SINGULAR VALUE DECOMPOSITION IN RECOMMENDER SYSTEMS

by

Anh Nguyen

Submitted in partial fulfillment of the

requirements for Departmental Honors in

the Department of Mathematics

Texas Christian University

Fort Worth, Texas

May 2, 2016

SINGULAR VALUE DECOMPOSITIONS IN RECOMMENDER SYSTEMS

Project Approved:

Supervising Professor: Ken Richardson

Department of Mathematics

Scott Nollet

Department of Mathematics

Igor Prokhorenkov

Department of Mathematics

ABSTRACT

Many websites nowadays such as Amazon, Ebay have used different kinds of *Recommender Systems* to predict ratings of items from their clients, so that they could suggest which items are more likely to be purchased. Nevertheless, problems arise as a data set can be both too large and too sparse to predict ratings. Hence, the size of a data set and its information sufficiency should be taken into consideration in order to make accurate predictions efficiently. In this project, we examine a method of matrix factorization called *Singular Value Decomposition* to approach the aforementioned problems by applying the method to real data that we retrieve from Grouplens.

1 Introduction

As discussed in [1], *recommender systems* are used to assist people with problems of information overload. Recommender systems feature two basic entities: users and items. Users first provide their opinions about items that they have viewed or purchased in the past, so that recommender systems generate recommendations about new items for users based on given information about previous ratings.

However, recommender systems encounter fundamental problems such as *sparsity*, which is caused by an insufficient amount of rating data from users and *scalability*, which is caused by large, intractable data. There exist several tools to approach these problems, yet we will focus on *Singular Value Decomposition* (SVD).

The report is organized as follows: We introduce the matrix factorization SVD in Section 2. Then, we discuss the truncated SVD algorithm in Section 3. In Section 4, we present item-based filtering algorithms popularly used in recommender systems and SVD enhanced item-based filtering algorithms. In Section 5, we describe several methods of imputation to solve the sparsity problems of data sets. In Section 6, we briefly describe our mathematical approach and assumptions to ease sparsity and scalability problems. In Section 7, we conduct experiments with data retrieved from ©Grouplens website to test our hypothesis. Next, in Section 8, we present the results of our experiments. We sum up our research in Section 9. Finally, we discuss our future work of the project in section 10 and present our code in Appendix A.

2 Singular Value Decomposition (SVD)

The Singular Value Decomposition (SVD) is a matrix factorization that takes an m by n matrix A of rank r and decomposes as

$$A = U\Sigma V^T, \tag{1}$$

where U and V are orthogonal matrices of size m by m and n by n respectively and Σ is an m by n matrix where $[\Sigma]_{ii} = \sigma_i$ for $i = 1, 2, ..., r$ and $\sigma_1 \geq \cdots \geq \sigma_r > 0$:

$$\begin{pmatrix} \sigma_1 & 0 & \cdots & \cdots \\ 0 & \sigma_2 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \sigma_r & \cdots \\ 0 & \cdots & \cdots & \cdots \end{pmatrix}$$

The non-negative $\sigma_1 \geq \cdots \geq \sigma_r$ are uniquely determined and are called the singular values of A. We now show how to find U, V, Σ as defined.

First, I want to show that A^TA is symmetric and positive semi-definite. Observe that $(A^TA)^T = A^TA$. Thus, A^TA is symmetric. Next, we prove that A^TA is positive semi-definite. For all $x \in \mathbb{R}^n$, see that $x^T(A^TA)x = (Ax)^T(Ax) = (Ax) \cdot (Ax) \geq 0$. Hence, A^TA is positive semi-definite, meaning that A^TA has all nonnegative eigenvalues $\alpha_1 \geq \alpha_2 ... \geq \alpha_r > 0 = 0 = ... = 0$. Let $\sigma_j = \sqrt{\alpha_j}$.

Because A^TA is symmetric and positive semi-definite, by the Spectral Theorem for symmetric matrices, there exists an orthogonal matrix V whose columns contain the eigenvectors of A^TA corresponding to the eigenvalues. Then:

$$A^TA = V \begin{pmatrix} \alpha_1 & 0 & \cdots & \cdots \\ 0 & \alpha_2 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \alpha_r & \cdots \\ 0 & \cdots & \cdots & \cdots \end{pmatrix} V^T = VDV^T. \tag{2}$$

Then the matrix D can be written as $\Sigma^T\Sigma$, where Σ is the m by n matrix:

$$\Sigma = \begin{pmatrix} \sqrt{\alpha_1} & 0 & \cdots & \cdots \\ 0 & \sqrt{\alpha_2} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \sqrt{\alpha_r} & \cdots \\ 0 & \cdots & \cdots & \cdots \end{pmatrix} = \begin{pmatrix} \sigma_1 & 0 & \cdots & \cdots \\ 0 & \sigma_2 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \sigma_r & \cdots \\ 0 & \cdots & \cdots & \cdots \end{pmatrix}. \tag{3}$$

Now, we need to find U.

There are two ways to find U. The first method is to calculate AA^T and by spectral theorem, U can be chosen similarly to V. We have $AA^T = U\Sigma\Sigma^TU^T$. Similarly, by the Spectral Theorem for symmetric matrices, there exists such an orthogonal matrix U whose columns contain all the eigenvectors of AA^T corresponding to each eigenvalues of AA^T in matrix $\Sigma\Sigma^T$. Hence, $\Sigma\Sigma^T$ and $\Sigma^T\Sigma$ both have $\alpha_1, \cdots, \alpha_r$ as the diagonal entries.

We now use Σ and $V = \begin{pmatrix} v_1 \mid v_2 \mid \cdots \mid v_n \end{pmatrix}$ to find U. Since A has rank r, $\{v_{r+1}, \cdots, v_n\}$ is an orthonormal basis for null space of A (i.e, 0 eigenspace).

Now, we let $\tilde{V}, \tilde{\Sigma}$ be defined by:

$$\tilde{V} = \begin{pmatrix} v_1 & v_2 & \cdots & v_r \end{pmatrix},$$

$$\tilde{\Sigma} = \begin{pmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \sigma_r \end{pmatrix}.$$

Then, since V is an orthogonal matrix, equation (1) is equivalent to:

$$AV = U\Sigma \tag{4}$$

Ignoring equations of the form $0 = 0$, an equivalent system is

$$A\tilde{V} = \tilde{U}\tilde{\Sigma}, \tag{5}$$

where matrix $\tilde{V}$ is an $n \times r$, $\tilde{U}$ is an $m \times r$ and $\tilde{\Sigma}$ is an $r \times r$ invertible matrix.

We know $\tilde{V}, \tilde{\Sigma}$ and wish to solve for $\tilde{U}$.

Notice that $\tilde{\Sigma}$ is a diagonal matrix with nonnegative singular values σ_i, so that $\tilde{\Sigma}$ is invertible. Hence, equation (5) implies

$$\tilde{U} = A\tilde{V}\tilde{\Sigma}^{-1}. \tag{6}$$

Next we will prove that $\tilde{U}^T\tilde{U} = I_r$ by computing $\tilde{U}^T\tilde{U}$:

$$\tilde{U}^T\tilde{U} = (A\tilde{V}\tilde{\Sigma}^{-1})^T A\tilde{V}\tilde{\Sigma}^{-1} \tag{7}$$

$$= (\tilde{\Sigma}^{-1})^T(\tilde{V})^T A^T A\tilde{V}\tilde{\Sigma}^{-1} \tag{8}$$

From equation (2), we have:

$$\tilde{U}^T\tilde{U} = (\tilde{\Sigma}^{-1})^T(\tilde{V})^T V\Sigma^T\Sigma V^T\tilde{V}\tilde{\Sigma}^{-1} \tag{9}$$

$$= (\tilde{\Sigma}^{-1})^T \begin{pmatrix} 1 & 0 & \cdots & \cdots \\ 0 & 1 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 1 & \cdots \end{pmatrix}^{r\times n} \Sigma^T\Sigma \begin{pmatrix} 1 & 0 & \cdots \\ 0 & 1 & \cdots \\ \cdots & \cdots & \cdots \\ 0 & \cdots & 1 \\ 0 & \cdots & 0 \end{pmatrix}^{n\times r} \tilde{\Sigma}^{-1} \tag{10}$$

$$= \begin{pmatrix} \sigma_1^{-1} & 0 & \cdots \\ 0 & \sigma_2^{-1} & \cdots \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \sigma_r^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} \sigma_1^2 & 0 & \cdots & \cdots \\ 0 & \sigma_2^2 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \sigma_r^2 & \cdots \\ 0 & \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} I \\ 0 \end{pmatrix} \begin{pmatrix} \sigma_1^{-1} & 0 & \cdots \\ 0 & \sigma_2^{-1} & \cdots \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \sigma_r^{-1} \end{pmatrix} \tag{11}$$

$$= \begin{pmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \sigma_r \end{pmatrix} \begin{pmatrix} \sigma_1^{-1} & 0 & \cdots \\ 0 & \sigma_2^{-1} & \cdots \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \sigma_r^{-1} \end{pmatrix} \tag{12}$$

$$= I_r \tag{13}$$

Using the $n \times r$ matrix $\tilde{U} = \begin{pmatrix} u_1 \,\middle|\, \cdots \,\middle|\, u_r \end{pmatrix}$, we then produce matrix U of size $m \times m$ by completing the basis $u_1, \cdots, u_r$ to an orthonormal basis of $\mathbb{R}^m$. Thus, $A = U\Sigma V^T$, as desired.

3 Truncated SVD Algorithm

The truncated SVD algorithm is used to compute a small number of singular values instead of calculating all the singular values of a matrix. By using the truncated SVD, we can calculate an approximation of a matrix using less data than the original matrix. That is, we write $A \approx U'\Sigma'(V')^T$, where Σ' only contains the largest singular values corresponding to eigenvectors of A^TA in the first few columns of V', and the rest of the columns of V' are constructed to be a completed basis of $\mathbb{R}^n$.

In the following subsection, we focus on Krylov-Schur approach to the truncated SVD, which is described in [3].

Before we discuss the Krylov-Schur Approach, we first introduce Householder transformations and bidiagonal factorization, which are necessary for the Krylov-Schur Approach.

3.1 Householder transformations

A Householder transformation is an orthogonal transformation. Reflection across the plane orthogonal to a unit normal vector v can be expressed as:

$$H = I - 2vv^T \tag{14}$$

A Householder matrix H has the following properties:

$$H^* = H \tag{15}$$
$$H^*H = H^2 = I \tag{16}$$

Moreover, H has one -1 eigenvalue and the rest of its eigenvalues are 1. First, consider Hv, we have:

$$Hv = (I - 2vv^T)v = Iv - 2vv^Tv \tag{17}$$

Since v is a unit vector, we have equation (17):

$$Hv = v - 2v = -v \tag{18}$$

Thus, the first eigenvalue of H is -1.
On the other hand, consider Hw where w is any unit vector perpendicular to v. We have

$$(I - 2vv^T)w = Iw - 2vv^Tw. \tag{19}$$

Since the vector v is orthogonal to w

$$Hw = Iw = w. \tag{20}$$

Hence, the rest of the eigenvalues of H are 1.

3.2 Bidiagonal factorization

This is a very crucial process in the SVD computation. Bidiagonal factorization is introduced in [4]. Let A be an $m \times n$ matrix. Without any loss of generality, let $m \geq n$, $A = A^{(1)}$ and $A^{(3/2)}, A^{(2)}, \cdots, A^{(n)}, A^{(n+1/2)}$ be defined by

$$A^{(k+1/2)} = P^{(k)}A^{(k)}, \quad k = 1, 2, \cdots, n, \tag{21}$$
$$A^{(k+1)} = A^{(k+1/2)}Q^{(k)}, \quad k = 1, 2, \cdots, n-1. \tag{22}$$

where $P^{(k)}$ and $Q^{(k)}$ are Householder matrices of form

$$P^{(k)} = I - 2x^{(k)}x^{(k)*}, \quad x^{(k)*}x^{(k)} = 1, \tag{23}$$
$$Q^{(k)} = I - 2y^{(k)}y^{(k)*}, \quad y^{(k)*}y^{(k)} = 1. \tag{24}$$

Now, we will choose $x^{(k)}$ and $y^{(k)}$ such that

$$a_{i,k}^{(k+1/2)} = 0, \quad i = k+1, \cdots, m, \tag{25}$$

$$a_{k,j}^{(k+1)} = 0, \quad j = k+2, \cdots, n. \tag{26}$$

In order to choose such $x^{(k)}$ to satisfy equation (25), we denote $x^{(k)}$ as

$$x^{(k)} = \frac{a^k - ||a^k||e_k}{||a^k - ||a^k||e_k||}, \tag{27}$$

where a^k is the k^{th} column of the matrix $A^{(k)}$ and e_k is the standard basis with 1 as its k^{th} entry.
After having computed $x^{(k)}$ and $P^{(k)}$, we can find $A^{(k+1/2)}$ in equation (21). Now, in order to find $A^{(k+1)}$, we need to compute $y^{(k)}$. This is done by using equation (27) but instead of using the k^{th} column of matrix $A^{(k)}$, we will use the first row of matrix $A^{(k+1/2)}$, which is a new matrix A updated by eliminating rows and columns with zeros. Hence, we can finally compute $A^{(k+1)}$.

Finally, after the bidiagonalization process, $A^{(k+1)}$ has the form

$$\begin{pmatrix} \alpha_1 & \beta_1 & 0 & . & . & & & & & \\ 0 & \alpha_2 & \beta_2 & 0 & . & & & & & \\ . & 0 & . & . & . & & 0 & & & \\ . & . & . & . & . & & & & & \\ & & & & \alpha_k & \beta_k & & & & \\ & & & & & * & * & . & . & . \\ & & 0 & & & * & * & . & . & . \\ & & & & & . & . & . & . & . \\ & & & & & . & . & . & . & . \end{pmatrix} \tag{28}$$

Furthermore, the bidiagonalization gives us two important equations

$$AV_m = U_m B_m, \tag{29}$$

$$A^T U_m = V_m B_m^T + \beta_m v_{m+1} e_m^T. \tag{30}$$

where $V_m = Q^{(1)}...Q^{(m)}$, $U_m = (P^{(1)})^T...(P^{(m)})^T$, $B_m = A^{(m)}$ and B has the form

$$\begin{pmatrix} \alpha_1 & \beta_2 & \cdots & \cdots \\ 0 & \alpha_2 & \beta_3 & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \alpha_{m-1} & \beta_m \\ 0 & \cdots & \cdots & \alpha_m \end{pmatrix}.$$

In equation (30), notice that there is a residual term $\beta_m v_{m+1} e_m^T$. This is because equation (29) is equivalent to:

$$(AV_m)^T = (U_m B_m)^T \tag{31}$$
$$V_m^T A^T = B_m^T U_m^T \tag{32}$$

Multiplying both sides on the left with V_m,

$$A^T = V_m B_m^T U_m^T. \tag{33}$$

Multiplying both sides on the right with U_m,

$$A^T U_m = V_m B_m^T. \tag{34}$$

We can indeed write the above equation as:

$$A^T \begin{pmatrix} u_1 \mid u_2 \mid \cdots \mid u_m \end{pmatrix} = \begin{pmatrix} v_1 \mid v_2 \mid \cdots \mid v_m \end{pmatrix} \begin{pmatrix} \alpha_1 & \beta_2 & \cdots & \cdots \\ 0 & \alpha_2 & \beta_3 & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \alpha_{m-1} & \beta_m \\ 0 & \cdots & \cdots & \alpha_m \end{pmatrix} \tag{35}$$

This is equivalent to:

$$A^T U_m = \begin{pmatrix} \alpha_1 v_1 + \beta_1 v_2 \mid \alpha_2 v_2 + \beta_2 v_3 \mid \cdots \mid \alpha_m v_m + \beta_m v_{m+1} \end{pmatrix} \tag{36}$$
$$= V_m B_m^T + \begin{pmatrix} 0 \mid 0 \mid \cdots \mid 0 \mid \beta_m v_{m+1} \end{pmatrix} \tag{37}$$
$$= V_m B_m^T + \beta_m v_{m+1} e_m^T \tag{38}$$

Thus, equation (38) is equivalent to equation (30).

3.3 Krylov-Schur Approach

Assume that we are looking for the l largest singular values of matrix A. We then create a search space m that is twice as large as size l, i.e, $m \approx 2l$.

By diagonal factorization (discussed above), we have equations (29) and (30). Now, we find the SVD for matrix $B_m = A^{(m)} = P_m \Sigma_m Q_m^T$ and replace B_m in equations (29) and (30)

$$AV_m = U_m P_m \Sigma_m Q_m^T, \tag{39}$$
$$A^T U_m = V_m Q_m \Sigma_m P_m^T + \beta_{m+1} v_{m+1} e_m^T. \tag{40}$$

We multiply by Q_m to the right of (39) and P_m on the right of (40), and we have the Krylov-Golub-Kahan (KGK) factorization

$$A\tilde{V_m} = \tilde{U_m}\Sigma_m, \tag{41}$$
$$A^T\tilde{U_m} = \tilde{V_m}\Sigma_m + \beta_{m+1}v_{m+1}p_m^T, \tag{42}$$

where $p_m^T = e_m^T P_m$, $\tilde{U_m} = U_m P_m$ and $\tilde{V_m} = V_m Q_m$.

Since Σ_m has real singular values as diagonal entries, we can put first l largest singular values to the left and the remaining $m-l$ to the right of Σ_m using permutation matrices to permute the columns. We call the permutation matrices as Π_m

$$A\tilde{V_m}\Pi_m = \tilde{U_m}\Pi_m\Pi_m^T\Sigma_m\Pi_m, \tag{43}$$
$$A^T\tilde{U_m}\Pi_m = \tilde{V_m}\Pi_m\Pi_m^T\Sigma_m\Pi_m + \beta_{m+1}v_{m+1}p_m^T\Pi_m, \tag{44}$$

where every column vector of permutation matrix Π_m is a standard basis vector. Specifically, for example, if we would like to switch the i^{th} column to the j^{th} column of Σ_m, we multiply Σ_m with Π_m such that $\Pi_m = \begin{pmatrix} e_1 & \cdots & e_j & \cdots & e_i \end{pmatrix}$, where e_j is in the i^{th} column and e_i is in the j^{th} column of Π_m.

After shrinking the factorization back to size l, we have

$$A\hat{V_l} = \hat{U_l}\hat{\Sigma_l}, \tag{45}$$
$$A^T\hat{U_l} = \hat{V_l}\hat{\Sigma_l} + \beta_{l+1}v_{l+1}\hat{p_l}^T, \tag{46}$$

where $\hat{V_l} = (\tilde{V_m}\Pi_m)_{1:l}$, $\hat{U_l} = (\tilde{U_m}\Pi_m)_{1:l}$ and $\hat{\Sigma_l} = (\Pi_m^T\Sigma_m\Pi_m)_{1:l}$.

We now will use the Householder matrix W_l to reduce the residual term, i.e, $\beta_{l+1}v_{l+1}\hat{p_l}^T W_l = \hat{\beta}_{l+1}v_{l+1}e_l^T$. Applying W_l

$$A\hat{V_l}W_l = \hat{U_l}W_lW_l\hat{\Sigma_l}W_l^T, \tag{47}$$
$$A^T\hat{U_l}W_l = \hat{V_l}W_lW_l\hat{\Sigma_l}W_l^T + \beta_{l+1}v_{l+1}\hat{p_l}^TW_l. \tag{48}$$

Because $W_l^T = W_l^{-1} = W_l$

$$A\check{V_l} = \check{U_l}\check{C_l}, \tag{49}$$
$$A^T = \check{V_l}\check{C_l}^T + \beta_{l+1}v_{l+1}e_l^T, \tag{50}$$

where $\check{U_l} = \hat{U_l}W_l$, $\check{V_l} = V_lW_l$ and $\check{C_l} = W_l\Sigma_lW_l^T$.

Now the 2 equations above look like bidiagonal factorization. However, the matrix $\check{C_l}$ is a dense matrix instead of a bidiagonal one. Thus, we wish to bidiagonal factorize $\check{C_l}$ without destroying the residual term $\beta_{l+1}v_{l+1}e_l^T$ by doing row reduction to bidiagonal form.

In other words, we would like to factorize $\check{C}_l = P_l B_l Q_l^T$ where P_l, Q_l are orthogonal matrices. Replacing $\check{C}_l$, we have the following

$$A\check{V}_l = \check{U}_l P_l B_l Q_l^T, \tag{51}$$
$$A^T\check{U}_l = \check{V}_l Q_l B_l^T P_l^T + \beta_{l+1} v_{l+1} e_l^T. \tag{52}$$

Multiplying equation (51) by Q_l on the right and equation (52) by P_l on the right, we have

$$A\check{V}_l Q_l = \check{U}_l P_l B_l, \tag{53}$$
$$A^T\check{U}_l P_l = \check{V}_l Q_l B_l^T + \beta_{l+1} v_{l+1} e_l^T, \tag{54}$$

with $e_l^T P_l = e_l^T$ because of the structure of P_l. Equations (53) and (54) can be written as a bidiagonal factorization

$$AV_l = U_l B_l, \tag{55}$$
$$A^T U_l P_l = V_l B_l^T + \beta_{l+1} v_{l+1} e_l^T, \tag{56}$$

where $\tilde{U}_l = \check{U}_l P_l$ and $\tilde{V}_l = \check{V}_l Q_l$.

3.4 Krylov-Schur SVD Algorithm (KSSVD)

The algorithm to create a truncated SVD of a given matrix introduced in [3] is as follows: Create bidiagonal factorization of dimension l. Then
for $k = 1, 2, \cdots$ do

1. expand bidiagonal factorization from size l to m
2. Do the SVD for B_m
3. Transform bidiagonal to Krylov-Golub-Kahan factorization
4. Use Krylov-Schur approach to order the singular values from largest to smallest
5. Set the l smallest eigenvalues to 0
6. Shrink factorization to order l
7. Transform residual term using Householder matrix
8. Bidiagonalize small matrix $\check{C}_l$ and obtain bidiagonal factorization of dimension l

end for loop.

3.5 SVDSECON Algorithm

We next introduce a second algorithm to truncate the SVD, SVDSECON, which is introduced in [6]. This algorithm is an alternative of the SVD in Matlab. The steps are described below.

Suppose the size of an original matrix A is size $m \times n$, where $m \leq n$. Then:

1. Let $C = AA^T$. Since $A = \tilde{U}\tilde{\Sigma}\tilde{V^T}$ from equation (6), C will be

$$C = \tilde{U}\tilde{\Sigma}\tilde{V^T}\tilde{V}\tilde{\Sigma^T}\tilde{U^T}, \tag{57}$$

$$= \tilde{U}\tilde{\Sigma}\tilde{\Sigma^T}\tilde{U^T}. \tag{58}$$

   Let $D = \tilde{\Sigma}\tilde{\Sigma^T}$ then equation (58) is:

$$C = \tilde{U}D\tilde{U^T} \tag{59}$$

2. Compute only the l largest eigenvalues and eigenvectors of C. In this case, D is the diagonal matrix with the squares of the l largest eigenvalues of C, and the columns of $\tilde{U}$ are orthonormal eigenvectors corresponding to l largest eigenvalues of C.

3. After calculating $\tilde{U}$ and $\tilde{\Sigma} = \sqrt{D}$, which has been explained in section 2, since $A = \tilde{U}\tilde{\Sigma}\tilde{V^T}$,

$$A^T = \tilde{V}\tilde{\Sigma}\tilde{U^T} \tag{60}$$

$$\Rightarrow A^T\tilde{U^T}\tilde{\Sigma^{-1}} = \tilde{V}. \tag{61}$$

else, if $m \geq n$ then we let $C = A^TA$ and do the same process above.

3.6 Numerical Issues: Condition of a matrix

A matrix A can be simplified by using Gaussian elimination. However, Gaussian elimination results in round-off errors. This creates a computational problem, since round-off errors are stored in its solution. Thus, the term *condition of a matrix* is introduced in [5].

A matrix A is called ill-conditioned if A is invertible but still can become non-invertible if some of the entries are changed very little. Hence, in order to measure of how ill-conditioned A is, we define the condition number of A, $cond(A)$, which can be found using A and A^{-1}

$$cond(A) = ||A||.||A^{-1}||, \tag{62}$$

where $||.||$ is the operator norm of A. Hence,

$$cond(A) = \frac{\lambda_{\max A}}{\lambda_{\min A}}, \tag{63}$$

where $\lambda_{\max A}$ is the maximum of the absolute value of the eigenvalues of A and $\lambda_{\min A}$ is the minimum of the absolute value of the eigenvalues of A.

The bigger the condition number is, the more ill-conditioned the matrix is. On the other hand, well-conditioned matrices have condition numbers close to 1. The condition number of a matrix A can be calculated by using Matlab ®, $cond(A)$.

In order to avoid such computational problems, instead of row-reducing A by Gaussian elimination, we choose to use Householder transformations (this is discussed in the beginning of Section 3). This is because Householder transformations are orthogonal and bidiagonalize A, making the unnecessary entries of A disappear and making $cond(A)$ close to 1. This makes it easier to do computations and to do them accurately.

We take an example of an orthogonal matrix, which is a well-conditioned matrix. We will calculated the condition number of A. We expect that the condition number of A should be 1. Let $A = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ A is on orthogonal matrix because $A^T A = AA^T = I$. The eigenvalues of A are $1, -1$. As A is a unitary matrix, all eigenvalues of A are of form $\lambda = e^{i\theta}$. Thus, the absolute value of eigenvalues of A is 1. Using equation (63), we have

$$cond(A) = \frac{1}{1} = 1. \tag{64}$$

Since $cond(A) = 1$, we know that A is a well-conditioned matrix.

Moreover, any unitary matrix has a condition number of 1. In particular, a Householder matrix discussed in section 3 has a condition number of 1. This is why Householder transformations are used in numerical computations to reduce a matrix instead of standard elementary row operations.

4 Enhanced SVD Item-based Filtering

4.1 Item-based Filtering

One popular algorithm used in recommender systems is item-based filtering, which is briefly introduced in [2]. Item-based filtering has 2 critical steps.

- Item Similarity Computation
  Compute the correlation between users and then to select the most similar item. Here, we will introduce the Adjusted Cosine Similarity (also known as the correlation formula) that is used in [1]

$$s_{i,j} = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_u)(R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_u)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_u)^2}}, \tag{65}$$

where U is the number of users, $R_{u,i}$ is the rating of user u on item i and $\bar{R}_u$ is the average of user u's ratings. If user u has not rated item i yet, then $R_{u,i}$ is set to some sort of average rating, which will be discussed later.

- Prediction Computation
  Once we have the set of most similar items based on $s_{i,j}$, the following target users' ratings are used to make predictions. In [1], the researching used weighted sum to calculate the predicted rating of user u on item i as

$$P_{u,i} = \frac{\sum_{\text{all items}}(s_{i,N} * R_{u,N})}{\sum_{\text{all items}}(|s_{i,N}|)}, \tag{66}$$

  where $s_{i,N}$ is the corresponding similarity between item i and N.

4.2 Enhanced SVD Item-based Filtering

As mentioned above, the recommender systems face two fundamental problems which are sparsity and scalability. We will use the SVD to address these problems. In [1], the SVD is combined with Item-based Filtering to make the original algorithm in 4.1 more effective.

1. Let matrix R of size m by n be the user-item matrix which contain ratings of m users on n items. Denote $R_{i,j}$ as the rating of user u on item j
2. Next, preprocess matrix R to eliminate missing data values by computing the average of each row and average of each column of R the replace all matrix entries that have no values with corresponding column average to get a new $R_{filled-in}$. Then, subtract the corresponding row average from all the entries of $R_{filled-in}$ to obtain the normalized matrix R_{norm}
3. Compute the singular values $\sigma_1, \cdots, \sigma_r$; then judiciously choose k such that $\sigma_1, \cdots, \sigma_k >> \sigma_{k+1}, \cdots, \sigma_r$
4. Use reduced matrices of rank k $\tilde{U}, \tilde{V}, \tilde{\Sigma}$ and do the computation to $\tilde{R} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$

Experiments in [1] indicate that the enhanced SVD item-based filtering proves to have more accurate predictions than the original Item-based Filtering. Besides, they also tested the effectiveness by combining the user-item data with demographic data.

5 Imputation

5.1 Different Approaches

As in [7], although the SVD methods have been successful in dealing with scalability, they have not efficiently solved the problem of sparsity. Sparsity leads to lower accuracy in

prediction in the recommender systems. For this reason, we also need to focus on how to fill the missing values in a set of data. There have existed several imputation methods in the literature to approximate missing values in the user-item matrices:

1. Filling by zeros: Fill missing values by zeros. This method is efficient in computation, but it does not take into account the underlying correlation structure of a data set. Thus, for a large set of data, this method could result in more inaccurate predictions.

2. Filling by random number: Fill missing values by random numbers. The advantages and disadvantages are the same for the previous method.

3. Filling by Normal Distribution: Fill the missing values by the normal distribution where μ is the user average rating and σ is the standard deviation of ratings given by other users.

4. Filling by Uniform Distribution: Fill the missing values by the uniform distribution on the interval $[a; b]$ where a is the lowest rating and b is the highest rating.

5. Filling by Item Average: Fill the missing values by the average rating given by all the users in the training set.

6. Filling by User Average: Fill the missing values by the average rating given by the active user in the training set. If the active user has not rated the item, fill the missing spot by zero. This method can distort the distribution and reduce the variance of the data, although it is simple.

7. Filling by the average of user and item averages: Fill the missing values by averaging the user's average rating and item's average rating.

8. Filling by user-based CF: There are three main steps to make predictions: (1) find all the users who have rated the target item, (2) find the similarity of the users with the active user and isolate these users called neighbors, (3) make predictions by calculating adjusted weight sum of ratings of neighbors. This method is both simple and gives accurate predictions.

9. Filling by item-based CF: The procedure is the same as the previous method but uses items instead of users.

10. Filling by the average of user- and item-based CF: Average predictions generated by user- and item-based CF.

11. Filling by support vector machine (SVM) Classifier: Replace the missing values by using the results obtained by the SVM Classifier in [10].

12. Filling by Naive Bayes Classifier: Replace the missing values by using the results obtained by the Naive Bayes Classifier in [11].

13. K Nearest Neighbor (KNN): Replace the missing values by KNN, which estimates missing values by searching K nearest neighbors and takes the weighted average of these K neighbors' ratings.

14. Filling by decision tree: Replace the results by using the decision tree in [13], which tries to minimize the error rate using training data for evaluation.

15. Filling by SVM regression: Replace missing values using SVM regression in [12] over the training set.

16. Filling by Linear Regression: Replace missing values using Linear regression. This method tries to lower the data variance of missing value estimates by using the underlying localized or global correlation structure of the data.

17. Filling by Logistic Regression: Replace missing values using the Logistic regression.

18. AdaBoost: Replace missing values using Ada Boost in [9] over the decision tree. This method is sensitive to noisy data but is less susceptible to over-fitting than most learning algorithms.

5.2 Results

Ghazanfar et al. [7] mainly focus on careful selection of imputation methods to replace missing values in data sets. They claim that with reasonable imputation approaches, it is possible to increase performance significantly. Hence, they run several experiments on four different observational data sets to evaluate the efficiency of the aforementioned imputation methods. Furthermore, during their experiments, Mean Absolute Error (MAE) is used as a predictive accuracy metric to measure how close the Recommender System's predicted value of a rating is with the true value of that rating assigned by the user.

The results of their experiments indicate that although imputation can make prediction more accurate, it is costly. Therefore, we need to consider when to impute and how much imputation is required. To answer when to impute, Ghazanfar et al. denote Θ_{sparse} as a sparsity parameter which shows the sparsity of a user's (item's) profiles in percentage. For instance, suppose $\Theta_{sparse} = 10\%$, then we only do imputation by the proposed methods from 5.1 when the sparsity of a data set is less than or equal to 10%; otherwise, we use the average of user and item averages to impute. The sparsity of data sets is calculated as follows:

$$1 - \frac{\text{number of vacant entries}}{\text{number of all possible entries}} \tag{67}$$

Reasonably, MAE is 0 when $\Theta_{sparse} = 100\%$. To answer how much imputation is needed, denote Θ_{dense} as the percentage up to which user's (or item's) profiles to be filled. For example, suppose that $\Theta_{dense} = 10\%$; then 10% of the missing values of a profile are to be filled using proposed imputation approaches, while the remaining 90% of them are filled by the average of user and item averages. In other words, 10% of imputation is sufficient to produce good accuracy.

In general, there are five interesting points mentioned in [7] from the experimental results:

1. Because of dataset characteristics such as sparsity, size, distribution, one approach may seem effective for one particular dataset but fail to produce desirable results for others.
2. Collaborative filtering and SVM prove to be more accurate and produce more computationally tractable results under all experiments.
3. While conventional approaches are straightforward, they do not produce good results. The same thing applies to many classifications and regression approaches.
4. Different recommendation algorithms, if combined systematically, can have complementary roles in generating recommendations.
5. Different imputation approaches can be used based on different situations and priorities such as time, cost, frequency of doing offline computations, available resources and so forth.

6 Main Goal

The main goal of this research is to observe whether there is any relationship between different users and items in a data set; that is, for a user-item matrix of dimension n (n choices that a user makes), we would like to consider what assumptions should be made and thus find if there is any relation such as linear and quadratic between n variables.

Mathematically speaking, assume we have a matrix R of size m by n, where $m \geq n$, m and n are denoted as the number of items and users respectively. Furthermore, every entries x_{ij} of R is between the lowest to the highest rating (from 1 to 5, or 1 to 10,...). Because we would like to determine if there is any approximate relationship among m variables of users, assume there exists a function representing such relation:

$$f(x_1, x_2, \cdots, x_n) = 0 \tag{68}$$

where $x_1, \cdots, x_n$ are the row vectors of matrix R.

A relation, for example, can be linear:

$$f(x) = c + \sum_{j=1}^{n} a_j x_j \tag{69}$$

or it can be quadratic:

$$f(x) = c + \sum_{j=1}^{n} a_j x_j + \sum_{j=1}^{n} a_j x_j^2 \tag{70}$$

The equation $f(x) = 0$ gives us a manifold in n-dimensional space. Let's say that there are k parameters that determine function f. If we know k choices that a user has made, we can predict the other choices. We suggest that if we are able to find parameter k of the function f, then k could also be the rank of matrix R (or the number of pseudo-users in [1]) after we do the truncated SVD.

Thirdly, we would like to combine imputation methods with the SVD to compare the singular values from different imputation approaches introduced in section 5.1.

In order to obtain such goals, we plan to generate a fake data set (as we would like to manipulate the characteristics of it) and apply several imputation methods and different SVD algorithms mentioned in section 3 on the fake data.

7 Experiments

We test the following algorithm to estimate ratings of users. This is an iterative procedure that produces a rating for each user on each item at the end of each iteration. The process continues until results converge.

The algorithm is tested on linear and quadratic relations between ratings of items.

If there is a quadratic relation between ratings of items r_j:

$$c + \sum_{j=1}^{n} a_j r_j + \sum_{j=1}^{n} a_j r_j^2 = 0 \tag{71}$$

If there is a linear relation between ratings of items r_j:

$$c + \sum_{j=1}^{n} a_j r_j = 0 \tag{72}$$

Our algorithm is as follows:

- Preprocess a user-item matrix R:
  1. Replace randomly chosen known entries with null entries
  2. Fill all of the null entries with 3 as imputation step
- Iterative Process

  For 20 iterations:

1. Do the truncated SVD where we vary k such that $1 \leq k \leq 10$ to find the most optimal rank k of a matrix
2. Force all the entries into range $[1, 5]$.
3. Calculate $nMAE_i$, the normalized Mean Absolute Error in percentage is

$$nMAE_i = \frac{1}{4} * 100 \sum_{b=1}^{n} \sum_{a=1}^{m} |R(i)_{ab} - R(i-1)_{ab}|, \tag{73}$$

where $R(i)_{ab}$ are new ratings of unknown entries at the i^{th} iteration of matrix R. If $nMAE_i$ decreases and approaches 0, then we confirm that all of the unknown entries appear to converge.

- Accuracy

  In order to determine how close the converged values are to the predicted values, we calculate $nMAE$ in percentage:

$$nMAE_2 = \frac{1}{4} * 100 * \sum_{b=1}^{n} \sum_{a=1}^{m} |R(i)_{ab} - R_{ab}|, \tag{74}$$

where $R(i)_{ab}$ are new ratings of randomly chosen known entries at the i^{th} iteration and R_{ab} are their original ratings.

In the end, we should expect to have the output *Rpredicted* matrix that can predict unknown entries from the original matrix.

8 Experimental Results

To begin with, we use the data set retrieved from the Grouplens website [8]. The data set is a 1682 by 943 matrix R where the number of rows are the number of items and the number of columns are the number of users. The entries of R are in the range $\{1, 2, 3, 4, 5\}$, where 1 is the least favorite and 5 is the most favorite.

In order to test our hypothesis, we only take the first 100 users and 100 items and call the matrix *Rreduce* to do the algorithm in section 7. We run our algorithm based on two assumptions mentioned in the previous section, approximate linear and quadratic relations between items. Then, we compare the accuracy by $nMAE_2$ and how fast unknown entries converge by $nMAE_i$ after doing a number of iterations in truncated SVD based on both assumptions.

8.1 Locating the optimal rank k

In order to locate the most optimal rank k in *Rreduce*, we vary k from 1 to 10 with 10 iterations of truncated SVD. We find out that for a linear relation, $k = 2$ yields the lowest $nMAE_2 = 5.0835\%$, while for a quadratic relation, $k = 4$ yields the lowest $nMAE_2 = 7.5096\%$. We see that in Figure 8.1, $\min nMAE_{2,lin} \leq \min nMAE_{2,quad}$. Hence, a linear relation seems to be a better fit to the data.

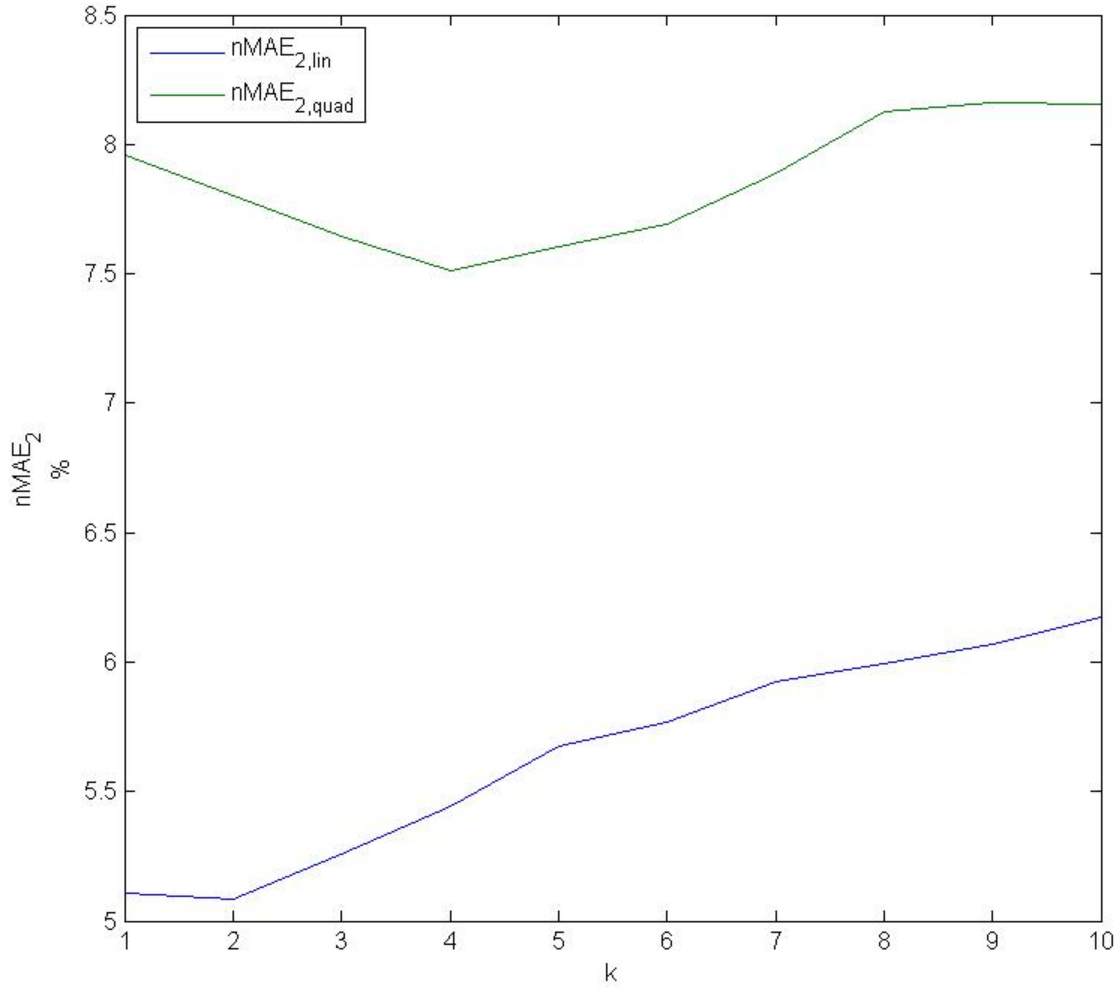

Figure 1: Comparing the most optimal k in linear and quadratic relations

8.2 Convergence of unknown entries

Next, we look at how fast all unknown entries converge after doing 20 iterations of truncated SVD. In order to do so, as we mention in section 7, we examine $nMAE_i$ by calculating the absolute value of the difference between sum of all predicted unknown values of i^{th} time and that of previous $i-1^{th}$ time. Since we expect that unknown entries converge to a certain value after being truncated, $nMAE_i$ should approach to 0.

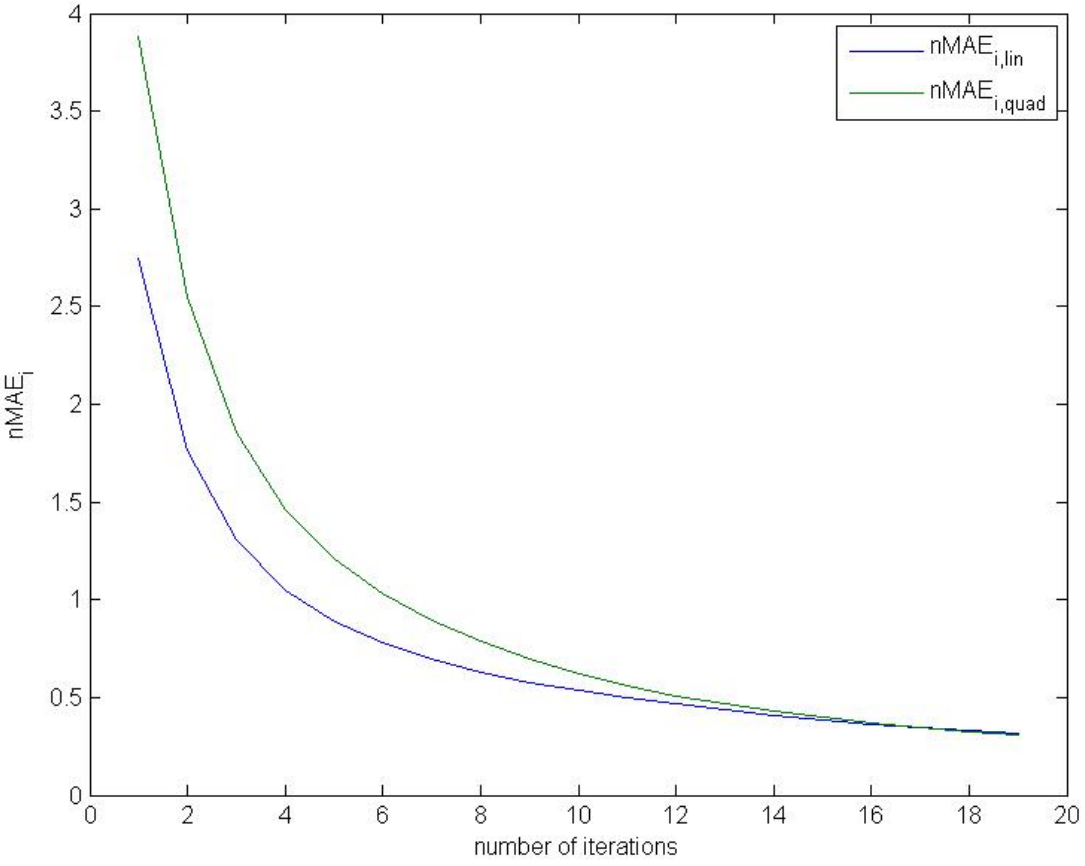

Figure 2: Comparing the convergence of unknown entries in linear and quadratic relations

Figure 8.2 indicates that in a linear relation, unknown entries converge faster than they do in a quadratic relation. Thus, it is better to assume that there is a linear relation among items.

9 Conclusion

To sum up, we see that by using only the truncated SVD algorithm, we can predict unknown entries with the smallest error. The SVD not only helps make more accurate suggestions but also only uses the most efficient rank k as possible. More specifically, instead of doing the SVD with $k = 100$ for a 100 by 100 matrix, we only have to do the SVD for only $k = 2$ if we assume there is a linear relation among items. This leads to less costly computations as well as more accurate predictions.

Moreover, even though we do not need to use all 100 singular values of the original matrix $Rreduce$, in a linear relation, only 2 largest singular values of $Rreduce$ will suffice to make accurate predictions for unknown ratings. This may be somewhat counter-intuitive, as people would think the more information they have, the better it is for them to predict the ratings. However, by only choosing the largest singular values, we both reduce the error (in this case, $nMAE_2$) as much as possible and eliminate unnecessary information (or noise) in a data set. Therefore, the whole process of prediction is much cheaper and more efficient.

Although we do not use the Item-based Filtering algorithm as described in section 4.1 during our algorithm, only using the truncated SVD already makes the prediction process much more efficient and less expensive. This indicates how powerful the SVD is in terms of alleviating the problems of scalability and sparsity as mentioned in section 1.

10 Future Work

10.1 Comparison Test

For the future work, we would like to see the effect of forcing all of the entries into the interval $[1, 5]$ (which is mentioned in section 7).

In order to do so, after locating the optimal k called k_{op}, we use this to do the truncated SVD iterative process, where we do not force all of the calculated entries into $[1, 5]$ for the last i^{th} iteration. We will call this particular predicted user-item matrix as $Rpredicted_{op}$.

We have 2 comparison tests. The first one is to compare the original ratings of randomly chosen known entries and their predicted ratings when we force all the calculated ratings into $[1, 5]$. The second one is to compare the original ratings of randomly chosen known entries and their predicted ratings when we do not force all the ratings into $[1, 5]$.

10.2 Random Test

Finally, we also want to see if the algorithm proposed in section 7 is better than if we randomly guess ratings of items that users have not yet rated. In order to do such, we will calculate the $nMAE_2$ mentioned in section 7. However, we will calculate the Mean Absolute Error in percentage $nMAE_{2rand}$ between randomly chosen known entries that are randomly

chosen from $\{1, 2, 3, 4, 5\}$ based on a uniform distribution. If $nMAE_2 << nMAE_{2rand}$, we can say that our algorithm is significantly better than random.

A Algorithm Code

```
%IMPORTANT VARIABLES
%Rreduce is the original user-item matrix
%RreducebisCheck is matrix with random entries of 0 with same size pf
%RreducebisNew is the final R predicted that we look for without range
%tCEll{numbInt} is the final R in range
%nMAE1 is MAE in % that tells whether the unknown entries converge after
%doing certain number of tSVD
%nMAE2 is MAE in $ that tells how accurate the predicted entries are
%compared to th original known entries
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
%LINEAR RELATIONSHIP
%u=importdata('C:\Users\anhvnguyen\Desktop\u.data');
u=importdata('C:\Users\duynguyen\Desktop\grouplens\u.data');
%%PARAMETERS
reduc=500; %number of rows we wanna use in u data to create matrix R
randPerm=30;
%number of permutations for Rreducebis-->number of zeros to put in known entries
%randPerm<length(ultd)
numbInt=10;
k=1:10; %k dimension <=min(n,m)

%% CREATE ULTD MATRIX
%create a smaller ultd matrix from u where we only take first reduc users
%on first reduc items
%of u
check = numel(u( u(:,1)<=reduc & u(:,2)<=reduc));
%check to see if have the same elements as the ones below
u=u(1:100000,1:3); %only take userid, itemid and ratings
ultd=sortrows(u,1); %sort ultd according to first col descendingly
limitUser=max(find(ultd(:,1)==reduc)); %only take rows where first col<reduc
ultd=ultd(1:limitUser,1:3);
ultd=sortrows(ultd,2); %sort ultd according to second col descendingly
limitItem=max(find(ultd(:,2)==reduc)); %only take rows where second col <reduc
ultd=ultd(1:limitItem,1:3);

%% CREATE MATRIX R (reduc by reduc size)
%reform user-item matrix where number of columns are users and number of
```

```
%rows are items, where the ratings are from 1 to 5
Rreduce=zeros(reduc,reduc);
Rcell=cell(reduc,1);
%Create cell array where each cell contains info of ratings of each user
for j=1:reduc %for number of users from 1 to reduc
Rcell{j}=ultd(ultd(:,1)==j,:);
for i=1:length(Rcell{j})
    Rreduce(Rcell{j}(i,2),j)=Rcell{j}(i,3);
end
end

%%
%%RREDUCE/PREPROCESS USER-ITEM MATRIX
randomVec=randperm(length(ultd),randPerm);
%randomly put randPerm number of zeros in randomly chosen entries
%from originally known ratings
Rreducebis=Rreduce; %create a new sparser matrix Rreducebis
for i=1:randPerm
index=ultd(randomVec(i),:);
%locate indices of randomly chosen known entries
%so that we can do the testforce, testnoforce
Rreducebis(index(2),index(1))=0;
end
RreducebisCheck=Rreducebis;
%create a new matrix RreducebisCheck where some of the known entries are now 0s
for row=1:reduc
    for col=1:reduc
if Rreducebis(row,col)==0
%fill 3 to any null entries of matrix R to create a new matrix R
%where null entries are 3
    Rreducebis(row,col)=3;
end
    end
end

%%
%%ITERATIVE PROCESS
tCell=cell(numbInt+1,1);
%create tCell as truncated Cell that stores matrices after number of tSVD
RreducebisNew=Rreducebis; %create matrix RreducebisNew that is originally Rreducebis
```

```
nMAE2=zeros(length(k),1);
%iterative process where we vary k and do tSVD for numbInt of times
for rank=1:length(k)
for p=1:numbInt+1
    tCell{p}=RreducebisNew;
    [U,S,V]=svdsecon(RreducebisNew,k(rank));
    %do tSVD for numbInt times with k dimension
    RreducebisNew=U*S*transpose(V);
    %calculate new RreducebisNew
    for r=1:reduc
        for c=1:reduc
            if RreducebisCheck(r,c)~=0
            %force all known entries of Rreducebis to original ones
                RreducebisNew(r,c)=RreducebisCheck(r,c);
            end
    if RreducebisNew(r,c)>5        %force all the entries in the range [1;5]
        RreducebisNew(r,c)=5;
    end
       if RreducebisNew(r,c)<1
           RreducebisNew(r,c)=1;
       end
        end
    end
end

%%
%%CALCULATE nMAE2 to see how accurate the prediction is
for r=1:reduc
    for c=1:reduc
        if (RreducebisCheck(r,c)==0) && (Rreduce(r,c)~=0)
%calculate only the entries that used to be randomly 0
%in RreducebisCheck but not 0 in original matrix Rreduce1
%randPerm is the number of elements that we put 0s
result2=1/4*100*1/randPerm*abs(RreducebisNew(r,c)-Rreduce(r,c));
nMAE2(rank,1)=result2+nMAE2(rank,1);
        end
    end
end
end
%%
```

```
%%COMPUTE AGAIN TSVD WITH AND WITHOUT RANGE AFTER LOCATING OPTIMAL K
%tCell becomes cell thats truncated when k=k_op

RreducebisNewop=Rreducebis;
k_op=find(nMAE2==min(nMAE2)) %locate the most optimal k as k_op to recompute

for p=1:numbInt
    tCell{p}=RreducebisNewop;
    [U,S,V]=svdsecon(RreducebisNewop,k_op);
    %do tSVD for numbInt times with k dimension
    RreducebisNewop=U*S*transpose(V); %calculate new RreducebisNew
    for r=1:reduc
        for c=1:reduc
            if RreducebisCheck(r,c)~=0
            %force all known entries of Rreducebis to original ones
                RreducebisNewop(r,c)=RreducebisCheck(r,c);
            end
    if RreducebisNewop(r,c)>5        %force all the entries in the range [1;5]
        RreducebisNewop(r,c)=5;
    end
    if RreducebisNewop(r,c)<1
           RreducebisNewop(r,c)=1;
     end
        end
    end
end
[U,S,V]=svdsecon(tCell{numbInt-1},k_op);
%the last time of tSVD, we don't force all the entries back to range
%[1;5] and [1;25] to check if it is close to the previous tSVD
RreducebisNewop=U*S*transpose(V);
plot(nMAE2)
```

References

[1] M. Vozalis and K. Margaritis, *Applying SVD on Generalized Item-based Filtering*, International Journal of Computer Science and Application (2006), 27-51.

[2] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, *Item-Based Collaborative Filtering Recommendation Algorithms*, GroupLens Research Group/Army HPC Research Center (2001).

[3] M. Stoll, *A Krylov-Schur Approach to Truncated SVD*, Linear Algebra and its Application (2012).

[4] G. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, J.Soc. Indust. Appl. Math. Ser. B Numer. Anal., 2 (1965), 205-224.

[5] G. Forsythe, M. Malcolm, and C. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, INC (1977), 41-44.

[6] V. Vijayan, *Fast SVD and PCA* (July, 2014). http://www.mathworks.com/matlabcentral/fileexchange/47132-fast-svd-and-pca.

[7] M. Ghazanfar and A. Prugel-Bennet, *The Advantage of Careful Imputation Sources in Sparse Data-Environment of Recommender Systems: Generating Improved SVD-based Recommendations*, Informatica (December, 2012), 61-92.

[8] Grouplens. http://grouplens.org/datasets/movielens/100k/.

[9] M. Collins, RE. Schapire, and Y. Singer, *Logistic regression, AdaBoost and Bregman distances*, Springer (2002).

[10] M. Ghazanfar and A. Prugell-Bennet, *Building Switching Hybrid Recommender System Using Machine Learning Classifiers and Collaborative Filtering*, IAENG International Journal of Computer Science (2010), 272-287.

[11] ———, *An Improved Switching Hybrid Recommender System Using Naive Bayes Classifier and Collaborative Filtering*, Proceedings of The International Multi Conference of Engineers and Computer Scientists 2010 (2010). http://eprints.ecs.soton.ac.uk/18483/.

[12] C. Chang and C. Lin, *Libsvm: A library for support vector machines*, ACM Trans. Intell.Syst. Technol. (May 2011). http://doi.acm.org/10.1145/.

[13] MA. Friedl and CE. Brodley, *Decision tree classification of land cover from remotely sensed data*, Remote sensing of environment, Elsevier (1997).