Summary and discussion of: "Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares"

MATH5472 report

Chengyu TAO

2021-12-10

1 Introduction

In this report, we will summarize Hastie et al. (2015)'s paper about matrix completion problem. The proposed method is named as **softImpute-ALS**. The main characteristic of **softImpute-ALS** is its fast computation speed compared with the two popular methods **softImpute** (Mazumder et al., 2010) and **MMMF**(maximum-margin matrix factorization) (Srebro et al., 2004). In fact, **softImpute-ALS** borrows ideas from **softImpute** and **MMMF**. It combines the two methods to achieve relatively lower time complexity. We will briefly summary **softImpute** and **MMMF** firstly.

softImpute solves the following problem

$$\min_{\boldsymbol{M}} \ F(\boldsymbol{M}) := \frac{1}{2}\|P_\Omega(\boldsymbol{X}-\boldsymbol{M})\|_F^2 + \lambda\|\boldsymbol{M}\|_*, \tag{1}$$

where $P_\Omega(\cdot)$ is a projector onto Ω and Ω is the positions of known entries of $\boldsymbol{X}$. Considering a Majority-Minimization (MM) framework, a surrogate function $Q(\boldsymbol{M}|\tilde{\boldsymbol{M}}) = \frac{1}{2}\left\|P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\tilde{\boldsymbol{M}}) - \boldsymbol{M}\right\|_F^2 + \lambda\|\boldsymbol{M}\|_*$ can be constructed such that $Q(\tilde{\boldsymbol{M}}|\tilde{\boldsymbol{M}}) = F(\tilde{\boldsymbol{M}})$ and $Q(\boldsymbol{M}|\tilde{\boldsymbol{M}}) \geqslant F(\boldsymbol{M})$. The second property can be derived using the identity $\boldsymbol{M} = P_\Omega(\boldsymbol{M}) + P_\Omega^\perp(\boldsymbol{M})$. Note that, Mazumder et al. (2010) did not explicitly interpret it by the MM framework. However, the **Lemma 2** (Mazumder et al., 2010) is indeed consistent with the MM method.

Therefore, the problem (1) can be replaced by solving the following problem iteratively

$$\boldsymbol{M}^{k+1} = \operatorname*{argmin}_{\boldsymbol{M}} \ Q(\boldsymbol{M}|\boldsymbol{M}^k) = \frac{1}{2}\left\|P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\boldsymbol{M}^k) - \boldsymbol{M}\right\|_F^2 + \lambda\|\boldsymbol{M}\|_*. \tag{2}$$

The above problem has closed-form solution based on singular value decomposition (SVD). Denote $\boldsymbol{X}^* = P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\boldsymbol{M}^k)$ and $X^* = UDV^T$ as its SVD, then, $\boldsymbol{M}^{k+1} = U\mathcal{S}_\lambda(D)V^T$, where $\mathcal{S}_\lambda(D)$ operates on each entry $D_{i,i}$ such that $\mathcal{S}_\lambda(D)_{i,i} = \max(D_{i,i}-\lambda, 0)$ and 0 elsewhere.

The main limitation of **softImpute** is the high time complexity of SVD when $\boldsymbol{X}^*$ has very large dimension. Generally, if $\boldsymbol{X}^* \in R^{m\times n}$, the time complexity of SVD of $\boldsymbol{X}^*$ is

$O(mn\min(m,n))$. Though the special structure of $\boldsymbol{X}^*$ and warm start of SVD can be explored, the computational load in high-dimension cases is still burdensome.

MMMF is an alternative method. Using the following inequality

$$\|\boldsymbol{M}\|_* = \min_{\boldsymbol{A},\boldsymbol{B}:\boldsymbol{M}=\boldsymbol{A}\boldsymbol{B}^T} \frac{1}{2}\left(\|\boldsymbol{A}\|_F^2 + \|\boldsymbol{B}\|_F^2\right). \tag{3}$$

Srebro et al. (2004)'s goal is to solve the following problem

$$\min_{\boldsymbol{A},\boldsymbol{B}} \; G(\boldsymbol{A},\boldsymbol{B}) := \frac{1}{2}\left\|P_\Omega(\boldsymbol{X}-\boldsymbol{A}\boldsymbol{B}^T)\right\|_F^2 + \frac{\lambda}{2}\left(\|\boldsymbol{A}\|_F^2 + \|\boldsymbol{B}\|_F^2\right), \tag{4}$$

where $\boldsymbol{M} \in R^{m\times n}$, $\boldsymbol{A} \in R^{m\times r}$, and $\boldsymbol{B} \in R^{n\times r}$. The two problems (3) and (1) produce equivalent results under certain conditions, due to the space limitation, for more details, we refer to the raw paper (Hastie et al., 2015). This problem is biconvex and can be solved by alternating least square method. When $\boldsymbol{A}$ is fixed, the optimization of $\boldsymbol{B}$ consists of a series of subproblems (r times) that each row of $\boldsymbol{B}$ can be obtained by an independent vector ridge regression. Symmetrically, optimizing $\boldsymbol{B}$ follows the same procedure. The independence is caused by the projector $P_\Omega(\cdot)$ that for each column the locations of known row entries are different such that the regression matrices are different.

The independence of vector ridge regressions will bring relatively higher computational complexity with r times larger than that of only one matrix ridge regression with an unchanged regression matrix. On the other side, if we factorize $\boldsymbol{M} = \boldsymbol{A}\boldsymbol{B}^T$ in Eq. (2) and use Eq. (3), we can solve the following problem in the framework of **softImpute** iteratively

$$\begin{aligned}\boldsymbol{A}^{k+1}, \boldsymbol{B}^{k+1} =& \operatorname*{argmin}_{\boldsymbol{A},\boldsymbol{B}} \; L(\boldsymbol{A},\boldsymbol{B}) = \\ & \frac{1}{2}\left\|P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\boldsymbol{A}^k(\boldsymbol{B}^k)^T) - \boldsymbol{A}\boldsymbol{B}^T\right\|_F^2 + \frac{\lambda}{2}\left(\|\boldsymbol{A}\|_F^2 + \|\boldsymbol{B}\|_F^2\right).\end{aligned} \tag{5}$$

We can find that the above problem (5) has the same form as the problem (4). Interestingly, $P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\boldsymbol{A}^k(\boldsymbol{B}^k)^T)$ do not contain any missing entries, which implies that using alternating least square method to solve (5) only requires matrix ridge regressions. From this perspective, it shows potential faster computation speed than that of **MMMF**. Moreover, SVD is also unnecessary in the above problem (5), thus, it is also expected to be better than **softImpute** in terms of time complexity. This problem (5) is the central step of **softImpute-ALS**. Next, we will show more details about **softImpute-ALS**.

2 Summary of softImpute-ALS

We will present the main procedure of **softImpute-ALS** at first, which is followed by some further comments.

SoftImpute-ALS is concluded in Algorithm 1. Actually, $\boldsymbol{A}$ and $\boldsymbol{B}$ are orthogonal factorized by $\boldsymbol{U}\boldsymbol{D}$ and $\boldsymbol{V}\boldsymbol{D}$ respectively. Then, when using alternating least square method to solve the problem (5), it consists of two matrix ridge regressions as shown in Algorithm 1. In each iteration, we have the following relationship

$$\begin{aligned} W(\bar{\boldsymbol{A}},\bar{\boldsymbol{B}}) \leqslant L(\boldsymbol{A}^k,\tilde{\boldsymbol{B}}) \leqslant L(\boldsymbol{A}^k,\boldsymbol{B}^k) = W(\boldsymbol{A}^k,\boldsymbol{B}^k) \\ W(\boldsymbol{A}^{k+1},\boldsymbol{B}^{k+1}) \leqslant L(\tilde{\boldsymbol{A}},\bar{\boldsymbol{B}}) \leqslant L(\bar{\boldsymbol{A}},\bar{\boldsymbol{B}}) = W(\bar{\boldsymbol{A}},\bar{\boldsymbol{B}}),\end{aligned} \tag{6}$$

where $W(\boldsymbol{A},\boldsymbol{B}) = \frac{1}{2}\left\|P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\boldsymbol{A}^k(\boldsymbol{B}^k)^T) - \boldsymbol{A}\boldsymbol{B}^T\right\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{A}\boldsymbol{B}^T\|_*$. The proof relies on Eq. (3), we do not give detailed proof here. Therefore, the orthogonal factorization [1] can be regarded as modifying the problem (5) that it directly deals with the following problem

$$\begin{aligned}\boldsymbol{A}^{k+1},\boldsymbol{B}^{k+1} =& \underset{\boldsymbol{A},\boldsymbol{B}}{\operatorname{argmin}} \; W(\boldsymbol{A},\boldsymbol{B}) = \\ & \frac{1}{2}\left\|P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\boldsymbol{A}^k(\boldsymbol{B}^k)^T) - \boldsymbol{A}\boldsymbol{B}^T\right\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{A}\boldsymbol{B}^T\|_*.\end{aligned} \tag{7}$$

Algorithm 1 **softImpute-ALS**

Input: matrix $\boldsymbol{X}$, stop threshold ϵ

Initialize: $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}$, where $\boldsymbol{U} \in R^{m\times r}$ is random orthogonal matrix and $\boldsymbol{D} = \boldsymbol{I}_r$. $\boldsymbol{B} = \boldsymbol{V}\boldsymbol{D}$, where $\boldsymbol{V} = \boldsymbol{0}$. Note that, any warm start can be used for initialization.

Output: $\boldsymbol{A}$,$\boldsymbol{B}$,$\boldsymbol{U}$,$\boldsymbol{D}$,$\boldsymbol{V}$.

repeat

Set $\boldsymbol{A}_{old} = \boldsymbol{A}$ and $\boldsymbol{B}_{old} = \boldsymbol{B}$.

Set $\boldsymbol{X}^* = P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\boldsymbol{A}\boldsymbol{B}^T)$, and then calculate $\tilde{\boldsymbol{B}}^T = (\boldsymbol{D}^2 + \lambda\boldsymbol{I}_r)^{-1}\boldsymbol{D}\boldsymbol{U}^T\boldsymbol{X}^*$.

Find the SVD $\tilde{\boldsymbol{B}}\boldsymbol{D} = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{D}}^2\tilde{\boldsymbol{V}}^T$.

Update $\boldsymbol{V} \leftarrow \tilde{\boldsymbol{U}}$, $\boldsymbol{D} \leftarrow \tilde{\boldsymbol{D}}$, $\boldsymbol{U} \leftarrow \boldsymbol{U}\tilde{\boldsymbol{V}}$, $\bar{\boldsymbol{A}} \leftarrow \boldsymbol{U}\boldsymbol{D}$, and $\bar{\boldsymbol{B}} \leftarrow \boldsymbol{V}\boldsymbol{D}$.

Set $\boldsymbol{X}^* = P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\bar{\boldsymbol{A}}\bar{\boldsymbol{B}}^T)$, and then calculate $\tilde{\boldsymbol{A}}^T = (\boldsymbol{D}^2 + \lambda\boldsymbol{I}_r)^{-1}\boldsymbol{D}\boldsymbol{V}^T(\boldsymbol{X}^*)^T$.

Find the SVD $\tilde{\boldsymbol{A}}\boldsymbol{D} = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{D}}^2\tilde{\boldsymbol{V}}^T$.

Update $\boldsymbol{U} \leftarrow \tilde{\boldsymbol{U}}$, $\boldsymbol{D} \leftarrow \tilde{\boldsymbol{D}}$, $\boldsymbol{V} \leftarrow \boldsymbol{V}\tilde{\boldsymbol{V}}$, $\boldsymbol{A} \leftarrow \boldsymbol{U}\boldsymbol{D}$, and $\boldsymbol{B} \leftarrow \boldsymbol{V}\boldsymbol{D}$.

until $\dfrac{\left\|\boldsymbol{A}\boldsymbol{B}^T - \boldsymbol{A}_{old}\boldsymbol{B}_{old}^T\right\|_F^2}{\left\|\boldsymbol{A}_{old}\boldsymbol{B}_{old}^T\right\|_F^2} \leqslant \epsilon$

Similarly, we provide the non-orthogonal version of **softImpute-ALS**, we name it as **softImpute-ALS-NSBS**(non-subspace iteration). This algorithm is illustrated in Algorithm 2.

Algorithm 2 **softImpute-ALS-NSBS**

Input: matrix $\boldsymbol{X}$, stop threshold ϵ

Initialize: $\boldsymbol{A} = \boldsymbol{A}_0$ and $\boldsymbol{B} = \boldsymbol{0}$. Note that, any warm start can be used for initialization.

Output: $\boldsymbol{A}$,$\boldsymbol{B}$

repeat

Set $\boldsymbol{A}_{old} = \boldsymbol{A}$ and $\boldsymbol{B}_{old} = \boldsymbol{B}$.

Set $\boldsymbol{X}^* = P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\boldsymbol{A}\boldsymbol{B}^T)$, and then update $\boldsymbol{B} \leftarrow (\boldsymbol{X}^*)^T\boldsymbol{A}(\boldsymbol{D}^2 + \lambda\boldsymbol{I}_r)^{-1}$.

Set $\boldsymbol{X}^* = P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\boldsymbol{A}\boldsymbol{B}^T)$, and then update $\boldsymbol{A} \leftarrow (\boldsymbol{X}^*)\boldsymbol{B}(\boldsymbol{D}^2 + \lambda\boldsymbol{I}_r)^{-1}$.

until $\dfrac{\left\|\boldsymbol{A}\boldsymbol{B}^T - \boldsymbol{A}_{old}\boldsymbol{B}_{old}^T\right\|_F^2}{\left\|\boldsymbol{A}_{old}\boldsymbol{B}_{old}^T\right\|_F^2} \leqslant \epsilon$

[1] In fact, the purpose of the above orthogonal factorization is not very clear. Panagoda et al. (2021) claim that it will produce faster convergence rate if $P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\boldsymbol{A}^k(\boldsymbol{B}^k)^T)$ is low rank, however, it is a kind of structure *sparse plus low rank*.

Another benefit from orthogonal factorization is that the output of **softImpute-ALS** can be used as warm start for **softImpute**, while **softImpute-ALS-NSBS** needs an additional SVD to refactorize. For large-scale problem, the refactorization is time-consuming. In addition, we should note that **softImpute-ALS** add a rank-r constraint, which plays a trade-off between accuracy and efficiency.

SoftImpute-ALS converges at the rate $O(\frac{1}{K})$. For detailed theoretic analysis of the convergence behavior of **SoftImpute-ALS**, one can refer the raw paper (Hastie et al., 2015). We are more concerned about the time-complexity. The total costs of an iteration of **SoftImpute-ALS** and **SoftImpute-ALS-NSBS** are $O(2r|\Omega|+mr^2+3nr^2+r^3)$. The cost of a rank-$r'$ SVD of **softImpute** with current rank $\tilde{r}$ is $O(r'|\Omega|+mr'\tilde{r}+nr'\tilde{r})$. At the first several steps, r' and $\tilde{r}$ are very large and there is no good warm start, so **softImpute** costs much more time than **SoftImpute-ALS**, while it may be comparable with **SoftImpute-ALS** after certain number of iterations. The cost of an iteration of **MMMF** is $O(2|\Omega|r^2+mr^3+nr^3)$, which is $O(r)$ times that of **SoftImpute-ALS** due to the independent vector ridge regressions.

3 Results and Discussion

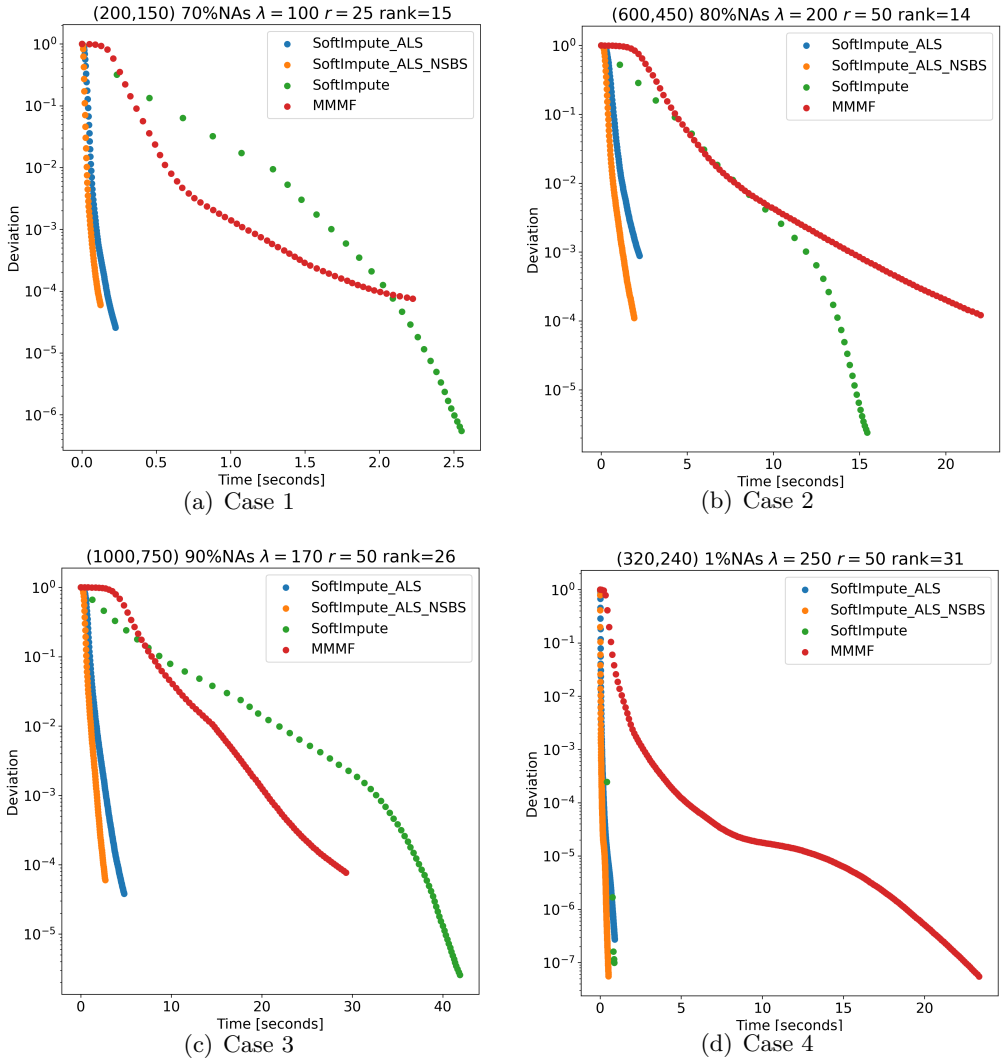
In this section, we show the performance of **SoftImpute-ALS**,**SoftImpute-ALS-NSBS**, **MMMF** and **softImpute** on simulation data. All results and codes are available on https://github.com/ctaoaa/SoftImpute_ALS .

3.1 Timing experiments

We generated $\boldsymbol{X}=\boldsymbol{A}\boldsymbol{B}^T+\boldsymbol{E}$, where each entry of $\boldsymbol{A},\boldsymbol{B}$ follows standard normal distribution. The entries $\boldsymbol{E}$ follows $e_{ij}\sim\mathcal{N}(0,\sigma^2)$. We set $\sigma=5$. To investigate the convergence rate, we denote the limiting solution of **SoftImpute-ALS** as $\boldsymbol{A}^*(\boldsymbol{B}^*)^T$ (we let the algorithm run by sufficient large number of iterations). Then, for each algorithm, we calculate the deviation $\|\boldsymbol{A}^k(\boldsymbol{B}^k)^T-\boldsymbol{A}^*(\boldsymbol{B}^*)^T\|_F^2/\|\boldsymbol{A}^*(\boldsymbol{B}^*)^T\|_F^2$. We will plot the deviation to time for four time experiments. These algorithms are implemented by Python. We follows the suggestion of Hastie et al. (2015) that the closed-form solution based on SVD of problem (2) in **softImpute** is implemented by alternating subspace iterations similar to the Algorithm 1 . We set the dimension of subspace as two times as the operating rank for other algorithms [2].

The Figure 1 shows the results of timing experiments. The shape of $\boldsymbol{X}$ are (200,150), (600,450), (1000,750), and (320,240) with true ranks 50, 80, 80, and 50 respectively. The proportion of missing values are 70%, 80%, 90%, and 1% respectively. The tuning parameter λ is manually set to make estimated rank smaller than the true rank. We use the same criteria $\epsilon=10^{-7}$ for all algorithms as described in Algorithm 1 in all experiments except that $\epsilon=10^{-10}$ for the last experiment. Each subfigure is labeled by the shape of $\boldsymbol{X}$, the proportion of missing values, the tuning parameter λ, the operating rank r for factorization, and estimated rank respectively.

[2]In order to make a fair comparison, Hastie et al. (2015) use alternating subspace iterations in their R package, while it is hard to say the best choice of the dimension of subspace.



(a) Case 1 (b) Case 2 (c) Case 3 (d) Case 4

Figure 1: Four timing experiments. Each subfigure is labeled by the shape of $\boldsymbol{X}$, the proportion of missing values, the tuning parameter λ, the operating rank r for factorization, and estimated rank respectively.

From Figure 1, we can know that **SoftImpute-ALS**,**SoftImpute-ALS-NSBS** have the fastest computation speed, while **MMMF** and **softImpute** are slower. The reasons are: (1) The cost of each iteration of **MMMF** is $O(r)$ times that of **SoftImpute-ALS**; (2) The cost of SVD of **softImpute** in the first several steps is much larger. The tails of the curves of **softImpute** also show that the cost of SVD will be much smaller due to the use of warm start using the previous solutions. Figure 1d is the case that only 1% entries are missing. In this case, **softImpute** performs pretty well that a good solution will be obtained by only few iterations. The interpretation for this phenomenon is that when the proportion of missing entries are very small, $P_\Omega(\boldsymbol{X}) + P_\Omega^\perp(\boldsymbol{A}^k(\boldsymbol{B}^k)^T)$ will change very small, the SVD can directly obtain a good solution. In all experiments, we can find that

SoftImpute-ALS and **SoftImpute-ALS-NSBS** have similar performance.

Next, we will show that using **SoftImpute-ALS** as warm start of **softImpute** performs better than only using **SoftImpute-ALS**. To validate this claim, we simulate another experiment. The shape of $\boldsymbol{X}$ is (2000,1500) with true rank 120 and 50% entries missing. We set the operating rank $r = 50$. We run **SoftImpute-ALS** by 150 iterations at first, and then the current solution will be used as warm start of **SoftImpute-ALS**. **SoftImpute-ALS** will stop after 200 iterations. **SoftImpute-ALS** will stop after 350 iterations. Since we do not know the limiting objective or solutions, we set the minimal objective f_{min} in the running process as the baseline, and then we define relative objective as $|f^k - f_{min}|/|f_{min}|$.

The following Figure 2 plots the result of relative objective to time, which shows the faster convergence rate of **softImpute** using warm start. Generally speaking, one iteration of **softImpute** can have more decline of current objective that that of **SoftImpute-ALS**, since SVD contains inner-loop alternating subspace methods similar to the explicit factorization in **SoftImpute-ALS**.

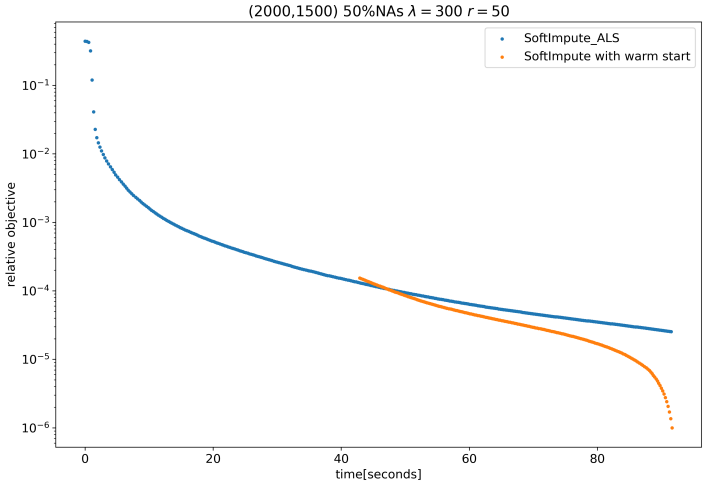


Figure 2: Warm start result.

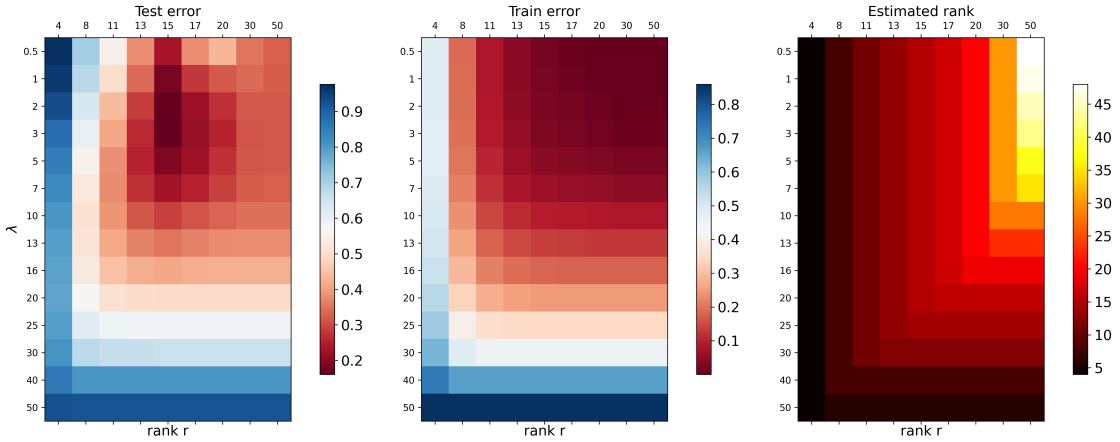
3.2 Train and test error experiments

In the above subsection, we have compared the computational speed. In this subsection, we want to show the train and test errors of comparison algorithms. We still use the model mentioned in the above experiments. The SNR is defined as $SNR = \sqrt{\text{var}(\boldsymbol{AB}^T)/\text{var}(\boldsymbol{E})}$. Therefore, when $\sigma = 1$, we have $SNR = 1$, and $SNR = 10$ if $\sigma = 0.1$. The shape of $\boldsymbol{X}$ is (100,100) with true rank 15. We test two experiments, there are 60% missing entries and $SNR = 1$ in the first case, while the second case we set 50% missing entries and $SNR = 10$. We define train error as $\|P_\Omega(\boldsymbol{X} - \boldsymbol{X}_{est})\|_F^2/\|P_\Omega(\boldsymbol{X})\|_F^2$, where $\boldsymbol{X}_{est}$ is the obtained solution under specific tuning parameters. The test error is defined as $\|P_\Omega^\perp(\boldsymbol{AB}^T - \boldsymbol{A}_{est}\boldsymbol{B}_{est}^T)\|_F^2/\|P_\Omega^\perp(\boldsymbol{AB}^T)\|_F^2$. **SoftImpute-ALS** and **softImpute** are compared. For **softImpute-ALS**, there are two parameters r and λ, while only λ for **softImpute**. The results are shown as Figure 3, Table 1, and Figure 4. For **softImpute**, we also

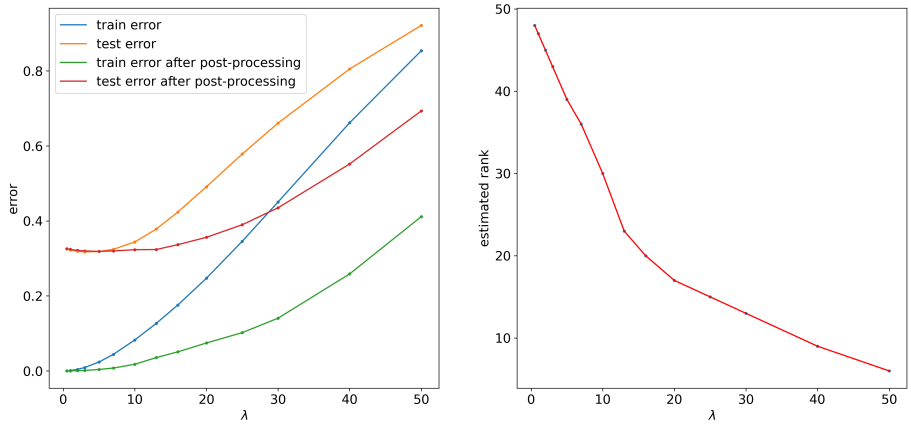
Table 1: Minimal test error of **softImpute-ALS** and **softImpute-SVD**.

	softImpute-ALS		**softImpute**	
	SNR=1	SNR=10	SNR=1	SNR=10
minimal test error	0.1609	0.0010	0.3189	0.0149

calculate the error after post-processing (Mazumder et al., 2010, Sec. 7).



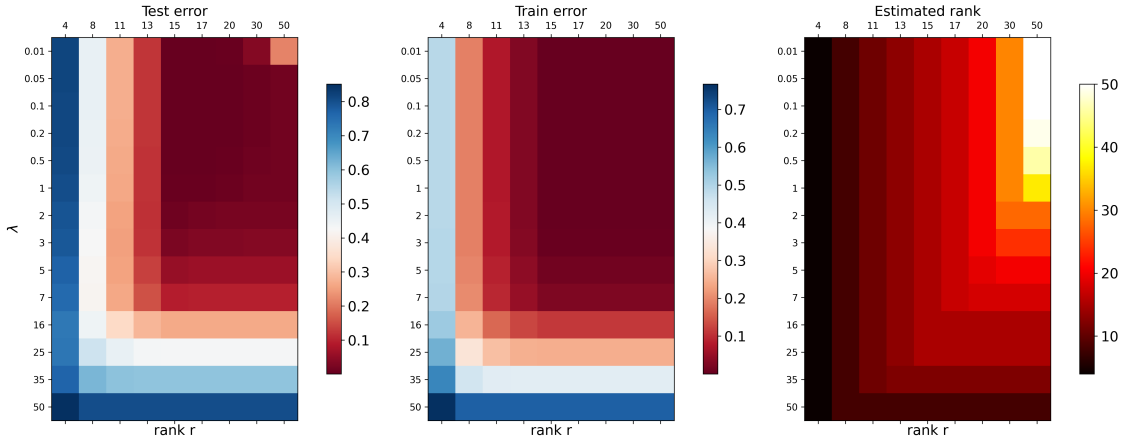
(a) Train and test error and estimated rank of **softImpute-ALS** when SNR=1.



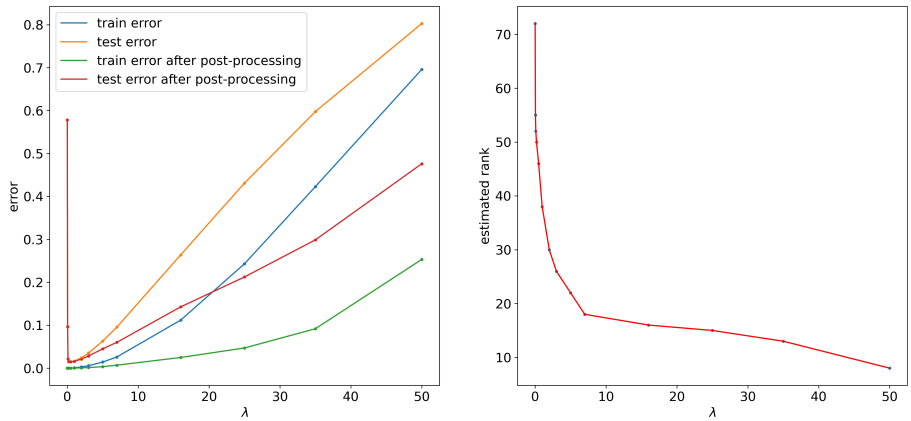
(b) Train and test error and estimated rank of **softImpute-SVD** when SNR=1.

Figure 3: The first case with 60% missing entries and $SNR = 1$.

The results suggest that **softImpute-ALS** is better than **softImpute** in terms of train error and test error. The reason may be the fact that **softImpute-ALS** contains another tuning parameter r, which can provide some prior information about our model. Indeed, we can observe from the test error colormap in Figure 3a that test error reach minimum when $r = 15$, which is the true rank. Table 1 illustrates that the additional regularization of r leads to much lower test error both in SNR=10 and SNR=1. Though **softImpute-ALS** works better, the selection of two-family tuning parameters is more difficult in practice.



(a) Train and test error and estimated rank of **softImpute-ALS** when SNR=10.



(b) Train and test error and estimated rank of **softImpute-SVD** when SNR=10.

Figure 4: The second case with 50% missing entries and $SNR = 10$.

4 Conclusion

This paper propose **softImpute-ALS** for matrix completion problem. This method combines **softImpute** and **MMMF**, which has faster computation speed. We validated this point by simulation experiments. Moreover, we found that using **softImpute-ALS** as warm start for **softImpute** can achieve the better performance than separate ones. We also calculated the train error and test error of **softImpute-ALS** and **softImpute**. The results show that **softImpute-ALS** has lower minimal test error due to the additional regularization of the operating rank r.

References

Hastie, T., R. Mazumder, J. D. Lee, and R. Zadeh (2015). Matrix completion and low-rank svd via fast alternating least squares. *The Journal of Machine Learning Research 16*(1), 3367–3402.

Mazumder, R., T. Hastie, and R. Tibshirani (2010). Spectral regularization algorithms for learning large incomplete matrices. *The Journal of Machine Learning Research 11*, 2287–2322.

Panagoda, M., T. Berry, and H. Antil (2021). Convergence analysis of the rank-restricted soft svd algorithm. *arXiv preprint arXiv:2104.01473*.

Srebro, N., J. D. Rennie, and T. S. Jaakkola (2004). Maximum-margin matrix factorization. In *NIPS*, Volume 17, pp. 1329–1336. Citeseer.