Million Song Dataset Challenge

Ran Li & Jian Liu

1. Motivation

In recent years the typical music consumption behavior has changed dramatically. With the growth of digital music technology nowadays, to recommend music to users become more and more important and helpful.

From the view of music app platform, to recommend people's favorite music can help singers to popularize their songs. If the users like the recommend songs they will download from the music websites and it also can help music websites to make greater profit.

2. Project Goals

Our goals for the recommendation system can be divided into three parts:

Firstly, there are several popular algorithms for music recommendation system: collaborative filtering, content-based methods, web crawling, even human oracle. After analyzing the characteristics of these algorithms, relying on Million Song Dataset we will focus on one algorithm and build a recommendation model.

Secondly, based on the recommendation model, we will predict user preferences from item content and recommend them to users.

Thirdly, after we get the results, a test set will be used to check whether the recommendation model actually works. We will use a metrics to analyze them, then we will adjust the parameter in our model, which can give the best recommend lists.

3. Introduction

Million Song Dataset Challenge is a large scale, music recommendation challenge posted in Kaggle. The task is to predict which songs a user will listen to and make a recommendation of 500 songs to each user given the user's listening history. The data involved in competition comes from the Million Song Dataset. Million Song Dataset (MSD) is a freely available collection of data for one million of contemporary songs (e.g. song titles, artists, year of publication, audio features, and much more).

Collaborative filtering (CF) is a technique often used by recommender systems. It is a method of making automatic predictions about the interest of a user by collecting preference from many users. Collaborative filtering approach is based on the assumption that if a person u has the same opinion as a person v on an issue, u is more likely to have v's opinion on another issue x.

In our report, we firstly implement the popularity algorithm. Then we implement item-based collaborative filtering algorithm and combine the stochastic aggregation into our algorithm. After computing the result, we use mAP as our

metrics to evaluate the result. According the value of mAP, we choose the best parameter for our model, which can achieve the highest value of mAP.

4. Recommendation Algorithm

(1) Popularity algorithm

Popularity algorithm is a basic algorithm to make the recommendation. The popularity of the song p_i is defined as the number of different people, who have listened to the song, which can be expressed as:

$$p_i = \sum_{u \in U} I_{ui}$$

Where I_{ui} is 1 or 0, r_{ui} = 1 means that user u has listened to the song i, I_{ui}= 0 means that user u has never listened to the song i.

Algorithm:

(1) According to the listening history, compute all the songs' popularity.

(2) Sorted them in decreasing order in List 1.

(3) Classify songs according to users in List 2.

(4) For each user, according to List 1, check whether the song appears in List 2. If it is in, move to the next song in List 1; if not, add the song to the recommended list. (When the number of songs reaches 500, compute the next user.)

(2) Stochastic Combination Algorithm

In general, there are two types of collaborative filtering: memory-based algorithm and model-based algorithm. Model-based CF algorithms are developed using data mining, machine learning algorithms to find patterns based on training data. These are used to make predictions for real data. And for memory-based algorithm, it uses users' rating data to compute similarity between users and items. It is widely used in many commercial systems for making recommendations. It is easy to implement and effective.

Memory-based algorithm can be divided into two types. One is a user-based recommendation, which gives a new user the set of suggestions that are computed based on users with similar tastes. Another type is item-based recommendation strategy, which computes the most similar items for the items that have been already purchased by the active users.

In our project, we choose the item-based recommendation strategy. Item-based model make recommendation by finding similar songs to the songs in certain user's listening history.

(2.1) Compute Similarity for songs

As for the song similarity, we define the similarity for two songs is [4] :

$$m(S_x, S_y) = \frac{\sum_{i \in U} X_i Y_i}{\sqrt{\sum_{i \in U} X_i}\ \sqrt{\sum_{i \in U} Y_i}}$$

Where U is the set of users, X_i and Y_i are whether the song X or song Y is listened by user i before or not, respectively.

(2.2) Parametric generalization of Similarity

The cosine similarity has the nice property to be symmetric but it might not be the better choice. As an alternative to the cosine similarity, we use the conditional probability measure, which can be estimated with the following formula:

$$m(S_x, S_y) = P(S_x | S_y) = \frac{\sum_{i \in U} X_i Y_i}{\sum_{i \in U} X_i}$$

This function can provide us with the information: how likely it is that a user will appreciate an item after we already know that the same user likes another item.

Then we can build a parametric generalization model of song similarity like this:

$$m(S_x, S_y) = \frac{\sum_{i \in U} X_i Y_i}{\left|\sum_{i \in U} X_i\right|^{\alpha} \left|\sum_{i \in U} Y_i\right|^{1-\alpha}}$$

Obviously, the cosine similarity case is when $\alpha = 0.5$. And we know that different value of α give different similarity, which will lead to different recommendations.

(2.3) Scoring function

We use the exponential function that is $f(m) = m^q$ to determine how much each individual scoring component influences the overall scoring. When q is high, smaller weights drop to zero while higher ones are (relatively) emphasized. At the other extreme, when q = 0, the aggregation is performed by simply adding up the ratings.

Thus the overall scoring function, on the basis of which the recommendation is made:

$$Score_{ux} = \sum_{y \in S} f\left(m(S_x, S_y)\right) * I_{uy}$$

where $Score_{ux}$ gives the score about how much the user u like the song x. And I_{uy} means whether user u has listened to song y.

(2.3) Stochastic Aggregation

It is not easy to determine a single strategy, which is able to correctly rank the songs. It may be, that particular ranker cannot suggest other songs that the users like. Hopefully, if the rankers are different, then the rankers can recommend different songs. We apply the stochastic aggregation strategy [1], which can perfectly do the job.

They assume we are provided with the list of songs, which are not yet rated by the active user, given in order of confidence, for all the basic strategies. On each step, the recommender randomly choose one of the lists according to a probability distribution p_i over the predictors and recommends the best scored item of the list which has not yet been inserted in the current recommendation. In their approach

the best p_i values are simply determined by validation on training data[1].

5. Implementation and Results

The data that we actually used in the recommendation can be divided into two parts. One is the full listening history for 1M users (that is train_triplets.txt). Another part is half of the listening history for 110K users (kaggle_visible_evaluation_triplets.txt) (10K validation set, 100K test set), and we must predict the missing half. With the help of python, we computed the recommend list for the 110k users based on the popularity algorithm and Stochastic Combination Algorithm.

(5.1) Popularity Algorithm Result

For the popularity algorithm, we don't need the training data. The only input is the 110k visible listening history. Here is the short cut of our result. As shown in figure 2, the fist line is the user ID. The numbers from the second line are the indexes of 500 songs, where each song corresponds to a unique song ID.

```
d7083f5e1d50c264277d624340edaaf3dc16095b
91177 12985 25150 14397 288653 217471 68212 54386 177172 87433 3140
333259 25323 221730 244143 319911 241705 165401 243769 307202 52478
183796 86545 348629 170541 38941 245936 311604 302369 291341 242151
307140 2078 334187 329834 8402 180413 207935 334240 212702 208383 5
341456 207916 311262 187097 126831 212017 243307 16220 383719 10719
211334 192716 285387 233632 354625 335634 305991 281075 337625 3537
223714 266750 351764 105694 115162 90822 19682 353035 55364 138672
```

Figure 1 Result of Popularity Algorithm

(5.2) Stochastic Combination Algorithm Result

For the stochastic combination algorithm, we need the training data to compute the best values p_i. [1] After that, according to the stochastic combination algorithm we have introduced above, we get the result of 500 recommendations for the 110k visible listening histories. Again, as the shown in the following picture, these indexes are the 500 songs we recommend for the first user in the Kaggle_users.txt. The parameter is $\alpha = 0.5$ and q = 5.

```
12985 127276 288653 311604 319911 307202 221730 315812 244143 289238 207935 25323 217471
91177 233124 48688 14397 82113 192716 177486 142602 311262 341456 79814 305991 334187
212702 291341 261596 302247 148050 384072 243307 177172 266750 314086 54386 57565 241705
177574 248603 49781 19682 130077 281075 86545 277126 16174 170541 115162 333259 52478
24825 38941 183796 292298 190270 243769 37133 307140 165401 373947 190006 302369 192425
26935 289658 138672 2242 2078 329869 310418 39842 45592 207916 187097 195955 353700 348629
212017 362714 237680 337625 236511 107193 8402 52176 357810 55364 125707 16220 334240
97972 126831 329834 208383 62467 108137 105434 241255 223714 251101 353035 180413 40311
142745 309384 49469 335634 245012 11011 211334 314350 383719 339774 245936 359476 311702
25706 322843 90822 285387 109807 147248 11123 202995 354625 239018 126487 154559 259912
105694 132813 233632 123457 183790 95103 30292 242151 345265 167248 302031 127365 242558
165753 74558 188332 257947 300918 114624 3525 281102 122591 273894 343207 262242 12554
```

Figure 2 Result of Stochastic Combination Algorithm

6. Evaluation and Analysis

In the Kaggle competition, mAP (mean average precision) is used as the evaluation metric. Its advantage is that it can emphasize the top recommendations. To do the evaluation, we need the visible listening history and the hidden listening history.

Luckily, when the first edition of the contest ended in August 2012, the MSD Challenge organizers released the test data, that is the hidden part of the kaggle_visible_evaluation_triplets. [5] This provides us with enough data to evaluate the algorithm.

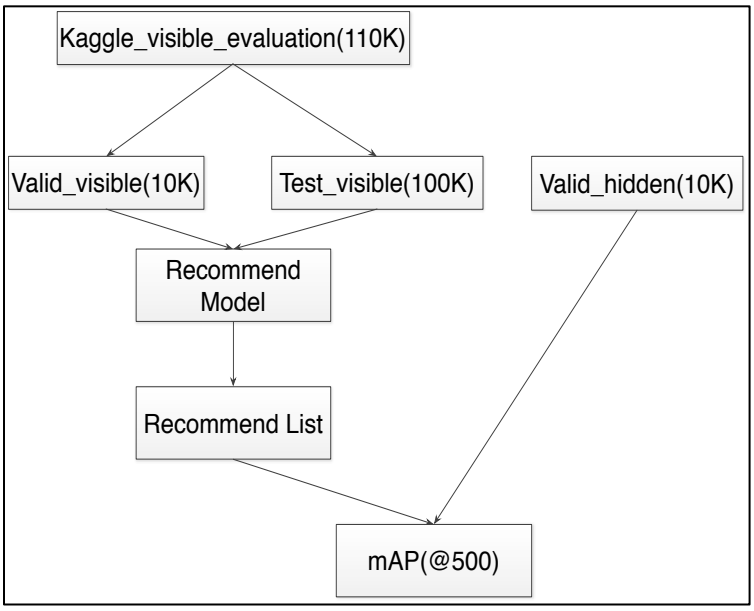

Figure 3 Evaluation Data Flow

We use our algorithm to compute the recommending list for the 110k users. The recommending lists of different users are ordered according to the kaggle_user.txt list. We have two parts of hidden parts, one is for valid triplets (10k users) and the other one is for test triplets (100k users). In consideration of the running time, we choose the valid hidden triplets to evaluate our algorithm.

In the result we get for 110k users. The 10k users in valid hidden triplets are included. For each user in 10k users, we can search the 500 songs in the result.

For user u, his hidden listening history including S_1, S_2 ..... S_n . All these songs has another rank in the 500 recommendation list R_1, R_2 ..... R_n. Thus for user u, the average precision is :

$$AP_u = \frac{\sum_{i=1}^{n} \frac{S_i}{R_i}}{n}$$

As for all the users, we compute the mean average precision:

$$mAP_U = \frac{\sum_{u \epsilon U} AP_u}{n_U}$$

where U is the set of users in hidden data. n_U is the number of users in hidden data. Obviously, the higher the mAP value is, the better the recommendation system is.

Because the popularity is a simple algorithm given by the competition organizer. So we refer the mAP value from [4]. mAP = 0.02262.

As for the stochastic combination algorithm we implement, firstly we compute the mAP value at fixed q = 5. We varied α to see how the value of α influences the value of mAP. Here is the result.

Table 1 Different value @ q=5

α	mAP
0	0.1339
0.3	0.1410
0.5	0.1435
0.8	0.1400
1	0.1315

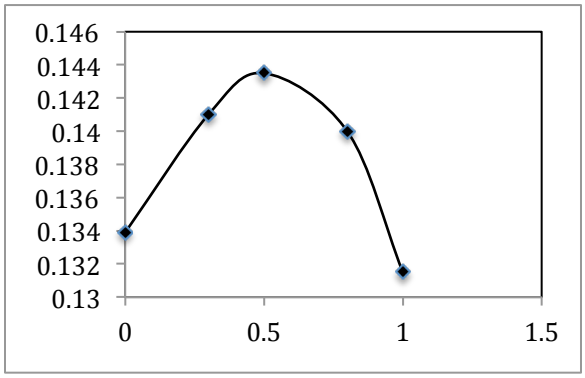

Figure 4 Different value @ q=5

According to the figure 5, when α between 0.4 and 0.6, the mAP has the maximum value. Thus, secondly we compute the mAP value at fixed α = 0.5. We varied α to see how the value of α influences the value of mAP.

Table 2 Different value @ α= 5

q	mAP
1	0.1650
2	0.1619
3	0.1547
4	0.1513
5	0.1478

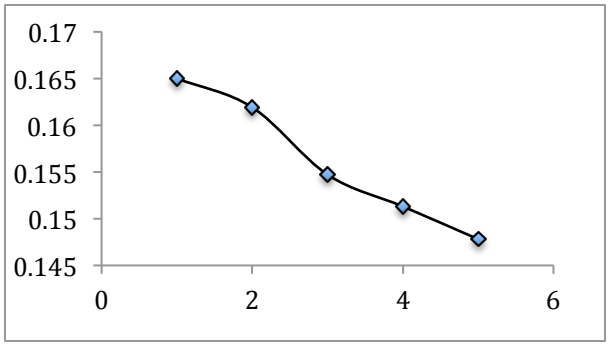

Figure 6 Different value @ α=0.5

According to figure 6, the mAP has its peak value at q = 1. Above all, we find that in our experiment, when q = 1, α=0.5 the mAP has maximum value, that is, we recommend system performs best at these two values.

7. Future Work:

Our project mainly focuses on the song similarity model. According to [4], we can combine the user similarity model and song similarity model in a linear way. Probably we can get a better prediction.

The MSD provides us with a huge amount of data. It could be useful to consider the additional meta-data, which are also available, and to construct alternative rankings based on that. And we don't use the information counts. We just assume that if a user once listened to this song, he liked it. Actually, this is not always true.

Our code takes a long time to compute the recommendation. We should focus on how we can improve the efficiency of our code so that it can cost less time to get the recommendation.

8. Reference:

[1] F. Aiolli, A Preliminary Study on a Recommender System for the Million Songs Dataset Challenge Preference Learning: Problems and Applications in AI (PL-12), ECAI-12 Workshop, Montpellier

[2] http://www.kaggle.com/c/msdchallenge!

[3] http://en.wikipedia.org/wiki/Collaborative_filtering

[4] Fenxuan Niu, Ming Yin, Cathy Tianjiao Zhang, Million Song Dataset Challenge

[5] http://labrosa.ee.columbia.edu/millionsong/challenge#data1.