Algorithms for Completing a User Ratings Matrix

Nick Asendorf, Madison McGaffin, Matt Prelee, Ben Schwartz

1 Motivation

There are many instances when we are interested in predicting user preferences, such as online news personalization, movies, music, or other commodity recommender systems. In these types of problems, we are given a set of user item preferences. Then for a given user, we use preferences from like-minded users to calculate a prediction for the user of interest.

Movie prediction can be cast as a matrix completion problem. In this setting, there is an unknown matrix, R, whose columns represent users and whose rows represent movies. An entry of R is a rating by a particular user for a particular movie. However, we only observe a subset of these ratings. From these known ratings we wish to recover, or complete, the full matrix R.

We impose structure on R which assumes that it has low rank. This is equivalent to assuming there are only a few factors which determine a user's taste in movies. This low-rank assumption has been used successfully in other matrix completion contexts [2], [4].

2 Problem Statement

$\mathcal{U}$:	Set of users	$\mathcal{M}$:	Set of movies
n:	Number of users, $\lvert\mathcal{U}\rvert$	d:	Number of movies, $\lvert\mathcal{M}\rvert$
R:	User ratings matrix, $R \in \mathbb{R}^{d\times n}$	$\widehat{R}$:	Estimate of R, $\widehat{R} \in \mathbb{R}^{d\times n}$
k:	Rank of R	$\widehat{k}$:	Estimate of k
T:	Sparse matrix of training data, $T \in \mathbb{R}^{d\times n}$	S:	Sparse matrix of test data, $S \in \mathbb{R}^{d\times n}$
Ω_T:	Set of indices of the nonzero entries of T	Ω_S:	Set of indices of the nonzero entries of S

Table 1: Notation

We make a few notes on our notation. Let Ω be a set of matrix indices. For a matrix X, we define the matrix projection operator $\mathcal{P}_\Omega$ on X as $X' = \mathcal{P}_\Omega(X)$, where

$$X'_{ij} = \begin{cases} X_{ij}, & (i,j) \in \Omega \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Both the training and testing data matrices are obtained through this projection operation, i.e., $T = \mathcal{P}_{\Omega_T}(R)$ and $S = \mathcal{P}_{\Omega_S}(R)$. The zero entries of S and T represent the user-movie pairs for which we do not have a rating. S and T are disjoint. That is, training data does not appear in the testing data.

Our problem is as follows. Given our incomplete training dataset of user-movie ratings, find the missing user-movie ratings that are not in the training dataset. Mathematically, we can express this as a matrix completion problem:

$$\text{Find } \widehat{R}, \text{ an estimate of } R \text{ using } T. \tag{P1}$$

All algorithms considered will be evaluated using a root mean squared error (RMSE) performance metric. The RMSE will be calculated using the provided test ratings. Mathematically, that

is

$$\text{RMSE}(\widehat{R}, S) = \sqrt{\frac{1}{|\Omega_S|} \sum_{(i,j)\in\Omega_S} \left(\widehat{R}_{ij} - S_{ij}\right)^2} = \left\| \frac{1}{|\Omega_S|} \mathcal{P}_{\Omega_S}(\widehat{R} - S) \right\|_F. \tag{2}$$

$\|\cdot\|_F$ denotes the usual Frobenius norm.

3 Data

The dataset that we will consider is the MovieLens datasset collected by the GroupLens Research Project at the University of Minnesota. The data was collected through the MovieLens web site (`http://movielens.umn.edu`) during the seven-month period from September 19th, 1997 through April 22nd, 1998. We refer to the set of users in a dataset as $\mathcal{U}$, the set of movies in a dataset as $\mathcal{M}$, and the set of ratings in a dataset as $\mathcal{R}$. There are a total of $|\mathcal{U}| = 943$ users, $|\mathcal{M}| = 1682$ movies and $|\mathcal{R}| = 100,000$ ratings.

Each user has rated at least 20 movies and every movie was rated by at least one user. The data is provided in 5 partitions where each partition splits the data into a training dataset and a testing dataset. Each training dataset containes 80% of the ratings and each testing dataset contains the remaining 20%. The 5 testing datasets are disjoint sets. The training datasets contain 5.04% of the entries in the user-movie matrix.

4 Completion Algorithms

4.1 Mean-Based Algorithms

4.1.1 Global Mean Algorithm

Simply stated, this algorithm takes all of the known ratings in our training matrix, and replaces the unknown entries with the mean of all known ratings. Formally, let

$$\mu_{global} = \frac{1}{|\Omega_T|} \sum_{(i,j)\in\Omega_T} T_{ij}. \tag{3}$$

Then

$$\widehat{R} = \begin{cases} T_{ij}, & (i,j) \in \Omega_T \\ \mu_{global}, & \text{otherwise.} \end{cases} \tag{4}$$

4.1.2 User Mean Algorithm

This is another intuitive algorithm. For every user, calculate their average rating using entries from the training matrix. Estimate all other unknown ratings for that user using this average. Let $\Omega_{T(:,j)} \subset \Omega_T$ be the indices of the known entries for user (column) j.

$$\mu_{\text{user}}(j) = \frac{1}{|\Omega_{T(:,j)}|} \sum_{(i,j)\in\Omega_{T(:,j)}} T_{ij}. \tag{5}$$

Then

$$\widehat{R} = \begin{cases} T_{ij}, & (i,j) \in \Omega_T \\ \mu_{\text{user}}(j), & \text{otherwise.} \end{cases} \tag{6}$$

4.1.3 Movie Mean Algorithm

This is the movie analogue to the User Mean Algorithm. Let $\Omega_{T(i,:)} \subset \Omega_T$ be the indices of the known entries for movie (row) i.

$$\mu_{\text{movie}}(i) = \frac{1}{|\Omega_{T(i,:)}|} \sum_{(i,j) \in \Omega_{T(i,:)}} T_{ij}. \tag{7}$$

Then

$$\widehat{R} = \begin{cases} T_{ij}, & (i,j) \in \Omega_T \\ \mu_{\text{movie}}(i), & \text{otherwise.} \end{cases} \tag{8}$$

4.1.4 Mixture Mean Algorithm

This algorithm assumes that a user's rating is governed by the global, user, and movie means. Mathematically, we form our ratings through a convex combination of these basic means via

$$\widehat{R}_{ij} = \alpha_1 \mu_{\text{user}}(j) + \alpha_2 \mu_{\text{movie}}(i) \tag{9}$$

where $\alpha_1 \geq 0, \alpha_2 \geq 0$ and $\alpha_1 + \alpha_2 = 1$.

To determine the values for α_1 and α_2, we conducted a parameter sweep to determine which values produced the minimum RMSE for each data partition. These values of partition specific optimal values α_1 and α_2 were then averaged over the 5 partitions. The optimal weightings were $\alpha_{1_{\text{opt}}} = 0.452$ and $\alpha_{2_{\text{opt}}} = 0.548$. The resulting user ratings matrix estimate is

$$\widehat{R}_{ij} = 0.452 \mu_{\text{user}}(j) + 0.548 \mu_{\text{movie}}(i). \tag{10}$$

4.2 Singular Value Thresholding

The singular value thresholding (SVT) algorithm is an iterative algorithm that solves the following optimization problem:

$$\begin{aligned} &\text{minimize} \quad \|\widehat{R}\|_* \\ &\text{subject to} \quad \widehat{R}_{ij} = T_{ij}, \qquad (i,j) \in \Omega_T. \end{aligned} \tag{11}$$

$\|\widehat{R}\|_*$ denotes the nuclear norm of $\widehat{R}$, i.e. the sum of the singular values of $\widehat{R}$. The minimization of $\|\widehat{R}\|_*$ is a convex relaxation of minimizing the rank of $\widehat{R}$. As given in [2], the following algorithm produces a sequence of matrices $\{\widehat{R}^q\}_q$ that converge to the solution to the above problem. Choose a fixed $\tau > 0$ and a sequence $\{\delta_q\}$ of step sizes. Initialize $Y^0 = 0$ and iterate the following steps until a convergence criterion is reached:

$$\begin{cases} \widehat{R}^q = \text{shrink } (Y^{q-1}, \tau), \\ Y^q = Y^{q-1} + \delta_q \mathcal{P}_{\Omega_T}(T - \widehat{R}^q). \end{cases} \tag{12}$$

The shrink operator performs a soft threshold on the singular values of its operand. That is, if X has the usual singular value decomposition $X = U\Sigma V^T$, then shrink $(X, \tau) = U\Sigma' V^T$, where $\Sigma' = \text{diag}\,\{(\sigma_i - \tau)_+\}$. The $(\cdot)_+$ operand denotes thresholding of negative values to zero.

The algorithm terminated when the relative error

$$\frac{\|\mathcal{P}_{\Omega_T}(\widehat{R} - T)\|_F}{\|\mathcal{P}_{\Omega_T}(T)\|_F} \leq \varepsilon \tag{13}$$

fell below a predetermined tolerance $\varepsilon > 0$.

4.2.1 Implementation and Basic Testing

The SVT algorithm was implemented in MATLAB according to Algorithm 1 in [2]. The free MATLAB package PROPACK was used to compute the rank-r SVD. To validate our implementation of this algorithm, we followed the testing procedure specified in [2]. Specifically, we formed a rank-r matrix R by generating two $n \times r$ matrices M_L and M_R, each being comprised of i.i.d. $\mathcal{N}(0, 1)$ Gaussian entries, and set $R = M_L M_R^T$. We successfully reproduced the results of the experiments specified in Table 1 of [2].

Finally, it is important to note that this algorithm fails when the training data is too sparse. Equation 1.2 in [2] states that if R is an $n \times n$ matrix with m known entries, then for some positive constant C,

$$m \geq Cn^{6/5} r \log n \tag{14}$$

ensures recovery of R. We simulated this behavior with our experiments with the i.i.d. Gaussian matrices. Indeed, below a certain sparsity, the algorithm does not converge

4.2.2 Parameter selection for ratings data

When running SVT on the MovieLens data, we chose our step size $\delta_q = \delta = 1.9$ for all iterations q. Although this is a small step size, [2] proved that the algorithm is guaranteed to converge (albeit slowly) for $0 < \delta < 2$. We chose our thresholding parameter $\tau = 5\sqrt{nd}$, which is similar to the suggested value in [2]. Finally, our relative error tolerance was $\varepsilon = 1 \times 10^{-4}$.

4.2.3 Quick SVT

To avoid overfitting, we also ran experiments with $\varepsilon = 0.2$. This not only decreased our testing error, but it also drastically cut down on the number of iterations required. Runtime was decreased from approximately 4.5 hours to 15 seconds on an Intel i7 processor. In Figure 1(a), note the minimum in testing error around the 150th iteration. This suggests that the SVT is overfitting the training error for small values of ε. Choosing a larger value of ε leads to an $\widehat{R}$ with a much smaller rank.

4.3 Weighted sum of others' ratings

We implemented the "weighted sum of others' ratings" (WSOR) collaborative filtering algorithm from Section 3.2.1 of [8]. Unlike the other algorithms considered in this project, WSOR does not

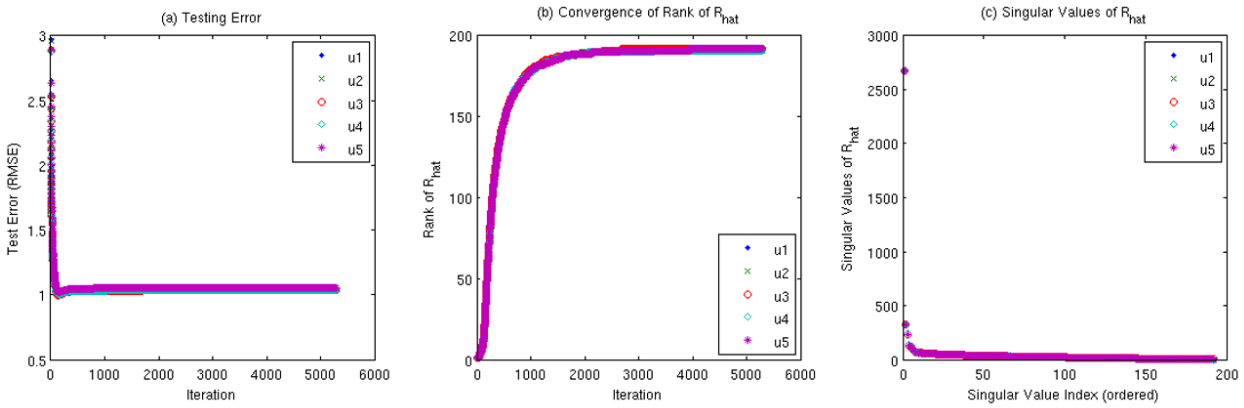

Figure 1: Results of running the SVT algorithm ($\varepsilon = 1 \times 10^{-4}$ as described in Section 4.2) to convergence. (a) shows the testing error at each iteration. (b) shows the rank of $\widehat{R}$ at each iteration. (c) plots the singular values of the final $\widehat{R}$.

make a low-rank assumption about R and instead estimates the j^{th} user's rating of movie i as a weighted sum of other users' ratings of the movie.

Define the user similarly matrix W with entries $W_{jk} = f\left(T(:,k), T(:,j)\right)$, where $f(u,v)$ is the vector cosine of u and v restricted to their nonzero entries:

$$f\left(u,v\right) = \frac{\sum_{j \in \Omega_u \cap \Omega_v} u_j v_j}{\sqrt{\sum_{j \in \Omega_u \cap \Omega_v} u_j^2} \sqrt{\sum_{j \in \Omega_u \cap \Omega_v} v_j^2}}. \tag{15}$$

The WSOR algorithm given in Figure 2 uses W to iteratively fill in estimates for each unranked movie using rankings from similar users.

4.4 K-SVD

The K-SVD algorithm [1] is an iterative procedure that attempts to solve the following non-convex sparse representation problem:

$$\begin{aligned} \underset{D,A}{\text{argmin}} \quad & ||T - DA||_F^2 \\ \text{subject to} \quad & ||a_i||_0 \leq L, \forall i, \end{aligned} \tag{17}$$

where the matrix $D \in \mathbb{R}^{d \times p}$ is commonly called a dictionary. The K-SVD algorithm draws inspiration from K-means, which solves problem (17) when $L = 1$ and the columns of A are restricted to be unit-norm. K-means alternates between updating the mean vectors and data associations; K-SVD follows a similar structure, alternating between updating the dictionary D and the coefficents A.

K-SVD has been applied to image and video denoising and inpainting, which differs from the matrix completion problem mostly in dimension and sparsity. Commonly for image- and video-processing problems, the data matrix T is overdetermined whereas the ratings matrix has more rows (movies) than columns (users). Furthermore, the data in image- and video-processing are

- Given T_0, the initial sparse movie ratings matrix.
- Scale the columns of T_0: $\widehat{T} = T_0 \cdot \text{diag}\left\{\frac{1}{\mu_{\text{user}}(j)}\right\}$.
- Repeat until no entries are updated:
  1. Build the user similarity matrix W from $\widehat{T}$ using (15).
  2. For unknown or unestimated each user j and movie i in $\widehat{T}$, estimate:

$$\widehat{T}_{ij} \leftarrow \frac{\sum_k w_{kj}\widehat{T}_{ik}}{\sum_k w_{kj}}. \tag{16}$$

     If the denominator of (16) is zero, do not perform the update.
- Scale the columns of T again: $\tilde{T} = \widehat{T} \cdot \text{diag}\ \{\mu_{\text{user}}(j)\}$.
- Fill in remaining unestimated values with the mixture mean estimate:

$$\widehat{R}_{ij} = \begin{cases} \tilde{T}_{ij}, & (i,j) \in \Omega_{\tilde{T}} \\ 0.452\mu_{\text{user}}(j) + 0.548\mu_{\text{movie}}(i), & \text{otherwise} \end{cases}$$

Figure 2: The weighted sum of others' ratings (WSOR) algorithm.

often more complete. Consequently, we consider the ratings matrix completion problem to be "more difficult."

4.4.1 Implementation details

The version of the K-SVD algorithm described in Figure 3 draws inspiration from the approximate K-SVD algorithm in [7] to provide a parallelizable version of the original K-SVD algorithm. In the original, the dictionary update step involves a rank-1 SVD of E_k and simultaneously updates A, incurring high computational cost and precluding parallelization. The parallel approximate K-SVD algorithm was implemented in multithreaded C++ and made accessible to MATLAB via a `mex`-file.

4.4.2 Parameter selection

We selected empirically optimal values of L and p, the representation rank and number of dictionary columns, by finding the minimum RMSE over a grid of values. The optimum was surprisingly reached at $L = 2$ and $p = 2$. When $L = p$, Problem (17) reduces to the problem solved by SVD-based methods.

- Let D^0 be some initial dictionary.
- Let $q = 1$ and iterate until convergence:
  1. Let A^q be the output of some vector selection algorithm applied to T and D^{q-1}. That is, let A^q be some approximate solution to problem (17), holding D constant.
  2. For each column d_k of D^{q-1}:
     (a) Compute the residual $E_k = T - \widehat{D}_{(k)}^{q-1} A$, where $\widehat{D}_{(k)}^{q-1}$ is equal to D^{q-1} with the kth column set to zero.
     (b) Let $g_k = E_k E_k^T d_k$.
     (c) Update $d_k \leftarrow g_k / ||g_k||$.

Figure 3: The parallel approximate K-SVD algorithm.

4.5 SVD Based Approaches

SVD approaches assume that users' ratings are governed by a relatively small number of factors, k. Mathematically, we assume the following model:

$$R = UX + Z$$

where $U \in \mathbb{R}^{d \times k}$ whose columns represent governing factors, $X \in \mathbb{R}^{k \times n}$ whose columns represent the user specific weights for each factor, and $Z \in \mathbb{R}^{d \times n}$ is a matrix of noise.

Given this model, a naïve approach to estimate R from the training data matrix, T, is to solve

$$\begin{aligned} &\text{minimize} \quad \|\widehat{R} - T\|_F \\ &\text{subject to} \quad \text{rank}(\widehat{R}) = k. \end{aligned} \tag{18}$$

The Eckhard-Young-Mirsky theorem states that the solution to this problem is the rank-k SVD of T. However, k is unknown so we instead take the rank-$\widehat{k}$ SVD of T, where $\widehat{k}$ is an estimate of k. We employ two strategies to estimate k.

4.5.1 Optimal $\widehat{k}$ Selection

The first strategy computes a rank-n SVD from T, yielding $U \in \mathbb{R}^{d \times n}, \Sigma \in \mathbb{R}^{n \times n}, V \in \mathbb{R}^{n \times n}$. Then for $k = 1, \ldots, n$, the rank-k reconstruction of T is used as our user ratings matrix estimate. That is, $\widehat{R}_k = U(:, 1{:}k)\Sigma(1{:}k, 1{:}k)V(:, 1{:}k)^T$. Our estimate, $\widehat{k}_{\text{opt}}$, is formed by selecting the k whose user ratings matrix estimate results in the lowest RMSE. That is,

$$\widehat{k}_{\text{opt}} = \underset{k}{\text{argmin}}\ \text{RMSE}(\widehat{R}_k). \tag{19}$$

4.5.2 Random Matrix Theory Selection of $\widehat{k}$

The second strategy employs random matrix theory which characterizes the accuracy of the singular value and singular vector estimates. These estimates are inaccurate due to noise introduced by

the missing entries of the training data and the randomness in user rankings. To apply these random matrix theory ideas, we make additional assumptions. First, we assume Gaussian noise, that is $Z_{ij} \sim \mathcal{N}(0, \sigma_n^2)$. Second, we assume that the columns of X, x_i , are governed by $x_i \sim \mathcal{N}(0, \Lambda)$ where $\Lambda = \text{diag}\,\{\lambda_1^2, \ldots, \lambda_k^2\}$ and $\lambda_1^2 > \cdots > \lambda_k^2 > 0$. These assumptions are most certainly inaccurate because this assumes that the covariance matrix, Λ, is the same for all users.

We extend Theorem 4 of [6]. Our modified theorem states that as $n, d \longrightarrow \infty$ with $d/n \to c$, when $\lambda_i^2 > \sqrt{c}$

$$\widehat{\lambda}_i^2 \sim f_{\widehat{\lambda}_i^2}(\lambda_i) := \mathcal{N}\left(\left(c\,\sigma_n^2 + \lambda_i^2 + \frac{c\,\sigma_n^4}{\lambda_i^2}\right), \frac{2\left(\lambda_i^2 + \sigma_n^2\right)^2}{n}\left(1 - \frac{c\,\sigma_n^8}{\lambda_i^4}\right)\right),$$

where $\widehat{\lambda}_i^2$ is the i^{th} largest eigenvalue of $\frac{1}{n}TT^T$. We form an improved estimate of the signal eigenvalue, λ_i^2, by employing maximum-likelihood (ML) estimation on $\widehat{\lambda}_i$. Specifically, we form the estimate:

$$\widehat{\lambda}_{i_{\text{rmt}}}^2 = \underset{\lambda_i^2}{\text{argmax}}\ \log\left(f_{\widehat{\lambda}_i^2}(\lambda_i)\right). \tag{20}$$

Let j be the first index such that $\widehat{\lambda}_{j_{\text{rmt}}}^2 > \widehat{\lambda}_{(j-1)_{\text{rmt}}}^2$. As we assumed that the feature variances, λ_i^2, are monotonically decreasing, this is the first j where this assumption is violated. Random matrix theory says that all eigenvectors v_k with $k > j$ are uninformative. The estimate for $\widehat{k}_{\text{rmt}}$ becomes

$$\widehat{k}_{\text{rmt}} = j - 1. \tag{21}$$

This dimensionality selection is completely data driven and not heuristic, which is very desirable. We also form an improved singular value estimate with

$$\Sigma_{\text{rmt}} = \text{diag}\,(\sqrt{m}\,\widehat{\lambda}_{1_{\text{rmt}}}, \ldots, \sqrt{m}\,\widehat{\lambda}_{\widehat{k}_{\text{rmt}}\text{rmt}}). \tag{22}$$

4.5.3 Implementation

Initial experimental results showed that generating user ratings matrix estimates from the rank-$\widehat{k}$ SVD of the training matrix T resulted in very poor RMSE values. Because T is very sparse, the low rank SVD overfits the abundance of zeros in T. To overcome this issue, we replaced the zeros in the the training matrix with the optimal mixture mean estimate in Section 4.1.4. The following augmented training matrix, $\tilde{T}$, was formed:

$$\tilde{T}_{ij} = \begin{cases} T_{ij}, & (i,j) \in \Omega_T \\ 0.452\mu_{\text{user}}(j) + 0.548\mu_{\text{movie}}(i) & \text{otherwise.} \end{cases} \tag{23}$$

Let $\tilde{T} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$ be the SVD of our augmented training matrix. Two user ratings matrix estimates were then formed. The first estimate used $\widehat{k}_{\text{opt}}$:

$$\widehat{R} = \tilde{U}(:, 1{:}\widehat{k}_{\text{opt}})\tilde{\Sigma}(:, 1{:}\widehat{k}_{\text{opt}})\tilde{V}(:, 1{:}\widehat{k}_{\text{opt}})^T. \tag{24}$$

The second estimate used $\widehat{k}_{\text{rmt}}$ with an experimentally determined $\sigma_n^2 = 0.12$:

$$\widehat{R} = \tilde{U}(:, 1{:}\widehat{k}_{\text{rmt}})\Sigma_{\text{rmt}}\tilde{V}(:, 1{:}\widehat{k}_{\text{rmt}})^T. \tag{25}$$

4.6 Robust Matrix Completion

In the case of robust matrix completion, we relax the assumption that R is low rank and instead assume that $R = L + B$, where L is low rank and B is sparse [3]. We observe $T = \mathcal{P}_{\Omega_T}(R)$, where $\mathcal{P}_{\Omega_T}$ is defined as in Section 4.2. To recover L we solve the following convex optimization problem:

$$\begin{aligned} \widehat{L} = \quad & \arg\min_{L,S} \|L\|_* + \lambda\|B\|_1 \\ \text{subject to} \quad & \mathcal{P}_{\Omega_T}(L+B) = R. \end{aligned} \tag{26}$$

We will use the Augmented Lagrangian method to solve this constrainted optimization problem. We can recast this problem as

$$\begin{aligned} \widehat{L} = \quad & \arg\min_{L,S} \|L\|_* + \lambda\|B\|_1 \\ \text{subject to} \quad & \begin{cases} L + B + E = R \\ \mathcal{P}_{\Omega_T}(E) = 0. \end{cases} \end{aligned} \tag{27}$$

Excluding the $\mathcal{P}_{\Omega_T}(E) = 0$ constraint, the partial Augmented Lagrangian is

$$\mathcal{L}(L, B, E, Y, \mu) = \|L\|_* + \lambda\|B\|_1 + \langle Y, L + B + E - R\rangle + \frac{\mu}{2}\|L + B + E - R\|_F^2, \tag{28}$$

where Y and μ are Lagrange variables. The inner product $\langle X, Y\rangle$ defined as the Euclidean inner product in $\mathbb{R}^{dn}$ induced by viewing matrices $X, Y \in \mathbb{R}^{d\times n}$ as vectors in $\mathbb{R}^{dn}$. The problem is convex, so we can solve it with the alternating directions method of multipliers.

This is an iterative algorithm in which we minimize the Lagrangian with respect to each variable whilst holding the others constant. It can be shown [5] that if each of the successive minimizations are implemented with finite total error, the algorithm will converge to a global minimum.

4.6.1 Implementation and testing of robust matrix completion

The robust matrix completion algorithm was implemented in MATLAB according to Figure 4. PROPACK's SVD routines were used. To test proper functionality of the code, we used the same tests as in Section 4.2.1. The algorithm was again able to recover the low rank matrix L with low RMSE, provided enough samples of the matrix were taken.

4.6.2 Parameter selection

Our algorithm involved choosing ρ, λ, and starting value for μ. The parameter ρ was chosen to be some number greater than but close to 1 in order to ensure that small enough step sizes would be taken. The algorithm was not especially sensitive to the choice of μ, so we set it to be inversely proportional to the largest singular value of T for numerical stability. For λ, contrary to [3] which advised setting $\lambda = 1/\sqrt{0.1 \cdot \max(n, d)}$, we found that choosing λ proportional to the largest singular value of T resulted in a better tradeoff between penalizing the rank of L and the sparsity of B. As a result, we also saw lower test error with this choice of λ.

- Define $D = R - L - E - \frac{1}{\mu}Y$, and let $\mathcal{P}_{\overline{\Omega}_T}$ be defined similar to $\mathcal{P}_{\Omega_T}$, but onto the complement of Ω_T. Initialize $Y = 0$, $\mu = 0$, $\rho > 1$, $E = 0$, and $B = 0$.
- Repeat until convergence:

1. $$L \leftarrow \text{shrink } \left(R - B - E - \frac{1}{\mu}Y, \frac{1}{\mu}\right)$$

2. $$B_{ij} \leftarrow \begin{cases} 0, & |d_{ij}| \leq \frac{\lambda}{\mu} \\ d_{ij} - \text{sign } (d_{ij})\frac{\lambda}{\mu}, & \text{otherwise} \end{cases}, \ \forall (i,j)$$

3. $$E \leftarrow \mathcal{P}_{\overline{\Omega}_T}\left(R - B - L - \frac{1}{\mu}Y\right)$$

4. $$Y \leftarrow Y + \mu(L + B + E - R)$$

5. $$\mu \leftarrow \rho\mu$$

Figure 4: The robust matrix completion alternating direction method of multipliers algorithm.

5 Analysis

5.1 Thresholding the entries of $\widehat{R}$

For all of our algorithms for estimating R, we can obtain a modified estimate $\widehat{R}'$ using a simple observation. We note that all entries of R must lie in the interval $[1, 5]$. Thus, for any estimated user ratings matrix $\widehat{R}$, we can produce a modified estimated user ratings matrix $\widehat{R}'$ by using the following thresholding rule:

$$\widehat{R}'_{ij} = \begin{cases} \widehat{R}_{ij}, & 1 \leq \widehat{R}_{ij} \leq 5 \\ 1, & R_{ij} < 1 \\ 5, & R_{ij} > 5 \end{cases} \tag{29}$$

The RMSE values for each algorithm after performing this thresholding step are shown in Table 2.

5.2 Overfitting and rank mismatch

When running the SVT algorithms, choosing a small tolerance for the training error led to a high rank solution. In Figure 1(b), notice that $\widehat{R}$ has a rank of nearly 200. However, evident from Figure 1(c), most of the matrix energy lies in a low-rank subspace. Even when we chose a larger value of ε, the rank of $\hat{R}$ is about 25.

We observed the same overfitting behavior in the K-SVD algorithm. We found that the optimal number of columns in D and column-sparsity of A were both 2. When the algorithm is allowed

Sec.	Algorithm	Part. 1	Part. 2	Part. 3	Part. 4	Part. 5	Average
4.1.1	Global Mean	1.1537	1.1307	1.1116	1.1133	1.1187	1.1256
4.1.2	User Mean	1.0630	1.0467	1.0329	1.0367	1.0393	1.0437
4.1.3	Movie Mean	1.0334	1.0305	1.0197	1.0169	1.0223	1.0246
4.1.4	Mixture Mean	0.9973	0.9861	0.9754	0.9747	0.9798	0.9826
4.2.2	SVT	1.0442	1.0270	1.0227	1.0266	1.0416	1.0324
4.2.3	SVT-quick	1.0080	0.9905	0.9846	0.9896	1.0033	0.9952
4.3	WSOR	0.9782	0.9679	0.9625	0.9595	0.9604	0.9657
4.4.1	Parallel K-SVD	0.9936	0.9741	0.9615	0.9596	0.9669	0.9712
4.5.3	SVD using $\widehat{k}_{\text{rmt}}$	0.9665	0.9515	0.9429	**0.9425**	0.9511	0.9509
4.5.3	SVD using $\widehat{k}_{\text{opt}}$	0.9653	0.9508	**0.9426**	0.9431	0.9508	0.9505
4.6	Robust MC	**0.9591**	**0.9502**	0.9461	0.9459	**0.9468**	**0.9496**

Table 2: Summary of RMSE values for each of our algorithms. The best-performing algorithm for each partition is indicated in bold.

to generate a dictionary with more columns, the RMSE increases dramatically; we interpret this as overfitting. This behavior rarely occurs in other applications of the K-SVD algorithm, in which there are significantly larger amounts of less sparse training data available.

The pattern of overfitting repeats again for SVD approaches, for which the primary challenge is finding an appropriately low-rank approximation. If the assumed rank, $\hat{k}$, is too large, the rank-$\hat{k}$ reconstruction overfits the training data, thereby increasing the RMSE. Although the $\hat{k}_{\text{opt}}$ SVD outperforms the $\hat{k}_{\text{rmt}}$ SVD, it selects the rank with the lowest test RMSE over all possible ranks. The $\hat{k}_{\text{rmt}}$ SVD selects the rank in a data driven manner. By doing so, it avoids overfitting and any further parameter tuning. This is evident in the low RMSE values for this algorithm. The mean value of $\hat{k}_{\text{opt}}$ was 15 whereas the mean value of $\hat{k}_{\text{rmt}}$ was 22.

Robust matrix completion seems particularly resistant to overfitting. The sparse matrix B in the robust matrix completion's model appears to "absorb" the perturbations which give rise to overfitting in other algorithms.

The WSOR algorithm also seems resistant to overfitting, but has limited predictive power. That is, many entries are predicted using the rank-one mean-based approximation after the iterative portion has terminated. The solution produced by WSOR shares the same sort of nearly-rank-1 singular value spectrum as the rest of the algorithms.

5.3 Model Mismatch

Model mismatch was evident in both our SVT and robust matrix completion experiments. For both algorithms, we were able to drive both our training error and our test error to close to zero ($\approx 1 \times 10^{-4}$) when we were using synthetic Gaussian data matrices. However, when we used the MovieLens data, driving the training error to zero resulted in overfitting in the SVT experiment. Additionally, in the robust matrix completion experiment, we were unable to drive the training error much below a relative error of 0.2. Both of these cases indicate a model mismatch.

5.4 Robust matrix completion's multiplicative model

When we ran robust matrix completion on the MovieLens dataset, we consistently found that the rank of $\widehat{L}$ was 1, with a single very large singular value. This was the case for all the MovieLens dataset partitions, even when varying the tuning parameters. This does not seem to be a mistake in our implementation of the algorithm, because we verified that our algorithm was able to recover a generated ratings matrix with known rank. In fact, we found that $\widehat{L} \approx c\mu_{\text{movie}}\mu_{\text{user}}^T$, where the entries of μ_{movie} and μ_{user}^T are drawn from equations (7) and (5), respectively and c a constant of proportionality. This is a multiplicative model of sorts: a user's rating of a movie is estimated as the product of a movie's average quality and a measure of how generous the user is. This is a different approach than the optimal mixture mean model, which models a user's generosity and the movie's quality as being additively combined to form a movie's rating.

6 Conclusions

The goal of this project was to complete a user ratings matrix given an incomplete user ratings matrix. We implemented several matrix completion algorithms to solve this problem; some of the algorithms were simple mean-based algorithms; others were based on standard matrix completion or collaborative filtering algorithms (SVT, SVD, WSOR); others applied non-standard algorithms (K-SVD, SVD-RMT). Adapting algorithms like K-SVD and random matrix theory inspired SVD algorithms showed that these algorithms may be used to solve the problem. We also developed a new robust matrix completion algorithm that outperformed all other algorithms tested. Compared to the standard SVT algorithm, the robust matrix completion algorithm decreased RMSE by approximately 5%. We expect that model mismatch and highly sparse training data limit performance. An interesting extension would be to develop a nonlinear prior on user ratings.

7 Labor Division

All members of our group contributed to the final report. Each member was responsible for the writeup regarding their individual experiments.

Nick spearheaded the SVD based approach initiative. This included a mathematical extension of theorems from random matrix theory which yielded essential insights on how to choose the appropriate rank for an SVD approximation. He also discovered ways of modifying these methods to supercharge performance.

Madison implemented the WSOR algorithm and developed the parallel approximate K-SVD. This involved parameter tuning, debugging, and a great deal of frustration. Somewhat experienced in the ways of the augmented Lagrangian, he provided invaluable guidance to Ben during his derivation of the augmented Lagrangian method for robust matrix completion.

Matt was in command of the singular value thresholding algorithm, including the MATLAB implementation, parameter tuning, and data analysis. He also procured the MovieLens dataset and converted it into `.mat` files, debugged the naïve algorithms, and assembled together the "bare-bones" version of the final report.

Ben was instrumental in deriving and implementing the augmented Lagrangian method for robust matrix completion. His brainchild, the robust matrix completion algorithm, won our hearts and position of top performer in our algorithmic shootout.

References

[1] Michal Aharon, Michael Elad, and Alfred Bruckstein. K-svd: Design of dictionaries for sparse representation. In *PROCEEDINGS OF SPARS05*, pages 9–12, 2005.

[2] J.F. Cai, E.J. Candes, and Z. Shen. A singular value thresholding algorithm for matrix completion. *Arxiv preprint Arxiv:0810.3286*, 2008.

[3] E.J. Candes, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Arxiv preprint ArXiv:0912.3599*, 2009.

[4] E.J. Candes and Y. Plan. Matrix completion with noise. *Arxiv preprint arXiv:0903.3131*, 2009.

[5] Jonathan Eckstein and Dimitri P. Bertsekas. On the douglas-rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55:293–318, 1992.

[6] D. Paul. Asymptotics of sample eigenstructure for a large dimensional spiked covariance model. *Statistica Sinica*, 17(4):1617, 2007.

[7] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Double sparsity: learning sparse dictionaries for sparse signal approximation. *Trans. Sig. Proc.*, 58:1553–1564, March 2010.

[8] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.