Vol. 37 (2017)
No. 5

数 学 杂 志
J. of Math. (PRC)

A MATRIX COMPLETION ALGORITHM USING RANDOMIZED SVD

XU Xue-min, XIANG Hua
(*School of Mathematics and Statistics, Wuhan University, Wuhan 430072, China*)

Abstract: In this paper, we investigate the large low-rank matrix completion problem. By using randomized singular value decomposition (RSVD) algorithm, we compute singular values of sparse matrix. Compared to the Lanczos method, the computational time is greatly reduced with the same error. The algorithm also can be used to solve the relatively low rank matrix.

Keywords: matrix completion; singular value thresholding; unclear norm minimization; randomized singular value decomposition

2010 MR Subject Classification: 65F30

Document code: A **Article ID:** 0255-7797(2017)05-0969-08

1 Introduction

In many situations we need to recover a matrix which has low rank or approximately low rank. The problem requires that we randomly select m entries from an $n \times n$ matrix M and find out the missing or unknown values based on the sampled entries. Such problems arise from many areas, such as multi-task learning [3], control [10], machine learning [1, 2], image processing, dimensionality reduction or recommender systems in e-commerce, and so on. A well known method for reconstructing low-rank matrices is based on convex optimization of the nuclear norm.

Let $M \in \mathbb{R}^{n\times n}$ be an unknown matrix with rank r satisfying $r \ll \min\{m,n\}$, and suppose that one has available m sampled entries $\{M_{ij} : (i\ ,j\ ) \in \Omega\}$, where Ω is a random subset of cardinality m, and $\Omega \subset \{1, 2, \cdots, n\} \times \{1, 2, \cdots, n\}$. The authors in [4] showed that most low rank matrices M can be perfectly recovered by solving the optimization problem

$$\begin{array}{ll} \text{minimize} & ||X||_*, \\ \text{subject to} & X_{ij} = M_{ij}\ , \qquad (i,j\ ) \in \Omega \end{array} \tag{1.1}$$

provided that the number of samples obeys $m \geq Cn^{6/5} r \log n$ for some positive numerical constant C, here the functional $\|\cdot\|_*$ stands for the nuclear norm of the matrix M, i.e., the summation of all singular values. The optimization problem (1.1) is convex and can be

* **Received date:** 2014-12-19 **Accepted date:** 2015-04-21

Foundation item: Supported by National Natural Science Foundation of China (10901125; 11471253).

Biography: Xu Xuemin (1991–), female, born at Nanyang, Henan, master, major in numerical algebra.

recast as a semidefinite programming [6, 7]. If there were only one low-rank object fitting the data, this would recover M. This is unfortunately of little practical usage because this optimization problem is NP-hard, and all known algorithms which provide exact solutions require time doubly exponential in the dimension n of the matrix in both theory and practice. Some solvers based on interior-point methods can deal with this problem, but they can only solve problems of size at most hundreds by hundreds on a moderate PC. Since the nuclear ball $\{X : \|X\|_* \leq 1\}$ is the convex hull of the set of rank-one matrices with spectral norm bounded by one, the nuclear norm minimization problem can be approximated by the rank minimization problem as its convex relaxation

$$\begin{array}{ll} \text{minimize} & \text{rank}(X), \\ \text{subject to} & X_{ij} = M_{ij}\,, \qquad (i,j\,) \in \Omega. \end{array} \tag{1.2}$$

2 Algorithms for Completing Matrix

2.1 The Singular Value Thresholding (SVT) Algorithm

Problem (1.1) is extended in [4] as follows

$$\begin{array}{ll} \text{minimize} & ||X||_*, \\ \text{subject to} & \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M), \end{array} \tag{2.1}$$

where X is an optimization variable. We can use a gradient ascent algorithm applied to the problem with a large parameter τ and scalar step sizes $\{\delta_k\}_{k\geq 1}$. That is, starting with $Y^0 = 0 \in \mathbb{R}^{n\times n}$, the singular value thresholding iteration is

$$\begin{cases} X^k = \mathcal{D}_\tau(Y^{k-1}), \\ Y^k = Y^{k-1} + \delta_k \mathcal{P}_\Omega(M - X_k), \end{cases} \tag{2.2}$$

where $\mathcal{D}_\tau(\cdot)$ uses a soft-thresholding rule at lever τ to the singular values of the input matrix. Consider the singular value decomposition (SVD) of a matrix $Z \in \mathbb{R}^{n\times n}$, and the rank of it is r. That is,

$$Z = U\Sigma V^*, \qquad \Sigma = \text{diag}(\{\sigma_i\}_{1\leq i\leq r}).$$

The definition of $\mathcal{D}_\tau(Z)$ is given as follows:

$$\mathcal{D}_\tau(Z) := U \begin{bmatrix} (\sigma_1 - \tau)_+ & & \\ & \ddots & \\ & & (\sigma_s - \tau)_+ \end{bmatrix} V^*,$$

$$\text{where } (\sigma_s - \tau)_+ = \begin{cases} \sigma_i - \tau, & \sigma_i - \tau > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

The most important property of (2.2) is that the sequence $\{X_k\}$ converges to the solution of the optimization problem (2.1) when the values of τ is large. We get the shrinkage iterations

with fixed $\tau > 0$ and scalar step sizes $\{\delta_k\}_{k\geq 1}$. Starting with Y_0, we define for $k = 1, 2, \cdots$, until the stopping criterion is satisfied.

The parameters in the iterations are needed to be given. Let $\tau = 5n$ and $p = m/n^2$. In general, we use constant step sizes $\delta = 1.2p^{-1}$ [4], and set the stopping criterion

$$||\mathcal{P}_\Omega(X^k - M||_F/||\mathcal{P}_\Omega M||_F < \epsilon_2. \tag{2.4}$$

Since the initial condition is $Y^0 = 0$, we need to have a big τ to make sure that the optimization problem has a close solution. Now we let k_0 be an integer and have the following condition

$$\frac{\tau}{\delta\|\mathcal{P}_\Omega(M)\|_2} \in (k_0 - 1, k_0]. \tag{2.5}$$

Because $Y^0 = 0$, we needn't compute the first several steps [4]. It's easy to know that $X^k = 0$ and $Y^k = k\delta\mathcal{P}_\Omega(M)$ when $k \leq k_0$. To reduce the computing time, we begin the iteration at the k_0 step.

2.2 The Randomized Algorithm

In SVT, we need to compute $[U^{k-1}, \Sigma^{k-1}, V^{k-1}]_{s_k}$, where $U^{k-1}, \Sigma^{k-1}, V^{k-1}$ are the SVD factors of Y^{k-1} and s_k is the parameter of Lanczos process. The SVT algorithm uses the Lanczos method via the package PROPACK [9] to compute the singular value decomposition of a huge matrix. The main disadvantage of the classical singular value thresholding algorithm is that we need to compute the SVD of a large matrix at each stage by using a Krylov subspace method such Lanczos or Arnoldi to compute the rank-k SVD. As we know, the efficiency of Krylov subspace depends on the spectrum of the matrix, and only BLAS-2 operations are applied. When the rank of the matrix is not very low, it will take a lot of time to achieve the SVD approximation.

Algorithm 1 (RSVD): Given $M \in \mathbb{R}^{m\times n}(m < n)$ and $l < m$, compute an approximate rank-l SVD: $M \approx U\Sigma V^T$ with $U \in \mathbb{R}^{m\times l}$, $\Sigma \in \mathbb{R}^{l\times l}$ and $V \in \mathbb{R}^{n\times l}$.

1: Generate an $l \times m$ Gaussian random matrix Ω .
2: Compute the $l \times n$ matrix $Y = \Omega M$.
3: Compute the $n \times l$ orthogonal matrix Q via QR factorization $Y^T = QR$.
4: Form the $m \times l$ matrix $B = MQ$.
5: Compute the SVD of a small matrix B: $B = U\Sigma W^T$.
6: Form the $n \times l$ matrix $V = QW$, then $M \approx U\Sigma V^T$.

We use the randomized algorithm [8] instead of the Lanczos method to compute the SVD. The Lanczos method is one of Krylov subspace method and can be unstable, while the randomized is robust and simply to be implemented. It is not dependent on the spectrum of the sampled matrix. What's more, the randomized algorithm is easy to be parallelized.

The idea of the randomized algorithm is that we project the matrix onto a smaller matrix which preserves most of the important information and ignore the less important information. The pseudo-codes of the randomized algorithm are given as follows (see Algorithm 1) [11].

Algorithm 2 : The R-SVT algorithm

Input: sampled entries $\mathcal{P}(M)$ and sampled set Ω , the limit value of error ϵ_1, the step size δ, the thresholding value τ, constant l, the maximum iteration numbers $k_{\max}$.

Output: X

Goal: Get a matrix X with low-rank, and recover of the sampled entries

Pseudo-code :

1: let $Y_0 = k_0\delta\mathcal{P}_\Omega(M)$, the definition of k_0 is given in (7)
2: let $r_0 = 0$
3: for $k = 1$ to $k_{\max}$
4: use RSVD algorithm to Compute $[U^{k-1}, \Sigma^{k-1}, V^{k-1}]_{s_k}$
5: let $X^k = \sum_{j=1}^{r}(\sigma_j^{k-1} > \tau)u_j^{k-1}v_j^{k-1*}$
6: if $\|\mathcal{P}_\Omega(X^k - M)\|_F/\|\mathcal{P}_\Omega M\|_F \leq \epsilon_1$ then break
7: end for k
8: let $X = X^k$

2.3 The SVT Algorithm Using RSVD

In SVT iterations, the SVD is needed in each step. Since the classical methods for SVD approximation are costly. We use the randomized SVD, i.e., Algorithm 1, to replace the classical one, and obtain the R-SVT algorithm (see Algorithm 2). We can clearly see that in Step 4 of the pseudo-code of Algorithm 2, RSVD is used instead, while the classical SVT algorithm uses Lanczos method to find the singular values. At the beginning of computing, we don't know the number of the singular values, so we have to spend much time to find this number, and it could be very slow.

On the other hand, in the randomized algorithm, we just preserves the important information and ignore the less important information, so the relatively error of our result can be larger than the SVT algorithm. To obtain a small relatively error at low cost, we combine the two algorithms together, and have the algorithm R-SVT* (see Algorithm 3). At the first stage we use SVT based on RSVD until the error is smaller than ϵ_1, for example 0.1. Then we switch to the classical SVT based on PROPACK, until the error is smaller than ϵ_2, for example $1e-4$. The pseudo-codes of R-SVT* algorithm are given as follows.

The classical methods use the PROPACK to compute the approximate SVD, based on Lanczos process. In the algorithm R-SVT*, we use RSVD instead to perform SVT, and later switch to the classical SVT. Lanczos procedure needs to access the coefficient matrix several times, and use the BLAS-2 operations. In RSVD, the large matrix is accessed by less times, and the BLAS-3 operations are used. So we can expect that the randomized algorithm can

be much faster than Lanczos process for SVD approximation. Note that our work is different from that in [5]. Here we use a different randomized algorithm, i.e., algorithm from [11], and we also apply the strategy of switching to the classical SVT in our algorithm R-SVT*.

Algorithm 3 : The R-SVT* algorithm

Input: sampled entries $\mathcal{P}(M)$ and sampled set Ω , the limit value of error ϵ_1 and ϵ_2, the step size δ, the thresholding value τ , constant l , the maximum iteration numbers $k_{\max}$.

Output: X

Goal: Get a matrix X with low-rank, and recover of the sampled entries

Pseudo-code :

let $Y_0 = k_0\delta\mathcal{P}_\Omega(M)$, the definition of k_0 is given in (7)
let $r_0 = 0$
while $\|\mathcal{P}_\Omega(X^k - M)\|_F/\|\mathcal{P}_\Omega M\|_F > \epsilon_1$
    use R-SVD to compute $[U^{k-1}, \Sigma^{k-1}, V^{k-1}]_{s_k}$
    let $X^k = \sum_{j=1}^{r}(\sigma_j^{k-1} > \tau)u_j^{k-1}v_j^{k-1*}$
    $k = k + 1$
end for while
for $k = k + 1$ to $k_{\max}$
    let $s_k = r_{k-1} + 1$
    repeat
        use SVT algorithm Compute $[U^{k-1}, \Sigma^{k-1}, V^{k-1}]_{s_k}$
        Let $s_k = s_k + l$
    until $\sigma_{s_k-l}^{k-1} \le \tau$
    let $r_k = \max\{j : \sigma_j^{k-1} > \tau\}$
    let $X^k = \sum_{j=1}^{r_k}(\sigma_j^{k-1} > \tau)u_j^{k-1}v_j^{k-1*}$
    if $\|\mathcal{P}_\Omega(X^k - M)\|_F/\|\mathcal{P}_\Omega M\|_F \le \epsilon_2$ then break
end for k
let $X = X^k$

3 Numerical Results

In our numerical tests, we use Matlab to implement the R-SVT algorithm, and all the results in this paper are obtained by a computer with 2.13 GHz CPU and 2 GB RAM. At first, we generate an $n \times n$ random matrix. Then, we generate a random data array with the length m. Next, we sample the entries of the matrix by the data array. We use the sampled matrix to complete the random matrix we generate.

First, setting the tolerance ϵ is 0.1, we compare the R-SVT with the SVT based on

PROPACK to complete the matrix. In Table 1, the matrices of size 500×500, 1000×1000, 2000×2000 are tested. We compare the computational time and solution accuracy of the classical SVT and our R-SVT. In Table 1, the notations T, iter, RE stand for the computational time, outer iteration number, and relative error, respectively. And in Table 1 we find that both our R-SVT and the classical SVT can achieve the final relative errors of almost the same order with almost the same number of iterations. According to the computational time, we also find that our R-SVT is faster than the SVT based on PROPACK, and the time difference becomes more obvious when the matrix is larger. For example, when the size of matrix is 2000×2000 with the rank of 400, the computational time of SVT is almost five times of that of R-SVT.

Second, we set the relative error as small as to be 10^{-4}. Based on the former algorithm R-SVT, we just make a small modification. We use the R-SVT until the error is 0.1, and then switch to the SVT based on PROPACK until the error is smaller than 10^{-4}. The computational result are shown in Table 2. We compare the results and can draw the similar conclusions as the first algorithm.

Table 1 Comparisons of SVT and R-SVT

		SVT (tol=0.1)			R-SVT (tol=0.1)		
size	rank	T(s)	iter	RE	T(s)	iter	RE
500×500	50	3.133	5	5.87E-02	2.435(k=250)	5	8.63E-02
	100	9.550	7	8.08E-02	3.310(k=250)	7	9.41E-02
	150	21.05	9	7.60E-02	6.923(k=300)	9	9.35E-02
1000×1000	100	23.92	5	5.58E-02	16.73(k=500)	5	8.48E-02
	200	90.34	7	8.58E-02	21.39(k=500)	7	9.94E-02
	300	201.3	9	7.95E-02	43.99(k=600)	9	9.87E-02
2000×2000	200	260.5	6	5.61E-02	182.6(k=1000)	6	8.59E-02
	300	544.2	6	8.76E-02	175.8(k=1000)	6	9.54E-02
	400	1092	7	8.26E-02	208.0(k=1000)	7	9.74E-02

4 Conclusion

In this paper, we consider the randomized SVT for matrix completion problems. When we nearly finish our work, we notice the work in [5]. But here we use a different randomized algorithm, i.e., the algorithm from [11], and we also apply the strategy of switching to the classical SVT in our algorithm R-SVT*. We use the random matrices to test our new algorithm. We can draw the conclusions as follows.

1. The computational time of our randomized-SVT algorithm is less than the classical SVT algorithm. And this advantage becomes more obvious when the rank of the matrix becomes larger. The amazing is that the our R-SVT algorithm works well for the matrix

Table 2 Comparisons of SVT and R-SVT*.

		SVT (tol=1e-4)			R-SVT* (tol=1e-4)		
size	rank	T(s)	iter	RE	T(s)	iter	RE
500×500	50	15.81	54	9.55E-05	12.12(k=250)	55	9.55E-05
	100	45.94	64	9.98E-05	32.56(k=250)	65	9.99E-05
	150	105.4	92	9.59E-05	86.94(k=300)	93	9.59E-05
1000×1000	100	87.59	53	9.18E-05	66.05(k=500)	54	9.18E-05
	200	266.0	64	9.52E-05	187.8(k=500)	65	9.52E-05
	300	947.2	91	9.70E-05	609.7(k=600)	92	9.71E-05
2000×2000	200	702.0	53	9.48E-05	460.8(k=1000)	54	9.49E-05
	300	1337	53	9.42E-05	806.7(k=1000)	54	9.43E-05
	400	2608	63	9.99E-05	1511(k=1000)	64	1.00E-04

whose rank is not very low, and this is a great improvement for matrix completion.

2. When the tolerance is very small, then the computational time of R-SVT will increase, but this can be overcome when we make a switch in R-SVT*. That is, when the tolerance is very small we switch from the R-SVT algorithm to SVT algorithm. Using this strategy in R-SVT*, the advantages of the R-SVT are still kept.

References

[1] Amit Y, Fink M, Srebro N, Ullman S. Uncovering shared structures in multiclass classification[A]. Proceedings of the 24th International Conference on Machine Learning[C]. Providence, RI: ACM, 2007: 17–24.

[2] Argyriou A, Evgeniou T, Pontil M. Multi-task feature learning[J]. Adv. Neural Inform. Proc. Syst., 2007: 41–48.

[3] Argyriou A, Evgeniou T, Pontil M. Convex multi-task feature learning[J]. Machine Learning, 2008, 73(3): 243–272.

[4] Cai Jianfeng, Candes J Ammanuel, Shen Zuowei. A singular value thresholding algorithm for matrix completion[J]. Soc. Indust. Appl. Math., 2010, 20(4): 1956–1982.

[5] Dhanjal Charanpal, Clemencon Stephan, Gaudel Romaric. Online matrix completion through nuclear norm regularisation[EB/OL]. http://arxiv.org/pdf/1401.2451.pdf. hal-00926605, Ver. 1, 9 Jan. 2014.

[6] Fazel M. Matrix rank minimization with applications[D]. Stanford, CA: Stanford University, 2002.

[7] Fazel M, Hindi H, Boyd S P. Log-det heuristic for matrix rank minimization with applications to Hankel and Euclidean distance matrices[J]. Proc. Amer. Control Conf., 2003, 3: 2156–2162.

[8] Halko N, Martinsson P G, Troop J A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions [J]. SIAM Review, 2011, 53(2), 217–288.

[9] Larsen R M. PROPACK-software for large and sparse SVD calculations[OL]. http://sun. stanford.edu/ rmunk/PROPACK/.

[10] Mesbahi M, Papavassilopoulos G P. On the rank minimization problem over a positive semidefinite-linear matrix inequality[J]. IEEE Trans Automat Control, 1997, 42(2): 239–243.

[11] Xiang Hua, Zou Jun. Regularization with randomized SVD for large-scale discrete inverse problems[J/EB]. Inverse Problem, 2013, 29(8): http://iopscience.iop.org/0266-5611/29/8/085008/.

[12] Guo Wei. Singular value decomposition and algorithm of o-symmetric matrix[J]. J. Math., 2009, 29(3): 346–350.

用随机奇异值分解算法求解矩阵恢复问题

许雪敏, 向 华

(武汉大学数学与统计学院, 湖北 武汉 430072)

摘要: 本文研究了大型低秩矩阵恢复问题. 利用随机奇异值分解(RSVD)算法, 对稀疏矩阵做奇异值分解. 该算法与Lanczos方法相比, 在误差精度一致的同时运算时间大大降低, 且该算法对相对低秩矩阵也有效.

关键词: 矩阵恢复; 奇异值阈值; 核范数最小化; 随机奇异值分解

MR(2010)**主题分类号**: 65F30 **中图分类号**: O241.6