CS-425

Algorithms for Web-Scale Data

Music Recommendation System

Final Report

Group 9

Muhammad Ramish Saeed

Eren Aslantürk

Hüseyin Eren Çalık

Babanazar Gutlygeldiyev

Arba Hoxha

13.05.2019

Table of Contents

1. Overview

There are many online music services which are similar to what we frequently watch and listen and then give a rating. To suggest the best possible recommendations for users, there are possible ways which are content-based and collaborative filtering. In this report we have implemented all of them to find out which algorithm is best suited for an online music recommendation system. Collaborative filtering includes item-item, user-user and latent factor. All of these types have different methods to recommend songs to the user so that they find it easy to listen to what pleases them.

2. Problem Description

Online music services have tens of millions of tracks. The content itself is broad and covers various musical genres as well as non-musical audio content such as radio plays and podcasts. The sheer scale and diversity of content make it difficult for a user to find relevant tracks. Relevant recommendations are therefore crucial for a good user experience [1]. Here we present a method to compute track-track similarities using collaborative filtering signals with side information such as artists, country, age, etc.

3. Algorithms Used

3.1. Content-Based

According to this approach, the recommendations are made based on the genres, which are 1998 reliable tags, in which users have higher listening events. We have used user's listening activities for this purpose. Depending on their activities we increment the value of the genres which the particular artist belongs to for that particular user. Then with that genre information, we find the similar artists using the Cosine Similarities and recommend the top three similar artists to the user. For example if a user has listened Megadeth 100 times, we increase "thrash metal", "heavy metal" and "metal" by 100. Then we look for the artists that the user hasn't listened to and according to their genre vector, using cosine similarity we calculate the value. By using this value, we decide whether we should recommend that artist to that user or not.

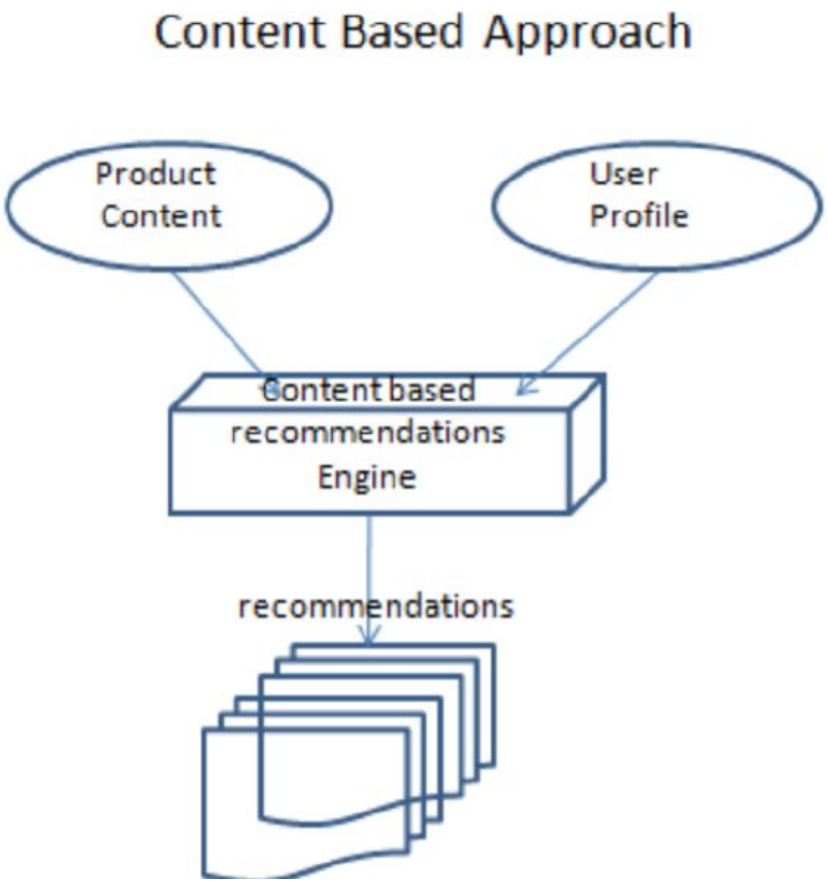

Figure 1. Content Based Approach

3.2. Latent-Factor

Latent factor model (LFM) factorizes user ratings of songs into user and item feature matrices. By this factorization, the model aims to find feature matrices to describe the data. This is accomplished by minimalizing Root Mean Square Error (RMSE). RMSE is minimalized using Stochastic Gradient Descent (SGD) technique.

In this project, this model is implemented using Python. Since LastFM dataset does not provide any ratings of users, LFM is run using Yahoo! Music Dataset. This dataset is originally divided into train and test part, each of which is 10 files. Moreover, since the dataset

contains 717000000 ratings of 136000 users to 1800000 songs, it is 0.029% full. Thus it is a sparse matrix. We deal with this sparsity storing the ratings in the array as a triple of songId, userId, and rating. Later, we convert this array to Numpy array of Python. Our P and Q matrices are also an instance of Numpy matrices filled with random numbers in the range [-1, 1]. However, dot products or other operations on matrices are implemented manually because our ratings are not in the form of the matrix.

In our tests, we used 3 and 4 number of features. Our learning rates, for the last results, are $\mu_1 = 0.002$ and $\mu_2 = 0.005$. Lastly, $\lambda_1 = 0.02$, $\lambda_2 = 0.02$, and threshold to stop calculation is 0.05.

When k=3 and with 50 users, RMSE = 1.66. It took 18 minutes to get the result. When k = 4 and with 50 users, RMSE = 1.53 with the same time as the above result.

There could be a few reasons for these results. Firstly, since the dataset is too big to fit in the RAM, we could not find the opportunity to use all ratings of the dataset. We tried to use train and test data in proportion to 80% and 20%. Secondly, the number of factors used to factorize might be different for this dataset. However, we tried to stick to what we learned in class. According to the slides, RMSE begins to rise for k > 2. Another reason could be the values of the constants and thresholds we chose.

3.3. Item-Item

In this approach, we have aimed to use a technique to reduce sparsity by choosing items as artists. User ratings are utilized to find similar items (User ratings are the features of

an artist). To improve the quality recommendations, the meta information of users which are mainly country and age are also utilized.

To recommend songs for a user, artist information is used. To accomplish this, features of artist items are extracted and item similarity matrix is conducted which is shown below in Figure 2. To calculate similarity, we have used cosine similarity as well as a hybrid version of Jaccard similarity. We created two vectors then subtracted one from the other one. Then sum of absolute values is taken and divided by dimension number which gives the average listening event numbers. After calculating the similarities with the songs the user has listened to of a specific artist, deviations are calculated. Then these deviations are multiplied with their according cosine similarity and estimate is found. Finally, these estimates are sorted and largest 20 values are recommended to the user with their artistid and the top 3 recommendation which are shown in the following table.

Artist Id	**1st preference**	**2nd preference**	**3rd preference**
2(Lil Wayne)	16252(Sean Kingston)	858(MC Hammer)	9950(Edwin Starr)
13(Limp Bizkit)	207(Fatboy Slim)	248(Slipknot)	9716(Nickelback)
21(Anastacia)	8429(Gloria Gaynor)	524(Haddaway)	15361(The Pointer Sisters)
32(The xx)	1648(Florence + the Machine)	461(Arcade Fire)	2590(MGMT)
44(Hurts)	12326(Wham!)	9577(Steppenwolf)	7197(Kool & The Gang)
45(The Kooks)	514(The Verve)	207(Fatboy Slim)	1650(Foster the People)
57(Fall Out Boy)	9615(Flo Rida)	319(Pitbull)	2789(Kaiser Chiefs)
95(AWOLNATION)	5782(Bonnie Tyler)	12326(Wham!)	2623(Survivor)
99(The Black Keys)	2596(Beck)	391(Eric Clapton)	272(Jimi Hendrix)

Figure 2. Example of recommendation

As it can be seen with a quick internet search, the artists we have found similar using the data of what kind of users they are listened to by, are in the same genre in most of the cases. Our approach can be considered as a novel one and apart from music recommendation, it can be used for artist similarity applications easily.

3.4. User-User

The datasets we are using in this algorithm is LastFm. We decide the similarity between the users by the artists the listen to. Since LastFm provides us with the userId, artistId we can easily make the connections between the userIds that listen to common artistIds. If two userIds have in common more than a few artistIds, then the artistsIds that are not in common between these two userIds are suggested to the users. In this way, we can determine similar user profiles and suggest artists to these users. In order to make our algorithm more precise and not leaving it to depend only in the artistsIds the users have in common, we decide to use listening events as well. The listening events are taken from LastFm. Accordingly, our algorithm checks not only the artistsIds in common but also how

many times a certain userId listens to a certain artist. If two userIds have a certain artistId in common, and the times they listen to this artist is similar then these two userIds are similar to each other.

3.4.1 Similarity Algorithm

While implementing the algorithm, we had to make a decision between which similarity algorithm needs to be used (cosine or jaccard similarity). By taking both of these similarity algorithms' benefits in consideration, we decide to use jaccard similarity since magnitude of the listening events is really important for our recommendation system. Since we need to decide user profiles, by taking into consideration the users' listening events, jaccard similarity gives better results while determining the similar users. We adapt jaccard similarity in a way that it integrates better with our logic of finding similar users. The jaccard similarity we use in user-user, is the same as in item-item, described in section 3.3.

4. Datasets

The datasets which we have used for our system are LastFM Dataset and Yahoo! Music Dataset. LastFM contains 17559530 unique data lines in the format of userId - artistname -

songname - numberofplays from 359347 unique users. Yahoo! Music Dataset contains 717 million ratings of 136 thousand songs given by 1.8 million unique users. The datasets will be organized in order to guarantee that they have required the information such as adding the genre information.

userId	songId	rating
0	7171	5
0	8637	4
0	21966	4
0	35821	5
0	82446	5
0	90409	5
0	107410	5
0	131919	5
0	132685	3
0	136507	3
1	3342	5
1	7522	1
1	25363	2
1	38997	5

songId	albumId	artistId	genreId
0	12070	8490	0
1	19512	7975	134
2	18953	3492	0
3	695	2653	0
4	243	2282	0
5	7783	3832	0
6	11704	1655	106
7	10126	6328	114
8	10672	4121	0
9	12302	9197	54
10	14464	6069	0
11	125	4656	0
12	16830	6857	134
13	1195	7770	0

Figure 4. Sample Screenshot of dataset

5. Libraries Used

The libraries we used are mostly python packages that are used in many scientific applications.

We used:

- Numpy for array operations.
- Scipy for calculating cosine distance.
- Pickle for disk operations.

Every other operation has been implemented by the group members.

6. Conclusion

This project helped us to see the real life applications of big data and more importantly how to deal with it and take advantage of it. It helped us to understand how programs that we are using everyday can do recommendations to us. We have implemented four different algorithms. We decided on some metrics to compare the algorithms to one another. We have seen that collaborative filtering algorithms provide the better results compared to other algorithms such as content-based recommendation.

Sparsity is the biggest issue we have seen on big data. Even though there are tons of data, matrices that we created were generally sparse and they impacted our run time. To increase efficiency, we eliminated the empty slots by using Python dict data types. Also, using artists as our recommendation items increased speed and accuracy.

As can be seen from our results, these calculations take an enormous amount of time for a tiny portion of the dataset. For example, in the LF it took 18 minutes for 23000 ratings. Original dataset has 717000000 ratings. Because of time constraints, it is almost impossible to run such an amount of data in our machines ($\frac{717000000}{23000} * \frac{1}{60} = 519\ hours$).

Collaborative filtering algorithms gave better results overall but they have the *cold start* problem. That means they can't effectively handle new items and new users because they are lacking data.

By the metrics that we have set, we can say that item-item collaborative filtering gives the best results. Also, we can learn more about the recommendation from that.

6.1 Metrics and Results

We could not use the same metrics in Latent Factor (LF) and the other methods because LF predicted ratings but other algorithms only recommend artists. Therefore, we used RMSE for LF and created our own metric (which is an accuracy value that also is in the range of 0 to 1) for other algorithms. To calculate our accuracy in the methods other than the

LF, we have created our own dataset. We have omitted the 1/3 of the total artists a particular user listened to and did random predictions for a thousand users.
The accuracy results we get from:

- User-User CF : 0.652
- Item-Item CF : 0.627
- Content-Based : 0.54
- Latent Factor : 1.53

As it can be seen, CF algorithms give better result than other methods. Also, our novel approach to find similar artists without normalizing the listening events into pseudo-ratings helps us having results that are close to state-of-art recommendation systems. Which also shows that the metrics we have chosen, which are listening events coupled with age and country information, are precise

References