HHS Public Access

Author manuscript

J Mach Learn Res. Author manuscript; available in PMC 2019 May 22.

Published in final edited form as:

J Mach Learn Res. 2015 ; 16: 3367–3402.

Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares

Trevor Hastie,
Department of Statistics, Stanford University, CA 94305, USA

Rahul Mazumder,
Department of Statistics, Columbia University, New York, NY 10027, USA

Jason D. Lee, and
Institute for Computational and Mathematical Engineering, Stanford University, CA 94305, USA

Reza Zadeh
Databricks, 2030 Addison Street, Suite 610, Berkeley, CA 94704, USA

Abstract

The matrix-completion problem has attracted a lot of attention, largely as a result of the celebrated Netflix competition. Two popular approaches for solving the problem are nuclear-norm-regularized matrix approximation (Candès and Tao, 2009; Mazumder et al., 2010), and maximum-margin matrix factorization (Srebro et al., 2005). These two procedures are in some cases solving equivalent problems, but with quite different algorithms. In this article we bring the two approaches together, leading to an efficient algorithm for large matrix factorization and completion that outperforms both of these. We develop a software package softlmpute in R for implementing our approaches, and a distributed version for very large matrices using the Spark cluster programming environment

Keywords

matrix completion; alternating least squares; svd; nuclear norm

1. Introduction

We have an $m \times n$ matrix X with observed entries indexed by the set Ω; i.e. $\Omega = \{(i, j) : X_{ij}$ is observed$\}$. Following Candès and Tao (2009) we define the projection $P_\Omega(X)$ to be the $m \times n$ matrix with the observed elements of X preserved, and the missing entries replaced with 0. Likewise $P_\Omega^\perp$ projects onto the complement of the set Ω.

Inspired by Candès and Tao (2009), Mazumder et al. (2010) posed the following convex-optimization problem for completing X:

$$\underset{M}{\text{minimize}}\ H(M) := \frac{1}{2}\|P_{\Omega}(X - M)\|_F^2 + \lambda\|M\|_*, \quad (1)$$

where the *nuclear norm* $\|M\|_*$ is the sum of the singular values of M (a convex relaxation of the rank). They developed a simple iterative algorithm for solving Problem (1), with the following two steps iterated till convergence:

1. Replace the missing entries in X with the corresponding entries from the current estimate $\widehat{M}$:

$$\hat{X} \leftarrow P_{\Omega}(X) + P_{\Omega}^{\perp}(\widehat{M}); \quad (2)$$

2. Update $\widehat{M}$ by computing the soft-thresholded SVD of $\hat{X}$:

$$\hat{X} = UDV^T \quad (3)$$

$$\widehat{M} \leftarrow U\mathcal{S}_{\lambda}(D)V^T, \quad (4)$$

where the soft-thresholding operator S_{λ} operates element-wise on the diagonal matrix D, and replaces D_{ii} with $(\mathrm{D}_{ii} - \lambda)_+$. With large λ many of the diagonal elements will be set to zero, leading to a low-rank solution for Problem (1).

For large matrices, step (3) could be a problematic bottleneck, since we need to compute the SVD of the filled matrix $\hat{X}$. In fact, for the Netflix problem $(m, n) \approx (400K, 20K)$, which requires storage of 8×10^9 floating-point numbers (32Gb in single precision), which in itself could pose a problem. However, since only about 1% of the entries are observed (for the Netflix dataset), sparse-matrix representations can be used.

Mazumder et al. (2010) use two tricks to avoid these computational nightmares:

1. Anticipating a low-rank solution, they compute a reduced-rank SVD in step (3); if the smallest of the computed singular values is less than λ, this gives the desired solution. A reduced-rank SVD can be computed by using an iterative Lanczos-style method as implemented in PROPACK (Larsen, 2004), or by other alternating-subspace methods (Golub and Van Loan, 2012).
2. They rewrite $\hat{X}$ in (2) as

$$\hat{X} = \left[P_{\Omega}(X) - P_{\Omega}(\widehat{M})\right] + \widehat{M}; \quad (5)$$

The first piece is as sparse as X, and hence inexpensive to store and compute. The second piece is low rank, and also inexpensive to store. Furthermore, the iterative methods mentioned in step (1) require left and right multiplications of $\hat{X}$ by *skinny* matrices, which can exploit this special structure.

This softlmpute algorithm works very well, and although an SVD needs to be computed each time step (3) is evaluated, this step can use the previous solution as a warm start. As one gets closer to the solution, the warm starts tend to be better, and so the final iterations tend to be faster.

Mazumder et al. (2010) also considered a path of such solutions, with decreasing values of λ. As λ decreases, the rank of the solutions tend to increase, and at each λ_ℓ the iterative algorithms can use the solution $\hat{X}_{\lambda_{\ell-1}}$ (with $\lambda_{\ell-1} > \lambda_\ell$) as warm starts, padded with some additional dimensions.

Rennie and Srebro (2005) consider a different approach. They impose a rank constraint, and consider the problem

$$\underset{A,B}{\text{minimize}} \quad F(A,B) := \frac{1}{2}\|P_\Omega(X - AB^T)\|_F^2 + \frac{\lambda}{2}\left(\|A\|_F^2 + \|B\|_F^2\right), \quad (6)$$

where A is $m \times r$ and B is $n \times r$. This so-called maximum-margin matrix factorization (MMMF) criterion[1] is not convex in A and B, but it is bi-convex — for fixed B the function $F(A, B)$ is convex in A, and for fixed A the function $F(A, B)$ is convex in B. Alternating minimization algorithms (ALS) are often used to minimize Problem (6). Consider A fixed, and we wish to solve Problem (6) for B. It is easy to see that this problem decouples into n separate ridge regressions, with each column X_j of X as a response, and the r-columns of A as predictors. Since some of the elements of X_j are missing, and hence ignored, the corresponding rows of A are deleted for the jth regression. So these are really *separate* ridge regressions, in that the regression matrices are all different (even though they all derive from A). By symmetry, with B fixed, solving for A amounts to m separate ridge regressions.

There is a remarkable fact that ties the solutions to Problems (6) and (1) (Mazumder et al., 2010, for example). If the solution to Problem (1) has rank $q \le r$, then it provides a solution to Problem (6). That solution is

$$\begin{aligned} \hat{A} &= U_r \mathcal{S}_\lambda(D_r)^{\frac{1}{2}} \\ \hat{B} &= V_r \mathcal{S}_\lambda(D_r)^{\frac{1}{2}}, \end{aligned} \quad (7)$$

[1] Actually MMMF also refers to the margin-based loss function that they used, but we will nevertheless use this acronym.

where U_r, for example, represents the sub-matrix formed by the first r columns of U, and likewise D_r is the top $r \times r$ diagonal block of D. Note that for any solution to Problem (6), multiplying $\hat{A}$ and $\hat{B}$ on the right by an orthonormal $r \times r$ matrix R would be an equivalent solution. Likewise, any solution to Problem (6) with rank $r \geq q$ gives a solution to Problem (1).

In this paper we propose a new algorithm that profitably draws on ideas used both in softlmpute and MMMF. Consider the two steps (3) and (4). We can alternatively solve the optimization problem

$$\underset{A,B}{\text{minimize}} \ \frac{1}{2}\|\hat{X} - AB^T\|_F^2 + \frac{\lambda}{2}\left(\|A\|_F^2 + \|B\|_B^2\right), \quad (8)$$

and as long as we use enough columns in A and B, we will have $\widehat{M} = \hat{A}\hat{B}^T$. There are several important advantages to this approach:

1. Since $\hat{X}$ is fully observed, the (ridge) regression operator is the same for each column, and so is computed just once. This reduces the computation of an update of A or B over ALS by a factor of r.
2. By orthogonalizing the r-column matrices A or B at each iteration, the regressions are simply matrix multiplies, very similar to those used in the alternating subspace algorithms for computing the SVD.
3. This quadratic regularization amounts to shrinking the higher-order components more than the lower-order components, and this tends to offer a convergence advantage over the previous approach (compute the SVD, then soft-threshold).
4. Just like before, these operations can make use of the *sparse plus low-rank* property of $\hat{X}$.

As an important additional modification, we replace $\hat{X}$ at each step using the most recently computed $\hat{A}$ or $\hat{B}$. All combined, this hybrid algorithm tends to be faster than either approach on their own; see the simulation results in Section 6.1

For the remainder of the paper, we present this softImpute-ALS algorithm in more detail, and show that it convergences to the solution to Problem (1) for r sufficiently large. We demonstrate its superior performance on simulated and real examples, including the Netflix data. We briefly highlight two publicly available software implementations, and describe a simple approach to centering and scaling of both the rows and columns of the (incomplete) matrix.

2. Rank-restricted Soft SVD

In this section we consider a complete matrix X, and develop a new algorithm for finding a rank-restricted SVD. In the next section we will adapt this approach to the matrix-completion problem. We first give two theorems that are likely known to experts; the proofs are very short, so we provide them here for convenience.

Theorem 1 *Let $X_{m\times n}$ be a matrix (fully observed), and let $0 < r \le \min(m, n)$. Consider the optimization problem*

$$\underset{Z:\,\mathrm{rank}(Z)\le r}{\text{minimize}} \; F_\lambda(Z) := \frac{1}{2}\|X - Z\|_F^2 + \lambda\|Z\|_*. \quad (9)$$

A solution is given by

$$\hat{Z} = U_r \mathcal{S}_\lambda(D_r) V_r^T, \quad (10)$$

where the rank-r SVD of X is $U_r D_r V_r^T$ and $S_\lambda(D_r) = diag[(\sigma_1 - \lambda)_+, \ldots, (\sigma_r - \lambda)_+]$.

Proof We will show that, for any Z the following inequality holds:

$$F_\lambda(Z) \ge f_\lambda(\boldsymbol{\sigma}(Z)) := \frac{1}{2}\|\boldsymbol{\sigma}(X) - \boldsymbol{\sigma}(Z)\|_2^2 + \lambda\sum_i \sigma_i(Z), \quad (11)$$

where, $f_\lambda(\boldsymbol{\sigma}(Z))$ is a function of the singular values of Z and $\boldsymbol{\sigma}(X)$ denotes the vector of singular values of X, such that $\sigma_i(X) \ge \sigma_{i+1}(X)$ for all $i = 1, \ldots, \min\{m, n\}$.

To show inequality (11) it suffices to show that:

$$\frac{1}{2}\|X - Z\|_F^2 \ge \frac{1}{2}\|\boldsymbol{\sigma}(X) - \boldsymbol{\sigma}(Z)\|_2^2,$$

which follows as an immediate consequence of the well-known Von-Neumann trace inequality (Mirsky, 1975; Stewart and Sun, 1990):

$$\mathrm{Tr}\left(X^T Z\right) := \left\langle X, Z \right\rangle \le \sum_{i=1}^{\min\{m,n\}} \sigma_i(X)\sigma_i(Y),$$

that provides an upper bound to the trace of the product of two matrices in terms of the inner product of their singular values.

Observing that

$$\mathrm{rank}(Z) \le r \;\Leftrightarrow\; \|\boldsymbol{\sigma}(Z)\|_0 \le r,$$

we have established:

$$\begin{aligned}&\min_{Z:\,\mathrm{rank}\,(Z)\le r}\left(\frac{1}{2}\|X-Z\|_F^2+\lambda\|Z\|_*\right)\\ \ge&\min_{\boldsymbol{\sigma}(Z):\,\|\boldsymbol{\sigma}(Z)\|_0\le r}\left(\frac{1}{2}\|\boldsymbol{\sigma}(X)-\boldsymbol{\sigma}(Z)\|_2^2+\lambda\sum_i\sigma_i(Z)\right)\end{aligned}\quad(12)$$

Observe that the optimization problem in the right hand side of (12) is a separable vector optimization problem. We claim that the optimum solutions of the two problems appearing in (12) are in fact equal. To see this, let

$$\widehat{\boldsymbol{\sigma}(Z)}=\operatorname*{argmin}_{\boldsymbol{\sigma}(Z):\,\|\boldsymbol{\sigma}(Z)\|_0\le r}\left(\frac{1}{2}\|\boldsymbol{\sigma}(X)-\boldsymbol{\sigma}(Z)\|_2^2+\lambda\sum_i\sigma_i(Z)\right).$$

If the SVD of X is given by UDV^T, then the choice $\hat{Z}=U\,\mathrm{diag}\,(\widehat{\boldsymbol{\sigma}(Z)})V^T$ satisfies

$$\mathrm{rank}\,(\hat{Z})\le r\quad\text{and}\quad F_\lambda(\hat{Z})=f_\lambda(\widehat{\sigma(Z)})$$

This shows that:

$$\begin{aligned}&\min_{Z:\,\mathrm{rank}\,(Z)\le r}\left(\frac{1}{2}\|X-Z\|_F^2+\lambda\|Z\|_*\right)\\ =&\min_{\boldsymbol{\sigma}(Z):\,\|\boldsymbol{\sigma}(Z)\|_0\le r}\left(\frac{1}{2}\|\boldsymbol{\sigma}(X)-\boldsymbol{\sigma}(Z)\|_2^2+\lambda\sum_i\sigma_i(Z)\right)\end{aligned}\quad(13)$$

and thus concludes the proof of the theorem. ■

This generalizes a similar result where there is no rank restriction, and the problem is convex in Z. For $r<\min(m,n)$, Problem (9) is not convex in Z, but the solution can be characterized in terms of the SVD of X.

The second theorem relates this problem to the corresponding matrix factorization problem

Theorem 2 *Let $X_{m\times n}$ be a matrix (fully observed), and let* $0<r\le\min(m,n)$. *Consider the optimization problem*

$$\min_{A_{m\times r},B_{n\times r}}\frac{1}{2}\|X-AB^T\|_F^2+\frac{\lambda}{2}\left(\|A\|_F^2+\|B\|_F^2\right)\quad(14)$$

A solution is given by $\hat{A}=U_r\mathcal{S}_\lambda\left(D_r\right)^{\frac{1}{2}}$ *and* $\hat{B}=V_r\mathcal{S}_\lambda\left(D_r\right)^{\frac{1}{2}}$, *and all solutions satisfy* $\hat{A}\hat{B}^T=\hat{Z}$, where, $\hat{Z}$ *is as given in Problem* (10).

We make use of the following lemma from Srebro et al. (2005); Mazumder et al. (2010), which we give without proof:

Lemma 1

$$\|Z\|_* = \min_{A,B:Z=AB^T} \frac{1}{2}\left(\|A\|_F^2 + \|B\|_F^2\right)$$

Proof (of Theorem 2). Using Lemma 1, we have that

$$\min_{A_{m\times r}, B_{n\times r}} \frac{1}{2}\|X - AB^T\|_F^2 + \frac{\lambda}{2}\|A\|_F^2 + \frac{\lambda}{2}\|B\|_F^2$$
$$= \min_{Z:\,\mathrm{rank}(Z)\le r} \frac{1}{2}\|X - Z\|_F^2 + \lambda\|Z\|_*$$

The conclusions follow from Theorem 1. ■

Note, in both theorems the solution might have rank less than r.

Inspired by the alternating subspace iteration algorithm (a.k.a. Orthogonal Iterations, Chapter 8, Golub and Van Loan, 2012) for the reduced-rank SVD, we present Algorithm 2.1, an alternating ridge-regression algorithm for finding the solution to Problem (9).

Remarks

1. At each step the algorithm keeps the current solution in "SVD" form, by representing A and B in terms of orthogonal matrices. The computational effort needed to do this is exactly that required to perform each ridge regression, and once done makes the subsequent ridge regression trivial.
2. The proof of convergence of this algorithm is essentially the same as that for an alternating subspace algorithm, a.k.a. Orthogonal Iterations (Chapter 8, Thm 8.2.2; Golub and Van Loan, 2012) (without shrinkage).
3. In principle step (7) is not necessary, but in practice it cleans up the rank nicely.
4. This algorithm lends itself naturally to distributed computing for very large matrices X; X can be chunked into smaller blocks, and the left and right matrix multiplies can be chunked accordingly. See Section 8.
5. There are many ways to check for convergence. Suppose we have a pair of iterates (U, D^2, V) (old) and $\left(\tilde{U}, \tilde{D}^2, \tilde{V}\right)$ (new), then the relative change in Frobenius norm is given by.

$$\nabla F = \frac{\left\|UD^2V^T - \widetilde{U}\widetilde{D}^2\widetilde{V}^T\right\|_F^2}{\left\|UD^2V^T\right\|_F^2} = \frac{\operatorname{tr}\left(D^4\right) + \operatorname{tr}\left(\widetilde{D}^4\right) - 2\operatorname{tr}\left(D^2U^T\widetilde{U}\widetilde{D}^2\widetilde{V}^TV\right)}{\operatorname{tr}\left(D^4\right)}, \quad (19)$$

which is not expensive to compute.

6. If X is sparse, then the left and right matrix multiplies can be achieved efficiently by using sparse matrix methods. Algorithm 2.1
Rank-Restricted Soft SVD

1. Initialize $A = UD$ where $U_{m\times r}$ is a randomly chosen matrix with orthonormal columns and $D = I_r$, the $r \times r$ identity matrix.
2. Given A, solve for B:

$$\underset{B}{\text{minimize}}\|X - AB^T\|_F^T + \lambda\|B\|_F^2. \quad (15)$$

This is a multiresponse ridge regression, with solution

$$\tilde{B}^T = \left(D^2 + \lambda I\right)^{-1}DU^TX. \quad (16)$$

This is simply matrix multiplication followed by coordinate-wise shrinkage.

3. Compute the SVD of $\tilde{B}D = \tilde{V}\tilde{D}^2R^T$, and let $V \leftarrow \tilde{V}, D \leftarrow \tilde{D}$, and $B = VD$.
4. Given B, solve for A:

$$\underset{A}{\text{minimize}}\|X - AB^T\|_F^T + \lambda\|A\|_F^2. \quad (17)$$

This is also a multiresponse ridge regression, with solution

$$\tilde{A} = XVD\left(D^2 + \lambda I\right)^{-1}. \quad (18)$$

Again matrix multiplication followed by coordinate-wise shrinkage.

5. Compute the SVD of $\tilde{A}D = \widetilde{U}\widetilde{D}^2R^T$, and let $U \leftarrow \widetilde{U}, D \leftarrow \widetilde{D}$, and $A = UD$.
6. Repeat steps $(2) - (5)$ until convergence of AB^T.
7. Compute $M = XV$, and then it's SVD: $M = UD_\sigma R^T$. Then output $U, V \leftarrow VR$ and $S_\lambda\left(D_\sigma\right) = \operatorname{diag}\left[\left(\sigma_1 - \lambda\right)_+, \ldots, \left(\sigma_r - \lambda\right)_+\right]$.

7. Likewise, if X is sparse, but has been column and/or row centered (see Section 9), it can be represented in "sparse plus low rank" form; once again left and right multiplication of such matrices can be done efficiently.

An interesting feature of this algorithm is that a reduced rank SVD of X is available from the solution, with the rank determined by the particular value of λ used. The singular values would have to be corrected by adding λ to each. There is empirical evidence that this is faster than without shrinkage, with accuracy biased more toward the larger singular values.

3. The softImpute-ALS Algorithm

Now we return to the case where X has missing values, and the non-missing entries are indexed by the set Ω. We present Algorithm 3.1 (softImpute-ALS) for solving Problem (6):

$$\underset{A,\,B}{\text{minimize}} \ \|P_{\Omega}\left(X - AB^T\right)\|_F^2 + \lambda\left(\|A\|_F^2 + \|B\|_F^2\right).$$

where $A_{m\times r}$ and $B_{n\times r}$ are each of rank at most $r \le \min(m, n)$.

The algorithm exploits the decomposition

$$P_{\Omega}\left(X - AB^T\right) = P_{\Omega}(X) + P_{\Omega}^{\perp}\left(AB^T\right) - AB^T. \quad (24)$$

Suppose we have current estimates for A and B, and we wish to compute the new $\tilde{B}$. We will replace the first occurrence of AB^T in the right-hand side of (24) with the current estimates, leading to a *filled in* $X^* = P_{\Omega}(X) + P_{\Omega}^{\perp}\left(AB^T\right)$, and then solve for $\tilde{B}$ in

$$\underset{\tilde{B}}{\text{minimize}} \ \|X^* - A\tilde{B}\|_F^2 + \lambda\|\tilde{B}\|_F^2.$$

Using the same notation, we can write (importantly)

$$X^* = P_{\Omega}(X) + P_{\Omega}^{\perp}\left(AB^T\right) = \left(P_{\Omega}(X) - P_{\Omega}\left(AB^T\right)\right) + AB^T; \quad (25)$$

This is the efficient *sparse* + *low-rank* representation for high-dimensional problems; efficient to store and also efficient for left and right multiplication.

Remarks

1. This algorithm is a slight modification of Algorithm 2.1, where in step 2(a) we use the latest imputed matrix X^* rather than X.
2. The computations in step 2(b) are particularly efficient. In (22) we use the current version of A and B to predict at the observed entries Ω, and then perform a multiplication of a sparse matrix on the left by a skinny matrix, followed by rescaling of the rows. In (23) we simply rescale the rows of the previous version for B^T.
3. After each update, we maintain the integrity of the current solution. By Lemma 1 we know that the solution to

$$\underset{A,\ B\,:AB^T\ =\ \tilde{A}\tilde{B}^T}{\text{minimize}} \left(\|A\|_F^2 + \|B\|_F^2\right) \quad (26)$$

Algorithm 3.1
Rank-Restricted Efficient Maximum-Margin Matrix Factorization: softImpute-ALS

ialize UD where $U_{m\times r}$ is a randomly chosen matrix with orthonormal columns and $D = I_r$, the $r \times r$ identity matrix, and $B = VD$ with $V = 0$. Alternatively, any prior solution $A = UD$ and

en A = and $B = VD$, approximately solve

$$\underset{\tilde{B}}{\text{minimize}}\ \frac{1}{2}\|P_{\Omega}\left(X - A\tilde{B}^T\right)\|_F^T + \frac{\lambda}{2}\|\tilde{B}\|_F^2 \qquad (20)$$

date E achieve that with the following steps:

Let X^* $_{\Omega}(X) - P_{\Omega}\left(AB^T\right)) + AB^T$, stored as *sparse plus low − rank*.

Solve

$$\underset{\tilde{B}}{\text{minimize}}\frac{1}{2}\|X^* - A\tilde{B}^T\|_F^2 + \frac{\lambda}{2}\|\tilde{B}\|_F^2, \qquad (21)$$

ith so

$$= \left(D^2 + \lambda I\right)^{-1} DU^T X^*$$

$$= \left(D^2 + \lambda I\right)^{-1} DU^T\left(P_{\Omega}(X) - P_{\Omega}\left(AB^T\right)\right) \qquad (22)$$

$$+\left(D^2 + \lambda I\right)^{-1} D^2 B^T. \qquad (23)$$

Use th ution for $\tilde{B}$ and update V and D:

i. bute the SVD decomposition $\tilde{B}D = \tilde{U}\tilde{D}^2\tilde{V}^T$;

ii. $\tilde{U}$, and $D \leftarrow \tilde{D}$.

en B = , solve for A. By symmetry, this is equivalent to step 2, with X^T replacing X, and B and A interchanged.

peat st) − (3) until convergence.

mpute X^*V, and then it's SVD: $M = UD_{\sigma}R^T$. Then output U, $V \leftarrow VR$ and $D_{\sigma,\lambda} =$ diag $\left[\left(\sigma_1 - \lambda\right)_+, \ldots, \left(\sigma_r - \lambda\right)_+\right]$.

is given by the SVD of $\tilde{A}\tilde{B}^T = UD^2V^T$, with $A = UD$ and $B = VD$. Our iterates maintain this each time A or B changes in step 2(c), with no additional significant computational cost.

4. The final step is as in Algorithm 2.1. We know the solution should have the form of a soft-thresholded SVD. The alternating ridge regression might not exactly reveal the rank of the solution. This final step tends to clean this up, by revealing exact zeros after the soft-thresholding.

5. In Section 5 we discuss (the lack of) optimality guarantees of fixed points of Algorithm 3.1 (in terms of criterion (1)). We note that the output of softImpute-ALS can easily be piped into softImpute as a warm start. This typically exits almost immediately in our R package softImpute.

4. Broader Perspective and Related Work

Block coordinate descent (for example, Bertsekas, 1999) is a classical method in optimization that is widely used in the statistical and machine learning communities (Hastie et al., 2009). This is useful especially when the optimization problems associated with each block is relatively simple. The algorithm presented in this paper is a stylized variant of block coordinate descent. At a high level *vanilla* block coordinate descent (which we call ALS) applied to Problem (6) performs a complete minimization wrt one variable with the other fixed, before it switches to over the other variable. softImpute-ALS instead, does a *partial*

minimization of a very specific form. Razaviyayn et al. (2013) study convergence properties of generalized block-coordinate methods that apply to a fairly large class of problems. The same paper presents asymptotic convergence guarantees, i.e., the iterates converge to a stationary point (Bertsekas, 1999). Asymptotic convergence is fairly straightforward to establish for softimpute-ALS. We also describe global convergence rate guarantees[2] for softimpute-ALS in terms of various metrics of convergence to a stationary point. Perhaps more interestingly, we connect the properties of the stationary points of the non-convex Problem (6) to the minimizers of the convex Problem (1), which seems to be well beyond the scope and intent of Razaviyayn et al. (2013).

Variations of alternating-minimization strategies are popularly used in the context of matrix completion (Chen et al., 2012; Koren et al., 2009; Zhou et al., 2008). Jain et al. (2013) analyze the statistical properties of vanilla alternating-minimization algorithms for Problem (6) with $\lambda = 0$, i.e.,

$$\underset{A,B}{\text{minimize}} \quad \|P_{\Omega}\left(X - AB^T\right)\|_F^2,$$

where, one attempts to minimize the above function via alternating least squares *i.e.* first minimizing with respect to A (with B fixed) and vice-versa. They establish statistical performance guarantees of the alternating strategy under incoherence conditions on the singular vectors of the *underlying* low-rank matrix—the assumptions are similar in spirit to the work of Candès and Tao (2009); Candès and Recht (2008). However, as pointed out by Jain et al. (2013), their alternating-minimization methods typically require $|\Omega|$ to be larger than than required by convex optimization based methods (Candès and Recht, 2008). We refer the interested reader to more recent work of Hardt (2014), analyzing the statistical properties of alternating minimization methods.

The flavor of our present work is in spirit different from that described above (Jain et al., 2013; Hardt, 2014). Our main goal here is to develop non-convex algorithms for the optimization of Problem (6) for arbitrary λ and rank r. A special case of our framework corresponds to the case where Problem (6) is equivalent to Problem (1), for proper choices of r, λ. In this particular case, we study in Section 5 when our algorithm softImpute-ALS converges to a global minimizer of Problem (1)—this can be verified by a minor check that requires computing the low-rank SVD of a matrix that can be written as the sum of a sparse and low-rank matrix. Thus softImpute-ALS can be thought of a *non-convex* algorithm that *solves* the convex nuclear norm regularized Problem (1). Hence softImpute-ALS inherits statistical properties of the convex Problem (1) as established in Candès and Tao (2009); Candès and Recht (2008). We have also demonstrated in Figures 1 and 3 that softimpute-ALS is much faster than the alternating least squares schemes analyzed in Jain et al. (2013); Hardt (2014).

[2]By global convergence rate, we mean an upper bound on the maximum number of iterations that need to be taken by an algorithm to reach an ϵ-accurate first-order stationary point. This rate applies for *any* starting point of the algorithm.

Note that the use of non-convex methods to obtain minimizers of convex problems have been studied in Burer and Monteiro (2005); Journée et al. (2010). The authors study nonlinear optimization algorithms using non-convex matrix factorization formulations to obtain global minimizers of convex SDPs. The results presented in the aforementioned papers also requires one to check whether a stationary point is a *local minimizer*—this typically requires checking the positive definiteness of a matrix of size $O(mr + nr) \times O(mr + nr)$ and can be computationally demanding if the problem size is large. In contrast, the condition (derived in this paper) that needs to be checked to certify whether softimpute-ALS, upon convergence, has reached the global solution to the convex optimization Problem (1), is fairly intuitive and straightforward.

5. Algorithmic Convergence Analysis

In this section we investigate the theoretical properties of the softImpute-ALS algorithm in the context of Problems (1) and (6).

We show that the softImpute-ALS algorithm converges to a first order stationary point for Problem (6) at a rate of $O(1/K)$, where K denotes the number of iterations of the algorithm. We also discuss the role played by λ in the convergence rates. We establish the limiting properties of the estimates produced by the softImpute-ALS algorithm: properties of the limit points of the sequence (A_k, B_k) in terms of Problems (1) and (6). We show that for any r in Problem (6) the sequence produced by the softImpute-ALS algorithm leads to a decreasing sequence of objective values for the convex Problem (1). A fixed point of the softImpute-ALS problem need not correspond to the minimum of the convex Problem (1). We derive simple necessary and sufficient conditions that must be satisfied for a stationary point of the algorithm to be a minimum for the Problem (1)—the conditions can be verified by a simple structured low-rank SVD computation.

We begin the section with a formal description of the updates produced by the algorithm in terms of the original objective function (6) and its majorizers (27) and (28). Theorem 3 establishes that the updates lead to a decreasing sequence of objective values $F(A_k, B_k)$ in (6). Section 5.1 (Theorem 4 and Corollary 1) derives the finite-time convergence rate properties of the proposed algorithm softImpute-ALS. Section 5.2 provides descriptions of the first order stationary conditions for Problem (6), the fixed points of the algorithm softImpute-ALS and the limiting behavior of the sequence (A_k, B_k), $k \geq 1$ as $k \to \infty$. Section 5.3 (Lemma 4) investigates the implications of the updates produced by softImpute-ALS for Problem (6) in terms of the Problem (1). Section 5.3.1 (Theorem 6) studies the stationarity conditions for Problem (6) vis-a-vis the optimality conditions for the convex Problem (1).

The softImpute-ALS algorithm may be thought of as an EM or more generally a MM-style algorithm (majorization minimization), where every imputation step leads to an upper bound to the training error part of the loss function. The resultant loss function is minimized wrt A—this leads to a partial minimization of the objective function wrt A. The process is repeated with the other factor B, and continued till convergence.

Recall the objective function in Problem (6):

$$F(A,B) := \frac{1}{2}\|P_\Omega\left(X - AB^T\right)\|_F^2 + \frac{\lambda}{2}\|A\|_F^2 + \frac{\lambda}{2}\|B\|_F^2.$$

We define the surrogate functions

$$\begin{aligned} Q_A(Z_1|A,B) \;:=\; & \frac{1}{2}\left\|P_\Omega\left(X - Z_1B^T\right) + P_\Omega^\perp\left(AB^T - Z_1B^T\right)\right\|_F^2 \\ & + \frac{\lambda}{2}\|Z_1\|_F^2 + \frac{\lambda}{2}\|B\|_F^2 \end{aligned} \tag{27}$$

$$\begin{aligned} Q_B(Z_2|A,B) \;:=\; & \frac{1}{2}\left\|P_\Omega\left(X - AZ_2^T\right) + P_\Omega^\perp\left(AB^T - AZ_2^T\right)\right\|_F^2 \\ & + \frac{\lambda}{2}\|A\|_F^2 + \frac{\lambda}{2}\|Z_2\|_F^2. \end{aligned} \tag{28}$$

Consider the function $g\left(AB^T\right) := \frac{1}{2}\|P_\Omega\left(X - AB^T\right)\|_F^2$ which is the training error as a function of the outer-product $Z = AB^T$, and observe that for any $Z, \bar{Z}$ we have:

$$\begin{aligned} g(Z) \;&\le\; \frac{1}{2}\left\|P_\Omega(X - Z) + P_\Omega^\perp(\bar{Z} - Z)\right\|_F^2 \\ &=\; \frac{1}{2}\left\|\left(P_\Omega(X) + P_\Omega^\perp(\bar{Z})\right) - Z\right\|_F^2 \end{aligned} \tag{29}$$

where, equality holds above at $Z = \bar{Z}$. This leads to the following simple but important observations:

$$Q_A(Z_1|A,B) \ge F(Z_1,B), \quad Q_B(Z_2|A,B) \ge F(A,Z_2), \tag{30}$$

suggesting that $Q_A(Z_1|A, B)$ is a majorizer of $F(Z_1,B)$ (as a function of Z_1); similarly, $Q_B(Z_2|A, B)$ majorizes $F(A, Z_2)$. In addition, equality holds as follows:

$$Q_A(A|A,B) = F(A,B) = Q_B(B|A,B). \tag{31}$$

We also define $X^*_{A,B} = P_\Omega(X) + P_\Omega^\perp\left(AB^T\right)$. Using these definitions, we can succinctly describe the softImpute-ALS algorithm in Algorithm 5.1. This is almost equivalent to Algorithm 3.1, but more convenient for theoretical analysis. It has the orthogonalization and redistribution of $\widetilde{D}$ in step 3 removed, and step 5 removed. Observe that the Algorithm 5.1

softImpute-ALS

Inputs: Data matrix X, initial iterates A_0 and B_0, and $k = 0$.

Outputs: (A^*, B^*) an estimate of the minimizer of Problem (6)

Repeat until Convergence

1. $k \leftarrow k + 1$.
2. $X^* \leftarrow P_\Omega(X) + P_\Omega^\perp\left(AB^T\right) = P_\Omega\left(X - AB^T\right) + AB^T$
3. $A \leftarrow X^*B\left(B^TB + \lambda I\right)^{-1} = \arg\min_{Z_1} Q_A\left(Z_1 \middle| A, B\right)$.
4. $X^* \leftarrow P_\Omega(X) + P_\Omega^\perp\left(AB^T\right)$
5. $B \leftarrow X^{*T}A\left(A^TA + \lambda I\right)^{-1} = \arg\min_{Z_2} Q_B\left(Z_2 \middle| A, B\right)$.

softImpute-ALS algorithm can be described as the following iterative procedure:

$$A_{k+1} \in \arg\min_{Z_1} Q_A\left(Z_1 \middle| A_k, B_k\right) \quad (32)$$

$$B_{k+1} \in \arg\min_{Z_2} Q_B\left(Z_2 \middle| A_{k+1}, B_k\right). \quad (33)$$

We will use the above notation in our proof.

We can easily establish that softImpute-ALS is a descent method, or its iterates never increase the function value.

Theorem 3 *Let* $\{(A_k, B_k)\}$ *be the iterates generated by* softImpute-ALS. *The function values are monotonically decreasing,*

$$F\left(A_k, B_k\right) \geq F\left(A_{k+1}, B_k\right) \geq F\left(A_{k+1}, B_{k+1}\right), \quad k \geq 1.$$

Proof Let the current iterate estimates be (A_k, B_k). We will first consider the update in A, leading to A_{k+1}, as defined in (32).

$$\min_{Z_1} Q_A\left(Z_1 \middle| A_k, B_k\right) \leq Q_A\left(A_k \middle| A_k, B_k\right) = F\left(A_k, B_k\right)$$

Note that, $\min_{Z_1} Q_A\left(Z_1 | A_k, B_k\right) = Q_A\left(A_{k+1} | A_k, B_k\right)$, by definition of A_{k+1} in (32).

Using (30) we get that $Q_A(A_{k+1}| A_k, B_k) \geq F(A_{k+1}, B_k)$. Putting together the pieces we get: $F(A_k, B_k) \geq F(A_{k+1}, B_k)$.

Using an argument exactly similar to the above for the update in B we have:

$$F(A_{k+1}, B_k) = Q_B(B_k | A_{k+1}, B_k) \geq Q_B(B_{k+1} | A_{k+1}, B_k) \geq F(A_{k+1}, B_{k+1}). \quad (34)$$

This establishes that $F(A_k, B_k) > F(A_{k+1}, B_{k+1})$ for all k, thereby completing the proof of the theorem. ■

5.1 softImpute-ALS: Rates of Convergence

The previous section derives some elementary properties of the softImpute-ALS algorithm, namely the updates lead to a decreasing sequence of objective values. We will now derive some results that inform us about the rate at which the softImpute-ALS algorithm reaches a stationary point.

We begin with the following lemma, which presents a lower bound on the successive difference in objective values of $F(A, B)$,

Lemma 2 *Let* (A_k, B_k) *denote the values of the factors at iteration k. We have the following*:

$$\begin{aligned} F(A_k, B_k) - F(A_{k+1}, B_{k+1}) \geq \frac{1}{2}\left(\|(A_k - A_{k+1})B_k^T\|_F^2 + \|A_{k+1}(B_{k+1} - B_k)^T\|_F^2\right) \\ + \frac{\lambda}{2}\left(\|A_k - A_{k+1}\|_F^2 + \|B_{k+1} - B_k\|_F^2\right) \end{aligned} \quad (35)$$

Proof *See* Section A.1 *for the proof.* ■

For any two matrices A and B respectively define A^+, B^+ as follows:

$$A^+ \in \arg\min_{Z_1} Q_A(Z_1 | A, B), \qquad B^+ \in \arg\min_{Z_2} Q_B(Z_2 | A, B) \quad (36)$$

We will consequently define the following:

$$\begin{aligned} \Delta((A, B), (A^+, B^+)) := \frac{1}{2}\left(\left\|(A - A^+)B^T\right\|_F^2 + \left\|A^+(B - B^+)^T\right\|_F^2\right) \\ + \frac{\lambda}{2}\left(\|A - A^+\|_F^2 + \|B - B^+\|_F^2\right) \end{aligned} \quad (37)$$

Lemma 3 $\Delta((A, B), (A^+, B^+)) = 0$ iff A, B *is a fixed point of* softImpute-ALS.

Proof *See* Section A.2, *for a proof.* ■

We will use the following notation

$$\eta_k := \Delta\big((A_k, B_k), (A_{k+1}, B_{k+1})\big) \quad (38)$$

Thus η_k can be used to quantify how close (A_k, B_k) is from a stationary point.

If $\eta_k > 0$ it means that Algorithm softImpute-ALS will make progress in improving the quality of the solution. As a consequence of the monotone decreasing property of the sequence of objective values $F(A_k, B_k)$ and Lemma 2, we have that, $\eta_k \to 0$ as $k \to \infty$. The following theorem characterizes the rate at which η_k converges to zero.

Theorem 4 *Let* (A_k, B_k),$k \geq 1$ *be the sequence generated by the* softImpute-ALS *algorithm. The decreasing sequence of objective values* $F(A_k, B_k)$ *converges to* $F^\infty \geq 0$ *(say) and the quantities* $\eta_k \to 0$.

Furthermore, we have the following finite convergence rate of the softImpute-ALS *algorithm*:

$$\min_{1 \leq k \leq K} \eta_k \leq \big(F(A_1, B_1) - F^\infty\big)/K \quad (39)$$

Proof *See* Section A.3 ■

The above theorem establishes a $O\left(\frac{1}{K}\right)$ convergence rate of softImpute-ALS; in other words, for any $\epsilon > 0$, we need at most $K = O\left(\frac{1}{\epsilon}\right)$ iterations to arrive at a point (A_{k^*}, B_{k^*}) such that $\eta_{k^*} \leq \epsilon$, where, $1 \leq k^* \leq K$.

Note that Theorem 4 establishes convergence of the algorithm for any value of $\lambda \geq 0$. We found in our numerical experiments that the value of λ has an important role to play in the speed of convergence of the algorithm. In the following corollary, we provide convergence rates that make the role of λ explicit.

The following corollary employs three different distance measures to measure the closeness of a point from stationarity.

Corollary 1 *Let* (A_k, B_k), $k \geq 1$ *be defined as above. Assume that for all* $k \geq 1$

$$\ell^U \mathbf{I} \succcurlyeq B_k^T B_k \succcurlyeq \ell^L \mathbf{I}, \quad \ell^U \mathbf{I} \succcurlyeq A_k^T A_k \succcurlyeq \ell^L \mathbf{I}, \quad (40)$$

where, ℓ^U, ℓ^L *are constants independent of k.*

Then we have the following:

$$\min_{1 \le k \le K} \left(\|(A_k - A_{k+1})\|_F^2 + \|B_k - B_{k+1}\|_F^2 \right) \le \frac{2}{\left(\ell^L + \lambda\right)} \left(\frac{F(A_1, B_1) - F^\infty}{K} \right) \quad (41)$$

$$\min_{1 \le k \le K} \left(\|(A_k - A_{k+1}) B_k^T\|_F^2 + \|A_{k+1}(B_k - B_{k+1})^T\|_F^2 \right) \le \frac{2\ell^U}{\lambda + \ell_U} \left(\frac{F(A_1, B_1) - F^\infty}{K} \right) \quad (42)$$

$$\min_{1 \le k \le K} \left(\|\nabla_A f(A_k, B_k)\|^2 + \|\nabla_B f(A_{k+1}, B_k)\|^2 \right) \le \frac{2\left(\ell^U\right)^2}{\left(\ell^L + \lambda\right)} \left(\frac{F(A_1, B_1) - F^\infty}{K} \right) \quad (43)$$

where, $\nabla_A f(A, B)$ *(respectively*, $\nabla_B f(A, B)$*) denotes the partial derivative of* $F(A, B)$ *wrt* A (*respectively, B*).

Proof *See* Section A.4. ■

Inequalities (41)–(43) are statements about different notions of distances between successive iterates. These may be employed to understand the convergence rate of softImpute-ALS.

Assumption (40) is a minor one. While it may not be straightforward to estimate ℓ^U prior to running the algorithm, a finite value of ℓ^U is guaranteed as soon as $\lambda > 0$. The lower bound $\ell_L > 0$, if both $A_1 \in \Re^{m \times r}$, $B_1 \in \Re^{n \times r}$ have rank r and the rank remains the same across the iterates. If the solution to Problem (6) has a rank smaller than r, then $\ell_L = 0$, however, this situation is typically avoided since a small value of r leads to lower computational cost per iteration of the softImpute-ALS algorithm. The constants appearing as a part of the rates in (41)–(43) are dependent upon λ. The constants are smaller for larger values of λ. Finally we note that the algorithm does not require any information about the constants ℓ^L, ℓ^U appearing as a part of the rate estimates.

5.2 softImpute-ALS: Asymptotic Convergence

In this section we derive some properties of the limiting behavior of the sequence (A_k, B_k), in particular we examine some elementary properties of the limit points of the sequence (A_k, B_k).

At the beginning, we recall the notion of first order stationarity of a point A_*, B_*. We say that A_*, B_* is said to be a first order stationary point for the Problem (6) if the following holds:

$$\nabla_A f(A_*, B_*) = 0, \quad \nabla_B f(A_*, B_*) = 0. \quad (44)$$

An equivalent restatement of condition (44) is:

$$A_* \in \arg\min_{Z_1} Q_A(Z_1 \mid A_*, B_*), \quad B_* \in \arg\min_{Z_2} Q_B(Z_2 \mid A_*, B_*), \quad (45)$$

i.e., A_*, B_* is a fixed point of the softImpute-ALS algorithm updates.

We now consider uniqueness properties of the limit points of (A_k, B_k), $k \geq 1$. Even in the fully observed case, the stationary points of Problem (6) are not unique in A_*, B_*; due to orthogonal invariance. Addressing convergence of (A_k, B_k) becomes trickier if two singular values of $A_* B_*^T$ are tied. In this vein we have the following result:

Theorem 5 *Let* $\{(A_k, B_k)\}_k$ *be the sequence of iterates generated by* Algorithm 5.1. *For* $\lambda > 0$, *we have:*

- **a.** *Every limit point of* $\{(A_k, B_k)\}_k$ *is a stationary point of Problem*, (6).
- **b.** *Let* B_* *be any limit point of the sequence* B_k, $k \geq 1$, *with* $B_\nu \rightarrow B_*$, *where,* ν *is a subsequence of* $\{1, 2, \ldots,\}$. *Then the sequence* A_ν *converges*.

  Similarly, let A_* *be any limit point of the sequence* A_k, $k \geq 1$, *with* $A_\mu \rightarrow B_*$, *where,* μ *is a subsequence of* $\{1, 2, \ldots,\}$. *Then the sequence* B_μ *converges*.

Proof *See* Section A.5 ■

The above theorem is a partial result about the uniqueness of the limit points of the sequence A_k, B_k. The theorem implies that if the sequence A_k converges, then the sequence B_k must converge and vice-versa. More generally, for every limit point of A_k, the associated B_k (sub)sequence will converge. The same result holds true for the sequence B_k.

Remark 1 *Note that the condition* $\lambda > 0$ *is enforced due to technical reasons so that the sequence* (A_k, B_k) *remains bounded. If* $\lambda = 0$, *then* $A \leftarrow cA$ *and* $B \leftarrow \frac{1}{c}B$ *for any* $c > 0$, *leaves the objective function unchanged. Thus one may take* $c \rightarrow \infty$ *making the sequence of updates unbounded without making any change to the values of the objective function*.

5.3 Implications of softImpute-ALS updates in term s of Problem (1)

The sequence (A_k, B_k) generated by Algorithm (5.1) are geared towards minimizing criterion (6), it is interesting to explore what implications the sequence might have for the convex Problem (1). In particular, we know that $F(A_k, B_k)$ is decreasing—does this imply a monotone sequence $H\left(A_k B_k^T\right)$? We show below that it is indeed possible to obtain a monotone decreasing sequence $H\left(A_k B_k^T\right)$ with a minor modification. These modifications are exactly those implemented in Algorithm 3.1 in step 3.

The idea that plays a crucial role in this modification is the following inequality (for a proof see Mazumder et al. (2010); see also remark 3 in Section 3):

$$\|AB^T\|_* \leq \frac{1}{2}\left(\|A\|_F^2 + \|B\|_F^2\right).$$

Note that equality holds above if we take a particular choice of A and B given by:

$$A = UD^{1/2}, B = VD^{1/2}, \quad \text{where,} \quad AB^T = UDV^T \quad \text{(SVD)}, \quad (46)$$

is the SVD of AB^T. The above observation implies that if (A_k, B_k) is generated by softImpute-ALS then

$$F\left(A_k, B_k\right) \geq H\left(A_k B_k^T\right)$$

with equality holding if A_k, B_k^T are represented via (46). Note that this re-parametrization does not change the training error portion of the objective $F(A_k, B_k)$, but decreases the ridge regularization term—and hence decreases the overall objective value when compared to that achieved by softImpute-ALS without the reparametrization (46).

We thus have the following Lemma:

Lemma 4 *Let the sequence* (A_k, B_k) *generated by* softImpute-ALS *be stored in terms of the factored SVD representation* (46). *This results in a decreasing sequence of objective values in the nuclear norm penalized Problem* (1):

$$H\left(A_k B_k^T\right) \geq H\left(A_{k+1} B_{k+1}^T\right)$$

with $H\left(A_k B_k^T\right) = F\left(A_k, B_k\right)$, *for all k. The sequence* $H\left(A_k B_k^T\right)$ thus converges to F^∞.

Note that, F^∞ need not be the minimum of the convex Problem (1). It is easy to see this, by taking r to be smaller than the rank of the optimal solution to Problem (1).

5.3.1 A Closer Look at the Stationary Conditions—In this section we inspect the first order stationary conditions of the non-convex Problem (6) alongside those for the convex Problem (1). We will see that a first order stationary point of the convex Problem (1) leads to factors (A, B) that are stationary for Problem (6). However, the converse of this statement need not be true in general. However, given an estimate delivered by softImpute-ALS (upon convergence) it is easy to verify whether it is a solution to Problem (1).

Note that Z^* is the optimal solution to the convex Problem (1) iff:

$$\partial H(Z^*) = P_\Omega(Z^* - X) + \lambda \operatorname{sgn}(Z^*) = 0,$$

where, sgn(Z^*) is a sub-gradient of the nuclear norm $\|Z\|_*$ at Z^*. Using the standard characterization (Lewis, 1996) of sgn(Z^*) the above condition is equivalent to:

$$P_\Omega(Z^* - X) + \lambda U_* \text{sgn}(D^*) V_*^T = 0 \quad (47)$$

where, the full SVD of Z^* is given by $U_* D_* V_*^T$; sgn(D^*) is a diagonal matrix with ith diagonal entry given by $\text{sgn}\left(d_{ii}^*\right)$, where, d_{ii}^* is the ith diagonal entry of D^*.

If a limit point $A_* B_*^T$ of the softImpute-ALS algorithm satisfies the stationarity condition (47) above, then it is the optimal solution of the convex problem. We note that $A_* B_*^T$ need not necessarily satisfy the stationarity condition (47).

(A, B) satisfy the stationarity conditions of softImpute-ALS if the following conditions are satisfied:

$$P_\Omega\left(AB^T - X\right)B + \lambda A = 0, \quad A^T\left(P_\Omega\left(AB^T - X\right)\right) + \lambda B^T = 0,$$

where, we assume that A, B are represented in terms of (46). This gives us:

$$P_\Omega\left(AB^T - X\right)V + \lambda U = 0, \quad U^T\left(P_\Omega\left(AB^T - X\right)\right) + \lambda V^T = 0, \quad (48)$$

where $AB^T = UDV^T$, being the reduced rank SVD i.e. all diagonal entries of D are strictly positive.

A stationary point of the convex problem corresponds to a stationary point of softImpute-ALS, as seen by a direct verification of the conditions above. In the following we investigate the converse: when does a stationary point of softImpute-ALS correspond to a stationarypoint of Problem (1); i.e. condition (47)? Towards this end, we make use of the ridged least-squares update used by softImpute-ALS. Assume that all matrices A_k, B_k have r rows.

At stationarity i.e. at a fixed point of softImpute-ALS we have the following:

$$A_* \in \quad \arg\min_A \frac{1}{2}\|P_\Omega\left(X - AB_*^T\right)\|_F^2 + \frac{\lambda}{2}\left(\|A\|_F^2 + \|B_*\|_F^2\right) \quad (49)$$

$$= \quad \arg\min_A \frac{1}{2}\|\left(P_\Omega(X) + P_\Omega^\perp\left(A_* B_*^T\right)\right) - AB_*^T\|_F^2 + \frac{\lambda}{2}\left(\|A\|_F^2 + \|B_*\|_F^2\right) \quad (50)$$

$$B_* \in \quad \arg\min_B \frac{1}{2}\|P_\Omega\left(X - A_* B^T\right)\|_F^2 + \frac{\lambda}{2}\left(\|A_*\|_F^2 + \|B\|_F^2\right) \quad (51)$$

$$= \quad \arg\min_B \frac{1}{2}\|\left(P_\Omega(X) + P_\Omega^\perp\left(A_* B_*^T\right)\right) - A_* B^T\|_F^2 + \frac{\lambda}{2}\left(\|A_*\|_F^2 + \|B\|_F^2\right) \quad (52)$$

Line (50) and (52) can be thought of doing alternating multiple ridge regressions for the fully observed matrix $P_\Omega(X) + P_\Omega^\perp\left(A_* B_*^T\right)$.

The above fixed point updates are very closely related to the following optimization problem:

$$\underset{A_{mxr}, B_{mxr}}{\text{minimize}} \ \frac{1}{2}\|\left(P_\Omega(X) + P_\Omega^\perp\left(A_* B_*^T\right)\right) - AB\|_F^2 + \frac{\lambda}{2}\left(\|A\|_F^2 + \|B\|_F^2\right) \quad (53)$$

The solution to (53) by Theorem 1 is given by the nuclear norm thresholding operation (with a rank r constraint) on the matrix $P_\Omega(X) + P_\Omega^\perp\left(A_* B_*^T\right)$:

$$\underset{Z:\ rank\,(Z) \le r}{\text{minimize}} \ \frac{1}{2}\|\left(P_\Omega(X) + P_\Omega^\perp\left(A_* B_*^T\right)\right) - Z\|_F^2 + \frac{\lambda}{2}\|Z\|_* . \quad (54)$$

Suppose the convex optimization Problem (1) has a solution Z^* with rank(Z^*) = r^*.

Then, for $A_* B_*^T$ to be a solution to the convex problem the following conditions are sufficient:

a. $r^* \le r$

b. A_*, B_* must be the global minimum of Problem (53). Equivalently, the outer product $A_* B_*^T$ must be the solution to the *fully observed* nuclear norm regularized problem:

$$A_* B_*^T \in \arg\min_Z \frac{1}{2}\|\left(P_\Omega(X) + P_\Omega^\perp\left(A_* B_*^T\right)\right) - Z\|_F^2 + \lambda\|Z\|_* . \quad (55)$$

The above condition (55) can be verified fairly easily; and requires doing a low-rank SVD of the matrix $P_\Omega(X) + P_\Omega^\perp\left(A_* B_*^T\right)$ as a direct application of Algorithm 2.1. This task is computationally attractive due to the "sparse plus low-rank structure" of the matrix: $P_\Omega(X) + P_\Omega^\perp\left(A_* B_*^T\right) = P_\Omega\left(X - A_* B_*^T\right) + A_* B_*^T$. We

summarize the above discussion in the form of the following theorem, where we assume of course that $\lambda > 0$.

Theorem 6 *Let* $A_k \in \Re^{m \times r}$, $B_k \in \Re^{n \times r}$ *be the sequence generated by* softImpute-ALS *and let* (A_*, B_*) *denote a limit point of the sequence. Suppose that Problem* (1) *has a minimizer with rank at most r. If* $Z_* = A_* B_*^T$ *solves the fully observed nuclear norm regularized problem* (55), *then* Z_* *is a solution to the convex Problem* (1).

5.4 Computational Complexity and Comparison to ALS

The computational cost of softImpute-ALS can be broken down into three steps. First consider only the cost of the update to A. The first step is forming the matrix $X^* = P_\Omega\left(X - AB^T\right) + AB^T$, which requires $O(r|\Omega|)$ flops for the $P_\Omega(AB^T)$ part, while the second part is never explicitly formed. The matrix $B(B^T B + \lambda I)^{-1}$ requires $O(2nr^2 + r^3)$ flops to form; although we keep it in SVD factored form, the cost is the same. The multiplication $X^* B\,(B^T B + \lambda I)^{-1}$ requires $O(r|\Omega| + mr^2 + nr^2)$ flops, using the sparse plus low-rank structure of X^*. The total cost of an iteration is $O(2r|\Omega| + mr^2 + 3nr^2 + r^3)$.

As mentioned in Section 1, alternating least squares (ALS) is a popular algorithm for solving the matrix factorization problem in Equation (6); see Algorithm 5.2. The ALS algorithm is an instance of block coordinate descent applied to (6).

Recall that the updates for ALS are given by

$$A_{k+1} \in \arg\min_A \; F\left(A, B_k\right) \quad (56)$$

$$B_{k+1} \in \arg\min_B \; F\left(A_k, B\right), \quad (57)$$

and each row of A and B can be computed via a separate ridge regression. The cost for each ridge regression is $O(|\Omega_j|r^2 + r^3)$, so the cost of one iteration is $O(2|\Omega|r^2 + mr^3 + nr^3)$. Hence the cost of one iteration of ALS is r times more flops than one iteration of softImpute-ALS. We will see in the next sections that while ALS may decrease the criterion at each iteration more than softImpute-ALS, it tends to be slower because the cost is higher by a factor $O(r)$.

Algorithm 5.2
Alternating least squares ALS

Inputs: Data matrix X, initial iterates A_0 and B_0, and $k = 0$.

Outputs: $(A^*, B^*) = \text{argmin}_{A,B} F(A, B)$

Repeat until Convergence

for i=1 to m **do**

$$A_i \leftarrow \left(\sum_{j \in \Omega_i} B_j B_j^T\right)^{-1} \left(\sum_{j \in \Omega_i} X_{ij} B_j\right)$$

end for

for $j = 1$ to n **do**

$$B_j \leftarrow \left(\sum_{i \in \Omega_j} A_i A_i^T\right)^{-1} \left(\sum_{i \in \Omega_j} X_{ij} A_i\right)$$

end for

Dependence of Computational Complexity on Ω: The computational guarantees derived in Section 5.1 present a worst-case viewpoint of the rate at which softimpute-ALS converge to an approximate stationary point—the results apply to *any* data and an arbitrary Ω. Tighter rates can be derived under additional assumptions. For example, for the special case where Ω corresponds to a fully observed matrix, softimpute-ALS becomes Algorithm 2.1. For $\lambda = 0$, Algorithm 2.1 with Ω fully observed becomes *exactly* equivalent to the Orthogonal Iteration algorithm of Golub and Van Loan (2012). Theorem 8.2.2 in Golub and Van Loan (2012) shows that the left orthogonal subspace corresponding to A converges to the left singular subspace of X, under the assumption that $\sigma_r(X) > \sigma_{r+1}(X)$—the rate is linear[3] and depends upon the ratio $\frac{\sigma_{r+1}(X)}{\sigma_r(X)_e}$. Similar results hold true for the left orthogonal subspace of B. Since the left subspaces of A and B generated by Algorithm 2.1 with $\lambda > 0$ are the same for $\lambda = 0$, the same linear rate of convergence holds true for Algorithm 2.1 for Problem (14).

For a general Ω it is not clear to us if the rates in Section 5.1 can be improved. However, for a sparse Ω the computational cost of every iteration of softimpute-ALS is significantly smaller than a dense observation pattern—the practical significance being that a large number of iterations can be performed at a very low cost.

6. Experiments

In this section we run some timing experiments on simulated and real datasets, and show performance results on the Netflix and MovieLens data.

6.1 Timing experiments

Figure 1 shows timing results on four datasets. The first three are simulation datasets of increasing size, and the last is the publicly available MovieLens 100K data. These experiments were all run in R using the softImpute package; see Section 7. Three methods are compared:

[3]Convergence is measured in terms of the usual notion of distance between subspaces (Golub and Van Loan, 2012); and it is also assumed that the initialization is not completely orthogonal to the target subspace, which is typically met in practice due to the presence of round-off errors.

1. ALS — Alternating Least Squares as in Algorithm 5.2;
2. softImpute-ALS — our new approach, as defined in Algorithm 3.1 or 5.1;
3. softImpute — the original algorithm of Mazumder et al. (2010), as layed out in steps (2)–(4).

We used an R implementation for each of these in order to make the fairest comparisons. In particular, algorithm softImpute requires a low-rank SVD of a complete matrix at each iteration. For this we used the function svd. als from our package, which uses alternating subspace iterations, rather than using other optimized code that is available for this task. Likewise, there exists optimized code for regular ALS for matrix completion, but instead we used our R version to make the comparisons fairer. We are trying to determine how the computational trade-offs play off, and thus need a level playing field.

Each subplot in Figure 6.1 is labeled according to the size of the problem, the fraction missing, the value of λ used, the operating rank of the algorithms r, and the rank of the solution obtained. All three methods involve alternating subspace methods; the first two are alternating ridge regressions, and the third alternating orthogonal regressions. These are conducted at the operating rank r, anticipating a solution of smaller rank. Upon convergence, softImpute-ALS performs step (5) in Algorithm 3.1, which can truncate the rank of the solution. Our implementation of ALS does the same.

For the three simulation examples, the data are generated from an underlying Gaussian factor model, with true ranks 50, 100, 100; the missing entries are then chosen at random. Their sizes are (300, 200), (800, 600) and (1200, 900) respectively, with between 70–90% missing. The MovieLens 100K data has 100K ratings (1–5) for 943 users and 1682 movies, and hence is 93% missing.

We picked a value of λ for each of these examples (through trial and error) so that the final solution had rank less than the operating rank. Under these circumstances, the solution to the criterion (6) coincides with the solution to (1), which is unique under non-degenerate situations.

There is a fairly consistent message from each of these experiments. softImpute-ALS wins handily in each case, and the reasons are clear:

- Even though it uses more iterations than ALS, they are much cheaper to execute (by a factor $O(r)$).
- softImpute wastes time on its early SVD, even though it is far from the solution. Thereafter it uses warm starts for its SVD calculations, which speeds each step up, but it does not catch up.

6.2 Netflix Competition Data

We used our softImpute package in R to fit a sequence of models on the Netflix competition data. Here there are 480,189 users, 17,770 movies and a total of 100,480,507 ratings, making the resulting matrix 98.8% missing. There is a designated test set (the “probe set”), a subset of 1,408,395 of the these ratings, leaving 99,072,112 for training.

Figure 2 compares the performance of hardImpute (Mazumder et al., 2010) with softImpute-ALS on these data. hardImpute uses rank-restricted SVDs iteratively to estimate the missing data, similar to softImpute but without shrinkage. The shrinkage helps here, leading to a best test-set RMSE of 0.943. This is a 1% improvement over the "Cinematch" score, somewhat short of the prize-winning improvement of 10%.

Both methods benefit greatly from using warm starts. hardImpute is solving a non-convex problem, while the intention is for softImpute-ALS to solve the convex Problem (1). This will be achieved if the operating rank is sufficiently large. The idea is to decide on a decreasing sequence of values for λ, starting from λ_{max} (the smallest value for which the solution $\widehat{M} = 0$, which corresponds to the largest singular value of $P_\Omega(X)$). Then for each value of λ, use an operating rank somewhat larger than the rank of the previous solution, with the goal of getting the solution rank smaller than the operating rank. The sequence of twenty models took under six hours of computing on a Linux cluster with 300Gb of ram (with a fairly liberal relative convergence criterion of 0.001), using the softImpute package in R.

Figure 3 (left panel) gives timing comparison results for one of the Netflix fits, this time implemented in Matlab. The right panel gives timing results on the smaller MovieLens 10M matrix. In these applications we need not get a very accurate solution, and so early stopping is an attractive option. softImpute-ALS reaches a solution close to the minimum in about 1/4 the time it takes ALS.

7. R Package softImpute

We have developed an R package softImpute for fitting these models (Hastie and Mazumder, 2013), which is available on CRAN. The package implements both softImpute and softImpute-ALS. It can accommodate large matrices if the number of missing entries is correspondingly large, by making use of sparse-matrix formats. There are functions for centering and scaling (see Section 9), and for making predictions from a fitted model. The package also has a function svd .als for computing a low-rank SVD of a large sparse matrix, with row and/or column centering. More details can be found in the package Vignette on the first authors web page, at http://web.stanford.edu/~hastie/swData/softimpute/vignette.html.

8. Distributed Implementation

We provide a distributed version of softimpute-ALS (given in Algorithm 5.1), built upon the Spark cluster programming framework.

8.1 Design

The input matrix to be factored is split row-by-row across many machines. The transpose of the input is also split row-by-row across the machines. The current model (i.e. the current guess for A, B) is repeated and held in memory on every machine. Thus the total time taken by the computation is proportional to the number of non-zeros divided by the number of CPU cores, with the restriction that the model should fit in memory.

At every iteration, the current model is broadcast to all machines, such that there is only one copy of the model on each machine. Each CPU core on a machine will process a partition of the input matrix, using the local copy of the model available. This means that even though one machine can have many cores acting on a subset of the input data, all those cores can share the same local copy of the model, thus saving RAM. This saving is especially pronounced on machines with many cores.

The implementation is available online at http://git.io/sparkfastals with documentation, in Scala. The implementation has a method named multByXstar, corresponding to line 3 of Algorithm 5.1 which multiplies X^* by another matrix on the right, exploiting the "sparse-plus-low-rank" structure of X^*. This method has signature:

multByXstar(X: IndexedRowMatrix, A: BDM[Double], B: BDM[Double], C:BDM[Double])

This method has four parameters. The first parameter X is a distributed matrix consisting of the input, split row-wise across machines. The full documentation for how this matrix is spread across machines is available online[4]. The multByXstar method takes a distributed matrix, along with local matrices A, B, and C, and performs line 3 of Algorithm 5.1 by multiplying X^* by C. Similarly, the method multByXstarTranspose performs line 5 of Algorithm 5.1.

After each call to multByXstar, the machines each will have calculated a portion of A. Once the call finishes, the machines each send their computed portion (which is small and can fit in memory on a single machine, since A can fit in memory on a single machine) to the master node, which will assemble the new guess for A and broadcast it to the worker machines. A similar process happens for multByXstarTranspose, and the whole process is repeated every iteration.

8.2 Experim ents

We report iteration times using an Amazon EC2 cluster with 10 slaves and one master, of instance type "c3.4xlarge". Each machine has 16 CPU cores and 30 GB of RAM. We ran softimpute-ALS on matrices of varying sizes with iteration runtimes available in Table 1, setting $k = 5$. Where possible, hardware acceleration was used for local linear algebraic operations, via breeze and BLAS.

The popular Netflix prize matrix has 17, 770 rows, 480,189 columns, and 100, 480, 507 non-zeros. We report results on several larger matrices in Table 1, up to 10 times larger.

9. Centering and Scaling

We often want to remove row and/or column means from a matrix before performing a low-rank SVD or running our matrix completion algorithms. Likewise we may wish to standardize the rows and or columns to have unit variance. In this section we present an

4.https://spark.apache.Org/docs/latest/mllib-basics.html#indexedrowmatrix

algorithm for doing this, in a way that is sensitive to the storage requirement of very large, sparse matrices. We first present our approach, and then discuss implementation details.

We have a two-dimensional array $X = \{X_{ij}\} \in \mathbb{R}^{m \times n}$, with pairs $(i, j) \in \Omega$ observed and the rest missing. The goal is to standardize the rows and columns of X to mean zero and variance one simultaneously. We consider the mean/variance model

$$X_{ij} \sim (\mu_{ij}, \sigma_{ij}^2) \quad (58)$$

with

$$\mu_{ij} = \alpha_i + \beta_j; \quad (59)$$

$$\sigma_{ij} = \tau_i \gamma_j. \quad (60)$$

Given the parameters of this model, we would standardized each observation via

$$\begin{aligned} \tilde{X}_{ij} &= \frac{X_{ij} - \mu_{ij}}{\sigma_{ij}} \\ &= \frac{X_{ij} - \alpha_i - \beta_j}{\tau_i \gamma_j}. \end{aligned} \quad (61)$$

If model (58) were correct, then each entry of the standardized matrix, viewed as a realization of a random variable, would have population mean/variance (0,1). A consequence would be that realized rows and columns would also have means and variances with expected values zero and one respectively. However, we would like the observed data to have these row and column properties.

Our representation (59)–(60) is not unique, but is easily fixed to be so. We can include a constant μ_0 in (59) and then have α_i and β_j average 0. Likewise, we can have an overall scaling σ_0, and then have log τ_i and log γ_j average 0. Since this is not an issue for us, we suppress this refinement.

We are not the first to attempt this dual centering and scaling. Indeed, Olshen and Rajaratnam (2010) implement a very similar algorithm for complete data, and discuss convergence issues. Our algorithm differs in two simple ways: it allows for missing data, and it learns the parameters of the centering/scaling model (61) (rather than just applying them). This latter feature will be important for us in our matrix-completion applications; once we have estimated the missing entries in the standardized matrix $\tilde{X}$, we will want to *reverse* the centering and scaling on our predictions.

In matrix notation we can write our model

$$\widetilde{\mathbf{X}} = \mathbf{D}_{\tau}^{-1}\left(\mathbf{X} - \boldsymbol{\alpha}\mathbf{1}^{T} - \mathbf{1}\boldsymbol{\beta}^{T}\right)\mathbf{D}_{\gamma}^{-1}, \quad (62)$$

where $\mathbf{D}_{\tau} = \mathrm{diag}(\tau_1, \tau_2, \ldots, \tau_m)$, similar for $\mathbf{D}_{\gamma}$, and the missing values are represented in the full matrix as NAs (e.g. as in R). Although it is not the focus of this paper, this centering model is also useful for large, complete, sparse matrices $\mathbf{X}$ (with many zeros, stored in sparse-matrix format). Centering would destroy the sparsity, but from (62) we can see we can store it in "sparse-plus-low-rank" format. Such a matrix can be left and right-multiplied easily, and hence is ideal for alternating subspace methods for computing a low-rank SVD. The function svd .als in the softImpute package (section 7) can accommodate such structure.

9.1 Method-of-moments Algorithm

We now present an algorithm for estimating the parameters. The idea is to write down four systems of estimating equations that demand that the transformed observed data have row means zero and variances one, and likewise for the columns. We then iteratively solve these equations, until all four conditions are satisfied simultaneously. We do not in general have any guarantees that this algorithm will always converge except in the noted special cases, but empirically we typically see rapid convergence.

Consider the estimating equation for the row-means condition (for each row i)

$$\begin{aligned}\frac{1}{n_i}\sum_{j\in\Omega_i}\widetilde{X}_{ij} &= \frac{1}{n_i}\sum_{j\in\Omega_i}\frac{X_{ij}-\alpha_i-\beta_j}{\tau_i\gamma_j} \quad (63)\\ &= 0,\end{aligned}$$

where $\Omega_i = \{j|(i,j)\in\Omega\}$, and $n_i = |\Omega_i| \le n$. Rearranging we get

$$\alpha_i = \frac{\sum_{j\in\Omega_i}\frac{1}{\gamma_j}\left(X_{ij}-\beta_j\right)}{\sum_{j\in\Omega_i}\frac{1}{\gamma_j}}, \quad i = 1,\ldots,m. \quad (64)$$

This is a weighted mean of the partial residuals $X_{ij} - \beta_j$ with weights inversely proportional to the column standard-deviation parameters γ_j. By symmetry, we get a similar equation for β_j,

$$\beta_j = \frac{\sum_{i\in\Omega^j}\frac{1}{\tau_i}\left(X_{ij}-\alpha_i\right)}{\sum_{i\in\Omega^j}\frac{1}{\tau_i}}, \quad j = 1,\ldots,n, \quad (65)$$

where $\Omega^j = \{i|(i,j)\in\Omega\}$, and $m_j = |\Omega^j| \le m$.

Similarly, the variance conditions for the rows are

$$\begin{aligned}\frac{1}{n_i}\sum_{j\in\Omega_i}\tilde{X}_{ij}^2 &= \frac{1}{n_i}\sum_{j\in\Omega_i}\frac{\left(X_{ij}-\alpha_i-\beta_j\right)^2}{\tau_i^2\gamma_j^2} \\ &= 1,\end{aligned} \quad (66)$$

which simply says

$$\tau_i^2=\frac{1}{n_i}\sum_{j\in\Omega_i}\frac{\left(X_{ij}-\alpha_i-\beta_j\right)^2}{\gamma_j^2},\quad i=1,\ldots,m. \quad (67)$$

Likewise

$$\gamma_j^2=\frac{1}{m_j}\sum_{i\in\Omega^j}\frac{\left(X_{ij}-\alpha_i-\beta_j\right)^2}{\tau_i^2},\quad j=1,\ldots,n. \quad (68)$$

The method-of-moments estimators require iterating these four sets of equations (64), (65), (67), (68) till convergence. We monitor the following functions of the "residuals"

$$R=\sum_{i=1}^{m}\left[\frac{1}{n_i}\sum_{j\in\Omega_i}\tilde{X}_{ij}\right]^2+\sum_{j=1}^{n}\left[\frac{1}{m_j}\sum_{i\in\Omega^j}\tilde{X}_{ij}\right]^2 \quad (69)$$

$$+\sum_{i=1}^{m}\log^2\left(\frac{1}{n_i}\sum_{j\in\Omega_i}\tilde{X}_{ij}^2\right)+\sum_{j=1}^{n}\log^2\left(\frac{1}{m_j}\sum_{i\in\Omega^j}\tilde{X}_{ij}^2\right) \quad (70)$$

In experiments it appears that R converges to zero very fast, perhaps linear convergence. In Appendix B we show slightly different versions of these estimators which are more suitable for sparse-matrix calculations.

In practice we may not wish to apply all four standardizations, but instead a subset. For example, we may wish to only standardize columns to have mean zero and variance one. In this case we simply set the omitted centering parameters to zero, and scaling parameters to one, and skip their steps in the iterative algorithm. In certain cases we have convergence guarantees:

- Column-only centering and/or scaling. Here no iteration is required; the centering step precedes the scaling step, and we are done. Likewise for row-only.

- Centering only, no scaling. Here the situation is exactly that of an unbalanced two-way ANOVA, and our algorithm is exactly the Gauss-Seidel algorithm for fitting the two-way ANOVA model. This is known to converge, modulo certain degenerate situations.

For the other cases we have no guarantees of convergence.

We present an alternative sequence of formulas in Appendix B which allows one to simultaneously apply the transformations, and learn the parameters.

10. Discussion

We have presented a new algorithm for matrix completion, suitable for solving Problem (1) for very large problems, as long as the solution rank is manageably low. Our algorithm capitalizes on the different strengths and weakness in each of the popular alternatives:

- ALS has to solve a different regression problem for every row/column, because of their different amount of missingness, and this can be costly. softImpute-ALS solves a single regression problem once and simultaneously for all the rows/columns, because it operates on a filled-in matrix which is complete. Although these steps are typically not as strong as those of ALS, the speed advantage more than compensates.
- softImpute wastes time in early iterations computing a low-rank SVD of a far-from-optimal estimate, in order to make its next imputation. One can think of softImpute-ALS as simultaneously filling in the matrix at each alternating step, as it is computing the SVD. By the time it is done, it has the the solution sought by softImpute, but with far fewer iterations.

softImpute allows for an extremely efficient distributed implementation (Section 8), and hence can scale to large problems, given a sufficiently large computing infrastructure.

Acknowledgments

The authors thank Balasubramanian Narasimhan for helpful discussions on distributed computing in R. The first author thanks Andreas Buja and Stephen Boyd for stimulating “footnote” discussions that led to the centering/scaling in Section 9. Trevor Hastie was partially supported by grant DMS-1407548 from the National Science Foundation, and grant R01-EB001988–15 from the National Institutes of Health. Rahul Mazumder was funded in part by Columbia University’s start-up funds and a grant from the Betty-Moore Sloan Foundation.

Appendix A.: Proofs from Section 5.1

Here, we gather some proofs and technical details from Section 5.1.

A.1 Proof of Lemma 2

To prove this we begin with the following elementary result concerning a ridge regression problem:

Lemma 5 *Consider a ridge regression problem*

$$H(\beta) := \frac{1}{2}\|y - M\beta\|_2^2 + \frac{\lambda}{2}\|\beta\|_2^2 \quad (71)$$

with $\beta^* \in \arg\min_\beta H(\beta)$. *Then the following inequality is true*:

$$H(\beta) - H(\beta^*) = \frac{1}{2}(\beta - \beta^*)^T(M^T M + \lambda \mathbf{I})(\beta - \beta^*) = \frac{1}{2} \| M(\beta - \beta^*) \|_2^2 + \frac{\lambda}{2} \| \beta - \beta^* \|_2^2$$

Proof *The proof follows from the second order Taylor Series expansion of* $H(\beta)$:

$$H(\beta) = H(\beta^*) + \langle \nabla H(\beta^*), \beta - \beta^* \rangle + \frac{1}{2}(\beta - \beta^*)^T\left(M^T M + \lambda \mathbf{I}\right)(\beta - \beta^*)$$

and observing that $\nabla H(\beta^*) = 0$. ■

We will need to obtain a lower bound on the difference $F(A_{k+1}, B_k) - F(A_k, B_k)$. Towards this end we make note of the following chain of inequalities:

$$F(A_k, B_k) = g\left(A_k B_k^T\right) + \frac{\lambda}{2}\left(\|A_k\|_F^2 + \|B_k\|_F^2\right) \quad (72)$$

$$= Q_A\left(A_k \middle| A_k, B_k\right) \quad (73)$$

$$\geq \min_{Z_1} Q_A\left(Z_1 \middle| A_k, B_k\right) \quad (74)$$

$$= Q_A\left(A_{k+1} \middle| A_k, B_k\right) \quad (75)$$

$$\geq g\left(A_{k+1} B_k^T\right) + \frac{\lambda}{2}\left(\|A_{k+1}\|_F^2 + \|B_k\|_F^2\right) \quad (76)$$

$$= F\left(A_{k+1}, B_k\right) \quad (77)$$

where, Line (73) follows from (31), and (76) follows from (30).

Clearly, from Lines (77) and (72) we have (78)

$$F(A_k, B_k) - F(A_{k+1}, B_k) \geq Q_A(A_k | A_k, B_k) - Q_A(A_{k+1} | A_k, B_k) \quad (78)$$

$$= \frac{1}{2}\|(A_{k+1} - A_k)B_k^T\|_2^2 + \frac{\lambda}{2}\|A_{k+1} - A_k\|_2^2, \quad (79)$$

where, (79) follows from (78) using Lemma 5.

Similarly, following the above steps for the B-update we have:

$$F(A_k, B_k) - F(A_{k+1}, B_{k+1}) \geq \frac{1}{2}\|A_{k+1}(B_{k+1} - B_k)^T\|_2^2 + \frac{\lambda}{2}\|B_{k+1} - B_k\|_2^2. \quad (80)$$

Adding (79) and (80) we get (35) concluding the proof of the lemma.

A.2 Proof of Lemma 3

Let us use the shorthand Δ in place of $\Delta((A, B), (A^+, B^+))$ as defined in (37).

First of all observe that the result (35) can be replaced with $(A_k, B_k) \leftarrow (A, B)$ and $(A_{k+1}, B_{k+1}) \leftarrow (A^+, B^+)$. This leads to the following:

$$\begin{aligned} F(A, B) - F(A^+, B^+) \geq\ & \frac{1}{2}\left(\|(A - A^+)B^T\|_F^2 + \|A^+(B^+ - B)^T\|_F^2\right) \\ & + \frac{\lambda}{2}\left(\|A - A^+\|_F^2 + \|B^+ - B\|_F^2\right). \end{aligned} \quad (81)$$

First of all, it is clear that if A, B is a fixed point then $\Delta = 0$.

Let us consider the converse, i.e., the case when $\Delta = 0$. Note that if $\Delta = 0$ then each of the summands appearing in the definition of Δ is also zero. We will now make use of the interesting result (that follows from the Proof of Lemma 2) in (78) and (79) which says:

$$Q_A(A | A, B) - Q_A(A^+ | A, B) = \frac{1}{2}\|(A^+ - A)B^T\|_2^2 + \frac{\lambda}{2}\|A^+ - A\|_2^2.$$

Now the right hand side of the above equation is zero (since $\Delta = 0$) which implies that, $Q_A(A|A, B) - Q_A(A^+|A,B) = 0$. An analogous result holds true for B.

Using the nesting property (34), it follows that $F(A, B) = F(A_+, B_+)$—thereby showing that (A, B) is a fixed point of the algorithm.

A.3 Proof of Theorem 4

We make use of (35) and add both sides of the inequality over $k = 1, \ldots, K$, which leads to:

$$\sum_{i=1}^{K}\left(F(A_k, B_k) - F(A_{k+1}, B_{k+1})\right) \geq \sum_{k=1}^{K}\eta_k \geq K\left(\min_{K \geq k \geq 1}\eta_k\right) \quad (82)$$

Since, $F(A_k, B_k)$ is a decreasing sequence (bounded below) it converges to F^{∞} say. It follows that:

$$\begin{aligned}\sum_{i=1}^{K}\left(F(A_k, B_k) - F(A_{k+1}, B_{k+1})\right) &= F\left(A^1, B^1\right) - F\left(A^{K+1}, B^{K+1}\right) \\ &\leq F\left(A^1, B^1\right) - F^{\infty}\end{aligned} \quad (83)$$

Using (83) along with (82) we have the following convergence rate:

$$\min_{1 \leq k \leq K}\eta_k \leq \left(F\left(A^1, B^1\right) - F\left(A^{\infty}, B^{\infty}\right)\right)/K,$$

thereby completing the proof of the theorem.

A.4 Proof of Corollary 1

Recall the definition of η_k

$$\eta_k = \frac{1}{2}\left(\|\left(A_k - A_{k+1}\right)B_k^T\|_F^2 + \|A_{k+1}\left(B_k - B_{k+1}\right)^T\|_F^2\right) + \frac{\lambda}{2}\left(\|A_k - A_{k+1}\|_F^2 + \|B_k - B_{k+1}\|_F^2\right)$$

Since we have assumed that

$$\ell^U\mathbf{I} \succcurlyeq B_k^T B_k \succcurlyeq \ell^L\mathbf{I}, \quad \ell^U\mathbf{I} \succcurlyeq A_k^T A_k \succcurlyeq \ell^L\mathbf{I}, \quad \forall k$$

we then have:

$$\eta_k \geq \left(\frac{\ell^L}{2} + \frac{\lambda}{2}\right)\|A_k - A_{k+1}\|_F^2 + \left(\frac{\ell^L}{2} + \frac{\lambda}{2}\right)\|B_k - B_{k+1}\|_F^2.$$

Using the above in (82) and assuming that $\ell_L > 0$, we have the bound:

$$\min_{1 \leq k \leq K}\left(\|\left(A_k - A_{k+1}\right)\|_F^2 + \|B_k - B_{k+1}\|_F^2\right) \leq \frac{2}{\left(\ell^L + \lambda\right)}\left(\frac{F\left(A^1, B^1\right) - F^{\infty}}{K}\right) \quad (84)$$

Suppose instead of the proximity measure:

$$\left(\|(A_k - A_{k+1})\|_F^2 + \|B_k - B_{k+1}\|_F^2\right),$$

we use the proximity measure:

$$\left(\|(A_k - A_{k+1})B_k^T\|_F^2 + \|A_{k+1}(B_k - B_{k+1})\|_F^2\right).$$

Then observing that:

$$\ell^U\|(A_k - A_{k+1})\|_F^2 \geq \|(A_k - A_{k+1})B_k^T\|_F^2, \quad \ell^U\|B_k - B_{k+1}\|_F^2 \geq \|A_{k+1}(B_k - B_{k+1})^T\|_F^2$$

we have:

$$\eta_k \geq \left(\frac{\lambda}{2\ell^U} + \frac{1}{2}\right)\left(\|(A_k - A_{k+1})B_k^T\|_F^2 + \|A_{k+1}(B_k - B_{k+1})\|_F^2\right).$$

Using the above bound in (82) we arrive at a bound which is similar in spirit to (41) but with a different proximity measure on the step-sizes:

$$\min_{1 \leq k \leq K}\left(\|(A_k - A_{k+1})B_k^T\|_F^2 + \|A_{k+1}(B_k - B_{k+1})\|_F^2\right) \leq \frac{2\ell^U}{\lambda + \ell_U}\left(\frac{F(A^1, B^1) - F^\infty}{K}\right) \quad (85)$$

It is useful to contrast results (41) and (42) with the case $\lambda = 0$.

$$\min_{1 \leq k \leq K}\left(\|(A_k - A_{k+1})B_k^T\|_F^2 + \|A_{k+1}(B_k - B_{k+1})\|_F^2\right) \leq \begin{cases} \frac{2\ell^U}{\lambda + \ell_U}\left(\frac{F(A^1, B^1) - F^\infty}{K}\right) & \lambda > 0 \\ 2\ell^U\left(\frac{F(A^1, B^1) - F^\infty}{K}\right) & \lambda = 0 \end{cases}$$

(86)

The convergence rate with the other proximity measure on the step-sizes have the following two cases:

$$\min_{1 \le k \le K}\left(\|(A_k - A_{k+1})\|_F^2 + \|B_k - B_{k+1}\|_F^2\right) \le \begin{cases} \frac{2}{(\ell^L + \lambda)}\left(\frac{F(A^1, B^1) - F^\infty}{K}\right) & \lambda > 0, \\ \frac{2}{\ell^L}\left(\frac{F(A^1, B^1) - F^\infty}{K}\right) & \lambda = 0. \end{cases} \quad (87)$$

The assumption (40) $\ell^U \mathbf{I} \succcurlyeq B_k^T B_k$ and $\ell^U \mathbf{I} \succcurlyeq A_k^T A_k$ can be interpreted as an upper bounds to the locally Lipschitz constants of the gradients of $Q_A(Z|A_k, B_k)$ and $Q_B(Z|A_{k+1}, B_k)$ for all k:

$$\begin{aligned} \|\nabla Q_A(A_{k+1}|A_k, B_k) - \nabla Q_A(A_k|A_k, B_k)\| &\le \ell^U \|A_{k+1} - A_k\|, \\ \|\nabla Q_B(B_k|A_{k+1}, B_k) - \nabla Q_B(B_{k+1}|A_{k+1}, B_k)\| &\le \ell^U \|B_{k+1} - B_k\|. \end{aligned} \quad (88)$$

The above leads to convergence rate bounds on the (partial) gradients of the function $F(A, B)$, i.e.,

$$\min_{1 \le k \le K}\left(\|\nabla_A f(A_k, B_k)\|^2 + \|\nabla_B f(A_{k+1}, B_k)\|^2\right) \le \frac{2(\ell^U)^2}{(\ell^L + \lambda)}\left(\frac{F(A^1, B^1) - F^\infty}{K}\right)$$

A.5 Proof of Theorem 5

Proof Part (a):

We make use of the convergence rate derived in Theorem 4. As $k \to \infty$, it follows that $\eta_k \to 0$. This describes the fate of the objective values $F(A_k, B_k)$, but does not inform us about the properties of the sequence A_k, B_k. Towards this end, note that if $\lambda > 0$, then the sequence A_k, B_k is bounded and thus has a limit point. Let A_*, B_* be any limit point of the sequence A_k, B_k. It follows by a simple subsequence argument that $F(A_k, B_k) \to F(A_*, B_*)$ and A_*, B_* is a fixed point of Algorithm 5.1 and in particular a first order stationary point of Problem (6).

Part (b):

The sequence (A_k, B_k) need not have a unique limit point, however, we show below: for every subsequence of B_k that converges, the corresponding subsequence of A_k also converges.

Suppose, $B_k \to B_*$ (along a subsequence $k \in \nu$). We will show that the sequence A_k for $k \in \nu$ has a unique limit point.

We argue by the method of contradiction. Suppose there are two limit points of A_k, $k \in \nu$, namely, $A_1$and A_2 and $A_{k_1} \to A_1, k_1 \in \nu_1 \subset \nu$ and $A_{k_2} \to A_2, k_2 \in \nu_2 \subset \nu$ with $A_1 \neq A_2$.

Consider the objective value sequence: $F(A_k, B_k)$. For fixed B_k the update in A from A_k to A_{k+1} results in

$$F\left(A_k, B_k\right) - F\left(A_{k+1}, B_k\right) \geq \frac{\lambda}{2}\|A_k - A_{k+1}\|_F^2.$$

Take $k_1 \in \nu_1$ and $k_2 \in \nu_2$, we have:

$$F\left(A_{k_2}, B_{k_2}\right) - F\left(A_{k_1+1}, B_{k_1}\right) = \left(F\left(A_{k_2}, B_{k_2}\right) - F\left(A_{k_2}, B_{k_1}\right)\right) \quad (89)$$
$$+ \left(F\left(A_{k_2}, B_{k_1}\right) - F\left(A_{k_1+1}, B_{k_1}\right)\right)$$

$$\geq \left(F\left(A_{k_2}, B_{k_2}\right) - F\left(A_{k_2}, B_{k_1}\right)\right) + \frac{\lambda}{2}\|A_{k_2} - A_{k_1+1}\|_F^2 \quad (90)$$

where Line 90 follows by using Lemma 5. As $k_1, k_2 \to \infty$, $B_{k_2}, B_{k_1} \to B_*$ hence,

$$F\left(A_{k_2}, B_{k_2}\right) - F\left(A_{k_2}, B_{k_1}\right) \to 0, \text{ and } \|A_{k_2} - A_{k_1+1}\|_F^2 \to \|A_2 - A_1\|_F^2$$

However, the lhs of (89) converges to zero, which is a contradiction. This implies that $\|A_2 - A_1\|_F^2 = 0$ i.e. A_k for $k \in \nu$ has a unique limit point.

Exactly the same argument holds true for the sequence A_k, leading to the conclusion of the other part of Part (b). ■

Appendix B.: Alternative Computing Formulas for Method of Moments

In this section we present the same algorithm, but use a slightly different representation. For matrix-completion problems, this does not make much of a difference in terms of computational load. But we also have other applications in mind, where the large matrix X may be fully observed, but is very sparse. In this case we do not want to actually apply the centering operations; instead we represent the matrix as a “sparse-plus-low-rank” object, a class for which we have methods for simple row and column operations.

Consider the row-means (for each row i). We can introduce a change Δ_i^α from the old α_i^o to the new α_i. Then we have

$$\sum_{j \in \Omega_i} \widetilde{X}_{ij} = \sum_{j \in \Omega_i} \frac{X_{ij} - \alpha_i^o - \Delta_i^\alpha - \beta_j}{\tau_i \gamma_j} \quad (91)$$

$$= 0,$$

where as before $\Omega_i = \{j | (i, j) \in \Omega\}$. Rearranging we get

$$\Delta_i^\alpha = \frac{\sum_{j \in \Omega_i} \widetilde{X}_{ij}^o}{\sum_{j \in \Omega_i} \frac{1}{\tau_i \gamma_j}}, \quad i = 1, \ldots, m, \quad (92)$$

where

$$\widetilde{X}_{ij}^o = \frac{X_{ij} - \alpha_i^o - \beta_j}{\tau_i \gamma_j}. \quad (93)$$

Then $\alpha_i = \alpha_i^o + \Delta_i^\alpha$. By symmetry, we get a similar equation for Δ_j^β,

Likewise for the variances.

$$\frac{1}{n_i} \sum_{j \in \Omega_i} \widetilde{X}_{ij}^2 = \frac{1}{n_i} \sum_{j \in \Omega_i} \frac{\left(X_{ij} - \alpha_i - \beta_j\right)^2}{\left(\tau_i \Delta_i^\tau\right)^2 \gamma_j^2} \quad (94)$$

$$= \frac{1}{n_i} \sum_{j \in \Omega_i} \left(\frac{\widetilde{X}_{ij}^o}{\Delta_i^\tau}\right)^2 \quad (95)$$

$$= 1.$$

Here we modify τ_i by a multiplicative factor Δ_i^τ. Here the solution is

$$\left(\Delta_i^\tau\right)^2 = \frac{1}{n_i} \sum_{j \in \Omega_i} \left(\widetilde{X}_{ij}^o\right)^2, \quad i = 1, \ldots, m. \quad (96)$$

By symmetry, we get a similar equation for Δ_j^γ,

The method-of-moments estimators amount to iterating these four sets of equations till convergence. Now we can monitor the changes via

$$R = \sum_{i=1}^{m} \Delta_i^{\alpha 2} + \sum_{j=1}^{n} \Delta_j^{\beta 2} + \sum_{i=1}^{m} \log^2 \Delta_i^{\tau} + \sum_{j=1}^{n} \log^2 \Delta_j^{\gamma} \quad (97)$$

which should converge to zero.

References

Bertsekas Dimitri P. Nonlinear Programming. Athena Scientific, Belmont, Massachusetts, 2nd edition, 1999 ISBN 1886529000. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1886529000.

Burer Samuel and Monteiro Renato D.C.. Local minima and convergence in low-rank semidefinite programming. Mathematical Programming, 103(3):427–631, 2005.

Emmanuel Candès and Benjamin Recht. Exact matrix completion via convex optimization. Foundations of Computational Mathematics, 2008. doi: 10.1007/s10208-009-9045-5. URL 10.1007/s10208-009-9045-5.

Emmanuel J Candès and Terence Tao. The power of convex relaxation: Near-optimal matrix completion, 2009 URL http://www.citebase.org/abstract?id=oai:arXiv.org:0903.1476.

Chen Caihua, He Bingsheng, and Yuan Xiaoming. Matrix completion via an alternating direction method. IMA Journal of Numerical Analysis, 32(1):227–245, 2012.

Golub G and Van Loan C. Matrix Computations. Johns Hopkins University Press, 3 edition, 2012.

Hardt Moritz. Understanding alternating minimization for matrix completion. In Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on, pages 651–660. IEEE, 2014.

Hastie Trevor and Mazumder Rahul. softImpute: Matrix Completion via Iterative Soft-Thresholded Svd, 2013 URL http://CRAN.R-project.org/package=softImpute. R package version 1.0.

Hastie Trevor, Tibshirani Robert, and Friedman Jerome. The Elements of Statistical Learning, Second Edition: Data Mining, Inference, and Prediction (Springer Series in Statistics). Springer New York, 2 edition, 2009 ISBN 0387848576.

Jain Prateek, Netrapalli Praneeth, and Sanghavi Sujay. Low-rank matrix completion using alternating minimization. In Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, pages* 665–674. ACM, 2013.

Michel Journée F Bach P-A Absil, and Sepulchre Rodolphe. Low-rank optimization on the cone of positive semidefinite matrices. SIAM Journal on Optimization, 20(5):2327–2351, 2010.

Koren Yehuda, Bell Robert, and Volinsky Chris. Matrix factorization techniques for recommender systems. Computer, 42(8):30–37, 2009.

Larsen RM. Propack-software for large and sparse svd calculations, 2004 URL http://sun.stanford.edu/~rmunk/PROPACK/.

Lewis A. Derivatives of spectral functions. Mathematics of Operations Research, 21(3): 576–588, 1996.

Mazumder Rahul, Hastie Trevor, and Tibshirani Rob. Spectral regularization algorithms for learning large incomplete matrices. Journal of Machine Learning Research, 11:2287–2322, 2010. [PubMed: 21552465]

Mirsky Leon. A trace inequality of John von Neumann. Monatshefte für Mathematik, 79 (4):303–306, 1975.

Olshen Richard and Rajaratnam Bala. Successive normalization of rectangular arrays. Annals of Statistics, 38(3):1638–1664, 2010. [PubMed: 20473354]

Razaviyayn Meisam, Hong Mingyi, and Luo Zhi-Quan. A unified convergence analysis of block successive minimization methods for nonsmooth optimization. SIAM Journal on Optimization, 23(2):1126–1153, 2013.

Rennie J and Srebro N. Fast maximum margin matrix factorization for collaborative prediction. In ICML, 2005.

Srebro Nathan, Rennie Jason, and Jaakkola Tommi. Maximum margin matrix factorization. Advances in Neural Information Processing Systems, 17, 2005.

Stewart G and Sun Ji-Guang. Matrix Perturbation Theory. Academic Press, Boston, 1 edition, 1990 ISBN 0126702306. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0126702306.

Zhou Yunhong, Wilkinson Dennis, Schreiber Robert, and Pan Rong. Large-scale parallel collaborative filtering for the netflix prize. In Algorithmic Aspects in Information and Management, pages 337–348. Springer, 2008.

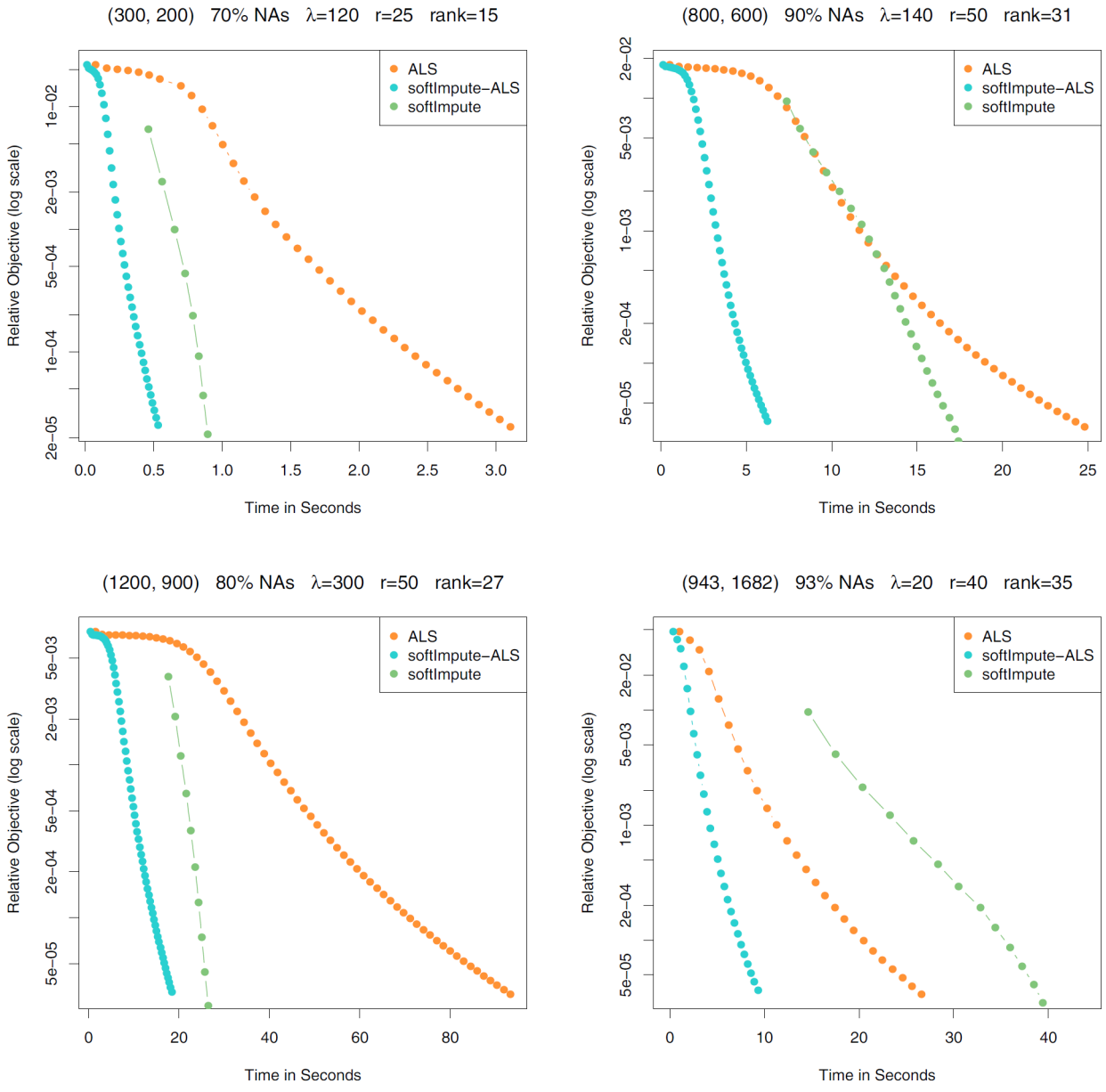

Figure 1:

Four timing experiments. Each figure is labelled according to size ($m \times n$), percentage of missing entries (NAs), value of λ used, rank r used in the ALS iterations, and rank of solution found. The first three are simulation examples, with increasing dimension. The last is the movielens 100K data. In all cases, softImpute-ALS (blue) wins handily against ALS (orange) and softImpute (green).

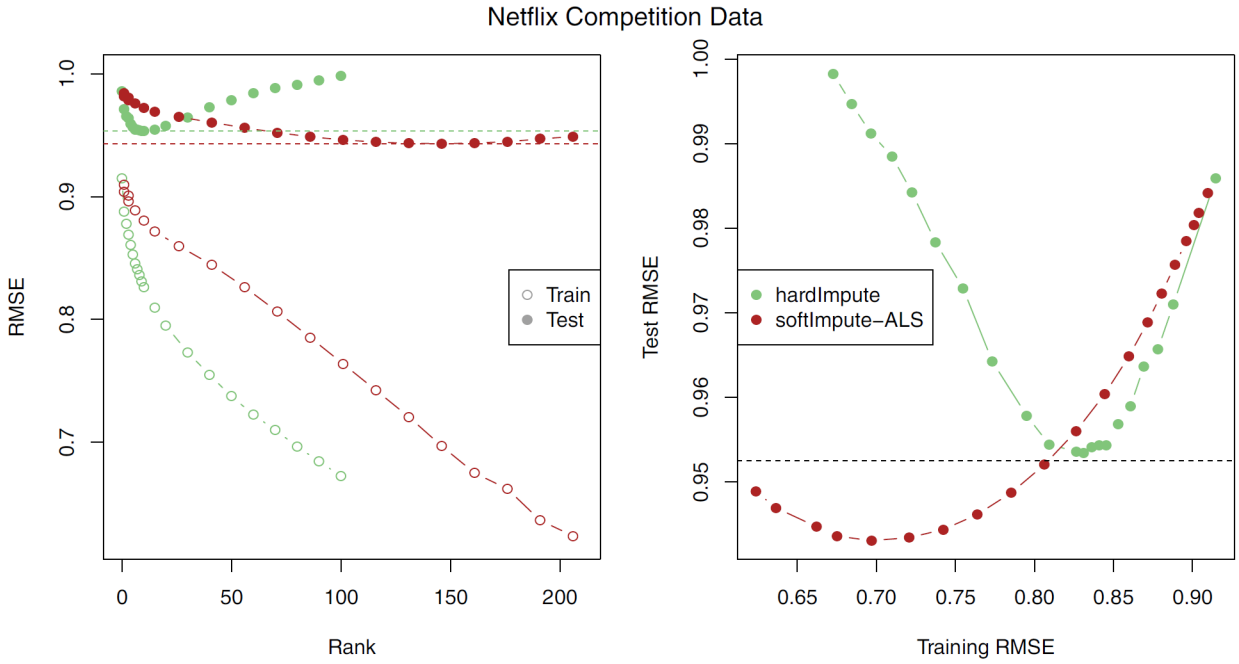

Figure 2:

Performance of hardImpute versus softImpute-ALS on the Netflix data. hardImpute uses a rank-restricted SVD at each step of the imputation, while softImpute-ALS does shrinking as well. The left panel shows the training and test error as a function of the rank of the solution —an imperfect calibration in light of the shrinkage. The right panel gives the test error as a function of the training error. hardImpute fits more aggressively, and overfits far sooner than softImpute-ALS. The horizontal dotted line is the “Cinematch” score, the target to beat in this competition.

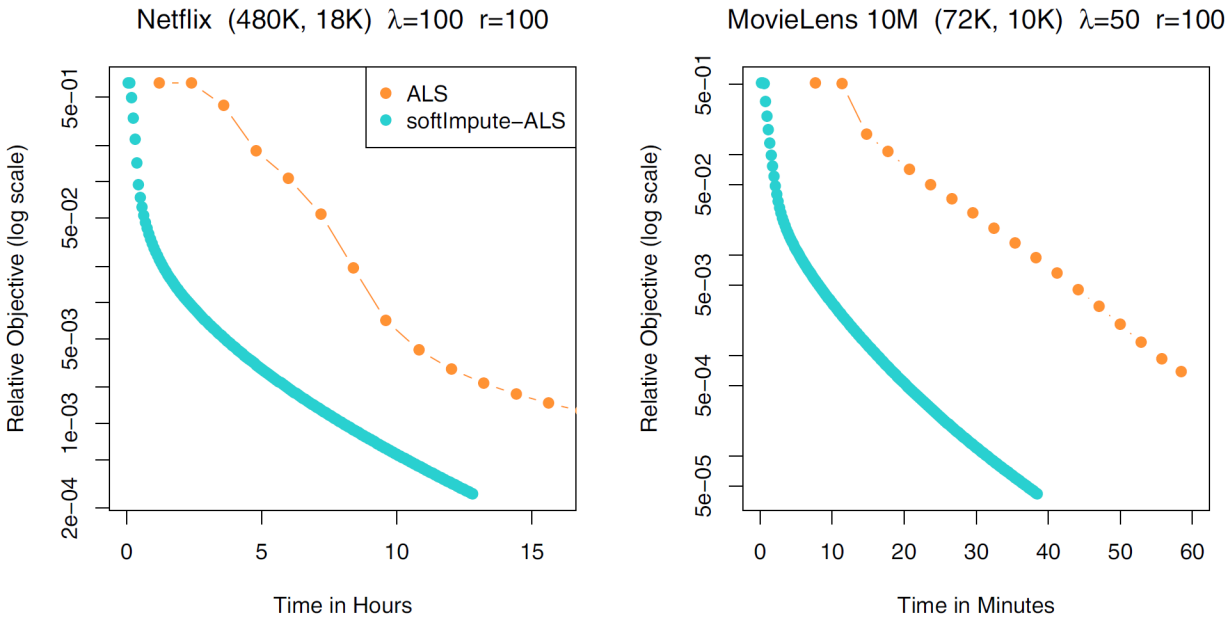

Figure 3:

Left: timing results on the Netflix matrix, comparing ALS with softImpute-ALS. Right: timing on the MovieLens 10M matrix. In both cases we see that while ALS makes bigger gains per iteration, each iteration is much more costly.

Table 1:

Running times for distributed softimpute-ALS

Matrix Size	Number of Nonzeros	Time per iteration (s)
$10^6 \times 10^6$	10^6	5
$10^6 \times 10^6$	10^9	6
$10^7 \times 10^7$	10^9	139