Project Summary – Music Recommendation System

Danish James

Project Design

The intent of this project was to create a music recommendation system that would function on the basis of collaborative filtering. The system would intake a user, compare their listening tastes to other users in the database, then make a recommendation for new music based on the most similar users. In particular, I wanted to make a multi-step process, where the first step would be user-based collaborative filtering, and then the second step would be an item-based content filter. Unfortunately the second step could not be achieved within the time allotted.

For the first step of this project, I needed to build up a user database as well as items (songs) and ratings. For the user database, I decided to utilize last.fm as they provide open access to users listening history through the API so long as you have a username. Getting the usernames was slightly tricky, but I found that while logged in, you can see the top followers for a particular artist. I built a scraper using selenium to iterate through the top followers page for several selected sets of artists and built up a list of approximately 7,000 users. Then, utilizing the last.fm API and the pylast python wrapper, I collected the top 750 tracks for each user as well as the amount of times each song was played. All of this data was stored in a mongodb collection. I also generated a rating system by taking the maximum plays of a single track for each user, dividing the play counts for each song by the max plays which produced a value between 0 and 1. As the EDA would show, the ratings were very heavily weighted towards the lower end as most people will listen to a wide variety of songs a few times, and a few select songs a significantly larger amount of times. This resulted in an uneven distribution, so I resolved this by taking the log of the ratings then adding a constant to make sure all ratings were positive. The final ratings varied between 0 to 10.

For the model, I needed a metric to understand how each model was performing, and while I didn't have a true accuracy metric to work with, I could utilize error as a metric. In order to do that, I needed a baseline to compare to. So I created a dummy model that only predicted the average rating of the entire dataset and calculated the RMSE on the model using a 75/25 train-test split. After cross-validating my results, I ended up with an error of about 1.3. That became my goal to beat in order to consider the model worthwhile. I tried a few different models such as a NormalPredictor which tried to map each user's ratings to a normal distribution and then estimate new ratings based on the average z-score for each new song. That model functioned very poorly, coming in at an RMSE of 1.9. I also attempted to use KNN, however in order to utilize KNN the model needed to construct a similarity matrix which ran into memory issues once the dataset started to get larger and larger. It was impossible to use it for more than 500 users at a time, so I chose not to proceed with it. I finally settled on the SVD model, which produced an RMSE of 0.7 which is almost half of the error that the dummy model produced.

Tools

- Python
  - Pandas, numpy
  - Pylast, pymongo, selenium
  - Surprise
- Mongodb
- Last.fm API
- Microsoft Powerpoint, Microsoft Word

What I Would Do Differently Next Time

I think in retrospect, I did not plan out how I was going to store and format my data properly. I started off with just the last.fm data, and I did not think about the difficulty in attaching a spotify track ID to the last.fm data. Once I actually got to the stage of trying to make connections, I realized that the spotify API search tends to have a lot of issues with making searches when the query is not perfectly in line with what it is expecting. For an example, there were a few songs that last.fm had with “radio mix” in the title, and the simple inclusion of that phrase caused the spotify search function to fail in finding any matches. It took a lot of time trying to clean up my track data from last.fm so that Spotify would be able to find the songs and match them to a track id. If I were to do this again, I would start with attaching the track ID from the start and just remove tracks that I could not find. It would trim down my dataset by a fair amount, but it would greatly simplify the second step of the algorithm.

In addition, I would also do more research into the documentation before I jumped directly into attempting to model. I tried to follow the guidelines in the tutorials without fully understanding what I was doing, and it made it difficult once I realized that the model and approach I was looking for could not be accomplished with the model I had already set up. The alternative model that would have provided the approach I was looking for, the KNN model, could not be completed on my device as the resultant similarity matrix the model builds consumed too much memory for my machine to handle. As a result I had to stick with the SVD model and make the best of what I had available.

Finally, though this was not entirely my fault, due to my laptop's charger breaking around the final week of modeling, I had to go back and re-scrape my entire database on a different machine to work with while I was waiting for my new charger to come in. Luckily, I had already backed up the code onto github. However, it was still a problem as I only had code backups and did not have backups for the database that I was using. For the future I would like to look into fully moving all of my work onto the cloud and working remotely via ssh into AWS or a different cloud service like Azure. This would allow me the ability to work on projects no matter what happens to my local machines.