Finding the Song of Your Ear - An Implementation of Collaborative Filtering-Based Recommender System

Amy Huang
amyh1@andrew.cmu.edu

Wei-Kung Wang
weikungw@andrew.cmu.edu

ABSTRACT

In this project, we are interested in utilizing collaborative filtering -based recommender system to provide recommendations to streaming-music service users when they are creating playlists. We believe the idea will add value to the streaming music business because, according to Music Business Association's publication in 2017, in the United States, over 30% of the music listening time comes from playlists, and thus a better automatic playlist continuation feature will help the industry attract and keep its users.

In our experiment, the matrix factorization based collaborative filtering algorithm yields a score of 7.527% for Precision at 10, and a score of 8.397% for Normalized Discounted Cumulative Gain (NDCG) at 10. The performances are significantly higher than the Baseline's 0.941% and 0.995% respectively. (The Baseline recommends the top 10 popular songs to the user.)

KEYWORDS

Recommender System, Collaborative Filtering, Matrix Factorization, Alternating Least Square

1 Introduction

1.1 Motivation

From what to wear to what to eat for dinner, we make hundreds of choices per day. But is it always good to have more choices? Probably not in the Streaming-Music Industry. According to the data released by Spotify[1], there are currently more than 35 million songs on the server. This raises the problem of "Choice Overload", as described in Iyengar and Lepper's research [6], that too many choices could potentially be demotivating. When faced with extremely complex choices, human tend to "defer choices, search for new alternatives, or even choose the default option (opt not to choose)" [6] This problem limits the potential benefits for both the consumer and the Streaming-Music service provider.

1.2 Problem Definition

In this project, we try to solve the problem of by building a collaborative filtering-based recommender system. It connects existing users with the right items by analyzing the relationship between users and items. Specifically, we would like to provide recommendations to streaming-music service users when they are creating playlists. We believe the idea will add value to the streaming music business because, according to Music Business Association's publication in 2017, over 30% of the music listening time comes from playlist for the listeners in the United States, and a better automatic playlist continuation feature will help the industry attract and keep its users.

This topic is part of the ACM RecSys Challenge 2018, organized by Spotify, The University of Massachusetts, and Johannes Kepler University. By participating the challenge, we have the access to the Million Playlist Dataset (MPD)[2].

2 Data Exploration and Procession

2.1 Dataset

The Million Playlist Dataset is provided by Spotify and contains 1,000,000 playlists created by the users on the Spotify platform. The dataset is composed of 1,000 JSON files; each of which consists 1,000 playlists. Each JSON file is about 33 MB, the total size of the dataset is 33.54 GB.

For each playlist, we have the name of the playlist, total numbers of unique artists and albums in it, number of tracks in the playlist, number of followers, duration of the playlist (millisecond)…etc. For each track, track name, album name, artist name, duration of the track (millisecond)… and some basic information are provided.

2.2 Data Understanding

In this project, we used a subset of the data. We randomly selected 20,000 playlists from the dataset. Among the 20,000 playlists we selected, there are 269,249 unique tracks. The number of tracks per playlist range from 5 to 245; the average number of tracks per playlist is 66.64 (the average tracks per playlist in the full dataset is 66.35). From Fig. 1, we can see that the distribution is skewed and close to a Poisson distribution.

[1] https://investors.spotify.com/financials/press-release-details/2018/Spotify-Technology-SA-Announces-Financial-Results-for-First-Quarter-2018/default.aspx

[2] https://recsys-challenge.spotify.com/

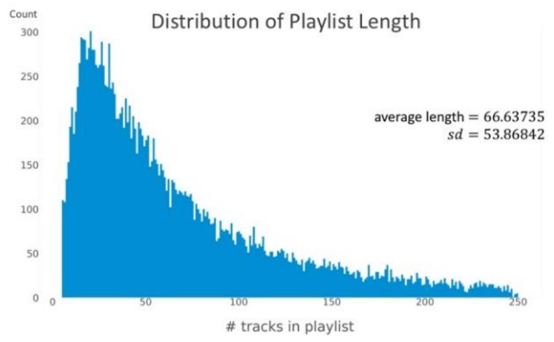


Figure 1: Distribution of playlist length where the x-axis is the number of tracks per playlist; y-axis is the count of playlist

The following are the distributions of the number of occurrences of tracks and artists. (Fig. 2 and Fig. 4) We can see that the distributions are both very skewed. The top 20 tracks appear 600 to 1,000 times while the majority of the tracks appears less than 300 times (Fig. 3). The distribution of the popularity of the artists is even more skewed; the occurrences of the majority of the artists are less than 2,500 while the number of occurrences of the No.1 is close to 17,500 (Fig. 5).

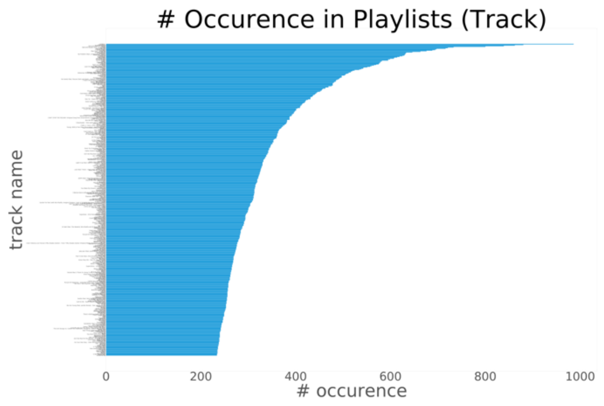


Figure 2: Distribution of number of occurrences of tracks

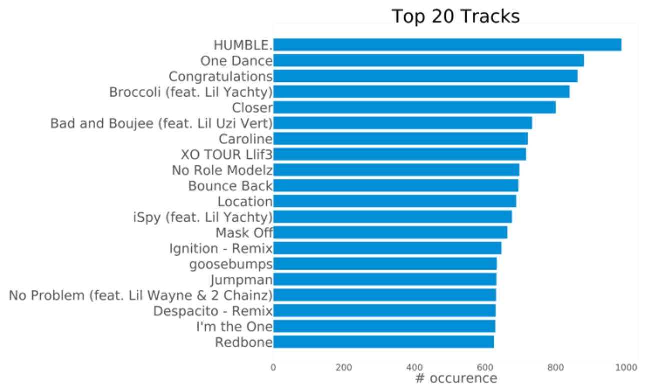


Figure 3: Top 20 tracks, ordered by number of occurrences in playlists

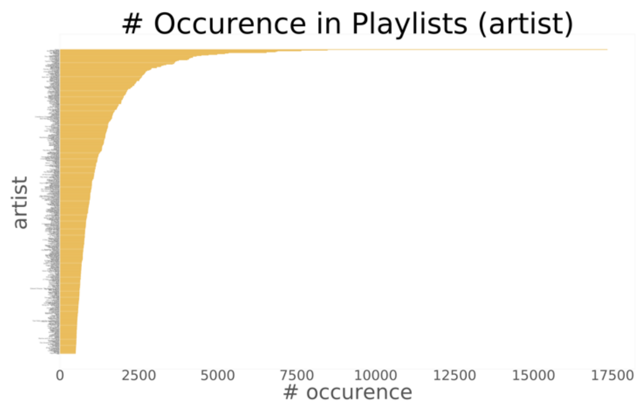


Figure 4: Distribution of number of occurrences of artists

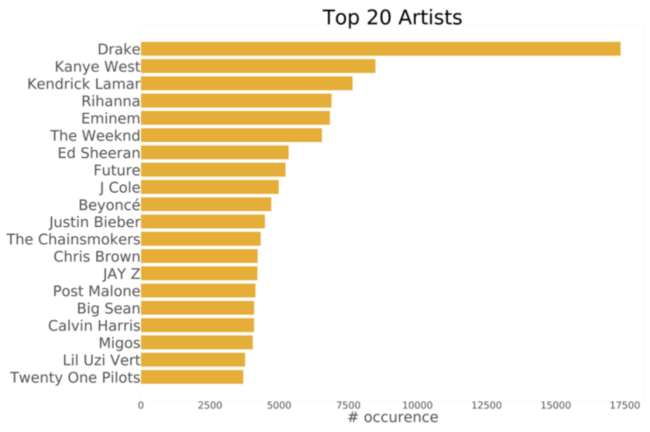


Figure 5: Top 20 artists, ordered by number of occurrences in playlists

2.3 Data Preparation

We constructed the data into a matrix with the size of n×m, where n is the number of playlists; m is the total number of unique tracks that are in the playlists. The (i, j) entry would be 1 if the i^{th} playlist contains the j^{th} track of the total list of tracks (Fig. 6). Since the number of tracks in each playlist is much smaller than the total number of unique tracks and the matrix is really large, we construct it as a sparse matrix. This would largely enhance the computational and storage efficiency. The resulting sparse matrix has a size of 20,000×269,249; the density is 0.0247%; the sparsity is 99.9753% (there are 1,332,747 non-zero elements in the matrix).

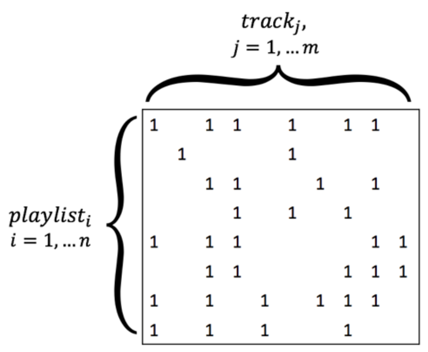


Figure 6: Playlist-track matrix $\boldsymbol{X}$; size = n×m; (i, j) = 1 if track$_j$ is in playlist$_i$.

3 Methodology

3.1 Collaborative Filtering and Matrix Factorization

This project focuses on utilizing matrix-factorization methods to build a collaborative filtering-based recommender system. We can break down the concept into two parts: Collaborative Filtering and Matrix Factorization.

3.1.1 Collaborative Filtering: Collaborative filtering utilizes only user-item information to make recommendations. The underlying assumption is that, if two users have the same "taste" for a specific item, they are more likely to have same "taste" on a different item than that of another randomly selected user.

This is particularly suitable for our dataset, as we do not need the playlist (user) specific information, such as the length, the modified date, the majority genre, etc, or the track (item) specific information, such as the tempo, the tonality, the duration of the track, etc. Instead, we only need the relationship between the playlist and the tracks. This can be represented by a playlist-track matrix, filled with 1's, indicating if a specific song is in the playlist or not. (See Fig. 6) Note that we only observe part of this matrix, and the goal is to fill in the missing part to form the recommendations of tracks for each playlist.

3.1.2 Matrix Factorization: The main assumption for the matrix factorization approach is that there are "latent features" correlated to the playlists and the tracks respectively. The relationships can be represented by two lower-rank matrices. By calculating the product of two lower-rank matrices, we get an approximation of the original playlist-track matrix, which contains the potential recommendations.

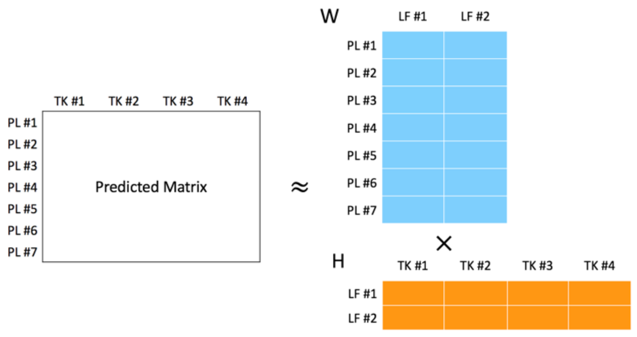


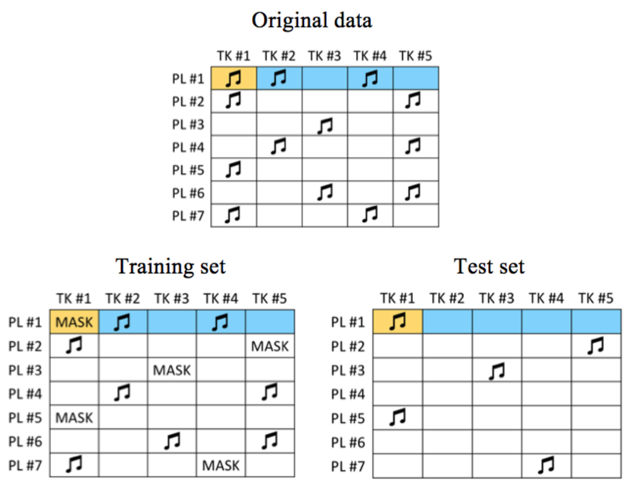
Figure 7: Production of playlist-latent feature matrix W and track-latent feature matrix H represents the prediction. Note that PL represents Playlist; TK represents Track; LF represents Latent Feature.

3.2 Train - Test Split

Before training the model, the dataset should be split into training set, validation set, and training set. Different from the traditional train-test split, we cannot randomly select rows (samples), as we need the same dimension for the matrices for the test and the training set.

In the ideal world, the evaluation of a recommender system is based on feedbacks from users. But since we cannot do this in the project, we randomly mask out tracks from each playlist. Fig. 8 gives an example of how masking works. Take playlist 1 for example, track 1, 2, 4 are in it. We randomly select track 1 to be in the test set. Therefore, in the train set, we only include track 2 and track 4 in playlist 1.

In this project, we split the data into 64% training set, 16% validation set and 20% test set.


Original data

	TK #1	TK #2	TK #3	TK #4	TK #5
PL #1	♫	♫		♫	
PL #2	♫				♫
PL #3			♫		
PL #4		♫			♫
PL #5	♫				
PL #6			♫		♫
PL #7	♫			♫	

Training set

	TK #1	TK #2	TK #3	TK #4	TK #5
PL #1	MASK	♫		♫	
PL #2	♫				MASK
PL #3			MASK		
PL #4		♫			♫
PL #5	MASK				
PL #6			♫		♫
PL #7	♫			MASK	

Test set

	TK #1	TK #2	TK #3	TK #4	TK #5
PL #1	♫				
PL #2					♫
PL #3			♫		
PL #4					
PL #5	♫				
PL #6					
PL #7				♫	

Figure 8: Test-train split. The note icon represents ‘1’ in our playlist-track matrix $\boldsymbol{X}$.

3.3 Objective Definition and Optimization with Alternating Least Square

Formally, we can structure the problem with the optimization objective, and the optimization algorithm as follows:

3.3.1 The optimization objective:

Recall that we approximate the original playlist-track matrix by calculating the product of the playlist-latent feature matrix W, and the track-latent feature matrix H. This can be written as:

$$X \approx \hat{X} = W \cdot H, \quad W \in \mathbb{R}^{n \times k}, H \in \mathbb{R}^{k \times m}$$

$$\widehat{X_{ij}} \equiv r_\theta(i,j) = w_i^T h_j$$

The quality of the approximation can be evaluated by the squared difference between the original value X_{ij} and its approximation $r_\theta(i,j)$:

$$\ell\big(r_\theta(i,j),\, X_{ij}\big) = (r_\theta(i,j) - X_{ij})^2$$

Note that not all the values in X are observed. As a result, the optimization problem is to minimize the sum of the squared loss across all the "observed value":

$$\underset{w_{1:n} h_{1:m}}{minimize} \sum_{i,j \in S} (w_i^T h_j - X_{ij})^2 + \alpha \parallel W \parallel_2^2 + \alpha \parallel H \parallel_2^2$$

3.3.2 The optimization algorithm - Alternating Least Square

Simultaneously optimizing the lower-rank matrix W and H is a non-convex and NP-hard problem. While seeking help from gradient descent is a common practice, it could cost a lot of iterations to converge and each iteration could be slow considered the size of the matrix. In this project, we alternate between optimizing one of the W or H at a time, while treating the other as constant, and repeat this process until convergence. Note that once W or H is fixed, our optimization objective becomes convex, and has an analytical solution. This is known as the Alternating Least Square algorithm and can be express by the following pseudo code:

```
Initialize W and H

Repeat:
    $w_i := \Big(\sum_{j:(i,j)\in S} h_j h_j^T + \alpha I_k\Big)^{-1} \Big(\sum_{j:(i,j)\in S} h_j X_{ij}\Big), \; i = 1, \dots, m$
    $h_j := \Big(\sum_{i:(i,j)\in S} w_i w_i^T + \alpha I_k\Big)^{-1} \Big(\sum_{i:(i,j)\in S} w_i X_{ij}\Big), \; j = 1, \dots, n$
Until Convergence
```

The trained models optimized by using ALS can be found here (https://drive.google.com/open?id=1klRLYmSvIBUvcnnVyHYQI5cwMdv8Rl4A).

3.3.3 A Detour - Non-Negative Matrix Factorization

As an additional reference, we build another matrix factorization model base on Non-Negative Matrix Factorization. The decomposition method imposed an additional constraint on the lower-rank matrices W and H, that the values in W and H cannot be negative. In the context of a recommender system, this could enhance the interpretability of the latent factor matrices.

For example, if more contexts are available to infer the meaning of the latent factors, we might be able to say that, the song "Hey Jude" is 0.2 * "70's Rock Ballad" (Latent factor 1) and 0.15 * "The Songs You Should Listen to Before You Die" (Latent factor 2). The concept is closely related to Topic Modeling in Natural Language Processing, but empirically, the recommendation qualities in terms of recommendation accuracy are not necessarily better.

We test the method out of curiosity, and for future side project of exploring the latent factors. In our experiments, we have trained 8 models using different hyper-parameters, and the score of Precision at 10 is not significantly different than the result of ALS. As a reference, using the hyper-parameter (latent features = 50, regularization coefficient alpha = 0.01), ALS and NMF yields 0.05408 and 0.05 respectively. A link to the NMF models is here (https://drive.google.com/open?id=1bDK1R_kjdoGLdLfjfX7d6rMjzVCigNpY).

4 Evaluation and Results

4.1 Evaluation Metrics

We decided to use two metrics to evaluate our outcome: Precision at K and Normalized Discounted Cumulative Gain (NDCG) at K. Two metrics are equally important, as Precision at K provides a more intuitive interpretation and NDCG accounts for a more realistic setting, in which the recommendation order matters. The details are explained as follows:

(Note: In the following, G is the ground truth set of tracks; R is the ordered list of recommended tracks.)

4.1.1 Precision at K

Precision at K is the number of tracks in the intersection of the ground truth and the top-K tracks of our recommendation divided by K.

$$\text{Precision at K} = \frac{|\mathrm{G} \cap \mathrm{R}_{1:\mathrm{K}}|}{\mathrm{K}}$$

This metric focus on the number of recommended tracks that match the ground truth, regardless of the order. For example, Precision at 5 measures "Among the 5 recommendations the system provides, how many of which are accepted by the user?"

4.1.2 Normalized discounted cumulative gain (NDCG) at K

$$\mathrm{NDCG} = \frac{\mathrm{DCG}}{\mathrm{IDCG}}$$

NDCG is the ratio between DCG and IDCG; DCG measures the ranking quality of the recommendation. That is, if tracks in the ground truth set G appear in the recommendation earlier, it will get a higher score; IDCG is the maximum score that a set of recommendation can achieved

$$\mathrm{DCG} = \mathrm{rel}_1 + \sum_{i=2}^{K} \frac{\mathrm{rel}_i}{\log_2 i} \qquad \mathrm{IDCG} = 1 + \sum_{i=2}^{|G \cap R_{1:K}|} \frac{1}{\log_2 i}$$

Note: $rel_i = 1$ when the i^{th} recommendation is in the ground truth set G. $rel_i = 0$ when the i^{th} recommendation is not in the ground truth set G.

The Normalized discounted cumulative gain (NDCG) at K focus more on the sequence of the recommendation. For a concrete example, consider the following settings:

The ground truth set is tracks {A, B, C, D, E}

The recommendation set 1 is {A, B, C, G, H}

The recommendation set 2 is {A, G, C, B, I}

Recommendation set 1 and set 2 will have the same Precision at 5 (both got 3 out of 5 right); however, Recommendation set 1 will get a higher NDCG since the tracks {A, B, C} in the ground truth set were recommended earlier.

To summarize, Precision at K focuses on the accuracy of recommendation with a more intuitive interpretation and NDCG focuses on a more realistic ranking quality that accounts for the order of recommendations. We use both metrics to evaluate the recommendation.

4.2 Results

We used the validation set for hyper-parameters selection. In particular, the hyper–parameters are number of latent features and the regularization term α. The selection is done by doing grid-search on {number of latent feature: 10, 25, 50, 100, 1000} and {α: 0.1, 0.01, 0.001}. The results are shown in Fig. 9 and Fig. 10.

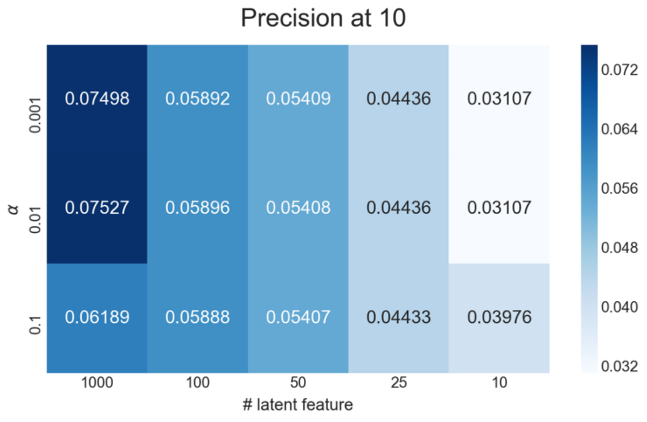


Figure 9: Grid search result of precision at K (K = 10)

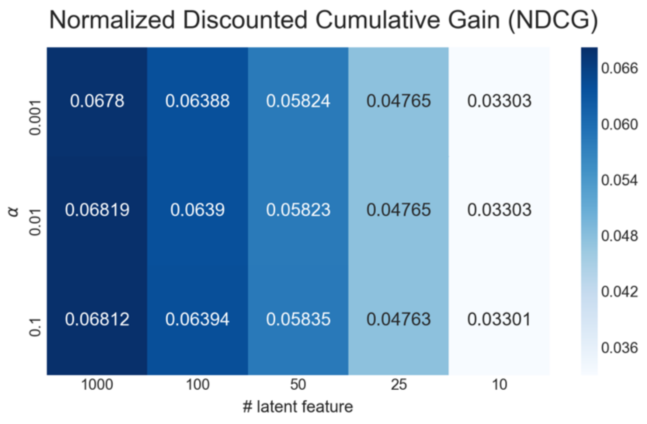


Figure 10: Grid search result of normalized discounted cumulative gain

From the result of the grid search, we can see that models using more latent features perform better, while the models using different α doesn't seem to perform very differently. For evaluation, lager Precision at K and NDCG are better; therefore, we chose the number of latent features = 1000 and α = 0.01 as our hyper-parameters. We stop testing the model performance with larger latent features because the incremental performance is diminishing, even when the number of the latent features used is increasing on a log scale.

Table 1 is the comparison between the result of the model performance on the test set and the performance of the baseline. The baseline we use is simply recommending top-K tracks (in our case, K = 10) to each playlist. Comparing to the baseline, our model performance is pretty well. Roughly speaking, for every 100 songs our system recommended, around 7.5 songs will be liked the user (or will be added to the playlist).

Table 1: Model performance

(K = 10)	Model	Baseline
Precision at K	7.527%	0.941%
NDCG at K	8.397%	0.995%

5 Conclusions

In this project, we implement a recommender system to provide streaming-music users recommendations for playlist completion. Our model yields a better result compared to the baseline, and have similar performance compared to models utilizing NMF algorithms.

For a better model performance, we can utilize the uri provided in the Million Playlist Dataset to get the track specific features. In that case, feature based matrix factorization model like Factorization Machine can be used. Empirically, it often performs better than pure matrix factorization approaches. This would be another interesting topic to explore [4].

References

[1] Benzi, K., Kalofolias, V., Bresson, X., & Vandergheynst, P. (2016). Song recommendation with non-negative matrix factorization and graph total variation. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2439-2443.

[2] Ben Frederickson. (2017). AlternatingLeastSquares — Implicit 0.3.4 documentation. Retrieved from http://implicit.readthedocs.io/en/latest/als.html

[3] Ben Frederickson. (2016). Faster Implicit Matrix Factorization. Retrieved from https://www.benfrederickson.com/fast-implicit-matrix-factorization/

[4] Chou, S., Yang, Y., & Lin, Y. (2015). Evaluating music recommendation in a real-world setting: On data splitting and evaluation metrics. *2015 IEEE International Conference on Multimedia and Expo (ICME)*, 1-6.

[5] Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative Filtering for Implicit Feedback Datasets. *2008 Eighth IEEE International Conference on Data Mining*, 263-272.

[6] Iyengar, S.S., & Lepper, M. (2000). When choice is demotivating: can one desire too much of a good thing*? Journal of personality and social psychology*, 79 6, 995-1006.

[7] Jesse Steinweg-Woods, P. (2016). A Gentle Introduction to Recommender Systems with Implicit Feedback. Retrieved from https://jessesw.com/Rec-System/

[8] Koren, Y., Bell, R.M., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. Computer, 42.

[9] Schedl, M., Zamani, H., Chen, C., Deldjoo, Y., & Elahi, M. (2017). Current Challenges and Visions in Music Recommender Systems Research. *CoRR, abs/1710.03208*.