Find Similar Users based on Music taste

Ziyue Wang
wang.ziyue1@northeastern.edu

Chamanthi Aki
aki.c@northeastern.edu

ABSTRACT

In this paper, we present a recommendation system to recommend similar users and their songs to a target user. We believe by enhancing the social networking aspect, it will further reinforce users' dependencies on music platforms and create more profits. Because it is difficult to measure the similar users we found for the target user explicitly, we conduct experiments on clustering to evaluate the validity of our pre-processing. The result demonstrates the effectiveness of scaling and averaging, and the authenticity of the similar users we recommended implicitly. Then we used the Surprise library to build a utility matrix and decompose it to recommend music and users such that the music is recommended only when the estimated listen count is above the mentioned threshold listen count.

Keywords

User Similarity; Dimensionality Reduction; SVD (Singular Value Decomposition); Clustering; Recommendation System

1. INTRODUCTION

In the audio streaming and music services industry, platforms like Spotify or Apple Music have already built industry-leading recommendation systems that preciously provide songs that we might like. However, there are two problems: first, they only focus on the music listening experience, while ignoring the social needs and community values when people are drowning in the ocean of music. Second, suggestions generated by algorithms cannot match our wide music tastes. People tend to discover the latest music mostly through friends' recommendations.[1] We want to bridge the gap and fully discover the power of music that could potentially bring people together, build relationships, and further expand their music community. Because of the strong social connections, this would strengthen the tie between users and the music streaming platform, reducing the churn rate and increasing profits as a result.

In our project, we recommend new friends and their favorite songs to users based on their similarity of music tastes. We utilize the user listening habits dataset collected from last.FM [2], and find similar users based on listening counts. We also performed clustering on these users to discover further insights.

In the second section, we introduce our dataset and the way we processed them. We also did some exploratory data analysis on the data. The third section describes the methodology in detail, including how we find similar users and build the song recommendation system. The fourth section describes how we evaluate our results. The 'Conclusion' section concludes what we did in this project. Lastly, we list our limitations and future works.

In this project, Ziyue is responsible to do the data pre-processing, dimensionality reduction, find similar users and conduct clustering experiments. Chamanthi is responsible to do the data pre-processing, music recommendation system and user recommendation system in the mentioned two approaches by implementing SVD algorithm.

2. DATASET AND PRE-PROCESSING

2.1 Dataset

This dataset is a 2.53 GB tsv file that contains up to 1k users' music listening records from 2005 to 2009 on the platform Last.fm. There are 6 columns: userid, timestamp, artid(artist id), artname (artist name), traid (track id), and traname (track name). Here is an example of the data:

user_000012 \t 2007-04-20T04:41:32Z \t b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d \t The Beatles \t 3fab7ff4-9502-4efd-8a22-8df790ac2a6b \t Happiness Is A Warm Gun

We only took the listening records in March 2008 as our input data due to the computation and memory limitation of our local machine, assuming we recommend users and music to them at the beginning of April 2008. There are 482386 entries, including 590 users and 178423 tracks from 29197 unique artists.

2.2 Exploratory Data Analysis

The most popular music from 2005 to 2009 is *Such Great Heights* from *The Postal Service* which has been listened to 3991 times and the most popular artist is *Radiohead* with 115099 listening counts in total (see Table A.1 and Table A.2). Through the hourly and monthly count distribution plot, we know that people tend to listen to music during the night and they are less likely to listen to lots of music during the summer (see Figure.1 and Figure.2).

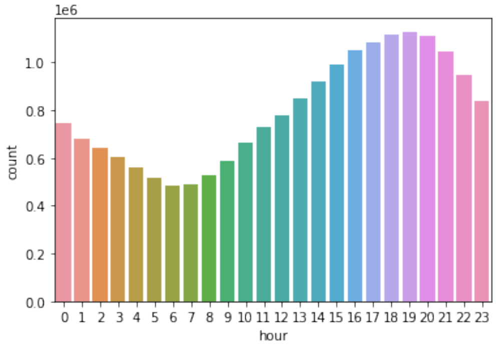


Figure.1 User hourly listening count distribution

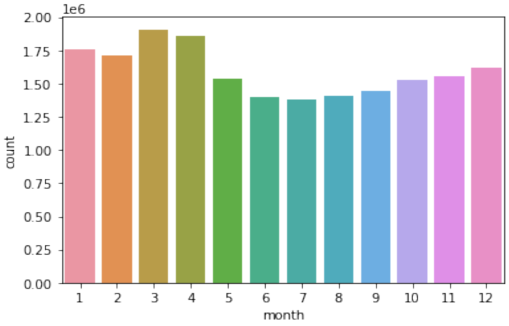


Figure.2 User monthly listening count distribution

In the listening history data in March 2008, the distribution is heavily right-skewed. Most of the songs were listened to only once. However, some of the crazy fans even listened to the same song more than 250 times.

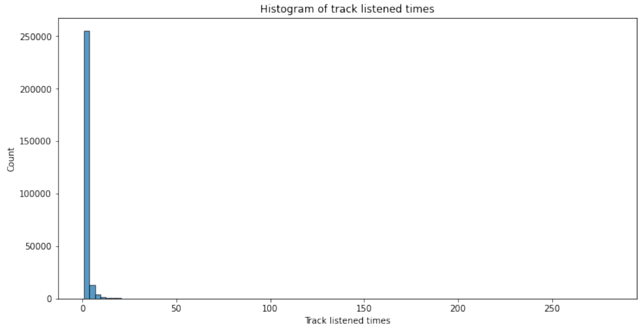


Figure.3 Track listening counts histogram

Because there are 11% missing values in track id, we decided to combine the artname and traname column with '_' as a unique representation of each track. Then we count the occurrence of different tracks for each user and pivot it to *nxm* matrix A, where *n* is the number of users and *m* stands for the number of music tracks. *A(i,j)* represents how many times a user *i* listened to song *j* in March 2008. Certainly, this matrix is a high-dimensionality and heavily sparse matrix with more than 178 k features.

3. PROPOSED APPROACH

3.1 Find Similar Users

3.1.1 Dimensionality Reduction

Data pre-processing in the last section provided us with a sparse and high-dimensional matrix. It's hard to do various matrix calculations on such high-dimensional data. So we first need to perform dimensionality reduction. We decided not to use PCA (principal component analysis) due to the reason that centering the data requires intense computation and also generates a big and dense matrix that may crash our machine during the PCA calculation. Instead, we use truncated SVD from scikit learn which does not require data to be centered, which means it can work with sparse matrices efficiently. Truncated SVD uses a randomized SVD algorithm, where for a matrix that is too large to fit in fast memory, the randomized techniques require only a constant number of passes over the data, as opposed to O(k) passes for classical algorithms. [3]

We visualized the data by taking the first and second-largest singular values to form the 2d scatter plot (see Figure.4). To determine the number of reduced features we need, we plot the graph of reconstructed error over n (number of features we keep). As we increase the number of components, the reconstructed error is getting smaller. Finally, we picked 200 as this amount of components keeps the most variance of the original data matrix (see Figure.5).

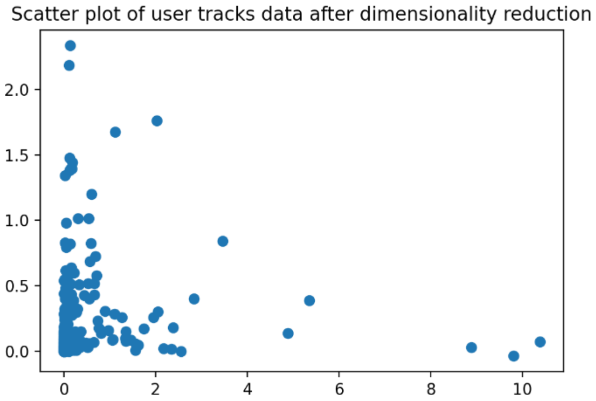


Figure.4 Scatter plot of users after dimensionality reduction

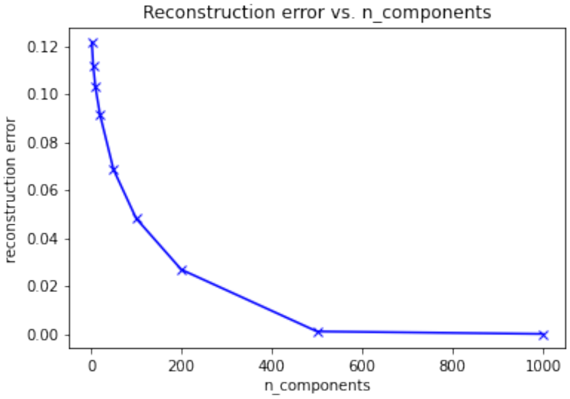


Figure.5 Reconstruction error over k in randomized SVD

3.1.2 Build Similarity Matrix

To find similar users, one naive solution is calculating the distances between users according to their listening counts. In our case, due to the large volume of songs, we used the dimensionally reduced data as our original input and calculated the similarity based on the Euclidean distance between each row. The similarity formula is as calculated as *s(i,j) = 1/(1+d(i,j))*. By filtering out users who have the top 5 largest similarity scores, we were able to find similar users for each of them.

However, there are two things we can do to improve this similarity matrix calculation. First, each song has different popularity. When calculating the similarity score, we don't want the weight of those popular songs to become too large. Second, each user has a different listening habit, some users listen to music every day while some only listen a few times a week, we don't want the latter to be ignored either. Accordingly, we normalize the listen counts of each track between 0 and 1 by MinMaxScaler. We also calculate the mean of each row (user) excluding empty values and subtract it from the original utility matrix. By doing this, we can obtain a matrix without the bias of popularity of song and listening variations of users.

3.1.3 Clustering

Since it's hard to measure the result of similar users we found explicitly, we tried to use different clustering algorithms to see how well we can cluster these users. Another reason we do clustering is to explore with only utility matrix between users and music tracks, what preprocessing techniques could improve our clustering performances. This will be elaborated in the Evaluation section. We tried K-means, HAC (Hierarchical Agglomerative Clustering), DBSCAN, and Spectral Clustering. One of the challenges to use parametric clustering–clustering algorithms that need k as a component parameter–is to determine how many clusters we want. In our case, we want to recommend up to 5 similar users to the target user. Therefore, 100 will be a good estimate of the cluster number we want. We also used T-SNE to visualize the data distribution. In summary, our workflow is described as follows:

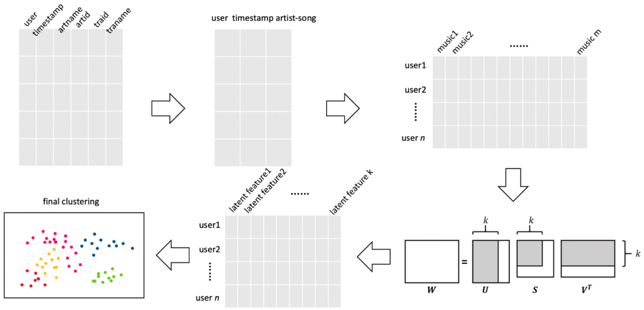


Figure.6 Flow chart to find similar users

3.2 Recommendation System

The recommendation system is modeled using the below 2 approaches:

3.2.1 Using scipy library for SVD

Matrix factorization can be used to discover features underlying the interactions between two different kinds of entities. And one obvious application is to predict ratings in collaborative filtering—in other words, to recommend items to users.

Matrix factorization discovers the hidden features between two entities and one of its applications is to predict listen count of the music tracks to the users to develop a music recommendation system. Recommendation systems use matrix factorization in a way that the matrix of listening counts (utility matrix) is factored into a product of matrices representing latent factors for the music tracks and the users.

The main advantage of using Matrix factorization is that it can utilize implicit feedback in analyzing users' behavior, such as frequently bought or viewed items. This behavior helps us to predict the listen count for the user to the music tracks which are not listened to which further leads to a personalized experience.

Utilized scipy library in implementing SVD algorithm. Developed a pivot matrix from the data on music tracks and users which store listen count for each track and user. Normalized the matrix by each user's mean and converted it to an array. Implemented Singular Value Decomposition choosing k=50 as the number of latent factors. Decomposed the matrix to find the predictions on listen count of the music tracks for each user.

Music tracks that are not listened to by the user will be recommended based on the condition of predicted listen counts greater than a specific value. Users who listen to the recommended music will also be recommended to the users.

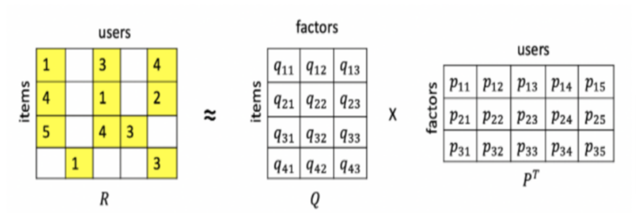

users
items
1
3
4
4
1
2
5
4
3
1
3
R
≈
factors
items
q_{11} q_{12} q_{13}
q_{21} q_{22} q_{23}
q_{31} q_{32} q_{33}
q_{41} q_{42} q_{43}
Q
X
users
factors
p_{11} p_{12} p_{13} p_{14} p_{15}
p_{21} p_{22} p_{23} p_{24} p_{25}
p_{31} p_{32} p_{33} p_{34} p_{35}
P^T

Figure.7 SVD matrix factorization

3.2.2 Compare RMSE on algorithm using surprise-library before and after Cross-Validation. [4]

Surprise is an open-source Python package that allows building recommender systems using explicit rating data for developers. Used this surprise package to develop music and user recommendation.

The dataset is trained and cross-validated on a model that implements SVD to build a recommendation system. SVD is a popular matrix factorization algorithm that can be used in recommendation systems. The model is cross-validated using three-fold cross-validation. The model is trained on the entire dataset using fit() method after cross-validation into a Surprise Trainset object using build_full_set() method. Further, implemented GridSearchCV to tune the hyperparameters on the SVD algorithm. The model is evaluated based on the RMSE metric.

4. EXPERIMENTS

To measure if scaling the listening count and averaging each user's habit contributes to our final result explicitly, we measure the silhouette score of clustering based on the data derived from the above processing. For the naive solution mentioned in the Method section, silhouette scores of DBSCAN and Spectral Clustering are negative (see Table.1). Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar. [5] As a comparison, silhouette scores increase in varying degrees after performing the scaling and averaging. Meanwhile, the distortion error of k-means when k is 100 decreases 56% after averaging and 70% after scaling. In addition, from the T-SNE plot, we can tell small clusters are distributed denser and slightly far from each other (see Figure.7).

Table.1 Comparison of Silhouette score of different clustering algorithms

	K-means	DBSCAN	Spectral Clustering	HAC
Without processing	0.5315	-0.6053	-0.0823	0.5124
After Averaging	0.5621	0.4213	0.4213	0.6263
After Scaling	0.3240	0.4284	0.4284	0.3112

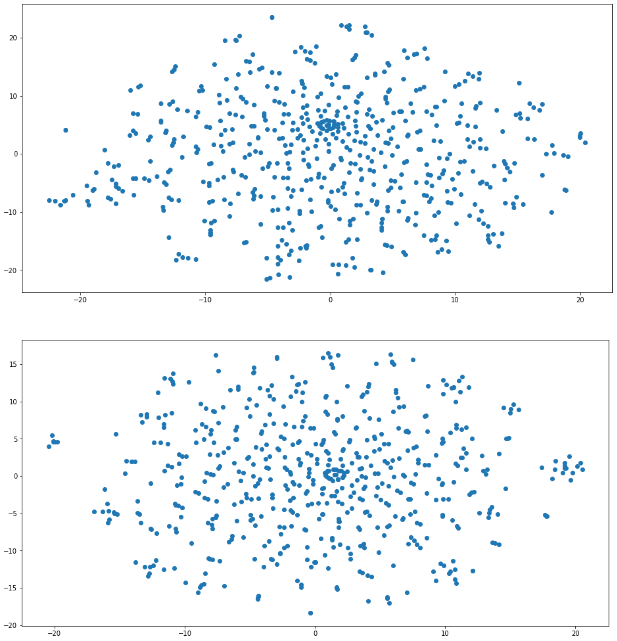


Figure.8 T-SNE plot of reduced data before and after processing

The Recommendation model showed a better RMSE value when cross-validation is implemented on the dataset. The model before cross-validation has an RMSE value of 3.017 and 2.41 RMSE after implementing cross-validation on the dataset.

5. CONCLUSION

Out of the motivation to enhance the social networking aspect of music service providers, our project focuses on finding similar users and recommending their favorite music based on music listening history on Last.FM. We believe by recommending similar users and their songs will further reinforce users' dependencies on music platforms and create more profits. Since it is difficult to measure the similar users we found for the target user, we conduct experiments on clustering to evaluate the validity of our pre-processing. The result demonstrates the effectiveness of scaling and averaging, and the authenticity of the similar users we recommended implicitly. We also developed a Recommendation system to recommend music tracks and users in 2 approaches using Singular Value Decomposition. Initially, scipy library is used for matrix decomposition and implementing SVD, whereas the second approach used the Surprise library to build a utility matrix and decompose it to recommend music and users such that the music is recommended only when the estimated listen count is above the mentioned threshold listen count. The model is evaluated on RMSE before and after cross-validation where the implementation of cross-validation showed a significant improvement in the model.

6. FUTURE WORK

Admittedly, due to time constraints, this project has some limitations. For example, although the dataset includes more than 170k music, the number of users is still relatively small. Meanwhile, we are recommending users and songs based on the data from a decade ago. Further work can be done to incorporate more social data so that we can recommend friends of recommended users. Additionally, we need to address the cold start problem since new users will not be able to recommend any users as they might not have listened to the music tracks above the given threshold value in the mentioned algorithm. Gray sheep problem might be also faced as there might exist some users whose opinions consistently would not match with any group of people and so their opinions will not benefit from collaborative filtering.

7. REFERENCE

[1] https://medium.com/@cindy.huang/spotify-case-study-discovering-music-through-friend-recommendations-ee21df5c24aa

[2] http://ocelma.net/MusicRecommendationDataset/lastfm-1K.html

[3] Halko, N., Martinsson, P.-G. & Tropp, J. A. (2009). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions arXiv.org > math > arXiv:0909.4061 . (cite arxiv:0909.4061)

[4] http://nicolas-hug.com/blog/matrix_facto_3

[5] Silhouette score https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

8. APPENDIX

Table A.1 Top 10 popular music with their total listen counts

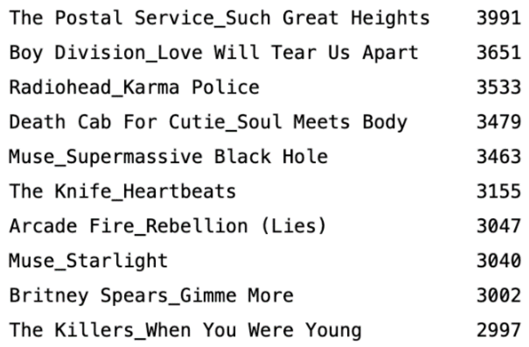
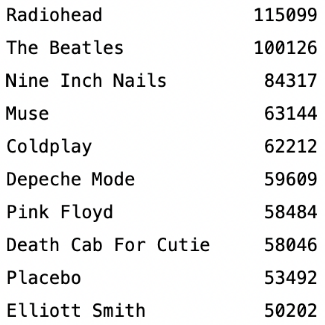

```
The Postal Service_Such Great Heights     3991
Boy Division_Love Will Tear Us Apart      3651
Radiohead_Karma Police                    3533
Death Cab For Cutie_Soul Meets Body       3479
Muse_Supermassive Black Hole              3463
The Knife_Heartbeats                      3155
Arcade Fire_Rebellion (Lies)              3047
Muse_Starlight                            3040
Britney Spears_Gimme More                 3002
The Killers_When You Were Young           2997
```

Table A.2 Top 10 popular artists with their total listen counts


```
Radiohead              115099
The Beatles            100126
Nine Inch Nails         84317
Muse                    63144
Coldplay                62212
Depeche Mode            59609
Pink Floyd              58484
Death Cab For Cutie     58046
Placebo                 53492
Elliott Smith           50202
```

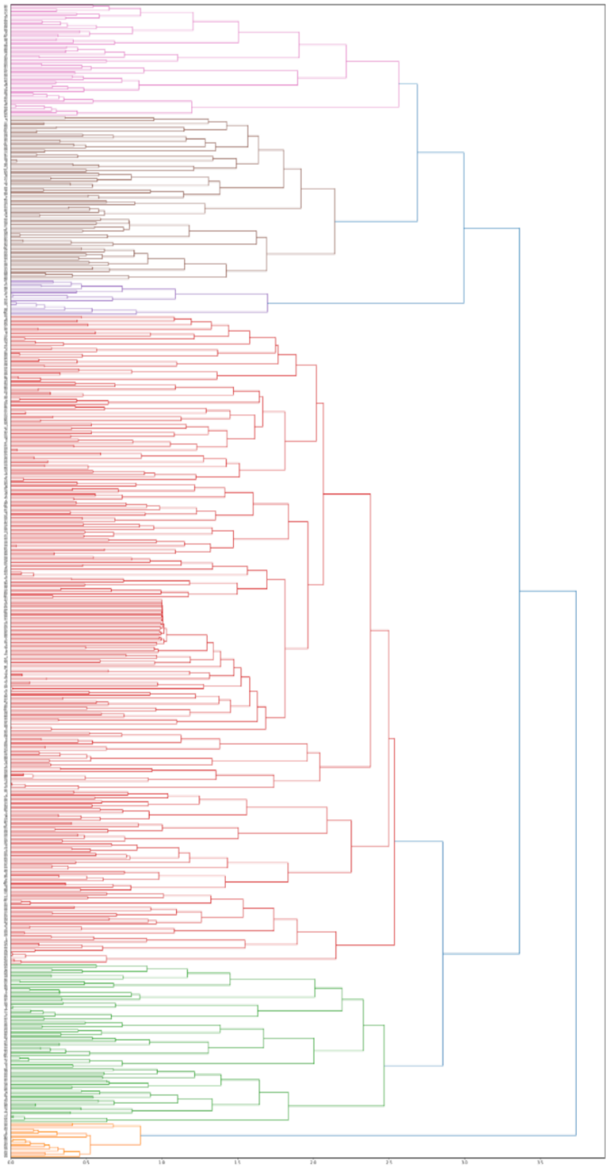
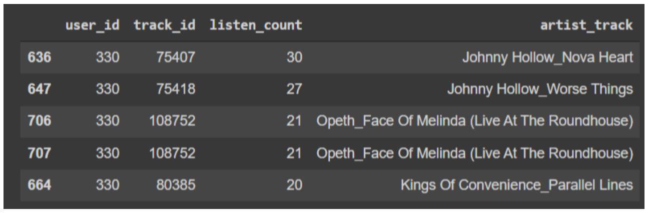


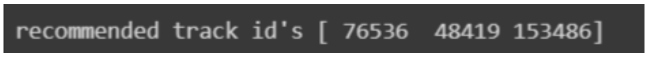
Figure A.1 Dendrogram of users after HAC using cosine distance and ward method

User and Music Recommendations using Scipy library:



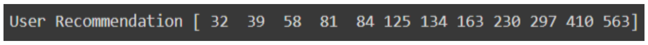
	user_id	track_id	listen_count	artist_track
636	330	75407	30	Johnny Hollow_Nova Heart
647	330	75418	27	Johnny Hollow_Worse Things
706	330	108752	21	Opeth_Face Of Melinda (Live At The Roundhouse)
707	330	108752	21	Opeth_Face Of Melinda (Live At The Roundhouse)
664	330	80385	20	Kings Of Convenience_Parallel Lines

Figure A.2 Recommended users and music tracks for user with user_id=330 using scipy library.



```
recommended track id's [ 76536  48419 153486]
```

Figure A.3 Recommended music tracks with user_id=330

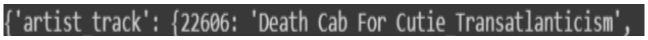


```
User Recommendation [ 32  39  58  81  84 125 134 163 230 297 410 563]
```

Figure A.4 User recommendations with user_id=330

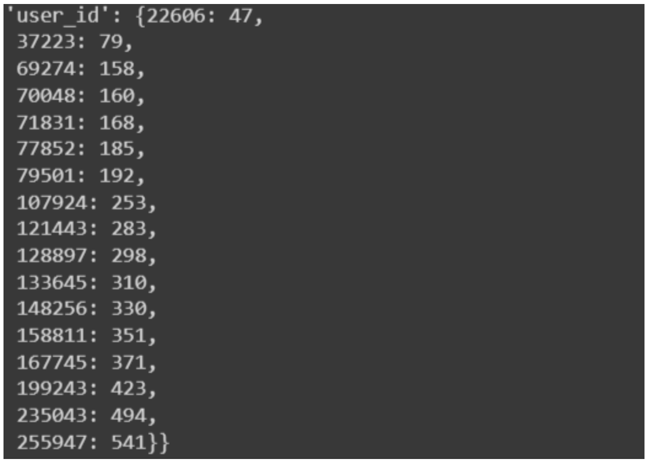
In this approach, the model is developed such that the music and users are recommended completely based on the overall user listen count prediction, where the above images describe the users and music recommendations.

Using surprise-library and Cross-Validation on the dataset:



```
{'artist track': {22606: 'Death Cab For Cutie Transatlanticism',
```

Figure A.5 Music Recommendation for the user_id=330



```
'user_id': {22606: 47,
 37223: 79,
 69274: 158,
 70048: 160,
 71831: 168,
 77852: 185,
 79501: 192,
 107924: 253,
 121443: 283,
 128897: 298,
 133645: 310,
 148256: 330,
 158811: 351,
 167745: 371,
 199243: 423,
 235043: 494,
 255947: 541}}
```

Figure A.6 Users recommendation for the user_id=330

Here, we developed a model in such a way that one can find user recommendations on a condition of a threshold listen count, and hence only one track is recommended to the user and many users are recommended.