Collaborative filtering for Music Recommendation System

Andrey Kulagin
Department of Computer Science
Innopolis University
Innopolis, Russia
a.kulagin@innopolis.ru

***Abstract*—Collaborative Filtering is one of the most popular techniques used in recommendation systems in general and particularly for music recommendations. In this article, we'll compare two kinds of Collaborative Filtering (user-based and item-based) on a real dataset and test various improvements in order to achieve better results.**

***Keywords*—collaborative filtering, recommendation systems.**

I. Introduction

Today recommendation systems is an essential feature for many services: music and video streaming, retail, feed in social networks... They increase sales and help retain users. Recommendation systems can be divided into two big categories: content-based filtering and collaborative filtering (CF) [1]. In this work we will consider the last one in the domain of music recommendations.

Collaborative filtering relies on behavior of users. The main idea is that people, who share interest in certain things will probably have similar taste for other things. And there are two types of CF: user-based, which we just described and item-based. Items-based could be easily understood on music example: songs which appear together in favorite lists of many users could be considered similar.

In this work, we will compare two methods against each other using real dataset, test some improvements and try to combine them together.

II. Dataset and evaluation

A. Dataset

Before going deep into algorithms, some information must be given about real dataset, used for evaluation. It is the Million Song Dataset (MSD) [2] - freely-available collection of audio features and metadata for a million popular music tracks, and particularly its Echo Nest Taste Profile Subset [3] - anonymous listening history in a form of *(user_id, song_id, play_count)* triplets. After releasing the dataset, authors found mismatches with a small fraction of songs [4]. For this work they are additionally fixed according to information here [5]. Some statistics: 1 million users, 380,000 songs. The user-item matrix is very sparse as the fraction of non-zero entries (the density) is only 0.01%.

For evaluation we will use not the whole dataset, but the most dense part of it, considering only active users (top users by the number of listened songs) and only popular songs (by the number of users, who have listened them). There are two reasons for it:

1) Get rid of noise in dataset
2) Reduce the time for calculations. Testing one set of hyper-parameters for the whole dataset takes more than a day. And we will consider a lot of different hyper-parameters.

The base "default" set to compare different versions of algorithms will contain 1000 users and 1000 songs. 0.2 of users will be used for testing, so we have 800 x 1000 user-song train matrix and 200 x 1000 user-song test matrix. But we will also use different shapes.

B. Data representation

Collaborative filtering techinques use a database in the form of a user-item matrix of preferences. As mentioned above we have splitted data into train matrix X and test matrix Y. We have a set U of n train users and a set I of m items. The entries of $X = x_{ui} \in \mathbb{R}^{n \times m}$ represent how much user u likes item i (for music recommendation system items are songs, we will use both terms interchangeably). We will start with assumption that $x_{ui} \in \{0, 1\}$, so data is binary: either user u has listened to a particular song i or not.

Also we have set T of k test users and test matrix $Y \in \mathbb{R}^{k \times m}$ which we will use for evaluation. Of course, $y_{ti} \in \{0, 1\}$ too.

To test a recommendation system we must hide some songs of test users to check how relevant these songs for the system. For each test user $t \in T$ let Rel_t be the set of relevant songs, so $Rel_t \subseteq I$ and $i \in Rel_t \iff y_{ti} = 1$.

Rel_t is splitted into two disjoint sets: $Open_t$ and Sec_t ($Open_t \cup Sec_t = Rel_t$). Sec_t is a set of secret relevant songs which ideally must be in the top of ranked results for the test user t. We must fill with zeros corresponding values in the matrix Y (as if user t has never listened to these songs): for each $i \in Sec_t$ make $y_{ti} = 0$.

C. Evaluation process

Algorithm evaluation consist of 3 steps:

1) **Fitting with train data.** Here algorithm fills some internal state, for example calculating user-user or song-song similarity matrices

2) **Predicting for test data.** For each test user t the prediction is calculated - ranked list of songs (without songs $i \in Open_t$)
3) **Evaluation** For evaluation we will use Mean Average Precision (MAP) metric.

III. Baseline results

The result of any algorithm we are considering is a matrix $Scores$. $score_{ti}$ represent how likely that the test user t will be happy to hear song i.

A. User-based filtering

For a user-based filtering user similarity matrix Sim is used. sim_{tu} represents how similar is test user t to train user u.

$$score_{ti} = \sum_{u \in U} sim_{tu} x_{ui}$$

The important question is how to measure similarity between two users. A large part of CF literature deals with problem of defining a good similarity measure. And common opinion is that it cannot exist a single similarity measure that can fit all possible domains. A euclidean distance is not a good choice because of data sparsity, so we will start with a simple cosine similarity - widely used measure.

Because we have binary data our cosine similarity easily expressed using sets: let $I(u)$ be the set of songs listened by user u, then the cosine similarity between two users t and u is defined by

$$sim_{tu} = \frac{|I(t) \cap I(u)|}{|I(t)|^{0.5} |I(u)|^{0.5}}$$

For default (1000x1000) dataset user-based strategy with simple cosine similarity gives **MAP = 0.1918**

B. Item-based filtering

For an item-based filtering item similarity matrix Sim is used. sim_{ij} represents how similar is song i to song j.

$$score_{ti} = \sum_{j \in I} sim_{ij} x_{tj}$$

Like for the user similarity matrix, by setting $U(i)$ the set of users who have listened item i, we have:

$$sim_{ij} = \frac{|U(i) \cap U(j)|}{|U(i)|^{0.5} |U(j)|^{0.5}}$$

For default dataset item-based strategy with simple cosine similarity gives **MAP = 0.2292**, which is better than user-based approach.

IV. Non symmetric similarity and emphasizing function

According to researches [6] and [7], performance of collaborative filtering can be greatly improved if we will use non symmetric similarity measure based on conditional probabilities and also emphasizing function. Let's start with the former. The cosine similarity has the nice property to be symmetric but, as we show later, it might not be the better choice. In fact for the item choice we are more interested in computing how likely it is that an item will be appreciated by a user when we already know that the same user likes another item [7]. It is clear that this definition is not symmetric. We can use the conditional probability measure which can be estimated with the following formulas:

$$sim_{tu} = P(t|u) = \frac{|I(t) \cap I(u)|}{|I(u)|}$$

$$sim_{ij} = P(i|j) = \frac{|U(i) \cap U(j)|}{|U(j)|}$$

And there is a generalization using combination of conditional probabilities:

$$sim_{tu} = P(u|t)^{\alpha} P(t|u)^{1-\alpha}$$

$$sim_{ij} = P(i|j)^{\alpha} P(j|i)^{1-\alpha}$$

Parameter $\alpha \in [0, 1]$ is tuned for the domain of interest.

Monotonic not decreasing emphasizing function $f(sim)$ can be used to adjust the locality of the scoring function, that is how many of the nearest users/items really matter in the computation. For our experiments we will use family of function $f(sim) = sim^q$, where $q \in \mathbb{N}$. The effect is following: when q is high, smaller weights drop to zero while higher ones are emphasized.

It is time for some results:

User-based 1000 users x 1000 songs (binary representation)

q \ α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	0.2069	0.2041	0.2018	0.1994	0.1960	0.1918	0.1894	0.1867	0.1840	0.1815	0.1791
2	0.2560	0.2542	0.2501	0.2459	0.2410	0.2360	0.2310	0.2253	0.2188	0.2130	0.2079
3	0.2585	0.2656	0.2665	0.2639	0.2581	0.2521	0.2459	0.2393	0.2321	0.2240	0.2152
4	0.2453	0.2555	0.2637	0.2677	0.2671	0.2597	0.2525	0.2453	0.2378	0.2290	0.2191
5	0.2322	0.2445	0.2556	0.2608	0.2656	0.2627	0.2552	0.2467	0.2389	0.2289	0.2179
6	0.2218	0.2340	0.2463	0.2562	0.2593	0.2582	0.2540	0.2467	0.2366	0.2270	0.2163
7	0.2132	0.2258	0.2385	0.2498	0.2537	0.2540	0.2502	0.2439	0.2338	0.2242	0.2137

Item-based 1000 users x 1000 songs (binary representation)

q \ α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	0.1612	0.1964	0.2229	0.2374	0.2363	0.2292	0.2212	0.2113	0.2004	0.1901	0.1816
2	0.1471	0.1823	0.2125	0.2314	0.2427	0.2442	0.2389	0.2308	0.2211	0.212	0.2013
3	0.1401	0.1775	0.2073	0.228	0.2411	0.2477	0.2486	0.2445	0.2364	0.2272	0.2165
4	0.1337	0.1745	0.2052	0.2249	0.2386	0.2472	0.2512	0.2505	0.2444	0.2374	0.2258
5	0.1281	0.1713	0.1998	0.2197	0.2343	0.2436	0.2502	0.2514	0.2472	0.2409	0.2311
6	0.1236	0.168	0.1956	0.2146	0.2287	0.239	0.2466	0.25	0.2485	0.2428	0.2322
7	0.12	0.1646	0.1913	0.2093	0.224	0.2339	0.2425	0.2469	0.248	0.2426	0.2323

As you can see the best parameters for user-based strategy for 1000×1000 dataset is $\alpha = 0.3$, $q = 4$ giving **MAP = 0.2677**. For item-based strategy best parameters are $\alpha = 0.7$, $q = 5$ giving **MAP = 0.2514**

But is it true, that user-based CF is always better? No, it depends on data distribution. The following tables show MAP results for a 200×2000 dataset:

User-based 200 users x 2000 songs (binary representation)

q \ α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	0.160638	0.16031	0.159937	0.159413	0.1588	0.157525	0.156812	0.155951	0.155074	0.153926	0.152885
2	0.206317	0.205817	0.204592	0.202436	0.199975	0.197355	0.194573	0.191539	0.188119	0.185013	0.182495
3	0.217105	0.219351	0.219958	0.218893	0.216247	0.211862	0.207671	0.20318	0.198997	0.19412	0.189947
4	0.214996	0.218628	0.220262	0.219819	0.218505	0.21586	0.211204	0.205251	0.199879	0.194926	0.189886
5	0.209628	0.215345	0.217424	0.217556	0.215555	0.21373	0.209794	0.204122	0.198922	0.193395	0.188265
6	0.204153	0.210688	0.214201	0.214728	0.212728	0.210853	0.208417	0.202762	0.197475	0.191434	0.187028
7	0.200183	0.20745	0.210293	0.211396	0.210351	0.207986	0.205727	0.200942	0.194535	0.190215	0.185645

Item-based 200 users x 2000 songs (binary representation)

q \ α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	0.152978	0.173363	0.194146	0.205639	0.205874	0.199071	0.190903	0.181693	0.17349	0.165854	0.158773
2	0.128055	0.154688	0.183258	0.20619	0.218672	0.218058	0.208682	0.198665	0.189481	0.179989	0.170701
3	0.116721	0.146833	0.178266	0.203857	0.221402	0.227762	0.223139	0.213103	0.202448	0.192085	0.180812
4	0.10847	0.140217	0.174971	0.203494	0.221641	0.230849	0.231121	0.224031	0.212666	0.200054	0.186231
5	0.101853	0.134573	0.172875	0.200739	0.218783	0.228956	0.231799	0.229899	0.219696	0.205348	0.188336
6	0.096242	0.131191	0.170083	0.198047	0.214834	0.22529	0.229923	0.229064	0.221219	0.207664	0.186905
7	0.091843	0.127799	0.167727	0.19593	0.212568	0.222649	0.228359	0.228432	0.219686	0.205785	0.185398

And here item-based wins.

V. Exact play count information

So far we have been using binary data: user either has listened to a particular song or not. But The Echo Nest Taste Profile offers more. For completeness here are results for exact play counts (i. e. x_{ui} contains how many times user u has listened to the song i):

User-based 1000 users x 1000 songs (raw play counts)

q \ α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	0.187881	0.186279	0.184112	0.180739	0.17786	0.173206	0.16695	0.160688	0.153821	0.145661	0.138194
2	0.195342	0.204117	0.206891	0.206342	0.201561	0.191307	0.179701	0.16706	0.15562	0.144181	0.132703
3	0.162005	0.179388	0.193391	0.199136	0.196695	0.18934	0.174988	0.160514	0.148622	0.136732	0.127029
4	0.137768	0.154608	0.171919	0.18644	0.188367	0.182005	0.16711	0.153124	0.14233	0.13222	0.123672
5	0.123024	0.138881	0.156239	0.171577	0.179449	0.175807	0.161885	0.14762	0.137601	0.128687	0.120993
6	0.115364	0.129444	0.145346	0.161579	0.171583	0.171102	0.157448	0.143745	0.134088	0.12656	0.119765
7	0.111189	0.123175	0.138166	0.154216	0.166243	0.167703	0.154962	0.141288	0.131586	0.124482	0.118611

Item-based 1000 users x 1000 songs (raw play counts)

q \ α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	0.086902	0.103635	0.125614	0.150055	0.175531	0.194715	0.193643	0.184748	0.170761	0.156772	0.142991
2	0.074985	0.089288	0.110365	0.136502	0.163792	0.188278	0.188148	0.178276	0.164817	0.147603	0.133882
3	0.06879	0.080438	0.09782	0.123695	0.152888	0.180646	0.180423	0.169874	0.154751	0.138162	0.125134
4	0.06501	0.075442	0.090643	0.113454	0.144093	0.173225	0.173862	0.161795	0.146298	0.131599	0.119026
5	0.062771	0.07194	0.086512	0.107582	0.135964	0.167375	0.168129	0.155325	0.140376	0.126755	0.114872
6	0.06143	0.069858	0.083866	0.103536	0.130007	0.162971	0.163937	0.150895	0.136322	0.123593	0.112551
7	0.06049	0.068537	0.081793	0.100534	0.126114	0.15933	0.159434	0.147035	0.133733	0.120978	0.111185

Results are much worse than binary representation.

The $log(x+1)$ representation will be a better choice:

User-based 1000 users x 1000 songs (log(x+1) representation)

q \ α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	0.2018	0.1997	0.1976	0.1954	0.1928	0.1901	0.1872	0.1850	0.1820	0.1789	0.1761
2	0.2446	0.2448	0.2428	0.2400	0.2357	0.2303	0.2240	0.2170	0.2089	0.2006	0.1939
3	0.2322	0.2453	0.2531	0.2522	0.2482	0.2423	0.2351	0.2260	0.2147	0.2020	0.1911
4	0.2091	0.2270	0.2427	0.2495	0.2494	0.2444	0.2365	0.2257	0.2136	0.1977	0.1846
5	0.1880	0.2111	0.2288	0.2416	0.2445	0.2419	0.2341	0.2224	0.2086	0.1928	0.1784

Item-based 1000 users x 1000 songs (log(x+1) representation)

q \ α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	0.1337	0.1595	0.1922	0.2180	0.2287	0.2268	0.2192	0.2090	0.1988	0.1881	0.1791
2	0.1227	0.1519	0.1832	0.2122	0.2309	0.2381	0.2327	0.2231	0.2129	0.2019	0.1916
3	0.1153	0.1433	0.1781	0.2081	0.2273	0.2390	0.2386	0.2309	0.2212	0.2091	0.1974
4	0.1094	0.1371	0.1739	0.2037	0.2231	0.2361	0.2396	0.2330	0.2241	0.2130	0.2005
5	0.1049	0.1323	0.1683	0.1998	0.2190	0.2336	0.2381	0.2331	0.2243	0.2126	0.2010

It gives results close to binary representation, but requires more time for computation because of logarithms.

VI. Combined filtering

Is it true, that we can combine both approaches (user-based and item-based CF) to achieve even better results? It will be similar to ensemble learning when by combining powers of several classifiers we achieve better accuracy. Let $Score^{[user]}$ be the score matrix, produced by user-based method and $Score^{[item]}$ be the matrix produced by item-based method. Then

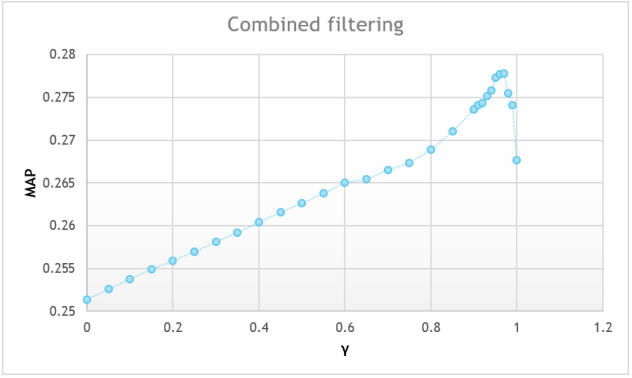
$$Score^{[comb]} = \gamma Score^{[user]} + (1-\gamma) Score^{[item]}$$

where $\gamma \in [0, 1]$. If $\gamma = 1$ it is pure user-based CF, if $\gamma = 0$ it is pure item-based CF. With $0 < \gamma < 1$ we have combined collaborative filtering. We will take user-based filtering with its best parameters ($\alpha = 0.3$, $q = 4$) and item-based filtering with its best parameters ($\alpha = 0.7$, $q = 5$)

Combined filtering

user_based	item_based	MAP
0	1	0.251366
0.05	0.95	0.252591
0.1	0.9	0.253759
0.15	0.85	0.254902
0.2	0.8	0.255882
0.25	0.75	0.256945
0.3	0.7	0.258135
0.35	0.65	0.259187
0.4	0.6	0.260363
0.45	0.55	0.261567
0.5	0.5	0.262616
0.55	0.45	0.263811
0.6	0.4	0.265012
0.65	0.35	0.265424
0.7	0.3	0.266503
0.75	0.25	0.267312
0.8	0.2	0.268903
0.85	0.15	0.271011
0.9	0.1	0.27356
0.91	0.09	0.274092
0.92	0.08	0.274353
0.93	0.07	0.275135
0.94	0.06	0.275848
0.95	0.05	0.277313
0.96	0.04	0.277674
0.97	0.03	0.277761
0.98	0.02	0.27548
0.99	0.01	0.274079
1	0	0.267662

We achieved even better performance for the default dataset: with $\gamma = 0.97$ we have **MAP = 0.2778** - the highest result.



VII. Conclusion

As a summary we achieved the following insights:

1) It is better not to use exact play counts: either smooth them by for example log(x+1) or use binary representation
2) Non symmetric weights based on conditional probabilities improve results
3) Emphasizing function improves results

4) Sometimes user-based filtering is better, sometimes - item-based one
5) By combining strategies we can achieve better outcomes than using them individually

All source code is available here [8]

REFERENCES

[1] Collaborative filtering, https://en.wikipedia.org/wiki/Collaborative_filtering

[2] Million Song Dataset https://labrosa.ee.columbia.edu/millionsong/

[3] The Echo Nest Taste Profile Subset https://labrosa.ee.columbia.edu/millionsong/tasteprofile

[4] Matching errors - Taste Profile and MSD https://labrosa.ee.columbia.edu/millionsong/blog/12-1-2-matching-errors-taste-profile-and-msd

[5] Fixing matching errors https://labrosa.ee.columbia.edu/millionsong/blog/12-2-12-fixing-matching-errors

[6] Shakirova, Elena. "Collaborative filtering for music recommender system." Young Researchers in Electrical and Electronic Engineering (EIConRus), 2017 IEEE Conference of Russian. IEEE, 2017.

[7] Aiolli, Fabio. "A preliminary study on a recommender system for the million songs dataset challenge." Preference Learning: Problems and Applications in AI 1 (2012).

[8] https://github.com/and-kul/music_recommendation_system