Recommendation Systems

Boran Hao
Dept. of Systems Engineering
brhao@bu.edu

Runzhou Han
Dept. of Electrical and Computer Engineering
rzhan@bu.edu

Yuning Xie
Dept. of Electrical and Computer Engineering
ynxie@bu.edu

Abstract

This report reviews the concept and development of recommendation systems, and detailedly researches three collaborative filtering algorithms, including User/ItemCF, Truncated SVD, and Probabilistic Matrix Factorization. We analyze the theory and key mathematical idea behind these methods, and apply them on MovieLens dataset. Through the experimental observation, we discuss the reasonability of results, and conclude the characteristic of each method.

1. Introduction

Recommendation system is a concept relevant to search engine. Instead of using the intentional input from users, recommendation system algorithms analyze users' information and behaviors to find their potential requirements, then recommend items users potentially like. Recommendation system deals with seriously missing data, which is the most significant difference from other machine learning problems. Apart from content-based and demographic-based methods, what we research called collaborative filtering (CF) is the most important class of algorithms in this area, and we concentrate on using such methods to solve a type of rating matrix recovery problem, which can be widely applied to movie, catering and other industries.

2. Literature Review

The first milestone of collaborative filtering was considered to be GroupLens developed in University of Minnesota, 1994 [1]. The original GroupLens system used Pearson correlations to judge similarity between different users, and gave a prediction by adding a weighted average of deviations from correlated neighbor's mean value. The idea of User/ItemCF in our project is based on the paper from AT&T Lab in 2001[2]. The similarity measurement in our CF algorithms is mainly referred from it.

In 2009, another idea of CF, latent factor model(LFM) came into people's eyes as Koren's paper won the one million dollars prize offered by Netflix by applying singular value decomposition (SVD) in CF [3]. Actually, as a method of matrix factorization, the application of SVD in other areas has a relatively longer history than conventional collaborative filtering based on nearest neighbor. Horn elaborated SVD in both of his book published in 1985 and his paper in 1991[4][5]. Based on SVD, the Truncated SVD (TSVD) in our project use the idea of truncation to remove noise caused from data initialization and observation in our rating matrix.

As time goes on, Many LFM algorithms based on statistical understanding were created. One of those algorithms, probabilistic matrix factorization (PMF), which we applied in our project is a model based on assuming a Gaussian distribution on observed ratings. In PMF, in order to get best result, we have to tune hyperparameters. Slakhutdinov and Mnih did some advanced work about how to tune hyperparameters automatically [6]. In their paper, they presented that model complexity of PMF can be automatically controlled by integrating over all model. In our project, we found PMF 's performance is not as good as our expectation in high matrix sparsity. Correspondingly, Slakhutdinov's paper also introduced a way to constrain the model in the sparse situation.

3. Problem formation and algorithms

3.1. Problem formation

To research collaborative filtering methods, we need to build the relationship between users and items. We formulate our problem as follows: suppose we have M users, N items and a sparse M*N user-item rating matrix R, where R_{ij} means the rating from user i towards item j and we have the null value index set $S = \{(i,j) | R_{ij}\ unknown\}$, then how should we predict R_{ij} for $(i,j) \in S$ to recover matrix R, and use highest predicted values to make the recommendation for corresponding users.

To use the information in R, a most simple way is filling null values with a baseline rating:

$$B_{ij} = E[R_{ik} | (i,k) \notin S], \qquad (i,j) \in S \qquad (1)$$

which means the average rating of user i's rated items. This baseline rating reflect the user's rating habit like some 'tough' or 'easy' raters always give low or high scores, which gives us a most conservative estimation. However, there will be no ranking if we use those equal predictions

for each user to make recommendation, so strategy based on this baseline is equivalent to 'random guessing'. We use this baseline to be a basic standard, if the error made by some methods exceeds the baseline error, then the method does not make sense since it performs even worse than randomly guessing what item would user like.
Also, we can centralize our data using those baseline values.

$$\tilde{R}_{ij} = R_{ij} - E[R_{ik}|(i,k) \notin S], \qquad (i,j) \notin S \tag{2}$$

To make the observed ratings of each user have mean 0, then the problem of tough/easy rater is solved. In this report, all data are centralized before further processing, and the notation will still be R_{ij}.

3.2. User/ Item Collaborative Filtering

User/Item Collaborative Filtering (User/ItemCF) is an algorithm based on nearest neighbor idea[1], and a basic assumption here is users with similar rating behaviors like similar items. Basically, UserCF uses the m-th row R_m in R to be the features vector of m-th user, and ItemCF uses the n-th column $\overline{R}_n$ to be the feature vector of n-th item. By calculating the similarity between user/item feature vectors and building the weighted adjacency matrix W, we find the k nearest neighbor users/items for each user/item, and then use the corresponding ratings and similarities to make predictions. To simplify the interpretation, we use UserCF to clarify the method, and the implementing of ItemCF is to simply transpose original R matrix before centralized.

3.2.1. Similarity measurement

To build W where $W_{ij} = sim(R_i, R_j)$, first we need to choose a similarity metric. We choose the cosine similarity:

$$sim(R_i, R_j) = \frac{R_i \cdot R_j}{|R_i||R_j|} \in [-1,1] \tag{3}$$

The main reason here is cosine similarity is symmetric to 0, and we will not use a negative correlated user to participate into the prediction, even if he is among the the k nearest neighbors. If we use other positive similarity like Gaussian, it requires more complicated estimation to decide above what value shall we confirm the two vectors are 'enough' similar.

3.2.2. Null value problem

Another problem is how to deal with null values. A simple strategy is to ignore null values and only compute the similarity between the features that are completed for both users. Using this way will make the dataset even sparser, and we may lose some information since a rating is null may because the user does not want to buy or rate it at all, then the actual rating should be even less than minimum possible rating. Since our dataset has been centralized to 0-mean, we can make the basic assumption: a positive rating means like and negative means dislike, when 0 means 'average'. If we set null values to be this average value 0, then our cosine similarity between two users' vectors R_i and R_j becomes:

$$\frac{R_i \bullet R_j}{|R_i||R_j|} = \frac{\frac{1}{N}\sum_{n=1}^{N}(R_{in}-\mu_i)(R_{jn}-\mu_j)}{\frac{1}{N}\sqrt{\sum_{n=1}^{N}(R_{in}-\mu_i)^2}\bullet\sqrt{\sum_{n=1}^{N}(R_{jn}-\mu_j)^2}} = \frac{cov(R_i,R_j)}{\sigma_i\sigma_j} = \rho(R_i,R_j) \tag{4}$$

where $\rho(R_i, R_j)$ is exactly our *Pearson Correlation* between user row vectors R_i and R_j. This strategy is widely used in most nearest neighbor based recommendation systems, and our result shows that in most cases Pearson correlation coefficient performs better. In this report we use Pearson Correlation to be the similarity measurement.

3.2.3. UserCF algorithm

To implement UserCF algorithm, the tuning parameter we need to fix is the nearest neighbor number k, then our algorithm is:

(a) Centralize data, set $R_{ij} = 0, \forall\ (i,j) \in S$
(b) Build weighted adjacency matrix W, where $W_{ij} = sim(R_i, R_j)$
(c) For $(i,j) \in S$, find at most k indexes $\{(i,v_1) \dots (i,v_l)|\ v \neq i, l \leq k, W_{iv} > 0\}$ of the largest positive entries in W_i, $l \leq$ k since there might be only less than k+1 positive entries in some rows
(d) For indexes $v_1 \dots v_l$ found in (3), compute the weighted average:

$$\hat{R}_{ij} = \sum_{\substack{p=1 \\ (v_p,j)\notin S}}^{l} W_{iv_p} R_{v_p j} \Bigg/ \sum_{\substack{p=1 \\ (v_p,j)\notin S}}^{l} W_{iv_p} \tag{5}$$

If all $(v_p, j) \in S$, then we won't make prediction. Back to (a), until we tried all $(i,j) \in S$
Through this algorithm, it is possible that many null values will not be predicted since we only use the observed ratings from positive correlated users. The time complexity of UserCF and ItemCF are $O(3M^2N)$ and $O(3N^2M)$, and the main time consuming part is similarity computing.

3.3. Truncated SVD

Besides nearest neighbor based methods, another class of algorithms in collaborative filtering is called Latent Factor Model (LFM), and Truncated SVD (TSVD) is one representative method among LFMs. The basic assumption of LFM is: between users and items, there are K linearly independent latent factors for both users and items. If we assume P_{ik} means the extent user i likes the factor j, and Q_{kj}

means the extent item j has the factor k, then the idea of linear LFM is that ratings are the linear combination of P_{ik} and Q_{kj}:

$$R_{ij} = \sum_{k=1}^{K} P_{ik} Q_{kj} \tag{6}$$

This expression is equivalent to decompose R into two User-Factor matrix P and Item-Factor matrix Q:

$$R_{M*N} = P_{M*K} Q_{K*N} \tag{7}$$

And the main task for LFM is to achieve this decomposition while extracting most information from R. Such mathematical idea is generally called matrix low-rank decomposition.

3.3.1. SVD and TSVD

For any matrix A, through singular value decomposing (SVD), we can always find two sets of orthonormal basis $v = \{v_1, v_2, \dots, v_n\}$ and $u = \{u_1, u_2, \dots, u_n\}$, such that $Av_i = \sigma_i u_i$ $(u, v \neq \mathbf{0})$, which means the mapped basis vectors are still orthogonal, but scaled by σ_i on each degree. Those orthonormal basis vectors are linearly independent, so we try to use u and v to build our linearly independent latent factors.

Now suppose for a full rating matrix R, through SVD we can find such basis vectors v_k and u_k:

$$R^{M*N} v_k^{N*1} = \sigma_k u_k^{M*1} \tag{8}$$

and furthermore:

$$R_i^{1*N} v_k^{N*1} = \sigma_k u_{ki}^{1*1} \tag{9}$$

if we assume v_k and u_k are the k-th latent factor vectors for item and user, then the left hand side of (9) is the linear combination of 'extent user i likes the items 1~N' and 'extent items 1~N have factor k', this result is exactly the right hand side 'extent user i likes the factor k', scaled by σ_i. This relationship indicates that the orthonormal basis u, v and singular values can be used to describe our LFM factors under SVD, and the rating matrix R actually linearly maps the factor basis vectors from 'item space' to 'user space', then the factor basis vectors whose corresponding σ_i are largest describe the most feature of this mapping matrix, i.e. rating matrix R.

In PCA, we remain the covariance matrix eigenvectors whose eigenvalues are largest to extract the main features of data, and remove noise. Similarly, we can achieve this in SVD through Truncated SVD (TSVD) algorithm. Suppose we set null values in R to be some average value and have the decomposition $U_{m\times m} \sum_{m\times n} V_{n\times n}$, then by setting the smallest singular values to be 0, we remove the noise caused from average initialization and observation, mostly extract the information given by R.

3.3.2. TSVD algorithm

To implement TSVD, the tuning parameter K is the number of largest singular values we remain, then the algorithm is:

(a) Centralize data, set $R_{ij} = 0, \forall\, (i,j) \in S$
(b) Implement SVD: $R = U\Sigma V^T$
(c) Remain K largest singular values in Σ and set others to be 0, we get Σ^{new}
(d) Compute $\widehat{R} = U\Sigma^{new} V^T$, and null values will be filled

The time complexity of TSVD is $O(M^2N + N^2M)$

3.4. Probabilistic Matrix Factorization

Relevant to TSVD, Probabilistic Matrix Factorization (PMF) uses a statistical model to express the latent factor model. Through Maximum a Posterior estimation, we can estimate P and Q as parameters.

3.4.1. Model construction

Based on our LFM assumption, we know that the observed rating R_{ij} is decided by the i-th row P_i and j-th column $\overline{Q}_j$ of User-Factor and Item-Factor matrices. Like the formulation of linear regression, we assume that observed values $R_{ij}, (i,j) \notin S$ are IID Gaussian samples with mean $P_i \overline{Q}_j$ and observation variance σ^2, then we have the model:

$$p(R|P,Q,\sigma^2) = \prod_{i=1}^{M} \prod_{j=1}^{N} \left[\mathcal{N}(R_{ij} | P_i \overline{Q}_j, \sigma^2)\right]^{I_{ij}} \tag{10}$$

where I_{ij} is an indicator function, equal to 1 if $(i,j) \notin S$ .To prevent overfitting problem during parameter estimation, we set spherical Gaussian prior distribution to constrain the parameter P and Q, namely:

$$p(P|\sigma_P^2) = \prod_{i=1}^{M} \mathcal{N}(P_i | 0, \sigma_P^2 I) \tag{11}$$

$$p\left(Q|\sigma_Q^2\right) = \prod_{j=1}^{N} \mathcal{N}\left(\overline{Q}_j | 0, \sigma_Q^2 I\right) \tag{12}$$

with both 0 mean and σ_P^2, σ_Q^2 prior variance.

3.4.2. Parameter learning

With the training samples in R and model we constructed, we can use maximum a posteriori to estimate the parameters:

$$\hat{\theta}_{MAP} = arg\max_{P,Q} ln\, p(P,Q|R,\sigma^2,\sigma_P^2,\sigma_Q^2)$$
$$= arg\max_{P,Q} ln[\,p(R|P,Q,\sigma^2)p(P|\sigma_P^2)p\big(Q\big|\sigma_Q^2$$
$$= arg\max_{P,Q}\left\{-\frac{1}{2\sigma^2}\sum_{i=1}^{M}\sum_{j=1}^{N} I_{ij}\big(R_{ij}-P_i\bar{Q}_j\big)^2\right. \quad (13)$$
$$\left.-\frac{1}{2\sigma_P^2}\sum_{i=1}^{M}P_iP_i^T-\frac{1}{2\sigma_Q^2}\sum_{j=1}^{N}\bar{Q}_j{}^T\bar{Q}_j)+C\right\}$$

and C means terms not related to parameter P and Q. If we set $\lambda_P = \sigma^2/\sigma_P^2$and $\lambda_Q = \sigma^2/\sigma_Q^2$, then we have the final minimization problem:

$$\min_{P,Q}\sum_{i=1}^{M}\sum_{j=1}^{N} I_{ij}\left(R_{ij}-P_i\bar{Q}_j\right)^2+\lambda_P\|P\|_F^2 \quad (14)$$
$$+\lambda_Q\|Q\|_F^2\ (20)$$

where $\|\cdot\|_F$ is the Frobenius norm of matrix. We can see that the final optimization problem we need to solve is a non-convex l_2 regularized problem, without a closed form global minimum solution, so we implement numerical method to get a local optimal solution.

3.4.3. PMF algorithm

Since the objective function f in (20) is a sum of Lipschitz continuously differential functions, we solve it through incremental gradient method, which means when problem shapes like:

$$\min f(x)=\sum_{i=1}^{|S|}f_i(x) \quad (15)$$

we can take the iteration step

$$x^{k+1}=x^k-\alpha^k\nabla f_{i_k}(x^k) \quad (16)$$

where α^k is the stepsize and we choose i_k from set $\{1,\dots,|S|\}$ cyclically. It can be proved that x^k converges to some $x(\alpha)$, and error $\|x^k - x(\alpha)\|$ has the order $O(\alpha)$. Generally incremental gradient method converges faster than steepest descent except for the very last iterations around optimal, which leads to less iteration steps. Here we cyclically choose the terms corresponding to null indexes, together with the regularization terms to compute gradient. Since now the observed ratings are random variables, this method can be also understood as stochastic gradient descent.

4. Implantation

We use Python 3.6.1 on Ubuntu (x64) to achieve the algorithms. We build a Python class and set every algorithms and necessary steps (e.g. centralize) to be class methods, which result in a very simple main function to further apply our validations. We set random seeds every time before using random related functions, to make sure a definite result. Our Python code is attached in appendix.

5. Experimental results

5.1. Data set

To implement our algorithms, we choose MovieLens 1M as our real world dataset. This dataset includes more than 1,000,000 integer ratings (1-5) toward 4000 movies from 6000 users, which can be regard as a 6000*4000 sparse rating matrix. To evaluate the prediction, we need the ground truth ratings, so we extract a full 28*41 matrix R from the original dataset, and then randomly remove values from it to make it 'sparse'. Since recommendation system problem is highly related to reality, our synthetic dataset is relevantly simple and attached in appendix, basically generated by assuming certain users like some type of movies.

5.2. Performance metrices

The metric we used to evaluate accuracy is Rooted Mean Squared Error (RMSE), defined by:

$$\mathrm{RMSE} = \sqrt{\frac{1}{|\mathrm{S}|}\sum_{\mathrm{ij}\in \mathrm{S}}(\widetilde{\mathrm{R}}_{\mathrm{ij}}-\widehat{\mathrm{R}}_{\mathrm{ij}})^2} \quad (17)$$

where $\widetilde{\mathrm{R}}_{\mathrm{ij}}$ are the centralized groundtruth ratings in full matrix. RMSE is widely used in recommendation system researches. If we compute the RMSE of baseline ratings, then this is exactly the error made by 'random guessing'.

Another metric we used to evaluate the recommendation diversity is Coverage:

$$\mathrm{C}=\frac{1}{\mathrm{N}}\left|\bigcup_{\mathrm{i}\in \mathrm{U}} G_i\right| \quad (18)$$

where U is the user set and G_i is the movie list recommended to user i. A good recommendation algorithm should suggest user a relevantly wide range of items to all users and give them more chances to try new things.

5.3. User/ItemCF performance

As nearest neighbor based algorithm, the nearest neighbor number k is a most essential parameter we should concern.

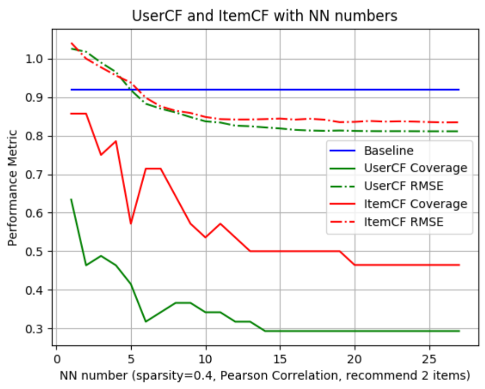

Figure.1

Figure.1 shows the comparison of User and ItemCF performance with k ranging from 1-28, under a 0.4 sparsity matrix (40% ratings are null). For accuracy, it can be observed that when k increases, both RMSE of User/ItemCF decrease passing through the baseline error and converges after about k>20. This trend indicates that even if we find a few nearest neighbors of a user or item, the information they could provide can be limited and misleading. Taking more neighbors into account will provide a more comprehensive prediction. According to the theory in[?], in most cases a proper k number is around 20, since too few neighbors will give wrong information while too much neighbors will increase the time complexity and let into more noise. Our observation confirms this theory. Another observation is that with error decreasing, recommendation coverage also decreases. This indicates a trade-off between recommendation accuracy and diversity. An accurate prediction can be more certain and constrain the recommendation results into a small range of items that user most likely like, meanwhile provide less chance for users to see the fresh items. For a practical application, we should consider both metrices to decide that nearest neighbor number parameter. Also, we can find that in most cases the accuracy performances of UserCF and ItemCF are similar, but the coverage of ItemCF is higher, which means ItemCF has higher recommendation diversity.

5.4. TSVD performance

For a LFM model, the most important parameter to be fixed is the latent factor number K.

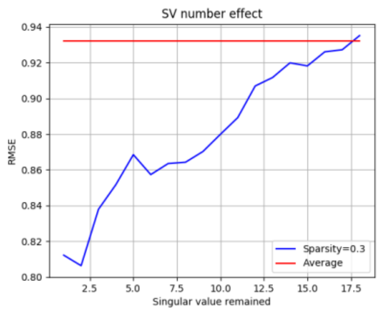

Figure.2

And our result shows that the prediction accuracy is highly related to this number. In figure.2 we notice that under 0.3 sparsity, the best K is 2. However if we keep increasing K, the error will quickly increases and finally even exceeds the baseline. This trend makes sense because remaining too many singular values leads to remaining too much noise. Even if there are chances for getting more information, those information explained by the corresponding extremely small singular values are too less. As the results showed in [], even for some billion number-level rating matrix, the optimal latent factor number is generally 10-20. That's the reason why idea behind LFM is called 'low-rank decomposition'.

5.5. PMF performance

The final shape of the PMF parameter estimation is like a supervised learning process, the observed ratings are our training set and the null values to be predicted is the test set. We can find that the objective function (20) is actually the sum of squared error for each observed rating terms, then the rooted expectation of those squared error is exactly our training RMSE.

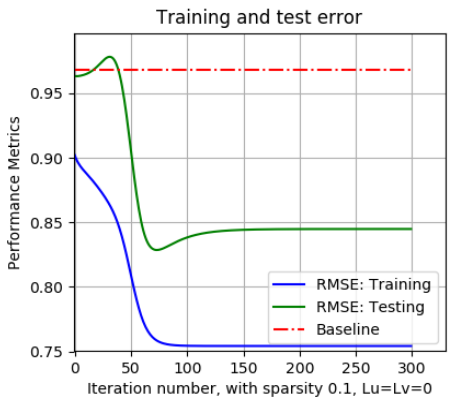

Figure.3

In figure.3, without regularization, for each iteration we compute the RMSE on both training and test set. As the objective function value keeps decreasing, our training error absolutely decreases. However, we can observe that since about the 75th iteration, test error increases and converges. This is a typical overfitting performance since our parameters P and Q fits the training set too much, which decrease the generalization ability on test set. In fact Tab.? shows that if we use larger latent factor number K, which means a more complicated model to fit the training samples, the training accuracy and test error will both increase. This problem is similar to using a high dimensional polynomial to fit the training set in our linear regression.

Tab. Training and test RMSE with K

K	1	2	3	4
Training	0.7541	0.6807	0.6207	0.5701

Testing	0.8442	0.8856	1.0780	1.1382

e=0.0001, no regularization

Figure.4 shows that if we set we set $\lambda_P = \lambda_Q$ after we add the regularization terms, we can actually improve the test performance by sacrificing some training accuracy. Using cross validation, we can estimate the optimal regularization parameters before learning the whole training set.

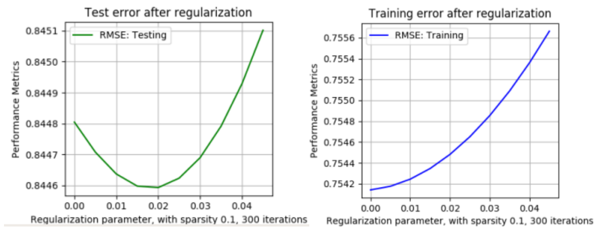

Figure.4

5.6. Performance comparison

Using the best tuning parameters for test RMSE, finally we compare the performances of algorithms given different rating matrix sparsity. In Figure.5 we notice that when sparsity rate is below 0.3, all 4 algorithms reaches the similar accuracy. However, when data sparsity reaches 80%-90%, Item/UserCF fail to find the ideal prediction and the error exceeds the baseline, when TSVD still keeps a reasonable performance. This indicates the reason why LFM becomes more and more popular in recommendation systems is that in practical application User-Item dataset usually has the sparsity rate more than 90%. For PMF, we found that for some higher sparsity and $\epsilon = 0.0001$, the convergence value of test error becomes strangely higher, even if we choose a very small stepsize. However the related research shows that PMF performs very well in sparse and unbalanced rating matrix, so there should be more theory to fix the initial point, stepsize and stop criterion.

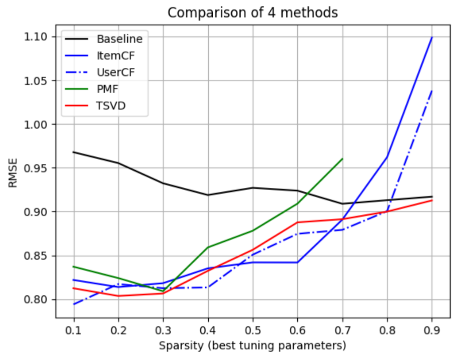

Figure.5

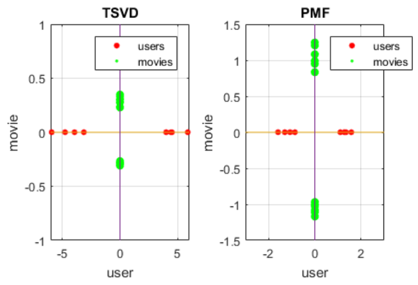

Figure.6

Figure.6 is the result we get on synthetic dataset (see appendix). Using TSVD and PMF under K=1, $\lambda_P = \lambda_Q = 10^{-17}$, we can get P and Q as User-Factor and Item-Factor vectors. If we visualize those vectors on user axis and movie axis, we can find the users and movies are divided into 2 clusters, which means 'like' or 'dislike' such factor on User axis and 'have' or 'don't have' such factor on Movie axis. This result is exactly the same as our setting, and intuitively shows how LFMs work.

6. Review

6.1. Algorithm conclusion

Through our research of 3 algorithms, we have the following main conclusions:

(a) The Nearest neighbor based CF methods works well in low sparsity condition. ItemCF and UserCF have the similar accuracy, when ItemCF has better coverage.

(b) LFM methods (TSVD and PMF) performance is highly depended on the latent factor number selection, and this number is generally far smaller than user and item number.

(c) For highly sparse rating matrix, TSVD performs much better than nearest neighbor based methods. PMF performance also highly relies on the optimization algorithm setting.

(d) Through adding proper regularization in PMF, the overfitting problem can be avoided to some extent, and test performance will be improved.

6.2. Future work

(a) Try our algorithms on larger, biased datasets like NetFlix to get more practical performance observations of algorithms. Try not to make null from full matrix, but from an originally sparse rating matrix to keep some null position information.

(b) For PMF, find a better way to control the optimization iteration performance and decide the initial P and Q, then try diminishing stepsize or using line search. Try to use stochastic gradient method theories to improve the convergence.

7. References

[1] R. Paul, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens. An Open Architecture for Collaborative Filtering of Netnews. In *proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work - CSCW 94*, 1994.
[2] L. G. Terveen, W. Hill. Beyond Recommender Systems: Helping People Help Each Other. In *HCI In The New Millennium*, Jack Carroll, ed., Addison-Wesley, 2001.
[3] Y. Koren, B. Robert. Advances in Collaborative Filtering. In *Recommender Systems Handbook*, pages 77-118, 2015.
[4] R. A. Horn, C. R. Johnson. *Matrix Analysis*, Section 7.3, 1985.
[5] R. A. Horn, C. R. Johnson. *Topics in Matrix Analysis*, Chapter 3, 1991.
[6] R. Salakhutdinov, A. Mnih. Probabilistic Matrix Factorization. *Proceeding of the 20th International Conference on NIPS*, pages 1257-1264, 2007.