), then we use "CHAMPION-MASTERY-V3" endpoint to get certain player's all played champions information as well as "LEAGUE-V3" endpoint to get certain player's current rank level. We construct the feature based on the above.

Each piece of data in the dataset contains the following information.

- Summoner's ID. A summoner is a player in the game LOL.
- A list of champions' information. A champion is the virtual role controlled by a summoner in the game. In each item of this list, we have champion's ID, the level of this summoner's champion, and the points of this summoner's champion. The champion's level is related to the magnitude of the champion's point. A higher level implies a higher point and vice versa. In general, a summoner starts with a champion with level 1 and mastery point 0, he/she could achieve level 5 with a champion by playing 15-30 games with this champion, depending on the number of victories or defeats (a victory grants faster level up experience than a defeat). However, for level 6 and level 7, a summoner must use the champion to play very well in the game (most likely the top player or the top 2 players in a 10-people game) for 2 or 3 games to be able to level up. For champion mastery point it's simpler. There is no limit for how many points you can earn. In general, winning a game grants you 1000 mastery point and losing grants you 200 mastery point.
- Summoner's rank. Summoners advance their ranks by playing in the rank ladder mode in the game. Winning gives you rank point and losing deducts points. There are 7 tiers of rank based on the rank points. Rank 0: UNRANKED, Rank 1: BRONZE, Rank 2: SILVER, Rank 3: GOLD, Rank 4: PLATIUM, Rank 5: DIAMOND, Rank 6: MASTER, Rank 7: CHALLENGER.

2.2 Preliminary Analysis

In total, there are 51,139 unique summoners and 141 unique champions in our dataset.

First, we analyze the summoners in our dataset. A summoner may or may not have a rank and this rank indicates how much/well they play in this game. We are interested in learning about their ranks to get a sense of how experienced the summoners are in our dataset. Their level of experience will potentially affect what they like in this game. For example, it is likely that a more experienced player of the game may like to use champions that need more skills to control, while beginners like those are easy to control. Among 51,139 summoners, there are 1,014 summoners who have a

rank above 3. These summoners take up 1.98% of the whole dataset, which is practically negligible. Therefore, we will focus on the analysis of the summoners who have rank between 0 and 3 (both inclusive) We analyze the data and obtain Fig. 1. From this figure, we see that about one quarter of the users are highly experienced and skilled in this game because they have rank of 3. Slightly more than one-third of the summoners are in the middle range of the rank. There are relatively few summoners who are in rank 1. Lastly, there are almost 30% of summoners who don't have a rank. This group of summoners do not participate in the Ladder Tournament. Since we do not have direct information about their level of experience, we will assume they are average users whose rank is about 2 if we must use their rank in our model as a feature, which is historically a pretty accurate estimate for most players who doesn't enjoy ladder gameplay.

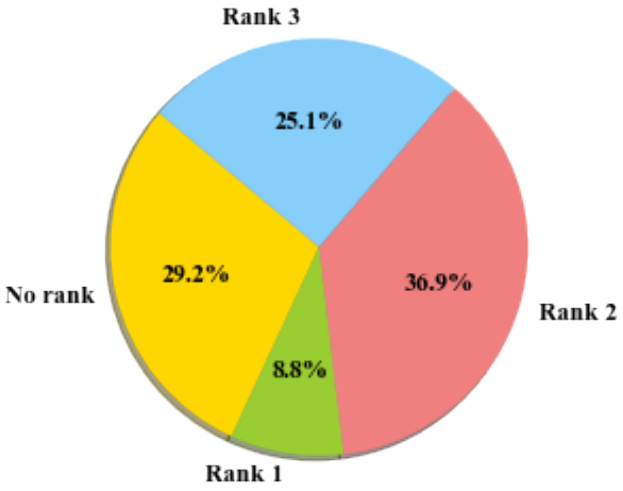

Figure 1: This pie chart shows the distribution of the summoners' ranks.

We suspect that a summoner would have a preference towards the same type of champions i.e. champions with the same tags. It's hard to check for every summoner whether this is true but we chose the first 10 summoners in the dataset to do a quick visualization. Fig. 2 shows the statistics of the first 10 summoners. It is seen that each summoner generally has a balanced distribution of the type of their champions. The reason for the even distribution is because of the fact that in every game, five players form a team that requires different roles to form a respectable composition. Thus, it is unlikely that players would only play certain positions since they would often need to fill needed roles in their teams. However, this does not mean that people doesn't have a preference, which is somewhat visible in our data. For most players, there is still a large difference between the champion type that the summoner has a high level and the one with a low level. For example, summoner 0 in this bar chart plays more champions of type "Fighter" than other types. Summoner 3 has a large number of type "Mage" champions played but the number of "Assassin" champions is relatively very small. Therefore, our original suspicion is valid to a limited extent. This observation may help us build a model with a decent performance.

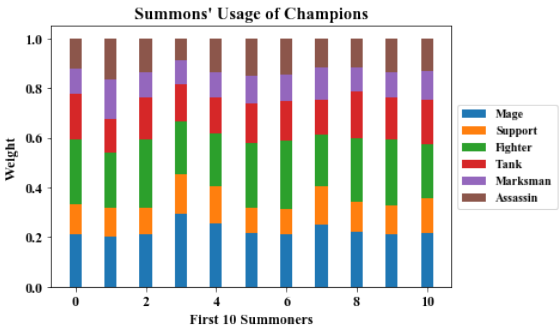

Figure 2: Stacked bar graph to show each summoner's champion usage.

Next, we analyze the properties of these champions. We hypothesize that there are champions that are relatively more popular than others if we consider the number of summoners who play a champion as the criterion for popularity. The analysis of their popularity would potentially allow us to find the good threshold if we want to design a model that recommends a champion to a summoner based on the popularity of the champions. Fig. 3 shows the result after we compute the number of summoners who have played the champion at least once for each champion and sort them. We see that the most popular champion is the absolutely number one in the ranking of popularity. Champion '22' (Ashe) which is the first one in the popularity ranking is played by 44,777 summoners in the dataset, whereas the second place Champion '21' (Miss Fortune) is played by 42,265 summoners. From the bar plot, we can tell that this difference is significantly large compared to the slow trend of decreasing popularity of the rest of the champions. From the third place onwards, the number of summoners who play each champions drops smoothly. Towards the last few champions in the sorted data, we see again the drop of popularity becomes relatively steep. This implies that there are a handful champions which are less preferred by the majority of the users. The least popular champion (Ivern) is played by 22,223 summoners. Comparing this amount to 44,777, which is the number of summoners who have played the most

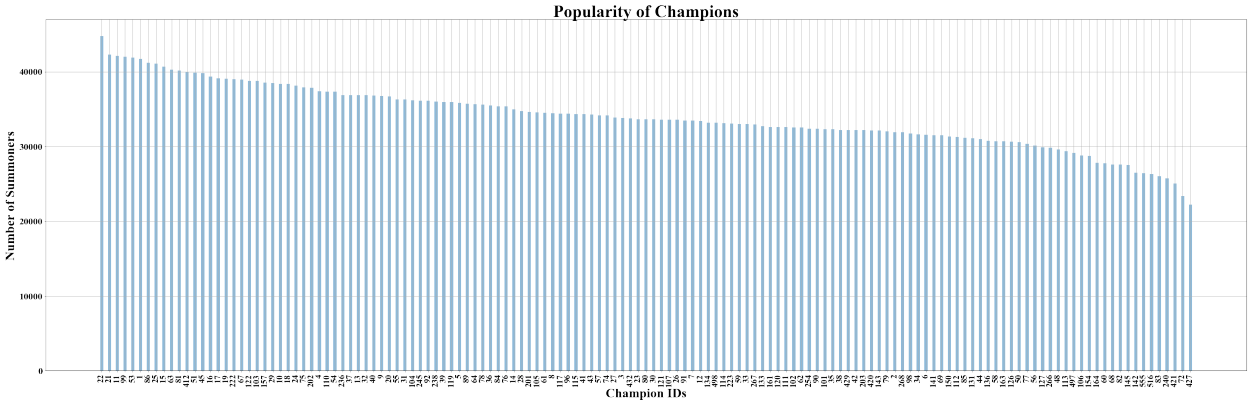

Figure 3: Bar graph of champions' popularity.

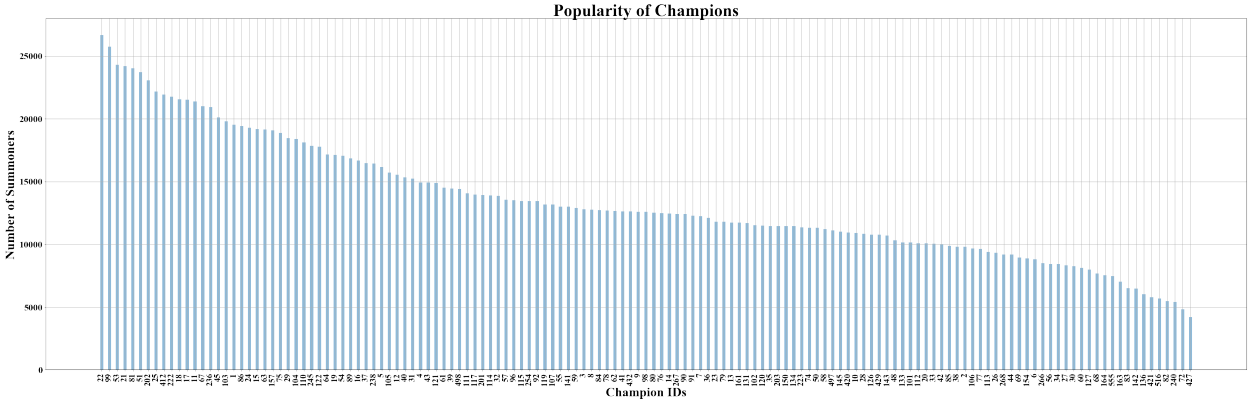

Figure 4: Bar graph of champions' popularity with the additional level restriction.

popular champion, we find this difference significantly large. The most popular champion is as twice popular as the least popular champion. Albeit this large difference between the most popular champion and the least one, it is still hard to clearly categorize all champions into clusters based on their popularity defined here because of the overall smooth change in popularity seen in the bar plot.

However, the above analysis of the popularity is not thorough enough. It's possible that a summoner tries to play a champion once or twice. Chances are the summoner uses the champion in the game but finds they don't like the experience overall. As a result, we can't conclude that this summoner likes the particular champion. Therefore, simply counting the number of summoners who have played each champion at least once may not give us an accurate representation of the champions' actual popularity. To solve the above problem, we count the number of summoners who have played each champion with an additional condition: the summoner's champion must have level higher than or equal to 3. With this, we obtain Fig. 4. From here, we see that the change of popularity is steep at first, and then it becomes quite smooth, until it reaches the last few champions, where the decrease in popularity becomes sharp again. This trend of change in popularity is illustrated in Fig. 5 more clearly. Based on this visualization, it seems reasonable to group the first 50 champions as the top popular champions, and within this group, the decrease in popularity is sharp from a champion to the next. Then the middle 70 champions are grouped together. This middle group has champions of the similar degree of popularity. The last 20 champions belong to the group of least popular and the decrease in popularity among these champions is sharp too.

3 Predictive Task

3.1 Overview

The goal of our model, as mentioned above, is to recommend champions that a summoner has never played before based on the historical data of champions that they have played. We have implemented three main ways to estimate the correlation (Feature based models, collaborative filtering and latent factor model).

These models sound reasonable solutions to this

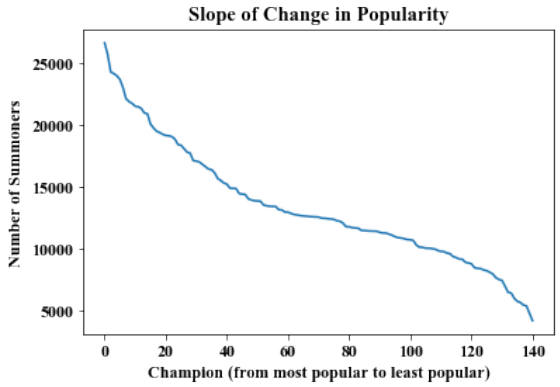

Figure 5: This line chart reflects the change of popularity from a champion to the next champion in the popularity ranking. Note that the numbers on the x-axis is not related to champions IDs. Rather, they refer to the rank of the champions. For example, the most popular champion corresponds to index 0.

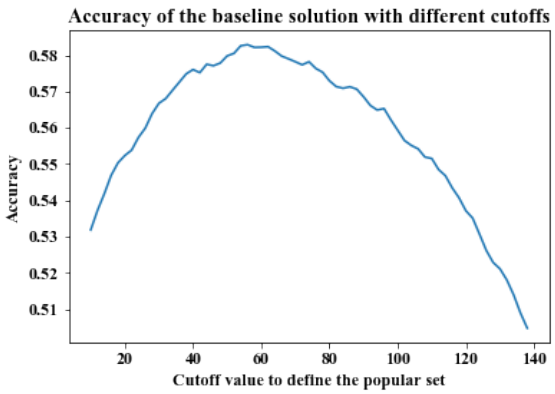

Figure 6: This plot indicates the different results of accuracy by varying the cutoff which defines if a champion is popular or not. It is seen from this graph that the best cutoff value is around 60, which aligns with the conclusion we made in the previous section that position 60 in the rank roughly separates the most popular group and the middle group.

task as we make a major assumption that summoners tend to have the habit to play similar champions in the game. This is the result of similar champions have similar difficulty levels and share some common tips of playing them well. Summoners, who are players of the game, will look for new champions who are similar to their favorite ones to seek new game experience but still maximize their sense of achievement in the game. Among these three similarity models, we will need to explore them using the validation set to check which one yields the best performance.

3.2 Baseline Solution

The baseline solution will be recommending champions based on their popularity. The idea is that if a champion is popular (i.e. played by many summoners), it is likely that a given summoner would also like it too. After experimenting various cutoffs to define if a champion is popular, we obtain Fig. 6. Using the cutoff value, 56, which generates the best result, we get an accuracy of 0.58287. We will then use this value to compare with other approaches as part of the valuation of other models.

3.3 Performance Assessment

Since the model is primarily user-centric, we divide the set of users (instead of the set of user-champion pairs) into training, validation and test sets. Every set constitute roughly 17,000 summoners.
Our task is mostly a binary prediction. It is possible for the model to predict mastery level instead of making a binary judgment, but we believe that it is mostly unnecessary. Therefore, we would primarily use recall as the main evaluation method for our models. The way we calculate the recalls is as follows: for every summoner in the validation and test set, we would hide around 20% of their champion pool from the model, and pass the rest into the model matrix. Then we calculate the recall by evaluating how many of the champions that were hidden gets recommended by our model.

3.4 Features[1]

- We will consider if a champion is popular in our models. This feature has already been considered and processed when we discuss the baseline solution. Essentially, we compute the number of times a champion is played frequently by all summoners in the dataset and rank this result for all champions to obtain a popularity ranking.

- We will consider the types of champions. In the game, each champion has 1 to 2 different tags. These tags are “Mage”, “Support”, “Fighter”, “Tank”, “Marksman” and “Assassin”. The tags reveal important traits of the champion which will large impact whether a summoner will like it or not. We can obtain the information about these tags from our dataset. We have gotten a json file which we can refer to for the tags of each champion.

[1]This section is applicable for only the feature based models.

- We will consider the properties of champions. Each champion has different properties. For example, they have various values of "attack", "defense", "magic" and "difficulty". These features seem to be important ones that affects a summoner's preference because players of this game tend to have a habit of either playing champions that are easy to play or playing champions that require more sophisticated skills. We can directly obtain this data from our dataset as we have a json file that contains the champions' properties.
- We will want to know the summoners' playing habits by analyzing their playing history. For example, we want to compute the number of times a summoner has played a certain type of champions. We need to process our data by counting the occurrence of each summoner-champion pair to get this data. We will also want to know the average values of each property of the champions a summoner has often played. We obtain this data by summing up the corresponding values of each property and dividing it by the number of champions a summoner has played before.

4 Model

4.1 Feature-based Similarity Comparison

4.1.1 Jaccard Similarity

Mathematically speaking, the model is based on the formula:

$$JaccardSimiliarty = \frac{|A \cap B|}{|A \cup B|}$$

We use three properties from the champion to complete this task. The first one is the popularity of the champion. We divide five clusters of popularity, to be specific, rank top 1/5 most popular played champions to least 1/5 played champions. The second one is the tag of the certain champion. For any champion, it will belong to these six tags, namely "Mage","Support", "Fighter", "Tank", "Marksman" and "Assassin". The third one is the "difficulty" of this champion, basically from 1-10. We construct a tuple that contains these 3 values of the champion, for each summoner, find the top 5 most played champion, and for each champion, use Jaccard similarity formula to calculate the score between this champion and the each rest of the champion. Find the champion with the most similar and recommend for the summoner. Do this for all five most played champions if there exists multiple highest similarity score champions, recommend them for summoner together.

- P: Popularity of a champion could be range from 1 to 5, with 1 meaning the top 1/5 most popular champions to 5 meaning the least 1/5 popular champions. This data is obtained by processing the popularity set mentioned in section 3.4.
- T: Tag of a champion could range from 1 to 6, with each number represent each type of the champion that has described in section 3.4.
- D: Difficulty level of playing of a champion, as specified in section 3.4.

The Jaccard Similarity equation is defined as follows:

$$SimilarityScore = \frac{[P_1 \stackrel{?}{=} P_2] + [T_1 \stackrel{?}{=} T_2] + [D_1 \stackrel{?}{=} D_2]}{3}$$

For each comparison, if two values are equal then produce a 1, 0 otherwise. So the Jaccard similarity would check the number of times a property is the same as the compared one, and would basically have 4 possible values. 0 indicates all three properties are different, 1/3 means one property is the same, 2/3 means two properties are the same, 1 means all are the same.

Jaccard Similarity model is very easy to implement, fast compute time, but solution is naive, will tend to recommend more champions to play than expected (the output number is usually large, contains 70% correct recommendations, and also 30% wrong recommendations), less accurate compared to other models. If the summoner is prepared to spend a lot of time on the game and try as many champions as possible, then this model would be a good fit.

4.1.2 Cosine Similarity

The cosine similarity is calculated to find how similar two vectors are. Mathematically,

$$cos(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||}$$

We make an assumption that a summoner will play a champion that is similar to the ones he has already played frequently. To select the set of champions which the summoner has played frequently, we pick the champions whose level is greater or equal to 3 among all the champions that this summoner has played. We choose this to be the criterion of "being played frequently" because a summoner needs to play around 10 to 20 times in order to have that champion raised to level

3. The main idea of this model is to use a feature vector to represent a given champion during prediction, and compare this feature vector with those of the frequently played champions. If the given champion's feature vector demonstrates a large degree of similarity compared to at least one of the feature vectors of the frequently played ones, we will recommend this champion to the user. Otherwise, we are not confident that this summoner will like the champion, and therefore we will not recommend it. Specifically, we choose the following properties to construct the feature vector for a champion:

- Whether the champion has a certain tag. Use one-hot encoding to represent this information.
- The properties of the champion. To represent this information, another 4 bits will be included in the feature vector as we use one-hot encoding again.
- The popularity of the champion. We calculated the popularity of each champion as described in section 2.2. If the champion appear to the the x^{th} one in the popularity rank, the corresponding bit in the vector will get the value of x.

Thus, we obtain a feature vector whose length is 11. Each champion will be represented by such a vector when we want to compute how similar two champions are.

Given a summoner-champion pair, we will compare this champion with every frequently played champion of this user. If at least one comparison indicates that the given champion is similar to the existing frequently played champion, we will predict the summoner will like this champion and thus, recommend. Similarly, if the given champion is similar the the ones that the summoner has played before but only have low levels (less than 3), we can conclude that the summoner will not be interested in playing this champion, thus do not recommend. We will set the threshold of distinguishing being similar from being dissimilar by experimenting the data to yield the best result.

The strength of this approach is that getting the necessary information only needs one iteration of the whole dataset. After that, creating a vector for a given champion will only take constant time. Furthermore, this approach demonstrates quick calculation in the process of prediction because each user typically has 20 to 30 champions that they have played, causing there to be a manageable number of comparison to be made every time.

The weakness of this model is that it currently performs very well in our validation set. Without an actual test set, we may have the risk of overfitting our validation set by selecting these features.

4.2 Collaborative Filtering with Pearson Similarity

We then consider a Pearson Similarity based approach that solely considers the summoner-champion correlation as its data. This part of our model features a memory-centric collaborative filtering approach that does not consider any artificially defined features.

4.2.1 Model Implementation

The model is very simple. Before the training process, we construct the summoner-champion table which considers mastery 7 as significant interest, mastery levels 4-6 as interest and ignores other data (as explained previously, due to the nature of the game, players might not necessarily be showing any interest to the champion if they only picked it once or twice). Then, we used collaborative filtering to generate all missing values in the summoner-champion table. The model was set up so that it operates as a champion-based filter instead of a user-based filter.

4.2.2 Model Performance

We analyzed the performance in two ways: statistical analysis and manual inspection. Inspecting results heuristically was useful as a sanity check for the model, as while it's difficult to evaluate whether or not the given predictions are satisfactory for a player from a mathematical standpoint, from a human perspective it's often easy to determine whether or not the output is expected.
The model performance was satisfactory in both parts:

- Statistically, the model had decently high recall with a tight threshold (we set the accept interest threshold as half of the maximum recommender score for this user), with around 75 percent for "interested" champions and near 100 percent recall for "significantly interested" champions. Since for the summoner-champion pairs that are designated in the "interested" category, many noisy situations can happen (such as champion reworks, role filling for a team composition, or simply experienced players experimenting with a high variety of champions), we believe that the current stats showcase an improvable yet healthy model that did not overfit to its data.

- As a performance testing, we manually checked the results of around 50 summoners, as well as tested the model on friends that have experience in the game. The recommended results are surprisingly decent, as in most of those cases, the recommendation not only provided recommendations within the type/role that the user liked to play, but also showed ability to incorporate some "human-like" evaluations in its recommendations, such as playstyle, difficulty, viability at different ranks and even the art style of different champions. Obviously, this is a very naive analysis on a small portion of data, but for the model to show these results with no pre-determined features being given to it is a reassurance of its viability.

4.3 Latent Factor Model

In this section, we consider a matrix factorization based approach that attempts to find underlying latent features that can represent and predict the dataset.

4.3.1 Model Preliminary

We employ a simple form of matrix factorization as the approximation goal model. Let F_U be the summoner-feature correlation matrix, F_I be the champion-feature correlation matrix and M denote the summoner-champion pair matrix, we want to find such feature matrices so that $F_U \times F_I^T \approx M$.
Evidently, the estimate value for a certain summoner-champion pairing can be approximated by the matrix. We therefore want to minimize the total of the mean-squared error of such pairings, which is given by $E = \sum_{i=1}^{n} \sum_{j=1}^{m} e_{ij}^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} (M_{ij} - \sum_{k=1}^{o} F_{U,ik} F_{I,kj})^2$. We then calculate the mathematical gradients of the error with respect to $F_{U,ik}$ and $F_{I,kj}$. Finally, we aim to minimize the total mean squared error by using stochastic gradient descent on the two objectives alternatively.

4.3.2 Model Implementation

The implementation of the model is not complicated, as we simply implement an alternating gradient descent that stops when converged or after a number of steps. For this model specifically, we decided to pre-filter the dataset for players that played more than a certain number of champions to make sure that the dataset is well-formed and not too sparse to cause issues.

4.3.3 Model Performance

As expected, the performance of the model has to do with how many latent features was employed. We ended up deciding that 10-20 latent factors are healthy amounts, since a number below that tend to underfit to the summoner (i.e. basically recommend the most popular champions), and numbers over that is too time-consuming and offers little improvement.
It is difficult to evaluate the performance of the latent factor model. Primarily, this is because of a difficulty in setting a meaningful threshold. When using the same half-max threshold idea as collaborative filtering, we find that the recall is slightly lower, and the "significant" recall is much lower; in many cases, a highly played champion has a pretty low evaluation score. This could be possible due to the nature of the model, since outliers (where a player mostly plays champions belonging to some category but plays one champion that is very different) are hard to account for without enough latent features. On the other hand, the overall non-zero mean squared error is comparable to that of collaborative filtering.
We also did an evaluation of manual checking of the style mentioned in 4.3.2, and found that while the results are mostly relevant and similar to the ones given by collaborative filtering, there are often weird singletons where the suggested champion isn't really of the style of anything that the summoner has played. In most of the cases, that suggested champion is a popular champion in the dataset. This could be part of the reason why these things happen.

5 Literature

Incorporating recommender system into games to improve players' experience is a relatively new research area [1]. It is not surprising that few efforts have been made to analyze a dataset like what we have in this assignment. Although RIOT API makes this data publicly available, we have not seen any similar projects that predict the preference of a user towards a given champion.

Reference [1] also collects players' information in a game and analyze it to make recommendations of items in the game to encourage users' purchases. From this paper, we find a few things that are similar to our findings, but we also find different observations as well.

Similar findings include that training data should be able to divide in mini-batches to ease the process of training but still makes the results scalable so that the resultant model remains applicable to the actual

games, which usually involve millions of users. Despite this similarity, the implementation of the model described in the paper to achieve this goal is different from ours. The model from the paper ensures scalability of the data by doing subsampling during training and produce a final ensemble by "combining many such subsets". However, we make sure our model still works simply by reading the available dataset once and choose a few representative features which allow us to make predictions. This difference is mainly a result of the different models we choose to use. The paper uses a deep neural network to make predictions, while our model is doing something much simpler - computing feature vector similarities.

One likely drawback in all our models is that we can only make good prediction if a summoner has played a decent amount of champions before. For a new player in the game, it will be extremely hard for us to come up with an accurate prediction. A more traditional approach to generate recommendations may likely improve this situation, which is a knowledge based system [2]. A knowledge-based recommender system does not depend on a base of user ratings or history. It actively asks a user to indicate their preference so as to narrow down the discrimination tree and yield an accurate recommendation eventually.

6 Results

In this part, we would combine and compare the performance of the three proposed models to discuss their relative strengths and weaknesses. Primarily, we would be looking at their recall performance on the test set. We're also going to explore the nature of the feedback: how well they perform when conforming to different needs (i.e. recommend a champion for each position), how well do they operate for different users (e.g. new players with a sparse data and old players with a big but noisy data), and how well it actually performs in reality. While it is difficult to evaluate these models without commercial testing and feedback, we believe that the above evaluation methods would provide some insight towards choosing and/or combining the approaches of these models to improve the performance moving forward.

6.1 Statistical Analysis

The recalls of the three models are comparable. The feature based model (with cosine similarity) had a recall of 0.70 on the validation set and 0.69 on the test set. The collaborative filtering model had a recall of 0.76 over the the two sets. The latent factor model with 20 latent features has a recall of 0.73 on the validation set and 0.69 on the test set. Overall collaborative filtering gave the best performance, but this is obviously subject to the chosen recommendation threshold, as well as the small sample size.

6.2 Scalability

In terms of calculation time, the feature based model is the fastest, followed closely by collaborative filtering. These two doesn't seem to show too much problems when increasing the dataset size. Obviously, if we further increase the dataset size (e.g. with a million users), feature based model would be much better as its cost does not scale with data size. Latent factor model is the least scalable one of the three, showing significantly worse runtimes as well as runtime increments when we change the dataset or increase the number of latent features.

6.3 Generalizations

We understand that if put into practical use, users would have various different needs. Thus, we want to look into how well the models perform when used on different purposes.
If the user is trying to look for champions in a position that they played a lot previously, collaborative filtering model should do the job quite well. However, if the user is trying to branch out and explore new options that are not quite similar, latent factor model is much better as it provides deeper insight in similarity between users. Finally, if the user can provide examples of champions that they do *not* enjoy, feature-based models are a fantastic tool as it makes an estimate on negative feedback as well as positive feedback, so having such information drastically increases its viability.

6.4 Some user testing

We invited three friends to test their data on the three models and gathered some feedback. All of them considered collaborative filtering to give the most accurate results, while feature-based is thought to be good but somewhat misleading (as it's affected by arbitrary judgment when constructing the feature vectors), and latent factor gave the wildest results with interesting suggestions.

6.5 Final thoughts

Overall, we consider collaborative filtering to be the most applicable model of the three since it showcased a strong performance overall and doesn't seem to have too much weaknesses. The feature based model is fast and very scalable, but is limited in what insight it can provide. The latent factor model has the highest potential, as we probably didn't implement it to the fullest potential, and it is also quite useful in certain scenarios.
We believe that a good model is probably combining the three, either functionally (by using different models) or mathematically (by sequentially applying the models).

References

[1] P. Bertens, A. Guitart, P. P. Chen and A. Perianez, "A Machine-Learning Item Recommendation System for Video Games," IEEE Computational Intelligence and Games (CIG), 1–4, 2018.

[2] R. Burke, "Knowledge-based Recommender Systems," Encyclopedia of Library and Information Science, 180–201, 2000.