A Comparison of Neighborhood-Based and Latent Factor-Based Collaborative Filtering for Music Recommendation

Jason Lefever
Drexel University
jtl86@drexel.edu

Mitali Sanwal
Drexel University
ms4952@drexel.edu

Sirui Wang
Drexel University
sw3336@drexel.edu

***Abstract*—Collaborative filtering is a method of building recommendation systems that use the totality of all user feedback of all items to give recommendations. This feedback can be either explicit, such as a 5-star rating, or implicit, such as a listen count. We compare two collaborative filtering approaches for recommending music based on a set of implicit feedback: a neighborhood approach and a latent model approach. We compare our results from these approaches with a third naive popularity recommender as a baseline. Our results found that the latent model approach performs the best while the neighborhood and popularity approach are roughly equal.**

I. Introduction

In the era of e-commerce and the Internet, big data leads to information overload. A recommendation system is a tool to solve the information overload problem. Recommendation system consists of two entities: users and items. The user is the online customer, and the item is the corresponding product. There are a large number of applications of recommendation systems, such as movie recommendation systems, book recommendation system, and music recommendation system. In a music recommendation system, the user is the listener and the item is the song.

Building a music recommendation is a common task that used by popular music service providers. The underlying task of music recommendation system is to generate the personal recommendation content based on relevant data of users.

II. Background

With the boost of digital content distribution, people now have access to music libraries on an unprecedented scale. Commercial music libraries easily exceed 15 million songs, which is vastly over the listening capability of any individuals. Even a million songs require more than seven years of listening to help users tackle with the rapidly expanding music libraries. Many academic efforts have been made to automate the search, annotation and retrieval of music content. The advancement of music information retrieval (MIR) technology can have far-reaching implications for commercial music recommendation systems, such as Amazon, Google, Pandora, and Spotify.

There are three families of techniques that have historically been used as recommendation systems: popularity, content-based, and collaborative filtering. Techniques based on popularity are the simplest. Simply recommend users the most popular songs regardless of their own tastes or listing habits. A content-based approach uses the content of the songs directly to make recommendations. This might either be metadata or the audio signal itself; this requires some similarity metric. collaborative Filtering relies on feedback from the user, either explicit or implicit, to make recommendations. Collaborative filtering attempts to capture correlation between user preferences and item features.

Collaborative Filtering is one of the most widely used algorithms in terms of recommendations, not only in music. Collaborative filtering builds a model from a user's past behaviors (items previously listened or selected or purchased and/or numerical ratings given to those items) as well as similar decisions made by other users. Generally, collaborative filtering comes in two flavors: neighborhood-based and latent factor-based.

A neighborhood approach requires a similarity metric between either users or items. Depending on which we use, we call say it is user-based or item-based. From there, we directly compare items or users against one another. For user-based one, the intuition is that users who listen to the same songs in the past tend to have similar interests or tastes and will probably listen to the same songs in the future. Similarly, the item-based approach is backed by the idea that songs that are often listened by the same user tend to be similar and are more likely to be listened together in future by some other user.

The latent model approach is an alternative method where we try to compute some "latent" or hidden features of both users and items, then use these latent features to estimate the preference of an unobserved user-item rating.

All three of these methods suffer from the "cold start" problem. How do we recommend a brand new item that does not have any interactions yet? This is out of scope for this project but we discuss some existing solutions to this in the future work section.

III. Related Work

Over the past decade 'Music Recommendation' has been a popular topic in the field of machine learning. But the related work starts as early as 1990s, when humming system was proposed, based on content-based model. Early query by

humming systems were using melodic contour which had been seen as the most discriminating features in songs.

Context-based Information Retrieval became very popular with the emergence of social networks such as Facebook, Youtube, Twitter etc . Recommendations in such sites can be seen in the form of People you may know, Pages you may like, Suggestions for you etc. Recommendations here are made using click-through rate, browsing behaviour and using user tags. Yazdanfar N, Thomo A in their work: 'collaborative-filtering for recommending URLS to Twitter users' proposed a neighbourhood based recommendation systems that makes recommendations to the Twitter users in the form of URLs.

Chang N, Irvan M, Terano in their work have proposed a smart and social TV program recommendation system that combines Internet and web 2.0 features with television set-top boxes. The proposed work also includes explanations of the recommendations so as to gain the user trust and to enhance the user-satisfaction .

Liebman et al. [3] introduced a different representation of songs, they use the audio features to represent a song, and the reward of a song is decided by both user's preference of the song and user's preference of the transition. The transition reward is represented as the weighted sum of utilities of current song and every previous songs. As for the learning process, both Liebman and Hu uses reinforcement learning,and in Liebman's latest paper, they improved the model by applying Monte Carlo Tree Search.

Hu et al. [5] introduced a method of Weighted Matrix Factorization (WMF) for latent models especially for implicit feedback data. This is an attempt to overcome the shortcomings of neighborhood approaches for implicit data where the similarity metric is not able to tell if a user does not like a song or has simply never had the opportunity to hear it.

IV. Methodology

A. Popularity-based Approach

We start with a naive popularity based approach which gives the same generic results to all users. We simply sort the songs in decreasing order of their total listen count. Our first recommendation to every user is the most popular song. Our second recommendation is the second most popular song. And so on. This is a surprisingly effective approach despite not being at all tailored to the user's tastes.

B. User-based Neighborhood Approach

A user-based neighborhood approach depends on a similarity metric between users. We use the cosine similarity between each user's listen count vector. Our goal is to predict the unobserved rating r_{ui} a user u might give an item i. We denote U as the set of k nearest users to u who rated i. The value of r_{ui} is a weighted average. This is a traditional k-nearest neighbors approach. See Fig. 2 for the exact calculation.

C. Latent Factor Approach

First, we lay out our m users and n items in a $n \times m$ "interaction" matrix where each value r_{ui} denotes the rating user u has given item i. This rating can be explicit such as a 5-star rating or implicit such as the number of times the user has listened to that song. Second, we decompose this interaction matrix into an $m \times f$ matrix X and an $n \times f$ matrix Y where f is the number of latent factors. Then to predict a user's rating for a certain item, we take the inner product $X_u^T Y_i$. One shortcoming with this approach is there is no intuition behind what these latent features mean. Another larger shortcoming is that there is no way to introduce a new user or item without recreating the entire model.

We are particularly interested in approaches that work well for implicit feedback. We borrowed the Weighted Matrix Factorization (WMF) method introduced by Hu et el. To work better with implicit feedback, WMF introduces two new values derived from r_{ui}. The first is preference, p_{ui}, which is simply a binary value denoting if user u has listened to song i at least once. The second is confidence, c_{ui}, which is a scalar-weighted value proportional to listen count. Confidence denotes how confident we are they like the song. One example confidence function is given below.

$$c_{ui} = 1 + \alpha r_{ui}$$

We need these new values to work around implicit feedback. A user's listen count for a song is not really a rating. If a user has never listened to a song, that does not mean they do not like it. They simply may have never heard it. Additionally, with only listen counts we have no way to tell if a user does not like a song. We make two assumptions: (1) if a user listened to a song once, they like it, (2) the more a user listened to a song, the more they like it. This is the motivation for the definitions of preference and confidence.

From there we borrow the objective function of Hu et al. to optimize the latent matrices X and Y.

$$argmin \sum_{u,i} c_{ui}(p_{ui} - X_u^T Y_i)^2 + \lambda(\sum_u \|X_u\|^2 + \sum_i \|Y_i\|^2)$$

This consists of a confidence-weighted mean squared error term and a lambda-weighted regularization term to curb overfitting. However because this function contains over $m \times n$ terms, this will quickly explode on a typical dataset. This prevents the usual stochastic gradient descent for optimization. Instead Hu et al. have proposed another method of minimizing this function called Alternating Least Squares (ALS). If you hold either the user-factors or item-factors fixed, the function becomes quadratic. So, this leads to alternating between computing user-factor and item-factor. By differentiation, we can find a closed form solution for X_u.

$$X_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

Here C^u is the square $n \times n$ matrix with the confidence values, c_{ui}, across its diagonal for a given user u. Additionally, $p(u)$ is the vector that contains all preference values, p_{ui}, for user u.

Fig. 1. The popularity percentage of each song (in order)

	song	listen_count	percentage
3820	Harmonia - Sehr kosmisch	8277	0.396608
4926	Kings Of Leon - Use Somebody	7952	0.381035
1118	Björk - Undo	7032	0.336952
3296	Florence + The Machine - Dog Days Are Over (Ra...	6949	0.332975
2730	Dwight Yoakam - You're The One	6412	0.307243
...	...	...	...
7251	Ricardo Arjona - Historia Del Portero	51	0.002444
9314	Three Days Grace - Scared	51	0.002444
434	Amparanoia - Don´t Leave Me Now	50	0.002396
4596	Juanes - No Creo En El Jamas	48	0.002300
5058	Ladytron - Ghosts (Toxic Avenger Mix)	48	0.002300

Similarly, we can also find a closed form solution for Y_i.

$$Y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i)$$

To minimize the objection function, we then just alternate with these calculations for some number of sweeps. Hu et al. had success with 10 sweeps. The algorithm scales linearly with the size of the interaction matrix.

V. Experiments and Results

We conducted three experiments on the same interaction matrix using the three methods addressed in the methodology section. We are most interested in the neighborhood and latent model approach while the popular approach serves as surprisingly effective baseline. We collected our data from the Echo Nest Taste Profile Subset of the Million Songs Dataset. This includes millions of (user id, song id, listen count) triplets which we used to construct our interaction matrix. From there, each user u and song i was addressed uniquely by their index on their respective axis of that matrix.

A. Popularity Approach

The most straightforward approach is the popularity-based recommendation. Each song has its own unique count number to indicate how many times this song has been listened to all users. This count number is the key indicator that we used to build the popularity-based model. By dividing the count number into the total number of songs count, we got the percentage for each song (See Fig. 1.) We then rank the songs according this percentage number. The top ten popular songs will be generated as the result of popularity-based model.

B. User-based Neighborhood Approach

User based model is built on the basis of searching for the nearest neighbor of the current user. In this way, we trained the KNN model to get the nearest neighbors of the current user. In order to find k nearest neighbors, we need to find the similarity between each pairs of users first. Once we have a similarity vector for user U2 having all other songs in the dataset in the similarity vector which contains the weight of their similarity. We then sorted in decreasing order and find k-most

Fig. 2. The similarity calculation formula

$$Cos(U_1, U_2) = \frac{\sum_{y \in R_{u_1} \bigcap \in R_{u_2}} R_{u1,y} * R_{u2,y}}{\sqrt{\sum_{y \in R_{U_1}} R_{u1,y}^2} * \sqrt{\sum_{y \in R_{U_2}} R_{u2,y}^2}}$$

$$Recommend(u, i) = \frac{\sum_{v \in U} Cos(u, v) * R_{v,i}}{\sqrt{\sum_{v \in U} |Cos(u, v)|}}$$

Fig. 3. Recommendation of top 10 songs based on popularity

most recommended songs based on popularity

	song
Sehr kosmisch - Harmonia	8277
Undo - Björk	7032
Dog Days Are Over (Radio Edit) - Florence + The...	6949
You're The One - Dwight Yoakam	6412
Revelry - Kings Of Leon	6145
Secrets - OneRepublic	5841
Horn Concerto No. 4 in E flat K495: II. Romance...	5385
Fireflies - Charttraxx Karaoke	4795
Hey_ Soul Sister - Train	4758
Tive Sim - Cartola	4548

similar neighbors. The formula for calculating the adjusted cosine similarity between a pair of user $U2$ and item y is given as follows, where U_2y is the similarity weight. (Fig 2.) After the KNN model is built, we use the recommend formula to generate the recommended songs.

C. Latent Factor Approach

We executed exactly what is described in the methodology section on our interaction matrix. We did 10 sweeps of alternating least squares. We experimentally found a confidence weight α of 140, a regularization constant λ of 0.5, and a latent factor count f of 20 to work best for our data. This is consistent with the hyper-parameters used by Hu et al. on a completely different dataset. From there we calculated the inner product $X_u^T Y_i$ for every u, i to get a real-value estimate, $\hat{r}_{ui}$, for if user u prefers song i. This "prefers" is the same notion of preference as discussed in the methodology section.

D. Evaluation

Ultimately the purpose of a recommendation system is provide suggestions to actual users. To properly evaluate our models, we would want to take a similar approach to what Hu et al. have done which is to train your model on some time frame of recent historical data, then present your system to those same users and record the results. Did the users listen to the songs suggested by your system? Unfortunately, we do not have access to the users this data was collected from. The data is completely anonymous. So we will have to find another approach.

Fig. 4. The final mean AUC for each model

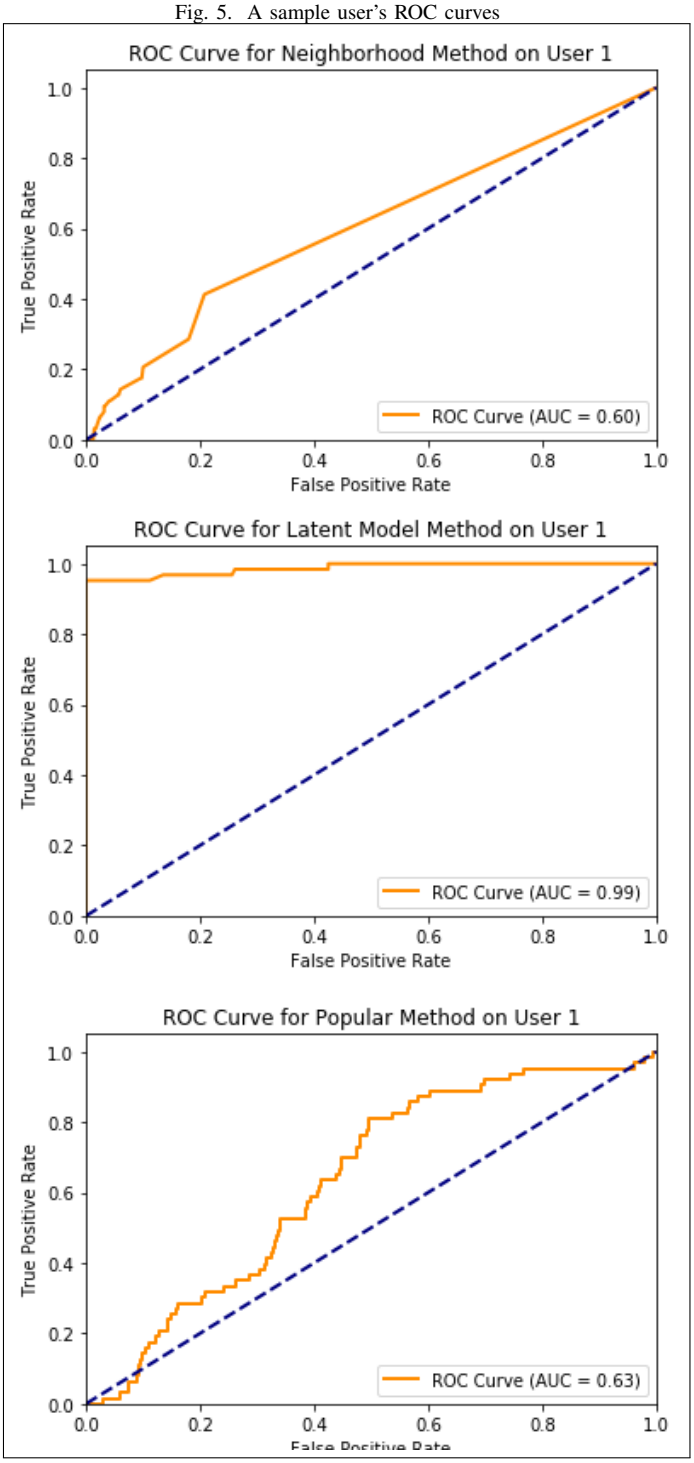
```
Mean AUC (Neighborhood): 0.584695042271343
Mean AUC (Latent Model): 0.9681248924909588
Mean AUC (Popular):      0.6112951213138439
```

Each of our three models is a binary classifier. We are attempting to assign a 0 or 1 to a user's preference for an item, p_{ui}. A preference of 1 indicates that the user likes the item. However, recall that a preference of 0 does *not* mean the user does not like the item, simply that they have not been exposed to it. So if there does not seem to be an actual negative case, how are we to assess if a false positive or true negative has occurred? After all, wouldn't false positives be a good sign that our recommmender was suggesting novel items?

Our approach was to treat the original interaction matrix as the test matrix. We derive our training matrix by copying the original interaction matrix and removing 20% of instances of positive preferences. We build our models from the training data and then check if our models were able to recommend the items we removed. To do this, we compute the ROC curve for each user we removed items from. Then to assign a final score to a model, we take the mean of every area under the ROC curve. Our models assign real estimates, not labels, so the ROC curve will address every potential threshold that would binarize these real values. Addressing every potential threshold does partially alleviate the negative case issue, but it does not resolve it. The negative case issue also makes it difficult to detect overfitting. We still cannot really know if we have made a bad recommendation. This is a fair evaluation but it still falls far short of actually running the recommendations by the live subjects.

We present our final mean AUCs for each model in Fig. 4. Fig. 5 and Fig. 6 show two ROC curves calculated from sample users. By our measure, we found the latent model to be far the best recommender with the popularity approach just beating out the KNN approach. This is close to the results of Hu et al, but their neighborhood approach was able to beat the popularity model but only by a small margin.

Fig. 5. A sample user's ROC curves



VI. Conclusions

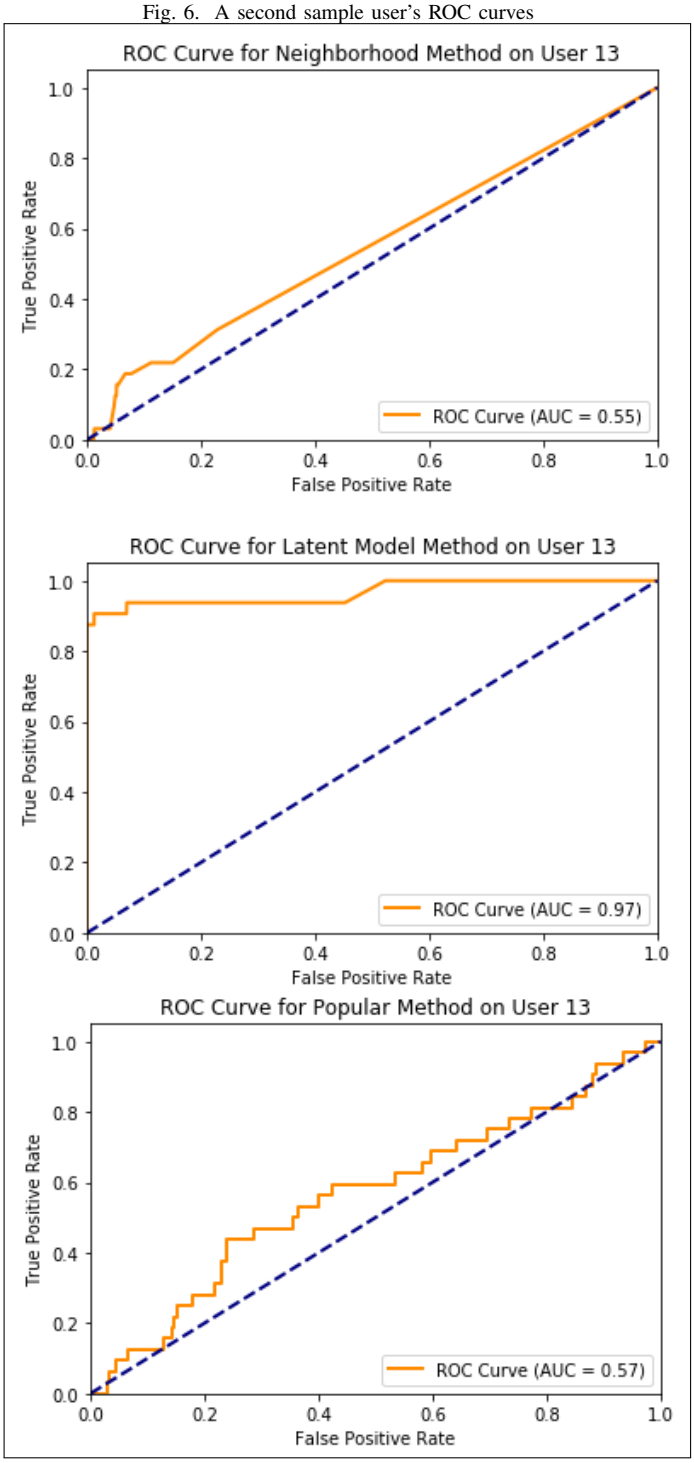
We covered two methods of collaborative filtering, neighborhood and latent model based, and shared the results of our own experiments with them. In addition, we compared these models against a more basic, but surprisingly effective, popularity model.

We also discussed some challenges we faced when attempting evaluate our models without having access to subjects to rate our recommendations. We presented our alternative evaluation and discussed some its limitations.

VII. Future Work

The subject of collaborative filtering on implicit feedback has many prospective and actual extensions. We list a few below.

Fig. 6. A second sample user's ROC curves



1. Train a classifier to predict the genre of a song given its latent feature vector. This is briefly discussed in Oord et al. [5].
2. Use a content-based method to predict a latent feature vector of a brand new song. This is an attempt to solve the cold start problem. Oord et al. attempt this using a convolutional neural network using features collected from the audio signal as input.
3. Algorithms for different aspects like user's mood, current time and day and so on.
4. Run the algorithms on a distributed system, like Hadoop or Condor, to parallelize the computation.

REFERENCES

[1] Yading Song, Simon Dixon and Marcus Pearce: A Survey of Music Recommendation Systems and Future Perspectives.

[2] Richa Sharma and Rahul Singh: Evolution of Recommender Systems from Ancient Times to Modern Era

[3] Elad Liebman, Maytal Saar-Tsechansky, and Peter Stone. Dj-mc: A reinforcement-learning agent for music playlist recommendation. In Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS '15, pages 591–599, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.

[4] Ms. Sonali Sonavane, Asst. Professor Computer Engineering Department G. H. Raisoni Institute of Engineering and Technology:Song Recommender System Using Machine Learning

[5] Yifan Hu, Yehuda Koren, and Chris Volinsky: Collaborative Filtering for Implicit Feedback Datasets

[6] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen: Deep content-based music recommendation