LoL Character Recommender

Marco Tan

Github repo

Google Drive

Original Plan

The original plan for my project was to create a restaurant recommender that would take a Google or Yelp user's ratings of restaurants and use those ratings to recommend new restaurants to the user. I planned to create a model similar to the YouTube recommendation system except instead of recommending videos, I would recommend restaurants. However, I could not call Google and Yelp APIs enough times to create a sufficient dataset and was also unable to find a public dataset with Google or Yelp review data.

Pivots

I pivoted from a restaurant recommendation system to a League of Legends character recommendation system since I had trouble finding good data for restaurant reviews. Instead of recommending a list of restaurants, I decided I would recommend champions (playable characters in League of Legends) based on a user's champion mastery data. Each time a player plays a game using a certain champion, they gain champion mastery points for that respective champion. This means that champion mastery data corresponds to playtime on the champion the points are assigned to. I got the data from Riot Game's API which has all champion mastery data for individual users. I pulled data from users from the ranked player base on the North American server.

I decided to make a champion recommendation because like restaurants, there are different types of characters which cater to specific people. There are several different roles to choose from in League of Legends and within each role there are different classes of characters to choose from. For example, within the mid lane role you can play supportive characters such as Karma or Lux or you can play an Assassin such as Zed or Akali. Additionally, there is a range of champion affinity among users. Some people exclusively play one champion and are dubbed one trick ponies while other people exclusively play ARAM (All Random All Middle), a gamemode that randomly selects your champion for you (with the option to trade with your other 4 teammates).

Originally I had planned to make an application that would ask the user for their riot ID and then output recommended champions for them after making an API call to Riot to get their champion mastery info. This could work in practice, but would require retraining the scikit surprise models everytime a new user was presented. This is because the scikit surprise models need the user’s rating data in its model to make predictions based on the user’s rating. I also decided not to do this because Riot API tokens only last for 24 hours, so running the code would require a good amount of setup. Instead, I created a user pool that can be randomly selected from. The users in the user pool only have the top 7 champion mastery scores in the database as opposed to all 165 champion scores registered. My program outputs a list of recommended champions and how much overlap there is with the user’s next 7 most played champions (that are not in the database).

Project Description

My project produces two League of Legends champion recommender models (kNN and SVD) from League of Legends champion mastery data. The mastery data was collected from Riot Games' API and contains 2986 users with champion mastery scores for each champion (n=165) per user. In the dataset, each user's maximum champion mastery scores was set to 1.0, so each user's individual champion mastery scores were divided by the user's maximum champion mastery score. The outputs of these models can be compared in example.py. This program loads the kNN and SVD models, then randomly chooses two users from the test sets and outputs the sample recommendations for those users.

Since scikit surprise requires that users be in the training dataset in order for them to be recognized for predictions, I put full user data (165 scores) for 90% of users into the dataset and then put the top 7 champion's master data in for 10% of users. In the example.py file the program selects two random users from the poll, calculates an expected rating for every champion not present in the model, and outputs the top 7 champions with the highest expected ratings.

These outputs are built by ordering the champion mastery score predictions from the kNN and SVD models on the champions not present in the dataset. An example output can be seen below.

```
Algorithm SVD
User ID: dkMTqn1HsyBz5imFuVVPXsM10ztfTr66Y1BeN8Rtk7fRQGqo
Most played: ['Yuumi', 'Lulu', 'Soraka', 'Janna', 'Lux', 'Seraphine', 'Nami']
Recommended: ['Ezreal', 'Yasuo', 'Irelia', 'Yone', 'Kaisa', 'Vayne', 'Kayn']
Actual: ['Cassiopeia', 'Leblanc', 'Syndra', 'Sona', 'Senna', 'Karma', 'Zoe']
Overlap: []


Algorithm KNN
User ID: dkMTqn1HsyBz5imFuVVPXsM10ztfTr66Y1BeN8Rtk7fRQGqo
Most played: ['Yuumi', 'Lulu', 'Soraka', 'Janna', 'Lux', 'Seraphine', 'Nami']
Recommended: ['Morgana', 'Sona', 'Karma', 'Leona', 'Zyra', 'Neeko', 'Ahri']
Actual: ['Cassiopeia', 'Leblanc', 'Syndra', 'Sona', 'Senna', 'Karma', 'Zoe']
Overlap: ['Sona', 'Karma']


Algorithm Random
User ID: dkMTqn1HsyBz5imFuVVPXsM10ztfTr66Y1BeN8Rtk7fRQGqo
Most played: ['Yuumi', 'Lulu', 'Soraka', 'Janna', 'Lux', 'Seraphine', 'Nami']
Recommended: ['Garen', 'Ahri', 'Kayn', 'Samira', 'Fizz', 'JarvanIV', 'KSante']
Actual: ['Cassiopeia', 'Leblanc', 'Syndra', 'Sona', 'Senna', 'Karma', 'Zoe']
Overlap: []
```

The project also contains .ipynb notebooks for visualizing the dataset and measuring performance of different various models in scikit surprise.

Self Evaluation

I expected the kNNBaseline model to give the best recommendations because it produced the best metrics during cross validation among the models in scikit surprise. I used 5-fold cross validation to get an FCP score and RMSE score for each algorithm on a smaller dataset of 750 users.

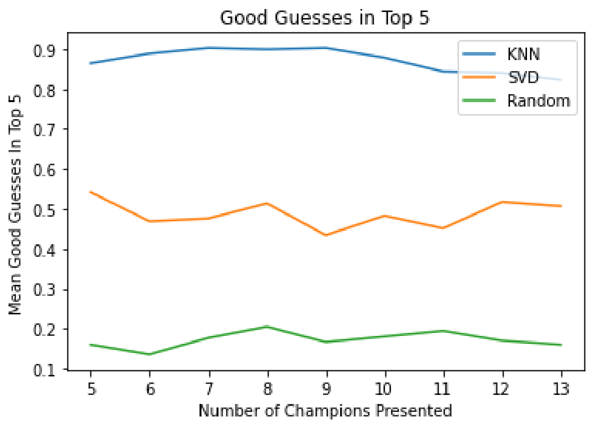
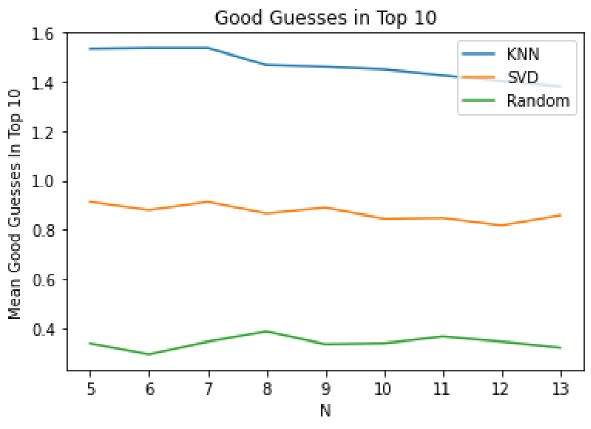
The Fraction of Concordant Pairs (FCP) score returns the proportion of known ratings for a user that are in the correct order compared to the total number of pairs. For example, if a user rated Garen a 0.3 and Yone a 0.1, and the model predicted Garen's rating to be 0.2 and Yone's as 0.15, then they would be in the correct order. This metric was the most crucial element for my problem since the actual ratings were not too important for my use case, but the ordering of ratings was.The results can be seen below.

Model	FCP	RMSE
Baseline	0.650	0.121
Co-Clustering	0.463	0.138
KNN	0.679	0.117
KNN Baseline	0.682	0.115
KNN With Means	0.681	0.115
NMF	0.658	0.134
Random	0.443	0.164
SVD	0.617	0.123
SVDpp	0.642	0.121
Slope One	0.610	0.124

I decided to continue using scikit surprise after seeing a similar error distribution to the Movielens 100k example data.

I also tested my data on the SVD model and Random model so I had benchmarks to compare the kNN model to. The random model randomly draws from a champion's rating distribution to output a prediction for each champion rating of a user. I chose the

SVD model because it was the best performing model for the Netflix Prize competition, although it was not the best model for my dataset based on its FCP and RMSE score. I wanted to see how useful my models were for my actual use case, so I checked how many guesses were relevant for each recommendation. I presented the model with full data for 90% of users and for the other 10% I gave the model the top N champions for that user and saved the user's top 5 champions that were not presented. I then made champion recommendations for the 10% of users who did not have full data in the model and counted the average number of champions that had overlap between the user's top 5 champions not present in the dataset and the system's expected top k champions not present in the dataset for the user. This number was called the number of good guesses.

 k = 5	 k = 10

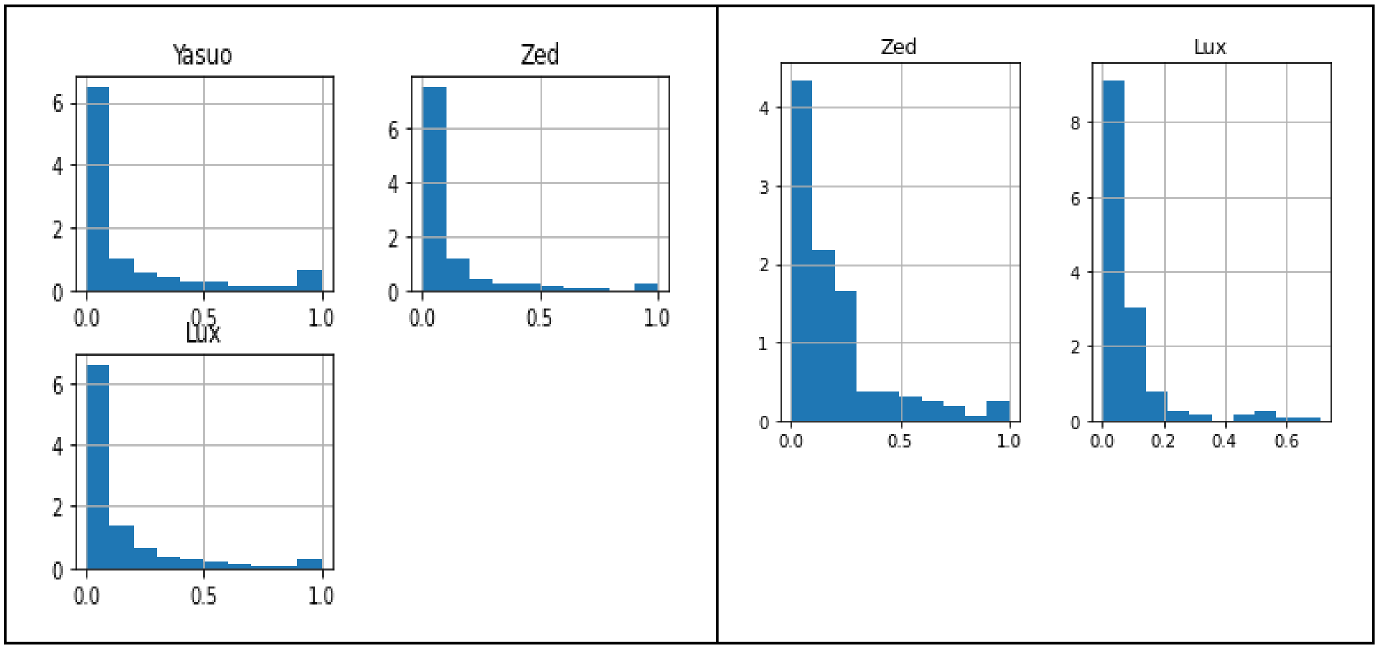
The kNN recommendation system recommended good guesses in the top 5 and top 10 recommendations about 5x more often than the Random model. It also made good guesses a little less than 2x more often than the SVD model.

I also counted the number of unique champions recommended in the 10% of users with hidden data to see how varied the recommendations were from my models. There were 283 users that got recommendations. This analysis revealed that SVD only recommended less than 10% of champions, while KNN recommended about 85% of champions, although it did not recommend every champion. There are 165 champions in total.

	SVD	**KNN**	**Random**
Unique Champs	13	140	165

The kNN recommendation system was clearly the best choice for my recommendation system. Although it did not produce perfect results, it was likely to recommend a relevant result in a list of 5 recommendations to the user. The SVD system was too narrow in its recommendations, since it seems it was not making recommendations mainly based on user preference, but instead based on champion popularity.

I think my recommendation system did fairly well considering the randomness of champion affinity in League of Legends. I looked at the effect that playing one champion had on the mastery score of an opposing champion by looking at Yasuo players preferences and how they compared to the general player base. The graphs below show the distribution of Yasuo, Zed, and Lux champion mastery scores among the general player base compared to the distribution of Zed and Lux champion mastery scores for Yasuo players. Although Zed's mastery score trends higher for Yasuo players and Lux's mastery score trends lower, the distributions do not change drastically. I think the system recommends the most logical champions that a user would play based on their most played champions, but the likelihood that they actually play those champions is fairly low.



In the future, I would move to a different KNN package which would let me input user data without being tied to a user in the system so I could input an unknown user without having to retrain the model. I would also get a developer Riot API key so that I could train on a much larger dataset since I know this model has potential now. This could help me make better recommendations for niche users.

I also think I could improve model performance by using a user’s ranked playrate instead of champion mastery score. This would cause the recommender system to work better for ranked players but worse for casual players. However, it would work better since mastery score is permanent, so someone who switches from a bottom lane player to a middle lane player would have a weird mix of marksmen and mid lane champions in their data even though the user now prefers mid laners. Ranked data is only saved for a season which lasts about a year, so it gives a better profile on what a user enjoys at the time and would probably make for better data on user preference. This would also help with parity for champion recommendations since mastery scores are skewed towards older champions since they have had more time to accumulate throughout the player base. Since ranked game info is divided by seasons, we would be able to look at a user’s champion playrate in a more controlled and even environment.

How to run my project

To look at the model in action:
Run src/example.py

To train and run the model:
Run src/training.py
Run src/example.py

To look at the dataset then see how it performs on various models:
View src/visualization.ipynb
Run src/processing.ipynb (optional)
View model_selection.ipynb
View rank_metric.ipynb
Run src/unique_champions

What each file does

Files for using the model:
src/training.py - build dataset and user pool, then train models

src/example.py - example of model in action on users with partially hidden data

Files for testing model performance:
src/unique_champions.py - prints number of unique champions per model

src/rank_metric.py - checks how many champions are recommended from top 5 champions not in dataset and saves results to csv

src/model_selection.ipynb - Cross validation for choosing model and choosing parameters for SVD and kNN

File for visualizing model performance:

src/rank_metric.ipynb - Visualizes results of tests on champion recommendation system, including number of relevant guesses

File for visualizing dataset:
src/visualization.ipynb - explores champion mastery score distribution and whether certain champion players prefer other champions

File for processing dataset:
src/processing.ipynb - creates full dataset and smaller dataset for data analysis

File for gathering data:
riot_api.py - make requests to riot API, requires token

Data
all_data is for the dataset where every mastery score was collected from users
25_data is for the dataset where 25 mastery scores were collected for each user

test_data - results of various tests from ipynb notebooks and python scripts

mastery_data.json - dataset where 25 mastery scores were collected for each user (5.9k users)

complete_mastery_info.json - dataset where all mastery scores were collected for each user (2.9k users)