Singular Value Thresholding for Matrix Completion

Sriram Ganesan[*]

Nitin Garg
gargn@umich.edu

Jeeheh Oh
jeeheh@umich.edu

Jonathan Stroud
stroud@umich.edu

Abstract

Matrix completion, is the task of filling in the unknown entries of a sparse matrix. This problem is central in recommender systems and collaborative filtering, which provide individualized product suggestions based on ratings gathered from a large set of users. Matrix completion algorithms are used effectively today to provide helpful recommendations for users of services such as Netflix and Amazon.com. In the present paper, the baseline algorithms for matrix completion, such as eigentaste, nearest K-neighbor, and column mean are compared against the more sophisticated matrix completion algorithms, such as Spectral Matrix Completion (SMC) and Singular Value Thresholding (SVT). The comparison metric used in the present paper include, accuracy, convergence time, and the nature of the input matrix. All of the presented methods have their advantages and disadvantages based on the problem. In the present paper, the detailed results are presented for synthetic data and the Jester data. While Eigentaste needs at least few fully filled columns to complete the sparse matrix, SMC and SVT does not have that limitation. Between SMC and SVT, SMC provides better accuracy but is much slower than the SVT. Detailed comparison of the methods is presented in the later sections.

1 Introduction

Collaborative Filtering is the process of learning patterns by aggregating information over multiple sources of data, typically multiple users of the same system. These techniques have been applied effectively on many difficult problems, most notably on recommender systems such as those found on Netflix, Amazon, Spotify, and others. The goal of these systems is to produce targeted product recommendations for individual users given the products the user has previously viewed or rated. Users have typically viewed or rated only a small subset of the available products, and different users typically have not rated the same set of products. These problems do not easily lie into any supervised learning framework, so we instead employ the framework of *matrix completion*.

In matrix completion, we reconstruct a matrix M from a set of known entries $M_{ij},\ (i,j) \in \Omega$. Relying on the assumption that M has low rank, we can pose this as a constrained rank-minimization problem

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & \operatorname{rank}(X) \\ \text{subject to} \quad & X_{ij} = M_{ij},\ \forall (i,j) \in \Omega. \end{aligned}$$

In this project, we explore several algorithms for matrix completion via rank-minimization and compare their performance on collaborative filtering baselines. We provide an implementation of the method described in “A Singular Value Thresholding Algorithm for Matrix Completion” [**?**] and

[*] The names are printed in alphabetical order by last name.

reproduce their experiments. We also provide an implementation of a competing algorithm from "Matrix Completion from a Few Entries" [**?**] and compare their effectiveness.

2 Spectral Matrix Completion

In the present paper, the Spectral Matrix Completion (SMC) algorithm presented in Keshavan et al. [**?**] is implemented. Lets say, M is the matrix whose entry $(i,j) \in [m] \times [n]$ corresponding to the rating user i would assign to movie/joke j. M^E is the $m \times n$ matrix that contains the revealed entries of M, and is filled with 0's in the other positions

$$M^E_{i,j} = \begin{cases} M_{i,j} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

As presented in [**?**], the SMC algorithm has the following structure:

1. Trim the M^E, and let $\widetilde{M}^E$.
2. Project $\widetilde{M}^E$ to $T_r(\widetilde{M}^E)$.
3. Clean residual errors by minimizing the discrepancy F(X,Y).

2.0.1 Trimming

In the trimming step, zero all columns in M^E with degree larger than $2|E|/n$ and set to zero all rows with degree larger than $2|E|/m$, where $|E|$ is the number of non-zero entries in M. Trimming leads to 'throwing out information' which makes the underlying true-rank structure more apparent. This effect becomes even more important when the number of revealed entries per row/column follows a heavy tail distribution, as for real data.

2.0.2 Projection

In the projection step, compute the singular value decomposition (SVD) of M^E (with $\sigma_1 \geq \sigma_2 \geq$ ..... ≥ 0)

$$M^E = \sum_{i=1}^{min(m,n)} \sigma_i x_i y_i^T \tag{2}$$

and, then the matrix $T_r(M^E)$ is $(mn/|E|) \sum_{i=1}^{r} \sigma_i x_i y_i^T$, obtained by setting to 0 all but the r largest singular values. Note that apart from the rescaling factor $(mn/|E|)$, $T_r(M^E)$ is the orthogonal projection of M^E onto the set of rank-r matrices.

2.0.3 Cleaning

This is the step where all the magic happens in SMC algorithm. Given $X \in R^{m \times r}$, $Y \in R^{n \times r}$ with $X^T X = m1$ and $Y^T Y = n1$,

$$F(X,Y) = \min_{S \in R^{r \times r}} F(X,Y,S) \tag{3}$$

$$F(X,Y,S) = \frac{1}{2} \sum_{(i,j) \in E} (M_{ij} - (XSY^T)_{ij})^2 \tag{4}$$

The cleaning step consists in writing $T_r(\widetilde{M}^E) = X_0 S_0 Y_0^T$ and minimizing $F(X,Y)$ locally with initial condition $X = X_0$, $Y = Y_0$. Note that $F(X,Y)$ is easy to evaluate since it is defined by minimizing the quadratic function $S \mapsto F(X,Y,S)$ over the low-dimensional matrix S. Further it depends on X and Y only through their column spaces. In geometric terms, F is a function defined over the cartesian product of two Grassmann manifolds. Optimization over Grassmann manifolds is a well understood topic [**?**] and efficient algorithms (in particular Newton and conjugate gradient) can be applied. In the present algorithm, gradient descent with line search is used to minimize $F(X,Y)$.

3 Singular Value Thresholding

Singular Value Thresholding (SVT) [**?**] is an algorithm proposed for *nuclear norm minimization* of a matrix. Formally, SVT addresses the optimization problem

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

where $\|\cdot\|_*$ is the *nuclear norm*, or the sum of the singular values and $\mathcal{P}_\Omega(\cdot)$ makes zero all entries $(i, j) \notin \Omega$. This can be thought of as a convex relaxation to the rank minimization problem, and the two are formally equivalent under some conditions. The rank minimization problem is, however, highly non-convex and therefore not a suitable candidate for black-box optimization algorithms.

Singular Value Thresholding works by iteratively constructing X using a low-rank, low-singular value approximation to an auxiliary sparse matrix Y. Y is then adjusted to ensure the resulting approximation in the subsequent step has matching entries $X_{ij} = M_{ij}$. Each iteration consists of the inductive steps

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

where $\text{shrink}(\cdot, \cdot)$ is the *singular value shrinkage operator*. Given a singular value decomposition $X = U\Sigma V^T$, $\Sigma = \text{diag}(\{\sigma_i\}_{1\leq i\leq r})$, we can write this as

$$\text{shrink}(X, \tau) = U\Sigma_\tau V^T, \;\; \Sigma_\tau = \text{diag}(\{(\sigma_i - \tau)_+\}).$$

These two operations, when repeated, approach a low-nuclear norm solution by repeatedly shrinking the singular values of X. This algorithm has shown success in recovering accurate low-rank solutions when the source of M is also low-rank, even though it does not optimize this objective directly. The original authors discuss its theoretical guarantees in detail, but we choose to omit them in this discussion.

In practice, this system has a number of hyperparameters that must be carefully tuned to guarantee convergence. The shrinkage value τ must be set fairly high in order for the algorithm to converge quickly, but not too high that it dwarfs the true singular values. The stepsizes δ_k are similarly sensitive. These can be set dynamically as well, though we choose to maintain a fixed stepsize throughout. We compute the decomposition of Y^K in batches, which introduces a new batch size parameter l. Also important is the initialization of Y^0, for which the authors provide helpful strategies. Finally, we use the relative error $\|\mathcal{P}_\Omega(X^k - M)\|_F / \|P_\Omega(M)\|$ as a stopping criterion. We terminate when this drops below a small ϵ.

4 Baseline Algorithms

4.1 Eigentaste

Eigentaste is a collaborative filtering algorithm which applies principal component analysis to the dense subset of user ratings, thus facilitating dimensionality reduction for offline clusters and rapid computation of recommendations. Mean rating of the jth item in the gauge set is given by

$$\mu_{ij} = \frac{1}{n} \sum_{i \epsilon U_j} \widetilde{r}_{ij}$$

$$\sigma_j^2 = \frac{1}{n-1} \sum_{i \epsilon U_j} (\widetilde{r}_{ij} - \mu_j)^2$$

In A, the normalized rating r_{ij} is set to $(\widetilde{r}_{ij} - \mu_j)/\sigma_j$. The global correlation matrix is given by

$$C = \frac{1}{n-1} A^T A = E^T \Lambda E$$

The data is projected along the first v eigenvectors $x = R{E_v}^T$

Recursive Rectangular Clustering:

1. Find the minimal rectangular cell that encloses all the points in the eigenplane.
2. Bisect along x and y axis to form 4 rectangular sub-cells.
3. Bisect the cells in step 2 with origin as a vertex to form sub-cells at next hierarchial level.

Online Computation of Recommendations

1. Collect ratings of all items in gauge set.
2. Use PCA to project this vector to eigenplane.
3. Find the representative cluster.
4. Look up appropriate recommendations, present them to the new user, and collect ratings.

4.2 K-Nearest Neighbor

Nearest neighbor algotihm predicts the ratings based on mean user rating and variation of the current rating from the mean rating of its nearest neighbors.

$$p_{ij} = \overline{r_i} + \kappa \sum_{k=1}^{n} w(i,p)(\overline{r_{pj}} - \overline{r_p})$$

where $\overline{r_i}$ is the average joke rating for user i, and κ is a normalizing factor ensuring that the absolute value of the weights sum to 1. We implemented the weighted nearest neighbor algorithm. We used a function of Euclidean distance from user i to user p as the weight w(i, p), and κ = $\sum_{k=1}^{n} w(i,p)$. Specifically, if we are interested in q nearest neighbors, w(i; p) = d(i, q +1)- d(i, p). This ensures that is closest neighbor has the largest weight.

4.3 Spectral Matrix Completion

In the present paper, the Spectral Matrix Completion (SMC) algorithm presented in Keshavan et al. [**?**] is implemented. Lets say, M is the matrix whose entry $(i,j) \in [m] \times [n]$ corresponding to the rating user i would assign to movie/joke j. M^E is the $m \times n$ matrix that contains the revealed entries of M, and is filled with 0's in the other positions

$$M_{i,j}^{E} = \begin{cases} M_{i,j} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

As presented in [**?**], the SMC algorithm has the following structure:

1. Trim the M^E, and let $\widetilde{M}^E$.
2. Project $\widetilde{M}^E$ to $T_r(\widetilde{M}^E)$.
3. Clean residual errors by minimizing the discrepancy F(X,Y).

4.3.1 Trimming

In the trimming step, zero all columns in M^E with degree larger than $2|E|/n$ and set to zero all rows with degree larger than $2|E|/m$, where $|E|$ is the number of non-zero entries in M. Trimming leads to 'throwing out information' which makes the underlying true-rank structure more apparent. This effect becomes even more important when the number of revealed entries per row/column follows a heavy tail distribution, as for real data.

4.3.2 Projection

In the projection step, compute the singular value decomposition (SVD) of M^E (with $\sigma_1 \geq \sigma_2 \geq \ldots.. \geq 0$)

$$M^E = \sum_{i=1}^{min(m,n)} \sigma_i x_i y_i^T \tag{6}$$

and, then the matrix $T_r(M^E)$ is $(mn/|E|)\sum\limits_{i=1}^{r}\sigma_i x_i y_i^T$, obtained by setting to 0 all but the r largest singular values. Note that apart from the rescaling factor $(mn/|E|)$, $T_r(M^E)$ is the orthogonal projection of M^E onto the set of rank-r matrices.

4.3.3 Cleaning

This is the step where all the magic happens in SMC algorithm. Given $X \in R^{m\times r}$, $Y \in R^{n\times r}$ with $X^T X = m1$ and $Y^T Y = n1$,

$$F(X,Y) = \min_{S\in R^{r\times r}} F(X,Y,S) \tag{7}$$

$$F(X,Y,S) = \frac{1}{2}\sum_{(i,j)\in E}(M_{ij} - (XSY^T)_{ij})^2 \tag{8}$$

The cleaning step consists in writing $T_r(\widetilde{M^E}) = X_0 S_0 Y_0^T$ and minimizing $F(X,Y)$ locally with initial condition $X = X_0$, $Y = Y_0$. Note that $F(X,Y)$ is easy to evaluate since it is defined by minimizing the quadratic function $S \mapsto F(X,Y,S)$ over the low-dimensional matrix S. Further it depends on X and Y only through their column spaces. In geometric terms, F is a function defined over the cartesian product of two Grassmann manifolds. Optimization over Grassmann manifolds is a well understood topic [**?**] and efficient algorithms (in particular Newton and conjugate gradient) can be applied. In the present algorithm, gradient descent with line search is used to minimize $F(X,Y)$.

5 Experimental Results

5.1 Synthetic Data

The performance of the SVT and SMC algorithms were compared on synthetically generated matrices of varying ranks, sizes and sparsity. Specifically, low rank square matrices of sizes 1000 and 5000 with ranks of 5, 10 and 20, and sparsity levels of 30%, 50% and 70% were used. As a baseline to SVT and SMC, a column mean matrix completion method was implemented. This method predicts the average column value for all missing entries. In the following figures we see mean absolute error and run time averaged over the categories of size, rank and sparsity. Mean absolute error is the absolute value of the average error on the unknown entries. Run time is measured in seconds and all experiments were run on CAEN lab computers (8GB memory with four core i7 processors).

Table 1: Mean Absolute Error

	Size		Rank			Sparsity		
Algorithm	1000	5000	5	10	20	30%	50%	70%
Mean	9.43	10.26	5.65	9.90	13.98	9.10	10.21	10.21
SVT	1.37	7.6E-04	2.05	7.2E-04	1.4E-03	2.05	8.4E-04	9.6E-04
SMC	1.4E-04	3.7E-07	2.1E-04	4.3E-07	4.5E-07	2.1E-04	4.2E-07	3.8E-07

Table 2: Run Tine (in seconds)

	Size		Rank			Sparsity		
Algorithm	1000	5000	5	10	20	30%	50%	70%
Mean	0.01	0.22	0.12	0.12	0.10	0.11	0.11	0.12
SVT	6	98	26	39	92	42	54	60
SMC	86	1433	252	405	1623	1021	510	748

Both SVT and SMC outperform the mean method. In addition, SMC consistently outperforms SVT in orders of magnitude of three. While SMC significantly outperforms SVT in terms of accuracy, SVT significantly outperforms SMC in terms of time. It appears that the time cost of SMC grows exponentially with rank size. Even at rank five, the smallest tested rank value, SMC runs 9.6 times slower than SVT.

5.2 Jester Dataset

6 Conclusions

7 Future Work

size ($n \times n$)	rank (r)	m/d_r	m/n^2	time (s)	#iters	relative error
1000	10	6	0.119	279.1	250.0	86.59246 x 10^{-4}
1000	50	4	0.390	800.4	220.4	0.99209 x 10^{-4}
1000	100	3	0.570	604.6	163.8	0.98189 x 10^{-4}
5000	10	6	0.024	12934.9	250.0	611.42446 x 10^{-4}
5000	50	5	0.100	-	-	-
5000	100	4	0.158	-	-	-
10000	10	6	0.012	-	-	-
10000	50	5	0.050	-	-	-
10000	100	4	0.080	-	-	-
20000	10	6	0.006	-	-	-
20000	50	5	0.025	-	-	-
30000	10	6	0.004	-	-	-

Table 3: Performance of Singular Value Thresholding on synthetic matrices of known rank. We generate two $n \times r$ matrices U and V whose entries are i.i.d. gaussian.We choose m random entries from $M = UV^T$ and measure convergence rates of SVT. m/d_r is the ratio of sampled entries m and the "true dimensionality" $d_r = r(2n - r)$.