Matrix Completion: Order and Extend Algorithm Evaluation

Qiwei Li, Xueting Shao, Yin Zhang
University of California, Davis

December 2015

Abstract

In this paper, we are going to do a review on "Order and Extend", which is an active matrix completion algorithm in the paper "Matrix Completion with Queries". In section 2, we will summarize the original approach. In section 3, we will present some challenges and assumptions we had when reproducing the algorithm. In section 4, we discuss the effect of the model parameter. In section 5, we will compare the algorithm with other traditional matrix completion method.

1 Introduction

Matrix Completion has been used in many fields, such as recommender systems (where entries of the matrix indicated user preferences over items), network traffic analysis (where the matrix contains the volumes of traffic among source destination pairs), and computer vision (image matrices). In many cases, only a small percentage of the matrix entries are observed. For example, the data used in the Netflix prize competition was a matrix of 480K users $\times$ 18K movies, but only 1% of the entries were known.

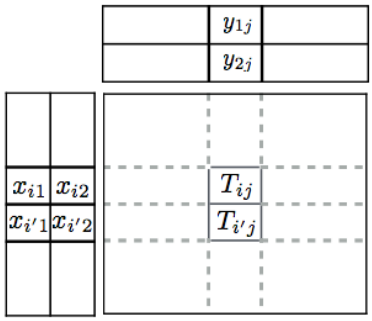
Recovering a low rank matrix from a given subset of its entries is a recurring problem in collaborative filtering, dimensional reduction, and multi-class learning. Iterative methods for matrix completion that use nuclear-norm regularization.There are several classic methods. The two popular approaches are singular value decomposition (SVD) and large-scale parallel matrix factorization.The one approach uses iterative soft-thresholded SVD to impute the missing values. The other approach recovering the matrix by recovering the matrix factorization. Alternating Least Squares (ALS) and Stochastic Gradient Descent (SGD) are two popular approaches to compute matrix factorization. An improved popular algorithm is large-scale parallel matrix factorization. So we will use SVD and large-scale parallel matrix factorization as the comparison with *Order & Extend*.

A new algorithm *Order & Extend* which proposed in Matrix Completion with Queries in 2015, has similar idea with SVD but a method for generating a small number of queries so as to ensure that the combination of observed and queried values provides sufficient information for an accurate completion.The difference between the classical matrix-completion problem and the new one is that in the former, the set of observed entries is fixed and the algorithm needs to find the best completion given these entries. In Active Completion, The authors design both a completion and a querying strategy in order to minimize the reconstruction error.

In this report, we first study the *Order & Extend* algorithm, then compare the performance with SVD and large-scale parallel matrix factorization.

2 Algorithm Description

The starting point for the design of Order and Extend is the low (effective) rank assumption of T (XY=T). The approach is to solve a sequence of linear systems, which means recovering rows in X and columns in Y one after one by setting up linear systems with information we have in T.



$$T_{ij} = x_{i1}y_{1j} + x_{i2}y_{2j}$$
$$T_{i'j} = x_{i'1}y_{1j} + x_{i'2}y_{2j}$$

Figure 1: An example of a linear system when solving for Y_j

For example, in Figure 1, we are trying to solve for Y_j by setting up linear systems with X_i, $X_{i'}$, T_{ij}, and $T_{i'j}$. Note that we have 2 columns in X and 2 rows in Y. This is because the assumed rank, r=2. When the linear system is not stably solvable, we query additional information in T to proceed the sequential steps. Thus, the algorithm has three main components. We will discuss each one below.

2.1 Finding the order

Each row in X is connecting to columns in Y though some entries in T. A good order, π, is one that minimize the occurrence of incomplete system, which happens when number of equations is less than unknown. Define the degree of a row in X or column in Y to be how many neighbors it connects to. We can find an order by the following.

1. Sort all X's and Y's by degree.
2. Put the last one at the end of π.
3. Remove this one from the matrix and recompute degrees
4. Put the last one in the next-to-last position in π.
5. Repeat until the all X's and Y's are in π

Then the algorithm perform the following adjustments on each u in the order to improve the efficiency.

1. if u has degree $\leq r$: it is repositioned to appear immediately after the neighbor v with the largest $\pi(v)$.
2. if u has degree ¿ r: it is repositioned to appear immediately after the neighbor its neighbor v with the the $r - th$ smallest $\pi(v)$.

These adjustments aim to construct an order such that when the implied directionality(solving from the largest u in π to smallest) is added, each u has degrees as close to r as possible.

2.2 Fixing incomplete systems

The definition of an incomplete system is when number of equations is less than unknown. Thus, we need to query additional information in T to complete the system. However, this information in T we query must have its corresponding row in X (if solving for Y) or corresponding column in Y (if solving for X) recovered.

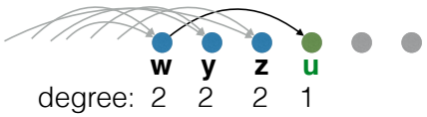


Figure 2: An example of incomplete system

For example, in Figure 2, $r = 2$. u has degree 1, which means it is only able to setup one equation when the system needs two to be complete. Thus, we need to query additional information in T. Recall that the information in T we query must have its corresponding row in X (if solving for Y) or corresponding column in Y (if solving for X) recovered. In Figure 2, an connection between {w,u} is already used. Therefore, the only candidates of information we can query are {y,u} and {x,u}.

2.3 Fixing unstable systems

If the linear system we set up is $A_x y = t$, then the solution of it is $y = A_x^{-1}t$. Let $A_x = U\Sigma V^T$ be the singular value decomposition of A_x with singular values $\alpha_1 > \alpha_2 > \ldots > \alpha_r$. If there exists α_j such that α_j is really small, then $\frac{1}{\alpha_j}$ will be very large. This will cause the solution $y = A_x^{-1}t$ unstable. To detect if a system is unstable, the algorithm defines the *local condition number*.

$$l(A_x, t) = ||A_x^{-1}||_2 \frac{||t||_2}{||y||_2}$$

. We can set a threshold of this value. If a linear system has *condition number* ¡ threshold, we query additional information in T, similarly in section 2.2, to stabilize the linear system. If the algorithm fails to stabilize the linear system after trying all candidates, we move it to the end of π.

3 Reproducing the Algorithm

The inventors of the algorithm do not have the code available, so we have to code the entire algorithm again. The language we used is R. Our code is available at **https://github.com/qiwei-li/order-and-extend**

At a high level, *Order and Extend* is very easy to understand. However, when we actually implement this, many issues came up. First, some details of this algorithm weren't discussed in the original paper. Since this algorithm is not mathematically based, those details really rely on the inventors' implementation. We had to skype with the inventor twice to understand some of the details. Second, each linear system is solved by taking a inverse of a matrix and the additional information is always queried in a un-deterministic fashion. This increases the complexity when debugging. Moreover, the algorithm uses different methods to solve the linear system in different situations(inverse and least square), which increases the complexity as well. Thirdly, the algorithm requirement a lot memory at some steps. For example, in the second adjustment part of the ordering, it requires to move a node in a list. Even though I used a linked list data structure, the computation is still slow. When taking the inverse, the computational time is also slow. Sometimes, the algorithm takes hundreds of inverses just to find the best candidate to stabilize one linear system.

On the other hand, we realized there could be many parts for improvements. We think the stabilize part has the most potential for improvements. Since every linear system uses previously computed information, the quality of those previously computed information is really important for the entire performance of the algorithm.

4 Model Parameter

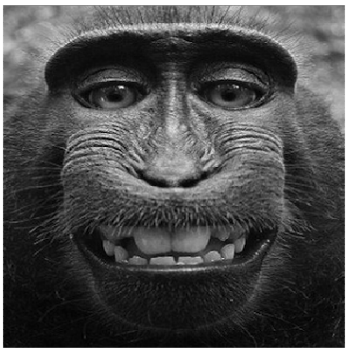


Figure 3: monkey selfie

We are testing *Order and Extend* using the monkey selfie in 3. The image can be treated as a matrix of pixel values. We will manually subset the matrix to be input of the algorithm as the incomplete matrix that needs recovery.

4.1 Assumed Rank

Other than the visible matrix as a input, *Order and Extend* needs the assumed rank of the low rank matrix and the local condition number threshold. The rank of the matrix is always determined by the singular values of the singular value decompostion.

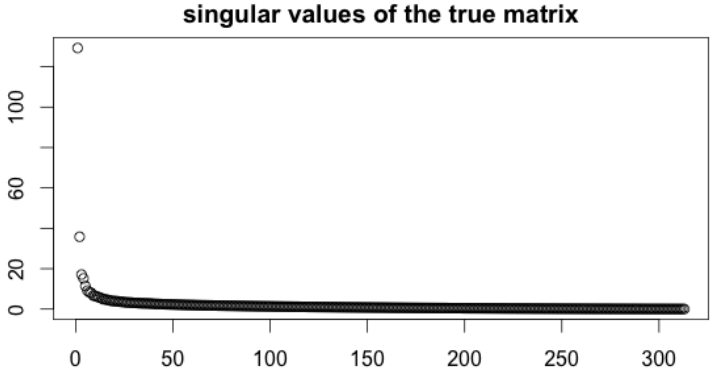


Figure 4: singular values

As we can see in Figure 4, the first 2 singular values are significantly large comparing to other singular values. On the other hand, after the 5^{th} largest singular values, the values stabilize. Therefore, we will apply *Order and Extend* by assuming the true matrix can be recovered by the product of two rank 2 once and two rank 5 matrices once.

4.2 Local Condition Number Threshold

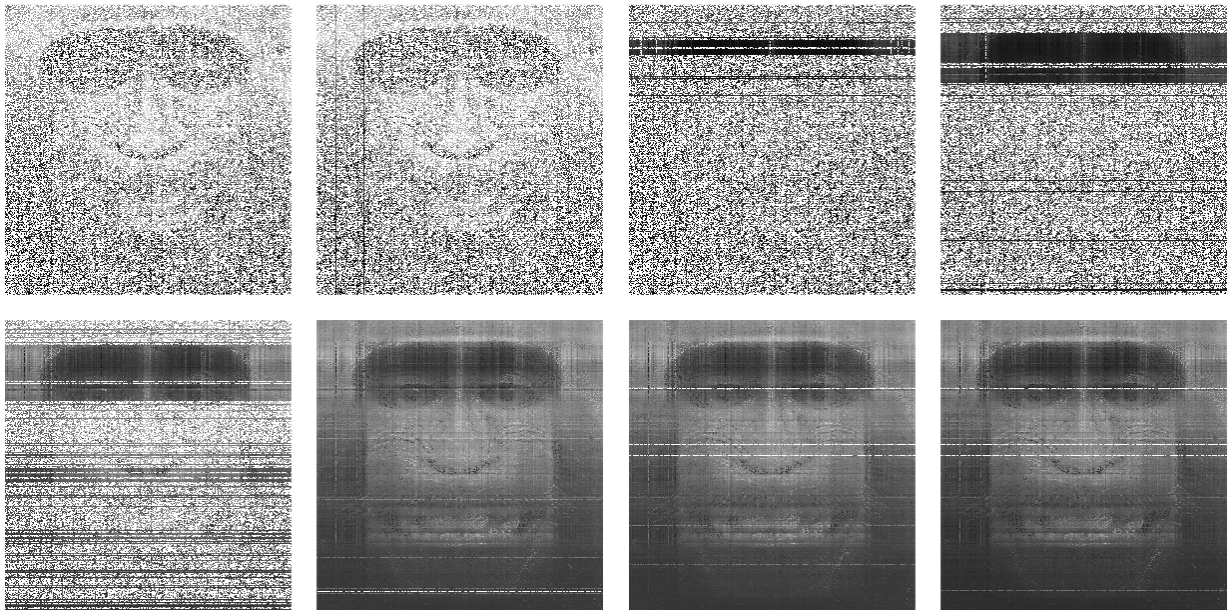


Figure 5: input matrix and 7 recover matrices when rank=2, threshold = 1, 1.5, 2, 2.5, 3, 3.5, 4

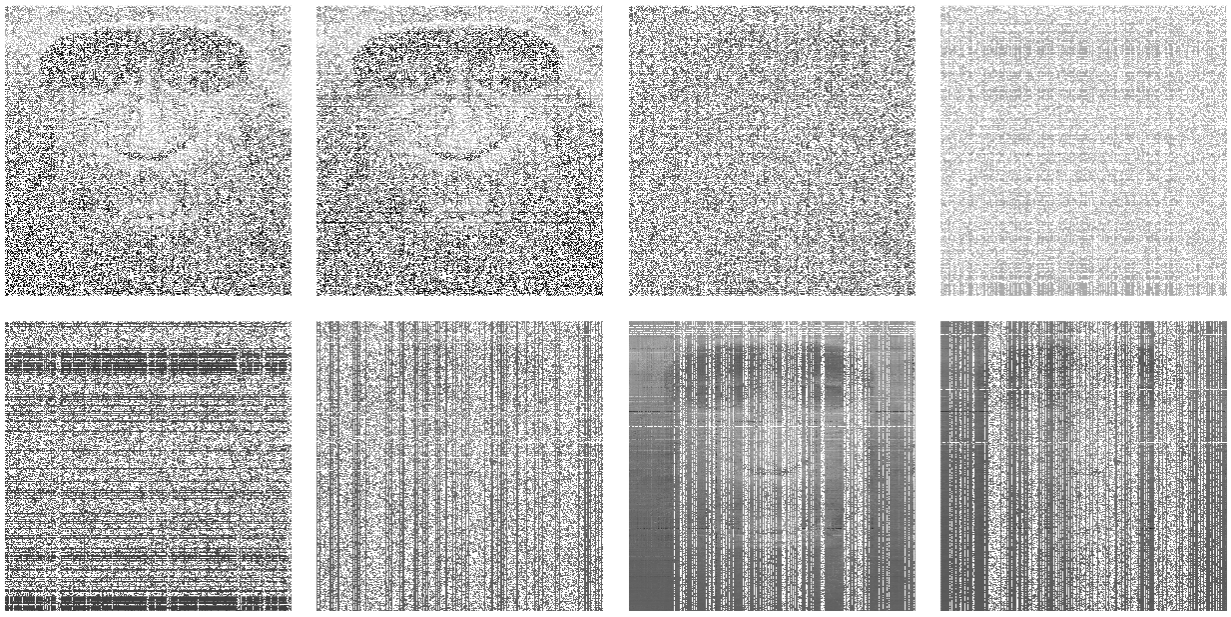


Figure 6: input matrix and 7 recover matrices when rank=5, threshold = 1, 1.5, 2, 2.5, 3, 3.5, 4

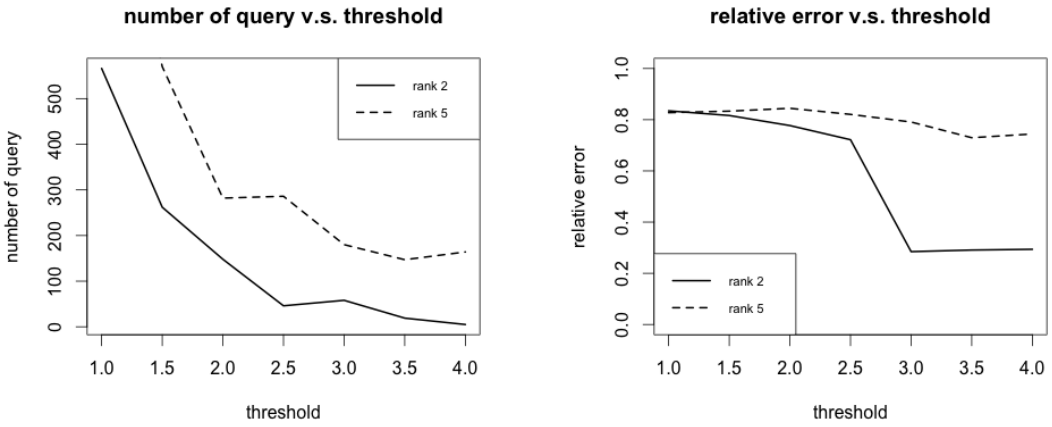


Figure 7: condition number threshold vs error and number of query

As you can see in Figure 5 and Figure 6, it is obvious that the assumption of rank = 5 does not help the algorithm. In Figure 7, we can see that as we being more generous on the condition number threshold, less query is needed because the algorithm is treating more systems to be stable. Moreover, if we set the local condition number threshold to be small, there will not be many solvable linear systems. If we set the local condition number threshold to be large, the quality of the solution will not be good. This phenomenon is very different from the original paper. The original paper states that setting the condition number threshold to be 1 will always give the best result on many real world application. This difference can be due to different implementations of our version.

5 Compare with Traditional Algorithms

5.1 SVD soft threshold imputation

SVD takes a rectangular matrix of gene expression data (defined as A, where A is a n x p matrix) in which the n rows represents the genes, and the p columns represents the experimental conditions. The SVD theorem states:

$$A_{n\times p} = U_{n\times n} S_{n\times p} V^T_{p\times p}$$

where $U^T U = I_{n\times n}$, $V^T V = I_{p\times p}$ (i.e. U and V are orthogonal).Where the columns of U are the left singular vectors (gene coefficient vectors); S (the same dimensions as A) has singular values and is diagonal (mode amplitudes); and V^T has rows that are the right singular vectors (expression level vectors). The SVD represents an expansion of the original data in a coordinate system where the covariance matrix is diagonal. The approach is to use iterative soft-thresholded svds to impute the missing values. In some sense, the algorithm has the "EM" flavor, in that at each iteration the matrix is completed. with the current estimate.

5.2 large-scale parallel matrix factorization

Large-scale parallel matrix factorization is a scalable and efficient coordinate descent method based matrix factorization method CCD++. The main idea is solve the problem:

$$\underset{\substack{W\in\Re^{m\times n}\\ H\in\Re^{n\times k}}}{min} \sum_{i,j\ in\Omega} (A_{i,j} - w_i^T h_j)^2 + \lambda(\|W\|_F^2 + \|H\|_F^2)$$

where Ω is the set of indices for observed ratings, λ is the regularization parameter, $\|\cdot\|_F$ denotes the Frobenius norm; wand w_i^T and h_j^T are the i^{th} and j^{th} row vectors of the matrices W and H, respectively. It is an efficient and easily parallelizable method for matrix factorization for large matrix completion.

5.3 Comparison

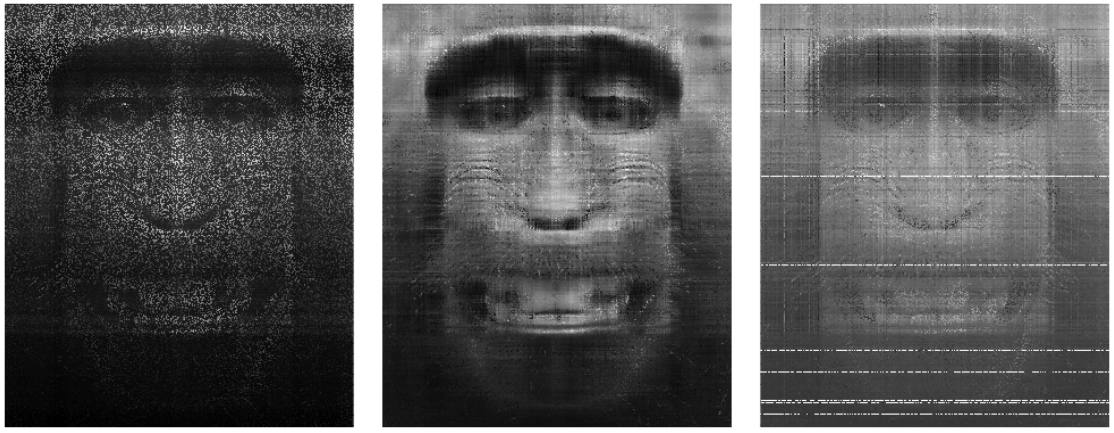


Figure 8: Recovered images from libpmf, svdImpute, and Order and Extend with the same input

From Figure 8, we see that *Order & Extend* does not perform as well as traditional algorithms, namely optimization approach and SVD approach.. Notice how *Order & Extend* is only able to cover some rows and columns. This is due to that *Order & Extend* gives up solving rows in X and columns in Y when they are never stable, causing the recovered matrix to be missing. Even though *Order & Extend* does not perform as well, as we discussed in section 2, there could be many improvements. Our implementation of *Order & Extend* is the most naive and basic version. It is worth to mention that **libpmf** is the fastest and really meant to be used for large scaled dataset.

6 Conclusion

In this report, we reviewed the active matrix completion algorithm, *Order & Extend*. In section 2, we summarized the algorithm. In section 3, we presented challenges we had when reproducing the algorithm and potential areas the algorithm has for improvements. In section 4, we studied the effect of model parameter on the number of query and relative error. In section 5, we compared the performance of *Order & Extend* to other traditional methods. In conclusion, *Order & Extend* is a new approach of active matrix completion and it has a lot potentials. Special thanks to one of the inventor of the algorithm, **Ruchansky Natali**, for guiding us through the whole process.

7 Reference

Ruchansky, Natali, Mark Crovella, and Evimaria Terzi. "Matrix Completion with Queries." Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015.

L. N. Trefethen and D. Bau III. Numerical linear algebra. SIAM Press, 1997.

Yu H F, Hsieh C J, Dhillon I. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. Data Mining (ICDM), 2012 IEEE 12th International Conference on. IEEE, 2012: 765-774.

Yu H F, Hsieh C J, Si S, et al. Parallel matrix factorization for recommender systems[J]. Knowledge and Information Systems, 2014, 41(3): 793-819.

Hansen P C. The truncatedsvd as a method for regularization[J]. BIT Numerical Mathematics, 1987, 27(4): 534-553.