Comparing Music Recommendation Techniques

By Christopher Kozich, Jeremy Cohen, and Zackary Hillman

Github: https://github.com/zackhillman/next-song-recommender

Abstract

In this paper we seek to compare the performance of 2 different AI techniques to perform a music recommendation task. The 2 techniques used were a collaborative filtering approach with matrix factorization, and a content based filtering approach with word2vec. We thought these results would be meaningful as the 2 approaches differ in that collaborative filtering focuses on user music preferences and behaviors and tries to group users together with similar music tastes and recommend music that is similarly liked in these groups while content based filtering focuses on specific song characteristics and tries to recommend songs with similar characteristics. To directly compare the techniques we performed an HRI experiment where it was found that the content based technique did a better job at recommending songs. Though due to differences in data and a small number of participants in the experiment, we are hesitant to conclude that the content based approach is the superior music recommendation technique.

1. Introduction

Spotify is a music platform that all of us use regularly. One of the features in Spotify is that song recommendations will appear under playlists you have made. As a group we were curious how this recommendation system works, and what sort of ways have been explored to accomplish the task of music recommendation. After conducting some preliminary research, we found that techniques for music recommendation generally fell into 2 camps, collaborative filtering and content based filtering.

Collaborative filtering in the lens of music recommendation is a technique that relies on user music preferences and behavior to recommend songs. For example if Spotify sees a group of users that generally have the same music tastes, and one user in the group hasn't yet played a song that all other users in the group have played and enjoy, then they might recommend that song because generally the group of users have the same music tastes and preferences. Content based filtering in the lens of music recommendation is a technique that relies on the specific characteristics of the songs the user listens to. For example if Spotify sees a user that listens to only rap music then it is more likely to recommend other rap songs rather than a genre like country music. We thought it would be meaningful to compare the performance of both these recommendation techniques as they inherently filter based on different metrics, and as a group we were interested in seeing which technique would perform better overall.

For collaborative filtering we collected data via a Google Form that included a set of songs which we told people to rate from 0 - 5. Then we used a matrix factorization model to recommend songs. For content based filtering we collected data by scraping the Spotify Million song dataset. Then we used a word2vec model to recommend songs. Finally, we performed an HRI experiment comparing the 2 techniques which is detailed

in the experiments section. Below are more details about our methodologies, experiments, and results. The specific project topics that we did are, Running HRI experiments with human subjects to determine what results they prefer (Ran our own HRI experiment), Collecting your own novel dataset - for example, with a survey (Collected data via a google form for collaborative filtering technique), Making use of existing machine learning implementations, such as k-nearest neighbor in scikit-learn or code you found for a more advanced method (matrix factorization for collaborative filtering), Collecting a novel dataset that requires significant cleaning or effort to collect (Scraping spotify API for playlist data), and Trying out different neural network architectures using a framework like Pytorch or Keras (Using word2vec for content based filtering).

2. Methodology

The first thing we did for our matrix factorization method was to sample some very preliminary data from close friends, and use this as some very basic baseline to get everything working off of. Because a major drawback of matrix factorization is the *cold start problem*, we knew we would have to skew our dataset to ensure a reasonable amount of candidate songs that we would expect most of our sampling demographic to know. As such, we assembled a preliminary dataset composed of some random songs, and some intentionally selected as likely baseline songs for our intended demographic. We sent out a form of about 40 songs, asking users to rate each song 1-5, or 0 if they had never heard the song before, and got about 60 responses. Next, songs that received too few unknown ratings were removed from the dataset. Then, we adjusted the hyperparameters for the factorization several times, and used user feedback to justify which hyperparameters worked best - though this conclusion was certainly reached

through a limited amount of available data. Next, about 45 songs were added, and 10 of the least known songs removed, from the dataset; of the added songs, emphasis was placed on ensuring the majority would be well known “classics” so as to strengthen our average listening profile before making recommendations. We then sent the new form out, and recorded 135 responses. This data was then cleaned for duplicate entries, entries that misused the rating system, and any other problems. This cleaned data was then converted into a list of lists, each list corresponding to a user’s ratings of each song, and that data was then used as the input to our *generate_scores(data)* method. This method used matrix factorization and the corresponding dot product of the user and item matrices to produce our model’s predicted scores for each song per user. To make recommendations, we then looked at every song a user marked as unknown, and returned those songs with the highest predicted score for that user. The output is a list of five tuples, in order, of the best recommendations and their respective predicted ratings.

Matrix Factorization: A Simple Tutorial and Implementation in Python - Matrix factorization code

https://forms.gle/i7bpQ5pTD1EUrmVr7 - Form for data collection

Boosting: why is the learning rate called a regularization parameter? - Reasoning about hyperparameter calibration

The first thing we did for content based filtering was trying to figure out how to get our data. For this we turned to the Spotify API and scraped and cleaned our own novel dataset. We scraped playlist data by looking up top users with a large amount of public playlists. Once we had the data, we attempted to determine which data to keep

and which data to drop. First, we dropped duplicate songs and songs with no artist or track name. We then grouped the songs back into their respective playlists and dropped any playlists that were empty. Next, we looked at playlist length. We found that the majority of playlists fell between 3 and 200 songs. Finally, we looked at removing playlists that obviously had no correlation between songs. From looking at the top playlist names, we dropped playlists that were named: starred, liked from radio, my shazam tracks, etc. After cleaning, we had 200,938 unique playlists and 1,570,980 unique tracks across all playlists.

We used Gensim's Word2Vec CBOW to derive vectors for each word (song) in our sentences (playlists). The input is a list of playlists where each playlist is a list of song id's. We took out 5% of our playlists to be used as test playlists. The output of our model is a mapping of song id's to vectors. We landed on the following hyperparameters: vector size of 256, window of 15, and negative of 5. The vector size is how many dimensions the model will encode each song in. The window is the maximum distance between the current and predicted song in the playlist. The negative tells the model how many "noise" words should be drawn. The end product takes in a list of songs and then outputs a list of songs with a similarity rating representing the cosine similarity between the average of all songs in the playlist, and the recommended song.

Word2Vec Documentation - Documentation for Gensim's Word2Vec

RNN-music-recommender - Github Repo used for inspiration on cleaning scraped data

Song2Vec Paper - Paper used for inspiration of using Word2Vec on playlist data

3. Experiment and Results

For the matrix factorization, we confirmed our methods effectiveness by receiving feedback from about 35 of the users who received recommendation. These users were not given the predicted ratings of those songs, and were simply asked to rate them in the same manner as the original dataset. Overall, the predictions appeared to be quite accurate, with several significant outliers - but the overall accuracy (average similarity between predicted and actual scores) was about 87%.

For our Word2Vec model, the output is a mapping of songs to vectors. These vectors, by themselves do not mean much, and therefore there is no clear way to discern whether the output of the model is correct or not. Despite this, we validated our results to the best of our ability. We decided to mask each song, one at a time, in each of our testing playlists. Then we calculated the hit rate as: how often does our model, which returns 30 likely candidate songs, recommend the masked song as one of the 30, given the context of songs around it.

First, we ran on our scraped data from spotify. The following table represents different experiments we ran. The rating scores were our arbitrary attempts at quantifying how well we thought the recommendations were. Unfortunately, running on google collab took 4+ hours to run causing this process to be excruciatingly slow.

Attempt #	**Size**	**Window**	**Negative**	**Hit Rate**	**Rating 1-5**
1	128	10	10	0.03588	2
2	256	10	10	0.08329	3
3	256	15	10	0.10111	4
4	256	20	10	0.09483	3
5	256	15	5	0.12700	4

We can see that even though we have *hit rate* as a way to compare different parameters, we have no context as to how good or bad a 3-13% hit rate is. To gain a better understanding, we decided to try out randomly recommending songs which gave us a hit rate of 0.00167. This means that our model is performing better than a random recommender.

Next, we decided to run our model on a premade spotify dataset. In 2018, Spotify released a million playlist dataset for this exact purpose. We used 200,000 of those playlists resulting in 679,321 unique songs. We used our best hyperparameter tunings again, and got a hit rate of 0.2104! In the future, we want to run more experiments using this pre cleaned dataset.

The last few experiments we ran were related to tuning the minimum number of playlists a unique song needs to appear in in order to be considered valid. We realized that this was set to one. When increasing this to two, 67% of our scraped songs are ignored. When trying this change with the pre cleaned dataset from Spotify, only 13% of the unique songs are ignored. This might be the cause of the poor performance from our scraped dataset. If a song only appears in a single playlist, that single playlist could drastically affect the resulting song vector.

To compare the two recommendation techniques we conducted an HRI Experiment as follows. We selected a subset of the people who filled out the google form (n=20). For each person we played them 4 songs randomly selected from the subset of the 75 songs used for the collaborative filtering method. We then fed these 4 songs into each algorithm and played the top 2 songs from each. The participant then selected the 2 songs that they thought were the most like the original 4 played. From this we

observed that 13 people thought the songs that the content based filtering approach recommended were better recommendations and 7 people thought the songs that the collaborative based filtering approach recommended were better recommendations. Still, we don't think we can conclude that the content based technique is superior, as it had a larger dataset to work with and the number of experiment participants was small. As such, we do not think these results are significant enough to conclude that one technique is better than the other.

4. Conclusions

The most important thing we learned throughout the project is the workings of 2 common ways to do music recommendation systems, collaborative and content based filtering. We believe it is inconclusive whether or not 1 technique works better than the other due to the small participant number in our HRI experiment and the discrepancies in the data size between the techniques. With more time we could have collected more participants for our experiment and tried to gather a larger manual song set of rated songs from participants, but without this the goal of trying to see which technique is "better" is inconclusive even though content based filtering was the higher voted technique.

As a bit of a side note we were originally going to build out a seq2seq RNN for the content based approach. We moved away from this due to time constraints and an overall lack of literature on the topic as it pertained to music recommendation systems. We did sink a decent amount of time into this researching the topic, and we believe that we have a solid understanding of how a seq2seq RNN would work in a music recommendation system and could have implemented it with more time. In conclusion, we figured out that trying to build out a seq2seq RNN from scratch is a very difficult

task, which is probably true of most AI algorithms where the literature is lacking, but definitely doable with a larger amount of time.

5. Appendix

This is detailed as a Readme in each respective file.