Music Recommendation Using Collaborative Filtering

Vuong Khuat
vdkhuat16@earlham.edu
Earlham College
Richmond, Indiana

ABSTRACT

With the increasing popularity of music streaming, music recommender systems are important instruments for increasing digital music consumption. However, collaborative filtering, a technique widely used in recommender systems, is not generally adapted in this domain because the technique does not scale well with large amounts of data. This paper proposes a recommender system using collaborative filtering with an improved runtime by using a more efficient data-representation scheme and considering a partial section of the dataset.

KEYWORDS

Recommender Systems, Collaborative Filtering, Recommendation Algorithms, Machine Learning

ACM Reference Format:
Vuong Khuat. 2018. Music Recommendation Using Collaborative Filtering. .

1 INTRODUCTION

In the last 25 years, the use of recommender systems has expanded rapidly [2, 14]. These systems, designed to produce recommendations that match users' interests, are incorporated into areas such as news, entertainment, and research [4, 13]. One common technique used by recommender systems is *collaborative filtering*, which produces recommendations for a user based on choices of users with similar preferences [9]. This paper describes a recommender system for music tracks using collaborative filtering.

In recent years, music streaming has become increasingly prevalent. Over the first half of 2018, more than 400 billion on-demand song streaming activities occurred in the United States, which equates to 360.1 million albums consumed [1]. Good recommender systems for music tracks can further increase the consumption for digital music and therefore are important tools for streaming platforms.

Different techniques have been published in the field of music recommendation, notably the deep-content approach proposed by van den Oord et al. [19] and the automatic tag generation method by Eck et al. [7]. Collaborative filtering is also used in this field; however, the algorithm does not scale well with large datasets [11]. This makes it hard for researchers to use this method with a dataset that can potentially have millions of users and music tracks. It should be noted that recommender systems of large streaming services such as Spotify or Pandora may be built using a combination of collaborative filtering with other techniques [5]. Nevertheless, details about how these systems are implemented are not publicly available.

It is important to build a recommender system using collaborative filtering that not only is accurate but also runs efficiently. In this project, I attempt to build a recommendation systems for music tracks using collaborative filtering with two optimization methods, dataset rescaling and automatic halting. These methods will be built and tested with different parameters to address the performance issue while retaining an acceptable accuracy.

2 BACKGROUND

2.1 Collaborative Filtering

Collaborative filtering is a successful and widely-used method for generating recommendation in various fields [8][3][6]. The method provides recommendations for a target user by estimating the utility of different items based on the habits or ratings of other users [14]. The advantage of this technique is that the behavior of a user can generally be predicted from users who are similar. Furthermore, systems that use collaborative filtering can work with any type of items and can generate recommendations of different types within the same system [10].

However, this technique also has some drawbacks. The first major drawback is scalability. In a commercial setting, collaborative filtering algorithms are required to search tens or hundreds of thousands of users to produce a recommendation. This makes it difficult to recommend items in real time, especially when the number of users and number of items are extensive. This also requires a substantial amount of space. Traditional collaborative filtering algorithms use a two-dimensional matrix to represent the data, where the columns represent individual users and the rows represent individual items. In practice, this matrix will be very sparse, as most users in many large commercial websites are estimated to consume less than 1% of the contents [15]. Therefore, the matrix representation can be an inefficient and space-consuming method.

In the next section, we discuss different types of rating systems. In sections 2.3 and 2.4 we describe prior attempts to improve collaborative filtering. The method in 2.3 uses neighborhood formation [14]; the method in section 2.4 uses a clustering model [18].

2.2 Types of User Rating Used by Recommender Systems

Determining the similarity of different users requires recommender systems to attain a numerical value for every user-item pair which represents the user's rating of the item. In this section we discuss two types of user ratings that can be used by recommender systems.

(1) *Explicit rating.* In this type of rating, users *explicitly* provides feedback for different items on a numerical scale (i.e. from 1 to 5 where 5 means "strongly like" and 1 means "strongly dislike"). Since ratings are provided directly by users and do not have to be inferred, it is generally considered a good rating system [15]. However, it is not practical for users to provide ratings for every item. Thus, recommender systems that work with this rating scheme can only make use of the rated items unless they have a method to derive feedback implicitly for the unrated items [15].
(2) *Implicit rating.* In this rating system, ratings are not provided by users, but are predicted based on users' activity. For instance, if someone has not provided a rating for a certain song but has listened to it more than 20 times, she will likely give the song a high rating.

2.3 Neighborhood Formation

The neighborhood formation method was proposed by Sarwar et al. [14]. We will first examine their approach and then consider an analysis of the approach by Su and Khoshgoftaar [17].

The neighborhood formation method aims to find a set of n users, given a user c, such that the ordered set $S = \{c_1, c_2, ..., c_n\}$ has the following property: the similarity function, $\text{sim}(c, c_i) \leq \text{sim}(c, c_j)$ for all $i < j$. The similarity function can be the Pearson correlation method, or the cosine method. The results are used to form neighborhoods. This can be done by using the k-nearest neighbors approach, which forms a neighborhood around any customer c by selecting k customers closest to c, or by using the aggregate method. After generating different neighborhoods, it generates recommendations in two ways: most-frequent item recommendation and association rule-based recommendation. In their evaluation model, they concluded that their method could scale better with large datasets. The method also produced higher-quality recommendations.

An analysis of this approach was published by Su and Khoshgoftaar [17]. The authors determined that the method was easy to implement, did not need to consider the content of the items recommended, and could scale well with items that were rated similarly. However, they also noted that the approach did not perform well with a sparse dataset.

The work of this paper is different from Sarwar et al.'s in that it takes the whole dataset into consideration rather than any particular neighborhood.

2.4 Cluster Model

The method of using a cluster model was proposed by Ungar and Foster [18]. In this section, We will summarize the details and results of their work, then compare it with the method proposed in this project.

The cluster model is an approach to reduce computing time and raise recommendation quality. In a traditional approach, all users and items are modeled using a matrix. However, Ungar and Foster argued that such representation are inefficient because users have different behavior and thus it is be better to divide the users into groups, or "clusters," where each group consists of more like-minded users. This method improved efficiency because recommendations were made within groups; the recommendations were also better because they were made from more similar groups of people.

Using this model on data of customers from CDNow, a shopping website for compact discs, the authors concluded that clustering people based on CDs they purchased was hard because the data were sparse (most people only buy one CD). However, the model produced great results when users were clustered using CD clusters (which were CDs by the same artist). An approach using cluster model may reduce the computation time, but does not guarantee good recommendations.

Compared to Ungar and Foster's work, the implementation collaborative filtering algorithm of this paper, which will be discussed in more detail in section 3, has a longer runtime. However, since the method considers the majority of the users in the dataset instead of a specific cluster, it is expected to produce better results.

3 METHODOLOGY

This methodology proposed in this paper consists of three main components: Data representation, the collaborative filtering function, and model evaluation (see figure 1). We first introduce a formal definition for the tasks this paper aims to solve. Then, we will discuss the dataset used in this work and how it is represented. The next section is the implementation of the collaborative filtering algorithm that will be used in the paper to produce recommendations. The final component introduces the steps to evaluate the model.

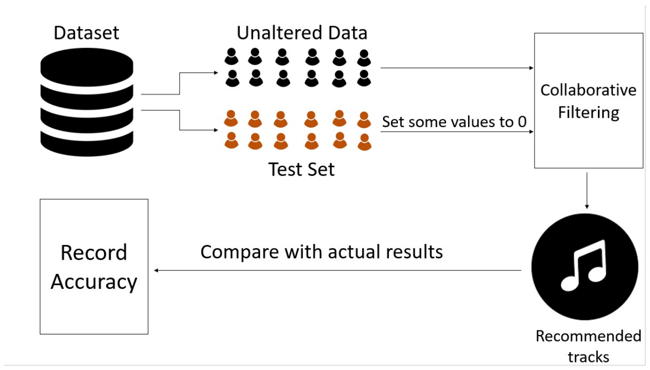

Figure 1: The Overall Framework of This Project.

3.1 Problem Definition

Formally, the goal of this project is as follows. Suppose we have a dataset containing a set of n users $U = \{u_1, u_2, ..., u_n\}$ and a set of m songs $S = \{s_1, s_2, ..., s_m\}$. Each user ($u_i$, for example) also has a set of songs they listened to, which we denote as S_{u_i}. Let u_a be the active user (the user we want to recommend songs to) and N be the number of songs to recommend. We want to find a set of N elements $I_r \subset I$ such that u_a would like them the most but has not reacted to them (as shown by the dataset).

3.2 Processing the Data

For this project, the data will be collected from the Echo Nest Taste Profile Subset, which provides play counts of more than 300,000 users for approximately 1 million songs [12]. Traditionally, the data is represented using a $m \times n$ matrix, where m is the number of songs, and n is the number of users, for example,

$$\begin{array}{c|cccc} & u_1 & u_2 & \dots & u_n \\ s_1 & 1 & 0 & \dots & 0 \\ s_2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ s_m & 1 & 1 & \dots & 1 \end{array}.$$

The value 1 at row u_i and column s_j means that user u_i has listened to the song s_j. In contrast, 0 means that the user has not listened to the song.

However, as noted in 2.1, this matrix is most likely sparse, which is not space-efficient. In fact, on this specific dataset, only 1.23% of the components are non-zeroes. To address this problem, this paper represents the dataset as a hash table that maps each user to the set of songs they listened to. Suppose $n_1, n_2, ..., n_n$ are the numbers of songs users $u_1, u_2, ...u_n$ react to, respectively. Then, the table is as follows:

$$\begin{aligned} u_1 &: \{s_{u_{11}}, s_{u_{12}}, ..., s_{u_{1n_1}}\}, \\ u_2 &: \{s_{u_{21}}, s_{u_{22}}, ..., s_{u_{1n_2}}\}, \\ &\vdots \\ u_n &: \{s_{u_{n1}}, s_{u_{n2}}, ..., s_{u_{1n_n}}\}. \end{aligned}$$

Using this representation, one can still determine the values of parts of the matrix if necessary. For instance, if we want to get all the values of a column u_i, then we can simply get non-zero values of the column from the key u_i of the table and fill the rest with zeroes.

3.3 Implementing the Collaborative Filtering Algorithm

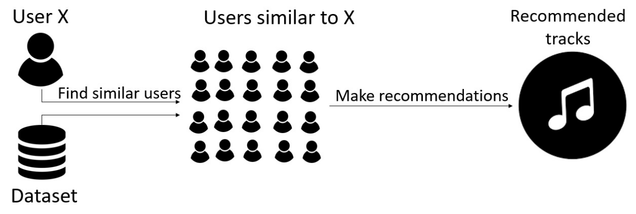

Figure 2: Procedure the collaborative filtering algorithm for music tracks in this proposal.

The overall implementation of the collaborative algorithm is given in figure 2. The collaborative filtering algorithm will be implemented as a function f. The function takes in the dataset d, a user u_a, a positive integer N and returns an list that contains N songs that best match u_a's interest. This is done in two steps.

First, f determines the users who are similar to u_a. This can be done using a iterative process: for each user u_i that is not u_a, if u_i and u_a are similar, then add u_i to the list of similar users.

Having produced a list of similar users, recommendations can then be generated for u_a from music tracks that the a specific proportion of similar users have listened to but u_a has not. For instance, suppose we have

$$\begin{array}{c|ccc} & u_a & u_2 & u_3 \\ s_1 & 1 & 1 & 1 \\ s_2 & 1 & 1 & 1 \\ s_3 & 1 & 1 & 1 \\ s_4 & 1 & 1 & 1 \\ s_5 & 0 & 1 & 1 \end{array}$$

The algorithm in this case may suggest that u_a, u_2, and u_3 are similar (since they all listen to s_1, s_2, s_3, s_4), and recommend track s_5, which has been listened to by both u_2 and u_3. We define a hyperparameter τ to represents the proportion of similar users that must have listened to a song for that song to be recommended. For example, if $\tau = 0.5$, then the algorithm recommend songs that at least half of the similar users have listened to.

3.4 Determining the Similarity Between Two Users

One critical part of the algorithm is being able to determine if two users are similar. To do this, this paper consider users as m-dimensional vectors, where m is the total number of songs in the dataset. With this representation, the similarity of two users, u_i and u_j can be determined by the cosine of the angle between the corresponding vectors:

$$cos(\vec{u_i}, \vec{u_j}) = \frac{\vec{u_i} \cdot \vec{u_j}}{||\vec{u_i}|| ||\vec{u_j}||}$$

This is one of the standard techniques for calculating similarity in collaborative filtering algorithms [14]. With the table representation proposed in this paper, the dot product of $\vec{u_i}$ and $\vec{u_j}$ can be calculated by finding the components in which both vectors have non-zero values, then sum up the products of the corresponding values. This works because the components with zero values do not affect the dot product. The strategy of only considering non-zero values is also used to determine the magnitude of each vector using the table representation.

We define a parameter θ to indicate the largest value the angle between $\vec{u_i}$ and $\vec{u_j}$ can have for u_i, u_j to be considered similar. In evaluation, we will test the model with different values of θ and analyze the results.

3.5 Testing and Evaluation

In this section we provide a scheme to evaluate the model in this paper. Given a user u_a, we retain k components of the vector $\vec{u_a}$, where k is a positive integer less than m, and set the remaining $m - k$ components to 0. We can assume that $\vec{u_a}$ has the form

$$\begin{array}{cc} s_1 & \text{real value} \\ s_2 & \text{real value} \\ \vdots & \vdots \\ s_k & \text{real value} \\ s_{k+1} & 0 \\ s_{k+2} & 0 \\ \vdots & \vdots \\ s_m & 0 \end{array}.$$

Then, we apply the collaborative filtering algorithm to generate recommendations for u_a. To prevent the model from producing too many recommendations, we set the maximum number of recommendations to be $10k$, and the songs chosen are the most frequent songs among the list of similar users. The components that were set to be 0 are now compared with the corresponding real data. To assess the performance of the model, we collect the following information:

- *Precision*: the number of songs which were hidden that are in the list of recommendations out of all recommended songs, and
- *Recall*: the number of songs which were hidden that are in the list of recommendations out of all hidden songs.

These two measurements will be used to assess the accuracy of the model. To assess its speed, we measure the average amount of time it takes the model to generate recommendations for a user.

3.6 Methods to Improve Efficiency

As noted by Linden et al. [11], one major challenge of the iterative collaborative filtering algorithm is its efficiency. The collaborative filtering algorithm works by iteratively finding users similar to the given user. Since there are n users, represented as m-dimensional vectors, the overall time complexity of the model is $O(mn)$. Thus, it does not scale well as the amount of data grows large. The representation presented in section 3.2 improves this runtime since for each user, only a small subset of songs is considered. This section introduces two methods that are used to reduce the runtime.

a. **Randomly select a subset of the dataset.** This method select a fixed-size subset of the dataset. The subset will be generated at random in order not to create bias in the data. The subset-size parameter will be determined from experiments to determine whether collaborative filtering model can run within a reasonable amount of time without decreasing the accuracy rate by a significant amount.
b. **Halt after finding enough similar users.** This method reduces the running time by stopping once the number of similar users found is sufficient to generate recommendations. In the case where the desired amount cannot be reached, we use a break point after having considered a certain amount of users and use the list of similar users gathered so far to generate recommendations. Similar to (a), different values for the sufficient number of similar users and the number of users for the break point will be experimented so that the run time of the algorithm and the accuracy rate are reasonable.

4 RESULTS/EVALUATION

We first evaluate the model with different values for τ and θ, as described in section 3.3 and 3.4, respectively. We pick values for τ from the set

$$\{0.1, 0.2, 0.3, 0.4, 0.5\}$$

and θ from the set

$$\{90^\circ, 84.3^\circ, 78.6^\circ, 72.5^\circ, 66.4^\circ, 60^\circ\}.$$

The rationale behind selecting these values is that experiments show with $\tau < 0.1$, the model recommends virtually every song in the dataset, while $\tau > 0.5$ does not produce any recommendation at all. Similarly, only values for the angle θ between 60° and 90° produces a reasonable number of recommendations. The recorded precision and recall rates for all combinations of τ and θ are shown in table 1 and table 2, respectively.

Table 1: Precision rates for different combinations of τ and θ

τ \ θ	90°	84.3°	78.6°	72.5°	66.4°	60°
0.1	0.0072	0.0212	0.0298	0.0413	0.0769	0.0787
0.2	0.0072	0.0268	0.0338	0.0335	0.0459	0.0815
0.3	0.0084	0.0260	0.0302	0.0451	0.0658	0.0600
0.4	0.0096	0.0180	0.0337	0.0409	0.0308	0.0200
0.5	0.0076	0.0256	0.0237	0.0392	0.0362	0.0400

Table 2: Recall rates for different combinations of τ and θ

τ \ θ	90°	84.3°	78.6°	72.5°	66.4°	60°
0.1	0.12	0.29	0.22	0.17	0.04	0.04
0.2	0.12	0.31	0.26	0.16	0.11	0.07
0.3	0.12	0.33	0.24	0.15	0.11	0.00
0.4	0.14	0.33	0.24	0.18	0.07	0.00
0.5	0.16	0.26	0.17	0.08	0.08	0.04

As $\theta = 84.3^\circ$ and $\tau = 0.4$ give the highest recall rate for the model, We use these values for the next step, which is to test the parameters in section 3.6. Denote s as the proportion of the generated subset compared to the full dataset, and m as the number of similar users threshold to halt. The execution time of different values for s and m are shown in table 3. The corresponding precision and recall rates of these tests are also documented in table 4 and 5, respectively.

Table 3: Execution time (in seconds) for different combinations of s and m

s \ m	500	1000	2500	5000
0.3	1.8247	1.938	2.001	2.0460
0.4	2.194	2.5218	2.590	2.628
0.5	2.760	3.0783	3.208	3.297

Table 4: Precision rates for different combinations of s and m

s \ m	500	1000	2500	5000
0.3	0.0336	0.0336	0.0344	0.0312
0.4	0.0344	0.0392	0.0318	0.0296
0.5	0.0314	0.0351	0.0320	0.0320

Table 5: Recall rates for different combinations of s and m

s \ m	500	1000	2500	5000
0.3	0.365	0.365	0.374	0.339
0.4	0.373	0.426	0.339	0.322
0.5	0.339	0.374	0.348	0.348

5 DISCUSSION

From table 1, different combinations of τ and θ result in precision rates ranging from 0.0072 to 0.0815, with the highest value occurring at $\tau = 0.2$ and $\theta = 60°$. An explanation for this is that these values correspond to the tighter constraints in the hyperparameter space; therefore we get higher precision. However, this combination gives thus a recall rate of only 0.07, which is significantly worse than 0.33, the highest value (at $\tau = 0.4$ and $\theta = 84.3°$.

Another observation from the experiments is that the highest precision rate being less than 10% means that less than 10% of the originally hidden songs came up in the recommendations. This might not necessarily reflect the quality of the model, as its role is to find songs that the active user might like but do not know about.

The schemes proposed in section 3.6 also result in an improvement in runtime. As mentioned in section 4, the average runtime before deploying these methods is 5.81 seconds, whereas the runtimes documented in table 3 range from 1.83 to 3.3 seconds. Lower values of s and m further reduces the runtime. However, we also need the precision and recall rates for these experiments in order to conclude whether this improvement also has a large impact on the accuracy of the model. Interestingly, at $s = 0.4$ and $m = 1000$, we attain the highest values for precision and recall rates at 0.0392 and 0.426, respectively. These values are both higher than the corresponding values in table 1 and 2. The results of these experiments show that the methods of section 3.6 both reduce the overall runtime and retain the accuracy of the model.

6 RELATED WORK

In this section, we described Linden et al.'s collaborative filtering algorithm using an item-based method as opposed to a user-based approach proposed in this paper. We then compare the method with the approaches discussed in sections 2.2, 2.3 and with the collaborative filtering implementation of this paper in section 3.

Linden et al. [11]'s method was used for generating recommendation by Amazon. The approach is different from other methods in that rather than finding like-minded users, Linden et al. developed an algorithm that linked a user's purchased or rated items to other similar items. The algorithm iteratively generates a table of similarity values between a products using the cosine method. Using this table, it can then quickly generates recommendations to users based on items they have rated or purchased.

The major factor that set the method of Linden et al. apart from the other implementations of collaborative filtering as well as approaches discussed in previous sections is its computation time. In the traditional collaborative filtering method, generating recommendations takes $O(mn)$ time where m is the number of users and n is the number of products. In Linden et al.'s method, however, creating the item-to-item similarity table takes $O(mn)$, but this work can be done offline. Once the table is created, making recommendations to user can be quickly done.

An analysis from Schafer et. al. [16] noted that this method saved memory and computation time because it took advantage of the fact that only a subset of the dataset was needed to generate recommendations. While clustering methods could also be efficient, their recommendation quality were relatively low. On the other hand, using a traditional approach may result in good recommendations, but their inefficiency made it difficult to implement in practice.

Contrary to Linden et al.'s approach, the collaborative filtering algorithm implementation is based on the similarities between users, not items. It is based on the traditional method, but makes use of the space and runtime optimization methods discussed in sections 3.2 and 3.6.

7 FUTURE WORK

An immediate extension of this project is to explore other methods that manipulate the dataset to reduce the runtime. Such methods would need to consider properties of the dataset so that the collaborative filtering function can make less comparison without losing accuracy.

Another extension is to compare the performance of the work proposed in this model with other recommendation techniques other than collaborative filtering.

8 ACKNOWLEDGMENTS

This project is a part of the Senior Capstone Experience of the Department of Computer Science at Earlham College. I would like to thank Dr. David Barbella, Dr. Ajit Chavan, Dr. Xunfei Jiang, and Dr. Charles Peck of Earlham College for their support in this project.

REFERENCES

[1] 2018. U.S. Music Mid-Year Report 2018. https://www.nielsen.com/us/en/insights/reports/2018/us-music-mid-year-report-2018.html

[2] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering* 6 (2005), 734–749.

[3] Hyung Jun Ahn. 2008. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences* 178, 1 (2008), 37–51.

[4] John S. Breese, David Heckerman, and Carl Kadie. 1998-07-24. Empirical analysis of predictive algorithms for collaborative filtering. *Proceeding UAI'98 Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence* (1998-07-24), 43–52.

[5] Sophia Ciocca. 2017. How Does Spotify Know You So Well? https://medium.com/s/story/spotifys-discover-weekly-how-machine-learning-finds-your-new-music-19a41ab76efe.

[6] Chris Cornelis, Jie Lu, Xuetao Guo, and Guanquang Zhang. 2007. One-and-only item recommendation with fuzzy logic techniques. *Information Sciences* 177, 22 (2007), 4906–4921.

[7] Douglas Eck, Paul Lamere, Thierry Bertin-Mahieux, and Stephen Green. 2008. Automatic generation of social tags for music recommendation. In *Advances in neural information processing systems*. 385–392.

[8] Xuetao Guo and Jie Lu. 2007. Intelligent e-government services with personalized recommendation techniques. *International Journal of Intelligent Systems* 22, 5 (2007), 401–417.

[9] Jonathan L. Herlocker, Joselph A. Konstan, Loren G. Terveen, and John Riedl. 2004-01. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004-01), 5–53.

[10] Seok Kee Lee, Yoon Ho Cho, and Soung Hie Kim. 2010. Collaborative filtering with ordinal scale-based implicit ratings for mobile music recommendations. *Information Sciences* 180, 11 (2010), 2142–2155.

[11] Greg Linden, Brent Smith, and Jeremy York. 2003-01. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7, 1 (2003-01), 76–80.

[12] Brian McFee, Thierry Bertin-Mahieux, Daniel PW Ellis, and Gert RG Lanckriet. 2012. The million song dataset challenge. In *Proceedings of the 21st International Conference on World Wide Web*. ACM, 909–916.

[13] Paul Resnick and Hal R Varian. 1997. Recommender systems. *Commun. ACM* 40, 3 (1997), 56–58.

[14] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2000-10-17. Analysis of recommendation algorithms for e-commerce. *Proceedings of the 2nd ACM conference on Electronic commerce* (2000-10-17), 158–167.

[15] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001-05-01. Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th international conference on World Wide Web* (2001-05-01), 285–295.

[16] Markus Schedl. 2016. The lfm-1b dataset for music retrieval and recommendation. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*. ACM, 103–110.

[17] Xiaoyuan Su and Taghi M Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in artificial intelligence* 2009 (2009).

[18] L. H. Ungar and D. P. Foster. 1998. Clustering Methods for Collaborative Filtering. cial Intelligence (1998).

[19] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *Advances in neural information processing systems*. 2643–2651.