Decode the Secrets Behind Recommender Systems: Matrix Factorization and Completion

Jie An
Department of Electrical Engineering
Columbia University
New York, NY, USA
jie.an@columbia.edu

***Abstract*—In this project, I try to decode the secretes behind media recommender systems, starting from the theory of low-rank matrix factorization which is an efficient algorithm provides a unified method for matrix completion. With the detailed explanations of the theory, I implement it on the public dataset MovieLens to build a naive recommender system for recommending movies according to users previous ratings.**

I. Introduction

A. Two Basic Strategies for Object Recommendation

Matching consumers to products is an important practical problem. Recommender systems use consumers feedback about subsets of products to help recommend new things to customers that they may like. Basically, there are two strategies for object recommendation: content filtering and collaborative filtering. Content filtering strategy uses known information about the products and users to characterize its nature profiles and make recommendations, which would requires a lot of information that difficult and expensive to collect. Collaborative filtering, on the other hand, uses users' past behavior (ratings) information to make future recommendations without requiring the creation of explicit profiles.

B. Two Primary Areas of Collaborative Filtering

The two primary areas of collaborative filtering are the neighborhood methods and latent factor models. Neighborhood methods are centered on computing the relationships between items or, alternatively, between users. Latent factor models are an alternative approach that tries to explain the ratings by characterizing both items and users on, say, 20 to 100 factors inferred from the ratings patterns. In this project, we focus on the latent factor models.

C. Netflix Prize and Matrix factorization

When talking about recommender systems, we cannot ignore the *Netflix Prize*. In 2006, Netflix announced the *Netflix Prize*, a competition to predict movie ratings on a 5-star scale. Netflix conducted this competition to find new ways to improve its recommendations. And used the root mean squared error (RMSE) between the predicted and actual rating to evaluate and quantify. Team BellKor took over the top spot in the competition in the summer of 2007, and won the 2007 Progress Prize with the best score at the time: 8.43 percent better than Netflix. The method they introduced was a realization of latent factor models based on matrix factorization.

Matrix factorization is the breaking down of one matrix into a product of multiple matrices. There are many different ways to factor matrices, but singular value decomposition (SVD) is particularly useful for making recommendations. We will go into details in the following section II-A.

D. Matrix Completion

We form a rating matrix M with N_1 users and N_2 objects that contains every user/object pair. Users rate objects based on the quality of their experience. It is clear that it will have many missing values. We wish to predict users ratings of items that they have not yet rated. Therefore, our goal is to fill in the missing entris of M as a matrix completion problem. The reason we are confident to predict every missing rating for the user is that M is a low-rank matrix with many objects have correlated ratings. This models correlations and the low-rank restriction gives hope for filling in missing data. And matrix factorization gives a way to learn user and object locations.

II. Technical Approach

A. Matrix Factorization via SVD

SVD is an algorithm that decomposes a matrix M into the best lower rank (i.e. smaller/simpler) approximation of the original matrix M. Mathematically, it decomposes M into two unitary matrices and a diagonal matrix:

$$M = U\Sigma V^T \tag{1}$$

where M is rating matrix, U is the user "features" matrix, Σ is the diagonal matrix of singular values, and V^T is the object "features" matrix. U and V^T are orthogonal with $U^TU = I$, $V^TV = I$ and Σ is diagonal with $\Sigma_{ii} \geq 0$.

After factorized, martix M can be illustrated as Fig. 1. The rank of matrix M then is represented by the rank of matrix U as notation d, which also means the number of latent factors of the models. Here being low rank, $d \ll min\{N_1, N_2\}$. And we try to learn a location $u_i \in \mathbb{R}^d$ for user i and $v_j \in \mathbb{R}^d$ for object j.

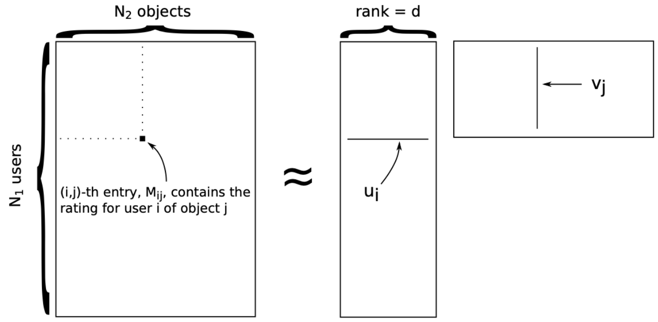

Fig. 1. Factorized Matrix M via SVD

B. Probabilistic Matrix Factorization

The convenient distribution assumption for user locations and object locations could be Gaussian distribution. Thus, initialize each u_i and v_j as $u_i \sim N(0, \lambda^{-1}I)$, $v_j \sim N(0, \lambda^{-1}I)$. Given these locations, the distribution on the data is $M_{ij} \sim N(u_i^T v_j, \sigma^2)$ for each $(i,j) \in \Omega$, where set Ω contains the pair (i,j) that are observed as $\Omega = \{(i,j) : M_{ij} \text{ is measured}\}$.

Let M_0 be the part of M that is observed and M_m the missing part. Then, try to maximize the joint likelihood $p(M_0, U, V)$ over U and V to learn the data.

C. Learning Algorithm

The joint likelihood of $p(M_0, U, V)$ can be factorized as:

$$p(M_0, U, V) = [\prod_{(i,j)\in\Omega} p(M_{ij}|u_i, v_j)] \times [\prod_{i=1}^{N_1} p(u_i)][\prod_{j=1}^{N_2} p(v_j)]. \tag{2}$$

The maximum a posteriori (MAP) solution for U and V is the maximum of the log joint likelihood as:

$$\begin{aligned} U_{MAP}, V_{MAP} = \arg\max_{U,V} & \sum_{(i,j)\in\Omega} \ln p(M_{ij}|u_i, v_j) \\ & + \sum_{i=1}^{N_1} \ln p(u_i) + \sum_{j=1}^{N_2} \ln p(v_j). \end{aligned} \tag{3}$$

Calling the MAP objection function $\mathcal{L}$, then try to maximize $\mathcal{L}$:

$$\begin{aligned} \mathcal{L} = - & \sum_{(i,j)\in\Omega} \frac{1}{2\sigma^2} ||M_{ij} - u_i^T v_j||^2 - \sum_{i=1}^{N_1} \frac{\lambda}{2} ||u_i||^2 \\ & - \sum_{j=1}^{N_2} \frac{\lambda}{2} ||v_j||^2 + \text{constant}. \end{aligned} \tag{4}$$

To update each u_i and v_j, take the derivative of $\mathcal{L}$ and set to zero:

$$\bigtriangledown_{u_i} \mathcal{L} = \sum_{j\in\Omega_{u_i}} \frac{1}{\sigma^2} (M_{ij} - u_i^T v_j) v_j - \lambda u_i = 0 \tag{5}$$

$$\bigtriangledown_{v_j} \mathcal{L} = \sum_{i\in\Omega_{v_j}} \frac{1}{\sigma^2} (M_{ij} - v_j^T u_i) u_i - \lambda v_j = 0 \tag{6}$$

Then, we can solve for each u_i and v_j individually:

$$u_i = (\lambda\sigma^2 I + \sum_{j\in\Omega_{u_i}} v_j v_j^T)^{-1} (\sum_{j\in\Omega_{u_i}} M_{ij} v_j) \tag{7}$$

$$v_j = (\lambda\sigma^2 I + \sum_{i\in\Omega_{v_j}} u_i u_i^T)^{-1} (\sum_{i\in\Omega_{v_j}} M_{ij} u_i) \tag{8}$$

We cannot solve for all u_i and v_j at once to find the MAP solution. So we use a coordinate ascent algorithm in several iterations to update.

D. MAP inference coordinate ascent algorithm

- **Input**: An incomplete rating matrix M, set Ω, rank d
- **Output**: N_1 user locations $u_i \in \mathbb{R}^d$, and N_2 object locations $v_j \in \mathbb{R}^d$
- **Initialization**: $u_i \sim N(0, \lambda^{-1}I)$, $v_j \sim N(0, \lambda^{-1}I)$

for each iteration **do**:

- **for** $i = 1, ..., N_1$ **update user location**

$$u_i = (\lambda\sigma^2 I + \sum_{j\in\Omega_{u_i}} v_j v_j^T)^{-1} (\sum_{j\in\Omega_{u_i}} M_{ij} v_j)^{(7)}$$

- **for** $j = 1, ..., N_2$ **update object location**

$$v_j = (\lambda\sigma^2 I + \sum_{i\in\Omega_{v_j}} u_i u_i^T)^{-1} (\sum_{i\in\Omega_{v_j}} M_{ij} u_i)^{(8)}$$

Then, the following section III would implement the algorithm on the public dataset MovieLens to build a naive recommender system.

III. Experiments

A. Dataset

The dataset used in the experiments is a simplified MovieLens dataset with 100,000 ratings (1-5) from 943 users on 1682 movies, and each user has rated at least 20 movies. Also, de-mean (normalize) the ratings data by each users ratings mean. Then split the original ratings dataset as training and testing set. The visualization for training and testing matrix are illustrated as Fig. 2 and Fig. 3.

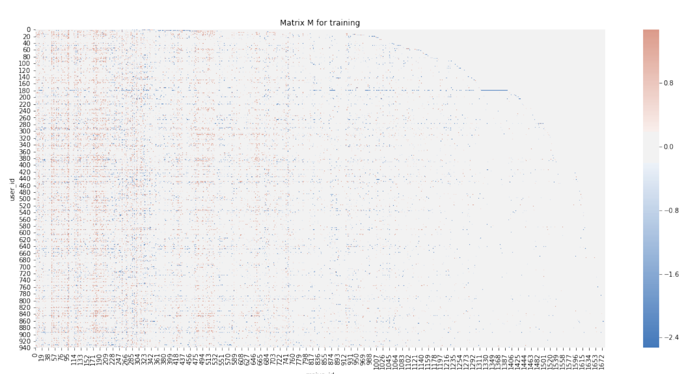

Fig. 2. Matrix M for training

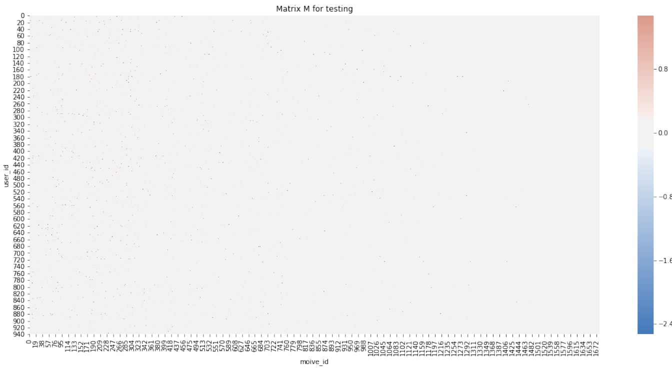

Fig. 3. Matrix M for testing

B. *Algorithm Execution*

Initialize user locations and object locations with $\sigma^2 = 0.25$ and $\lambda = 1$. And set rank (the number of latent fators) $d = 10$ in the algorithm execution. Run the algorithm for 10 times and each run iterates 100 times. Fig. 4 shows the the log joint likelihood for iterations 2 to 100 for each run. In the table I, each row shows the final value of the training objective function next to the RMSE on the testing set.

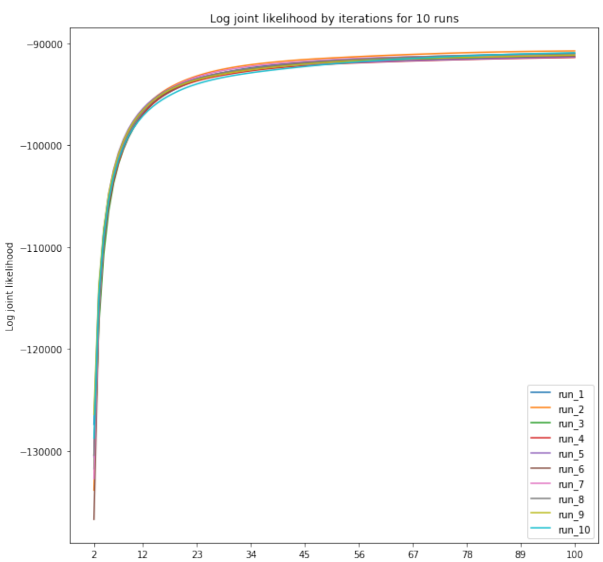

Fig. 4. Log joint likelihood by iterations for 10 runs

Pick the best run with the highest objective value, which is the 2^{nd} run according to the table I, multiply updated user i location and object j location as $u_i^T v_j$ to complete the predicting matrix M. Visualize the predicting matrix as Fig. 5.

After gain the predicting matrix, we could use it to imitate some use cases as a naive recommender system.

TABLE I
OBJECTIVE FUNCTION VALUE AND RMSE FOR 10 RUN

run	final value of obj	RMSE
2	-90725.489855	1.015077
10	-90902.895882	1.111587
1	-90972.762885	1.093415
8	-91014.111316	1.111624
7	-91134.543574	1.128258
9	-91136.570353	1.093788
6	-91214.313817	1.108230
5	-91312.961398	1.116357
3	-91315.854626	1.125137
4	-91359.263814	1.107143

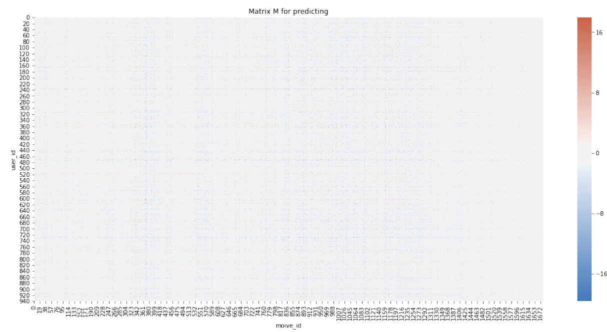

Fig. 5. Matrix M for predicting

IV. RECOMMENDER SYSTEM USE CASES

A. *Use Case 1: Finding Similar Movies*

Everytime a user create a new account on Netflix, it would ask the user to choose 3 movies/shows he/she likes (like showed in Fig. 6). Then, the platform would recommend tons of contents according to the user's choice. This use case can be simplified as finding the similar movies for a given one. To imitate this case, we could find the similar movies according to Euclidean distance using their respective locations v_j. Here, we use *Toy Story* as an example, and table II showed the 10 closet movies for *Toy Story* and their distances.

TABLE II
THE 10 CLOSET MOVIES FOR *Toy Story*

rank	moive name	distance
1	Beauty and the Beast (1991)	0.511122
2	Aladdin (1992)	0.540992
3	The Lion King (1994)	0.551604
4	Indiana Jones and the Last Crusade (1989)	0.563193
5	Searching for Bobby Fischer (1993)	0.631215
6	The Secret Garden (1993)	0.656768
7	The Sting (1973)	0.656768
8	Apollo 13 (1995)	0.681707
9	The Abyss (1989)	0.682708
10	Back to the Future (1985)	0.716056

As we can see from table II, the recommender system did return reasonable results that are similar to *Toy Story*.

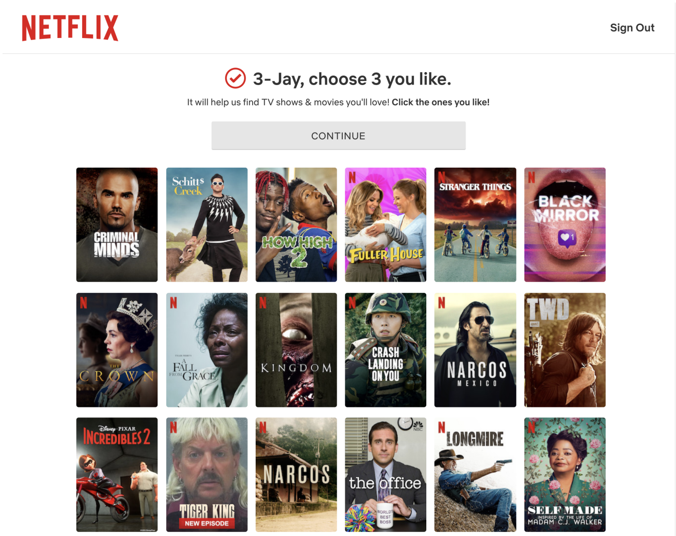

Fig. 6. A new user is asked to choose 3 liked ones

B. Use Case 2: Recommending Unrated Movies for the User

Another use case in Netflix is that users could check movies/shows they have watched and rated under their profile page (like showed in Fig. 7). The platform would update the recommendation contents according to users' watching and rating behaviors. To imitate this case, we could just simplify it as recommending 10 unrated movies that the user would like according to predicted ratings in the predicting matrix.

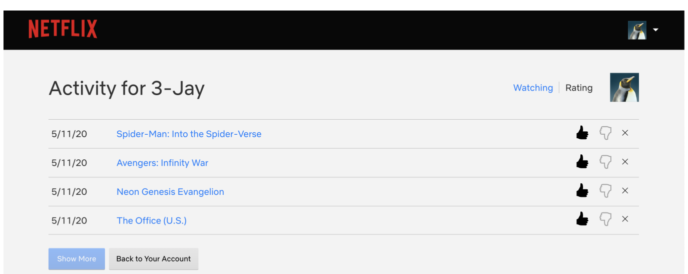

Fig. 7. Movied/shows the user has rated

Giving user 12 as an example. He/she has already rated 50 movies, and the recommender system should pick out 10 highest predicted ratings movies that the user would like most in the remaining 1632 unrated ones. Table III shows the recommendations (the ratings in the table is still the de-meaned value).

V. Conclusion

Matrix factorization provides a way to solve the low-rank matrix completion problem raised in recommender systems. It performs well to predict missing ratings for users only according to users' previous rating behavior. Recommender system use cases imitated in this project gives a taste of how real-world media recommendation works.

TABLE III
THE 10 HIGHEST PREDICTED RATINGS MOVIES

rank	moive name	rating
1	BJohnny Mnemonic (1995)	3.441464
2	A Goofy Movie (1995)	2.896367
3	I Shot Andy Warhol (1996)	2.755474
4	Bio-Dome (1996)	2.724087
5	Only You (1994)	2.604431
6	The Fox and the Hound (1981)	2.533775
7	Love & Human Remains (1993)	2.520267
8	City Hall (1996)	2.319934
9	Alice in Wonderland (1951)	2.314357
10	The Fifth Element (1997)	2.284594

References

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," Computer, 42(8), August 2009.
[2] X. Amatriain and J. Basilico, "Past, Present, and Future of Recommender Systems: An Industry Perspective," the 10th ACM Conference on Recommender Systems, New York, NY, USA, 211–214, 2016.
[3] J. D. M. Rennie and N. Srebro, "Fast Maximum Margin Matrix Factorization for Collaborative Prediction," the 22nd International Conference on Machine Learning (ICML), 2005.
[4] O. Sar Shalom, N. Koenigstein, U. Paquet, and H. Vanchinathan. "Beyond Collaborative Filtering: The List Recommendation Problem," WWW '16, 2016.