Matrix Factorization Method for Decentralized Recommender Systems

Zheng Wenjie

April 29, 2016

Abstract

Decentralized recommender system does not rely on the central service provider, and the users can keep the ownership of their ratings. This article brings the theoretically well-studied matrix factorization method into the decentralized recommender system, where the formerly prevalent algorithms are heuristic and hence lack of theoretical guarantee. Our preliminary simulation results show that this method is promising.

1 Introduction

Recommender Systems (RS) are a kind of system that seek to recommend to users what they are likely interested in. Unlike search engines, the users do not need to type any keyword. The RS’s will learn their interest automatically. For instance, if the user has just bought a numeric camera, the RS will recommend to him some SD memory cards; if a user watches a lot of action movies, the RS may suggest some other action movies to him. And this is the typical behaviors which we observe universally in Netflix (movies), Youtube (videos), Google Play (apps), Facebook (friends), Amazon (goods) and other platforms today.

Recommender systems play an important role in our daily life. They target user tastes and profiles and provide them with relevant information. This keeps users from drowning in the ocean of data, and helps them quickly find what they exactly want or like. This capacity of providing users with targeted information is achieved by collecting user-related data into a central database, and then running various recommendation algorithms on it.

Here comes the issue. The data, e.g. the ratings given to a movie by users, are generated by the users themselves. Logically, they are property of the users who generated them. But why in the end, the companies which provide the recommender system take possession of it? What if they sell them to a third party? What if the user privacy is leaked? A related example is the Netflix Prize, the Netflix company published 100 480 507 ratings that 480 189 users gave to 17 770 movies on the Internet in an anonymous way. However, one year later, two researchers from the University of Texas, de-anonymized some of the Netflix data by matching the data set with film ratings on the Internet Movie Database.

Therefore, it would be interesting if we could design a computational framework where the users can get recommendation without the help of a recommender service provider, and the users do not even transfer their data to others. Instead, they keep the data on their own computers and do all the computation collaboratively with other users in a decentralized way.
In fact, there are already some works in this direction\textsuperscript{[4, 5, 6, 7, 8]}. They use trust or some similarity metric to build a graph among the users, and then use some propagation algorithms to do the recommendation. These methods are straightforward and heuristic. However, they lack theoretical guarantee. Moreover, recent research has developed many powerful methods such as matrix factorization\textsuperscript{[9]}, which are incompatible to the network propagation framework.

The goal of this article is to introduce the matrix factorization method into the decentralized recommender system. Before our effort, there are already some trials in Network Distance Prediction\textsuperscript{[10, 11]}. This article will focus on the specialty of recommender systems. We first introduce this model and the associated theoretical analysis in Section 2. Then, we proceed in Section 3 to elaborate a decentralized algorithm to solve the model. Section 4 presents some preliminary simulation. We end this article with a discussion in Section ??.

## 2 Matrix factorization model and consistency

In this section, we first present the matrix factorization model of the RS problem. Then, we show that its estimator is consistent.

Suppose there are $m$ users and $n$ items in the universe. $\Theta_{m \times n}$ is the unknown matrix of true ratings that each user will give to each item. This is a dense matrix without any missing values. However, since there are so many items, users are not able to test every item and their ratings are corrupted by noise. What we finally observe is a sparse matrix $X_{m \times n}$, which could be regarded as some approximation of $\Theta$. The objective of RS is to recover $\Theta$ from $X$, so that it can recommend to users the items with the largest values in $\Theta$.

In order for that,\textsuperscript{[9]} uses a factorization method. It regards $\Theta$ as some low-rank matrix, i.e. \operatorname{rank}(\Theta) = r \ll \min(m, n). Thus, $\Theta$ can be approximated by the product of two low dimensional matrices $U_{r \times m}$ and $V_{r \times n}$: $\Theta = U^T V$. This representation has a nice interpretation. The $i$-th column of $U$ can be seen as the profile of the $i$-th user, and the $j$-th column of $V$ can be seen as the profile of the $j$-th item; the rating $\Theta_{ij}$ that the $i$-th user gives to the $j$-th item is just the inner product of these two profiles.

Then, the estimation of $\Theta$ is equivalent to the estimation of $U$ and $V$. In this way,\textsuperscript{[9]} proposes the following method:

$$\min_{U,V} \|P_{\Omega X} (U^T V - X)\|_F^2 + \lambda (\|U\|_F^2 + \|V\|_F^2),$$

where $\|\cdot\|_F^2$ represents the Frobenius norm, $\Omega X$ is the support of $X$ and $P$ is the projection operator. Besides the obvious approximation term, \textsuperscript{(1)} includes a regularization term as well.

\textsuperscript{[12]} shows that \textsuperscript{(1)} is equivalent to the following optimization problem:

$$\min_{\hat{M}} \|\hat{M}\|_* \text{ s.t. } \|P_{\Omega X} (\hat{M} - X)\|_F \leq \rho,$$

for some $\rho(\lambda)$ depending on the value of $\lambda$, where $\|\cdot\|_*$ represents the nuclear norm (a.k.a. trace norm). And the minimal solution $\hat{M}$ of \textsuperscript{(2)} is the product of the minimal solution $\hat{U}$ and $\hat{V}$ of \textsuperscript{(1)}: $\hat{M} = \hat{U}^T \hat{V}$.

\textsuperscript{[13]} proves that if the projection operator $P_{\Omega X}$ satisfies \textit{restricted isometry property} (RIP): $(1 - \alpha) \|A\|_F^2 \leq \frac{1}{p} \|P_{\Omega X} (A)\|_F^2 \leq (1 + \alpha) \|A\|_F^2$, for any matrix $A$ with sufficiently small rank and $\alpha \in (0,1)$ sufficiently small, where $p$ is the proportion of non-missing values of $X$, then

$$\|\hat{U}^T \hat{V} - \Theta\|_F \leq C_0 p^{-1/2} \rho.$$
This means that the matrix factorization algorithm approximately recovers the true ratings $\Theta$ with a few corrupted ratings $X$.

Therefore, as soon as a decentralized algorithm converges to the minimal solution of (1) (or approximately), it enjoys a nice theoretical guarantee.

### 3 Decentralized matrix factorization algorithm

In this section, we present our decentralized matrix factorization algorithm. We first clarify the computation environment. Then, we proceed to describe the algorithm. Then, we explain the relation between this algorithm and optimization problem (1). Finally, we show how to use the trained model to get recommendation.

At the beginning, each user $i$ holds his rating vector $x_i$ in his computer. And he also spares some place in the memory to store the user profile $u_i$, which is randomly initialized. To store the item profile $v_j$, we suppose that there are many spared computational entities scattered on the Internet. Such entity can be routers or volunteers’ PC. For ease of description, we will just use “router” to refer to it. The item profile is also randomly initialized.

Suppose that the user is in some P2P network. He does not know the global topology of the network, but he possesses a short list of routers that he can send messages to. This list can be dynamically updated. It is the same for the routers. They have a short list of users that they can send messages to. An additional thing is that each user/router should choose a learning rate $\eta$ for itself. This $\eta$ can vary between different users/routers. With a bit abuse of notation, we do not put any super/subscripts on $\eta$. Higher $\eta$ values stand for that the profiles are updated more aggressively.

Once a user/router is ready, it joins in the network and begins the communication with other routers/users. Each user/router will basically do two things: broadcast its profile parameter according to its list, and receive the broadcast profile parameter to update its own profile parameter. While for the broadcasting, users and routers do exactly the same thing, their behaviors are slightly different during the updating procedure. When user $i$ receives an item profile, say $v_j$, if rating $x_{ij}$ is not missing (i.e. user $i$ does rate item $j$), then user $i$ updates his user profile:

$$u_i \leftarrow u_i - \eta v_j (u_i^T v_j - x_{ij}) - \eta \lambda u_i.$$ (3)

When item $k$ receives a user profile $u_l$, if it does not know the rating $x_{lk}$, then it requests it from user $l$. Then it updates its item profile:

$$v_k \leftarrow v_k - \eta u_l (u_l^T v_k - x_{lk}) - \eta \lambda v_k.$$ (4)

Each user/router can freely join in or quit the learning process at any time. Therefore, the algorithm is robust against the breakdown of individual nodes. And when a user gives a new rating to an item (e.g. he just saw the movie yesterday), the learning process does not need to restart from scratch; it follows exactly the same protocol. And it is the same when new users or new items come into the system. The computation network can keep going on and it scales up well.

To answer the question why this algorithm can work well, it is sufficient to notice that (3) and (4) is a variant of stochastic gradient descent (SGD) algorithm to minimize the optimization problem (1). What should be accentuate is that our algorithm is not exactly SGD. For a true SGD, (3) and (4) should be done at the same moment and at the same place for a pair of profile parameters $(u_i, v_j)$. Therefore, its update sequence can only be something like $\ldots (u_i, v_j), (u_k, v_l) \ldots$. However, in our
algorithm, each time, only one profile gets updated, and a user/router only updates his own $u_i/v_j$.
As a consequence, the evolution of the profile parameters in our algorithm can have patterns like $\ldots u_i, u_k, v_j, v_l \ldots$, which will never appear in the true SGD.

The above is the learning process. For the recommendation, say if user $i$ would like to know whether he will appreciate item $j$, he just need to request the profile parameter $v_j$ from the router which hosts it. He calculates its inner product with $u_i$, and then he knows what rating he is likely to give to this item. In contrast to traditional recommender systems, which only give recommendation, our system actually gives the predicted rating. User can thus get a rough idea about the item that he is hesitating to purchase or the movie he is hesitating to watch.

4 Simulation

In this section, we test our algorithm on both synthetic dataset and real dataset. We will first show the experiment protocol. Then, we present the database and the result our algorithm yields.

Since all this thing is currently just a concept, we have not yet a genuine decentralized recommender system that we can experiment with. We will simulate it in a reasonable way. For this reason, we make a hypothesis to restrict the behavior that is allowed to happen in a genuine decentralized network, so that our simulation reflects more or less the phenomenon in practice.

We suppose that given two messages $D_1$ and $D_2$ sent by the same emitter, where $D_1$ is emitted before $D_2$, if both are ever received by a receptor, then the reception of $D_1$ always happens before the reception of $D_2$. This assures that a delayed message sent millions years ago will not erase the effect of a recent message.

This hypothesis is realistic. We can make sure it happens by timestamping the data and discarding the delayed data. For each message emitted, the emitter annotates it in adding the time information and the emitter’s MAC address. If a receiver receives a message with an earlier timestamp than the last message received from the same emitter, then it simply discards it.

In this way, we can simulate our algorithm in a single computer in an efficient way. Indeed, in this case, it is equivalent to repeatedly sampling a rating $x_{ij}$ and updating either $u_i$ or $v_j$ (only one of them, not both).

For synthetic data, we let $\Theta$ be a low-rank matrix with floating-point values close to $\{1, 2, 3, 4, 5\}$, and rank($\Theta$) = $r$ = 10. $X$ is generated by rounding $\Theta$ to the closest integers, and randomly deleting 20% values as missing ratings. We test for matrix size $m = n = 100, 200$ and 500. The evaluation criterion is the root mean square (RMS) error, defined as $\frac{1}{\sqrt{mn}} \| \hat{U}^T \hat{V} - \Theta \|_F$. The result is 0.17, 0.15 and 0.13 respectively. This means that the error of the estimator of the rating yielded by our algorithm is within $\pm0.2$ in average. Note that the difference between two successive ratings is 1, our algorithm successfully recovers the true ratings.

Next we move on to MovieLens 100k dataset. This dataset is composed of 943 users and 1682 movies. There are 100k user ratings with the values among $\{1, 2, 3, 4, 5\}$. We use this dataset to do 5-fold cross validation. During each cross validation, the whole dataset is divided into a training set with 80% ratings and a testing set with 20% ratings. This is done with the official division provided. Our algorithm yields an error of 0.963 slightly worse than the best error registered 0.894. This may be because that our model does not take into consideration of the intercepts (i.e. preprocessing by moving the rating average towards zero).
References

[1] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[2] James Bennett and Stan Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007.

[3] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE, 2008.

[4] Peng Han, Bo Xie, Fan Yang, and Ruimin Shen. A scalable p2p recommender system based on distributed collaborative filtering. *Expert systems with applications*, 27(2):203–210, 2004.

[5] Anne-Marie Kermarrec, Vincent Leroy, Afshin Moin, and Christopher Thraves. Application of random walks to decentralized recommender systems. In *Principles of Distributed Systems*, pages 48–63. Springer, 2010.

[6] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 17–24. ACM, 2007.

[7] John O’Donovan and Barry Smyth. Trust in recommender systems. In *Proceedings of the 10th international conference on Intelligent user interfaces*, pages 167–174. ACM, 2005.

[8] Cai-Nicolas Ziegler. *Towards decentralized recommender systems*. PhD thesis, Citeseer, 2005.

[9] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[10] Yongjun Liao, Pierre Geurts, and Guy Leduc. Network distance prediction based on decentralized matrix factorization. In *NETWORKING 2010*, pages 15–26. Springer, 2010.

[11] Yongjun Liao, Wei Du, Pierre Geurts, and Guy Leduc. Dmfsgd: A decentralized matrix factorization algorithm for network distance prediction. *Networking, IEEE/ACM Transactions on*, 21(5):1511–1524, 2013.

[12] Benjamin Recht, Maryam Fazel, and Pablo A Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM review*, 52(3):471–501, 2010.

[13] M Fazel, E Candes, B Recht, and P Parrilo. Compressed sensing and robust recovery of low rank matrices. In *Signals, Systems and Computers, 2008 42nd Asilomar Conference on*, pages 1043–1047. IEEE, 2008.