# A Guide to Singular Value Decomposition for Collaborative Filtering

Chih-Chao Ma

Department of Computer Science, National Taiwan University, Taipei, Taiwan

**Abstract**

As the market of electronic commerce grows explosively, it is important to provide customized suggestions for various consumers. Collaborative filtering is an important technique which models and analyzes the preferences of customers, and gives suitable recommendations. Singular Value Decomposition (SVD) is one of the popular algorithms used for collaborative filtering. However, directly applying conventional SVD algorithms to collaborative filtering may result in poor performance. In this report, we discuss problems that beginners may face and present effective SVD variants for collaborative filtering.

## 1 Introduction

Collaborative filtering is a technique of predicting the preferences and interests of users from the collections of taste information given by a large number of users. It has the basic assumption that people who agreed in the past will also agree in the future. Collaborative filtering can be used for a recommendation system that automatically suggests a user his/her preferred products.

### 1.1 Problem Definition

Suppose that a database collects the preferences of $n$ users for $m$ objects as numerical scores. For example, a user can score a movie he or she has watched by a rating of $1-5$ stars. Usually, a user does not score all objects in the database. Furthermore, some users may score many objects, but others only score a few.

Let $V \in \mathbb{R}^{n\times m}$ be the matrix of the collected scores in the database, and $I \in \{0,1\}^{n\times m}$ be its indicator such that $I_{ij} = 1$ if object $j$ is scored by user $i$ and 0 if the score is missing. In most cases, $V$ is sparse and unbalanced so that the numbers of scores for every user or object are unequal with a large variance. The existing scores in $V$ work as the training data of collaborative filtering algorithms, and the goal is to predict the missing scores in the database. Let $A \in \mathbb{R}^{n\times m}$ be a sparse matrix including all or part of the missing votes as its nonzero entries. A collaborative filtering algorithm aims to predict the values in $A$.

The performance of collaborative filtering can be measured by the error between the prediction values and the ground-truth. A common and efficient measure is *Root Mean Square Error* (RMSE). Consider the prediction matrix $P \in \mathbb{R}^{n\times m}$ and the ground-truth answer matrix $A \in \mathbb{R}^{n\times m}$. Let $J \in \{0,1\}^{n\times m}$ be the indicator of $A$. The RMSE between the prediction $P$ and the answer $A$ is defined as

$$\mathrm{RMSE}(P,A) = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{m} J_{ij}(A_{ij}-P_{ij})^2}{\sum_{i=1}^{n}\sum_{j=1}^{m} J_{ij}}}. \tag{1}$$

Besides the function of error measurement, the formation and distribution of matrix $A$ may also affect the evaluation of algorithms. For example, if the scores in $A$ are randomly sampled from all existing scores in the database, a user who gives more scores in the database tends to have more scores in $A$.

A good example of test data generation is *the Netflix Prize* [Bennett and Lanning, 2007], which is a grand contest for collaborative filtering on how people like or dislike movies. The database used for the Netflix Prize contains over 100 million scores for $480,189$ users and $17,770$ movies. The test data of the Netflix Prize, invisible to the competitors, are selected from the newest scores of each user. More precisely, they select a fixed number of most recent scores made by each user, regardless of the total number of scores given by that user. Then, the collection of these newest scores is randomly divided into three sets, named probe, quiz, and test sets. The ground-truth scores of probe set are given to the competitors as well as other older scores in the database. The competitors are then asked to predict the scores of quiz and test sets. Usually, the competitors use part of the training data to test their algorithms in an offline mode. That is, a validation dataset divided from the whole training data is needed.

These sets are generated so that the validation and test data have a similar distribution. Since the test data are selected from the newest scores in the database, this selection matches the goal of predicting future scores from previous ones. The similarity between validation and test data ensures that the performance of an algorithm is consistent in validation and testing. For each user, the number of unknown scores to be predicted is roughly the same. Hence, the error of predictions for users who gave only a few scores in the past tends to be larger due to the lack of information. This situation is challenging to collaborative filtering algorithms.

## 1.2 Related Work

Several conventional methods are available for collaborative filtering, and they appear in the competition of the Netflix Prize. We briefly describe these methods as well as their strengths and weaknesses. More details of some methods are shown in later chapters.

The most intuitive way to predict the score of an object given by a user is to ask other users who have scored the object. Instead of merely using the mean score of that object among all users, we consider only those users similar to the target user. This method is known as *K-Nearest Neighbor*, abbreviated as KNN. The KNN algorithm can be applied directly to the scores [Paterek, 2007], or to the residuals of another algorithm as a post-processing method [Bell et al., 2007a]. The key point of KNN methods is the definition of similarity. Usually the similarity is computed by the feature values of users and objects which represent the characteristics of them in numerical values, but these features are difficult to find.

Another type of method directly finds the feature values of each user and object, and predicts the unknown scores by a prediction function using those features. Usually such algorithms involve a matrix factorization which constructs a feature matrix for users and for objects, respectively. These kinds of algorithms includes *Non-Negative Matrix Factorization* and *Singular Value Decomposition*, which have numerous variants and also play important roles in the Progress Prize of the Netflix Prize in the year 2007 [Bell et al., 2007b].

These types of methods assumes that the score values depend on a predefined prediction function.

Besides the widely-used algorithms described above, there are also other methods which give good performance in the competition of the Netflix Prize. One of them uses a model called *Restricted Boltzmann Machines* [Salakhutdinov et al., 2007], which has proved to perform well in the Netflix Prize, either by itself or in a linear combination with other algorithms. Another approach is *Regression*, which predicts the unknown scores by taking the scores in the training data as observations. On the other hand, if one wants to combine the results of several algorithms for better accuracy, regression is also useful to find the weights of linear combination.

### 1.3 Summary

In this report, we focus on Singular Value Decomposition, which is the most popular algorithm for the Netflix Prize. Section 2 shows details of SVD algorithms, including the conventional way used for information retrieval and variants which are more suitable for collaborative filtering. We find that the choice of optimization methods is important for SVD algorithms to give good performance. In Section 3 we give the experimental results of algorithms. The conclusions are given in Section 4.

## 2 Singular Value Decomposition

Singular Value Decomposition, abbreviated as SVD, is one of the factorization algorithms for collaborative filtering [Zhang et al., 2005]. This type of algorithm finds the features of users and objects, and makes predictions based on these factors. Some factorization algorithms have additional restrictions on each single feature value, or between the feature vectors of multiple users, but Singular Value Decomposition does not impose restrictions and is easier to implement.

### 2.1 Formulation

Suppose $V \in \mathbb{R}^{n \times m}$ is the score matrix of $m$ objects and $n$ users, and $I \in \{0,1\}^{n \times m}$ is its indicator. The SVD algorithm finds two matrices $U \in \mathbb{R}^{f \times n}$ and $M \in \mathbb{R}^{f \times m}$ as the feature matrix of users and objects. That is, each user or object has an $f$-dimension feature vector and $f$ is called the dimension of SVD. A prediction function $p$ is used to predict the values in $V$. The value of a score $V_{ij}$ is estimated by $p(U_i, M_j)$, where $U_i$ and $M_j$ represent the feature vector of user $i$ and object $j$, respectively. Once $U$ and $M$ are found, the missing scores in $V$ can be predicted by the prediction function.

The optimization of $U$ and $M$ is performed by minimizing the sum of squared errors between the existing scores and their prediction values:

$$E = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} I_{ij} (V_{ij} - p(U_i, M_j))^2 + \frac{k_u}{2} \sum_{i=1}^{n} \|U_i\|^2 + \frac{k_m}{2} \sum_{j=1}^{m} \|M_j\|^2, \qquad (2)$$

where $k_u$ and $k_m$ are *regularization coefficients* to prevent overfitting.

The most common prediction function is the dot product of feature vectors. That is, $p(U_i, M_j) = {U_i}^T M_j$. The optimization of $U$ and $M$ thus becomes a

matrix factorization problem where $V \approx U^T M$. But in most applications, scores in $V$ are confined to be in an intervel $[a, b]$, where $a$ and $b$ are the minimal and maximal score values defined in the domain of data. For example, if the users rate the objects as $1-5$ stars, then the scores are bounded in the interval $[1, 5]$. One way is to clip the values of dot products. For example, we can bound the values of ${U_i}^T M_j$ in the interval $[0, b-a]$ and the prediction function becomes $a$ plus the bounded dot product. Hence, the prediction function is:

$$p(U_i, M_j) = \begin{cases} a & \text{if } {U_i}^T M_j < 0, \\ a + {U_i}^T M_j & \text{if } 0 \le {U_i}^T M_j \le b-a, \\ b & \text{if } {U_i}^T M_j > b-a. \end{cases} \tag{3}$$

When using the prediction function (3), the objective function and its negative gradients have the following forms:

$$\begin{aligned} E &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} I_{ij} (V_{ij} - p(U_i, M_j))^2 \qquad (4) \\ &+ \frac{k_u}{2} \sum_{i=1}^{n} \|U_i\|^2 + \frac{k_m}{2} \sum_{j=1}^{m} \|M_j\|^2, \\ -\frac{\partial E}{\partial U_i} &= \sum_{j=1}^{m} I_{ij} \left( (V_{ij} - p(U_i, M_j)) M_j \right) - k_u U_i, i = 1, ..., n, \qquad (5) \\ -\frac{\partial E}{\partial M_j} &= \sum_{i=1}^{n} I_{ij} \left( (V_{ij} - p(U_i, M_j)) U_i \right) - k_m M_j, j = 1, ..., m. \qquad (6) \end{aligned}$$

One can than perform the optimization of $U$ and $M$ by gradient descent:

**Algorithm 1 (Batch learning of Singular Value Decomposition)**
Select a learning rate $\mu$, and regularization coefficients $k_u, k_m$.

1. Set the starting values of matrices $U, M$.
2. Repeat
   (a) Compute gradients $\nabla_U$ and $\nabla_M$ by (5) and (6).
   (b) Set $U \leftarrow U - \mu \nabla_U$, $M \leftarrow M - \mu \nabla_M$.

   until the validation RMSE starts to increase.

The tuneable parameters are the learning rate $\mu$, and the regularization coefficients $k_u$ and $k_m$ for user and object features, respectively. The learning rate affects the learning time, and too large a value may lead to divergence. In general, a smaller learning rate gives better performance, but the learning time is also longer.

Another setting that affects the performance is the starting point of feature values. One way is to use random values in a specific range, but unstable performances may occur if these values are too random. In Algorithm 1, the starting point can be the average of all existing scores $\bar{V}$:

$$U_{ij}, M_{ij} = \sqrt{\frac{\bar{V} - a}{f}} + n(r) \text{ for each } i, j, \tag{7}$$

where $a$ is the lower bound of scores, $f$ is the dimension of SVD algorithms, and $n(r)$ is a random noise with uniform distribution in $[-r, r]$. According to the prediction function (3), it is likely to predict all values to the global average $\bar{V}$ with some noises at the start of the algorithm. Without the random noise in (7), the features of a user or an object will be the same as they always have the same gradients during optimization. A small value of $r$ is usually enough.

Batch learning is the standard method for SVD. However, it is not good for a large-scale but sparse training matrix $V$, which is common for collaborative filtering. The values of the gradients have a large variance under such a situation, and a small learning rate is required to prevent divergence. In Section 2.2, we show some more suitable variants for collaborative filtering.

## 2.2 Variants of SVD

We have mentioned that batch learning may not perform well. Instead, one can use *incremental learning*, which modifies only some feature values in $U$ and $M$ after scanning part of the training data. For example, if one user $i$ is considered at a time, the variables related to this user are:

(1) $U_i$: the feature vector of user $i$,

(2) each $M_j$ with $I_{ij} = 1$: the feature vectors of objects scored by user $i$.

If one considers only terms related to user $i$, the objective function becomes:

$$E_i = \frac{1}{2}\sum_{j=1}^{m} I_{ij}(V_{ij} - p(U_i, M_j))^2 + \frac{k_u}{2}\|U_i\|^2 + \frac{k_m}{2}\sum_{j=1}^{m} I_{ij}\left(\|M_j\|^2\right), \quad (8)$$

with the negative gradients

$$-\frac{\partial E_i}{\partial U_i} = \sum_{j=1}^{m} I_{ij}\left((V_{ij} - p(U_i, M_j))M_j\right) - k_u U_i, \quad (9)$$

$$\begin{aligned} -\frac{\partial E_i}{\partial M_j} &= I_{ij}\left((V_{ij} - p(U_i, M_j))U_i\right) - k_m I_{ij}\left(M_j\right) \\ &= I_{ij}\left[(V_{ij} - p(U_i, M_j))U_i - k_m M_j\right], j = 1, ..., m. \end{aligned} \quad (10)$$

Note that if $I_{ij} = 0$, then the gradients of $M_j$ is 0. Hence the variables of an object not scored by user $i$ are not updated.

This incremental learning approach is different from batch learning, as the sum of $E_i$ over all users does not lead to the objective function $E$ given in (4):

$$\begin{aligned} \sum_{i=1}^{n} E_i = &\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m} I_{ij}(V_{ij} - p(U_i, M_j))^2 \\ &+\frac{k_u}{2}\sum_{i=1}^{n}\|U_i\|^2 + \frac{k_m}{2}\sum_{j=1}^{m}\sum_{i=1}^{n} I_{ij}\left(\|M_j\|^2\right). \end{aligned} \quad (11)$$

Each object feature vector $M_j$ has an additional regularization coefficient $\sum_{i=1}^{n} I_{ij}$, which is equal to the number of existing scores for object $j$. Therefore, an object with more scores has a larger regularization coefficient in this incremental learning approach.

An extreme case of incremental learning is to update the features after looking at each single score. That is, we consider the following objective function and negative gradients for each $V_{ij}$ which is not missing:

$$E_{ij} = \frac{1}{2}(V_{ij} - p(U_i, M_j))^2 + \frac{k_u}{2}\|U_i\|^2 + \frac{k_m}{2}\|M_j\|^2, \quad (12)$$

$$-\frac{\partial E_{ij}}{\partial U_i} = (V_{ij} - p(U_i, M_j))M_j - k_u U_i, \quad (13)$$

$$-\frac{\partial E_{ij}}{\partial M_j} = (V_{ij} - p(U_i, M_j))U_i - k_m M_j. \quad (14)$$

We denote this type of learning as *complete incremental learning*, and the one in (8), which considers multiple scores at a time as *incomplete incremental learning*.

The summation of $E_{ij}$ over all existing votes is

$$\sum_{i=1}^{n}\sum_{j=1}^{m} E_{ij} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m} I_{ij}(V_{ij} - p(U_i, M_j))^2 + \frac{k_u}{2}\sum_{i=1}^{n}\sum_{j=1}^{m} I_{ij}\left(\|U_i\|^2\right) + \frac{k_m}{2}\sum_{j=1}^{m}\sum_{i=1}^{n} I_{ij}\left(\|M_j\|^2\right). \quad (15)$$

In this case, each user and object has a regularization coefficient which is proportional to the number of existing scores related to that user or object.

The following algorithms describe the optimization of $U$ and $M$ with incremental learning. Algorithm 2 performs incomplete incremental learning, and Algorithm 3 does complete incremental learning, which updates the feature values after examining each single score. The starting points of $U$ and $M$ in these algorithms can also be (7).

**Algorithm 2 (Incomplete incremental learning of Singular Value Decomposition)** Select a learning rate $\mu$, and regularization coefficients $k_u, k_m$.

1. Set the starting values of matrices $U, M$.
2. Repeat
   (a) For $i = 1$ to $n$
      i. Compute gradients $\nabla_{U_i}$ by (9).
      ii. Compute gradients $\nabla_{M_j}$ by (10) for $j = 1$ to $m$.
      iii. Set $U_i \leftarrow U_i - \mu\nabla_{U_i}$.
      iv. Set $M_j \leftarrow M_j - \mu\nabla_{M_j}$ for $j = 1$ to $m$.

   until the validation RMSE starts to increase.

**Algorithm 3 (Complete incremental learning of Singular Value Decomposition)** Select a learning rate $\mu$, and regularization coefficients $k_u, k_m$.

1. Set the starting values of matrices $U, M$.
2. Repeat
   (a) For each existing score $V_{ij}$ in training data

i. Compute gradients $\nabla_{U_i}$ and $\nabla_{M_j}$ by (13) and (14).
ii. Set $U_i \leftarrow U_i - \mu\nabla_{U_i}$, $M_j \leftarrow M_j - \mu\nabla_{M_j}$.

until the validation RMSE starts to increase.

Algorithms 2 and 3 cause different optimization results and performance, as the sums of objective functions (11) and (15) are different when regularization is used. If we modify the objective function (8) to

$$\begin{aligned} E_i &= \frac{1}{2}\sum_{j=1}^{m} I_{ij}(V_{ij} - p(U_i, M_j))^2 \\ &+ \frac{k_u}{2}\sum_{j=1}^{m} I_{ij}\left(\|U_i\|^2\right) + \frac{k_m}{2}\sum_{j=1}^{m} I_{ij}\left(\|M_j\|^2\right), \end{aligned} \tag{16}$$

then the sums of objective functions will be the same. In Section 3, we will use this approach for the comparison between incomplete and complete incremental learning in the experiments.

Instead of using incremental learning, we can also increase the learning speed of batch learning by adding a momentum $\lambda$ in the gradient descent procedure. That is, Algorithm 1 is modified to the following Algorithm:

**Algorithm 4 (Batch learning of SVD with Momentum)**
Select a learning rate $\mu$, and regularization coefficients $k_u, k_m$.

1. Set the starting values of matrices $U, M$.
2. Set the movement of matrices $\Delta U \leftarrow \mathbf{0}^{f\times n}, \Delta M \leftarrow \mathbf{0}^{f\times m}$.
3. Repeat
   (a) Set $\Delta U \leftarrow \lambda\Delta U, \Delta M \leftarrow \lambda\Delta M$.
   (b) Compute gradients $\nabla_U$ and $\nabla_M$ by (5) and (6).
   (c) Set $\Delta U \leftarrow \Delta U - \mu\nabla_U, \Delta M \leftarrow \Delta M - \mu\nabla_M$
   (d) Set $U \leftarrow U + \Delta U$, $M \leftarrow M + \Delta M$.

   until the validation RMSE starts to increase.

Batch learning usually suffers from divergence when using the same learning rate as incremental learning, but a small value of learning rate also makes the algorithm infeasible in time. The momentum term can accumulate the movements of variables, making the optimization faster under a small learning rate. In small-scale data sets, batch learning can outperform incremental learning by adding the momentum term. However, batch learning cannot give reasonable performances in large-scale data sets even with the momentum.

One can perform line search to find a better learning rate for gradient descent. This approach requires fewer iterations, but its computation time is longer due to the overhead of line search.

## 2.3 Further Improvements

With proper optimization settings, the SVD algorithm is able to give a good performance. However, some variants have the potential to make more accurate predictions. A simple one is to add a per-user bias $\boldsymbol{\alpha} \in \mathbb{R}^{n\times 1}$ and a per-object bias $\boldsymbol{\beta} \in \mathbb{R}^{m\times 1}$ on the prediction function [Paterek, 2007]:

$$p(U_i, M_j, \alpha_i, \beta_j) = a + {U_i}^T M_j + \alpha_i + \beta_j, \tag{17}$$

where $\alpha_i$ is the bias of user $i$ and $\beta_j$ is the bias of object $j$.

The biases are updated like the feature values. For the "complete incremental learning," using a single score $V_{ij}$ at a time gives the following negative gradients of bias $\alpha_i$ and $\beta_j$:

$$\begin{aligned} E_{ij} &= \frac{1}{2}(V_{ij} - p(U_i, M_j, \alpha_i, \beta_j))^2 \\ &\quad + \frac{k_u}{2}\|U_i\|^2 + \frac{k_m}{2}\|M_j\|^2 + \frac{k_b}{2}(\alpha_i^2 + \beta_j^2), \end{aligned} \tag{18}$$

$$-\frac{\partial E_{ij}}{\partial \alpha_i} = (V_{ij} - p(U_i, M_j, \alpha_i, \beta_j)) - k_b\alpha_i, \tag{19}$$

$$-\frac{\partial E_{ij}}{\partial \beta_j} = (V_{ij} - p(U_i, M_j, \alpha_i, \beta_j)) - k_b\beta_j, \tag{20}$$

where $k_b$, the regularization coefficient of biases, is similar to $k_m$ and $k_u$. Of course, the learning rate of the biases can be different from each other.

Another variant of SVD algorithms, called constrained SVD, adds additional constraints to each user feature vector $U_i$ [Salakhutdinov and Mnih, 2008]. In this algorithm, the user feature matrix $U \in \mathbb{R}^{f\times n}$ is replaced by a matrix $Y \in \mathbb{R}^{f\times n}$ of the same size, and an object constraint matrix $W \in \mathbb{R}^{f\times m}$ shifts the values of user features:

$$U_i = Y_i + \frac{\sum_{k=1}^{m} I_{ik} W_k}{\sum_{k=1}^{m} I_{ik}}, \tag{21}$$

where $W_k$ is the $k$-th column of the matrix $W$. In other words, the feature vector $U_i$ is the user-dependent feature vector $Y_i$ plus an offset $\frac{\sum_{k=1}^{m} I_{ik} W_k}{\sum_{k=1}^{m} I_{ik}}$, which is the mean of object constraint feature vectors on the objects scored by user $i$. Each $W_k$ contributes to a user feature $U_i$ if user $i$ scores object $k$.

Under this setting, the prediction function becomes:

$$p(U_i, M_j) = a + \left(Y_i + \frac{\sum_{k=1}^{m} I_{ik} W_k}{\sum_{k=1}^{m} I_{ik}}\right)^T M_j, \tag{22}$$

and the values of $Y$ and $W$ can also be updated by gradient descent with the use of regularization coefficients $k_y$ and $k_w$. However, the optimization has a high time complexity for complete incremental learning. We give further discussion and propose a feasible solution, called *compound SVD*, in [Ma, 2008]. A compound SVD algorithm which combines the biases and constraints is used, and it gives a significantly better performance than the SVD algorithm with complete incremental learning.

# 3 Experiments

We use the *Movielens* data set [Miller et al., 2003], the *Netflix* data set [Bennett and Lanning, 2007], and three smaller data sets sampled by the *Netflix* data set for evaluation. Each data set is divided into two disjointed training and test sets. The performances are measured by test RMSE (1), which is presented along with training time.

## 3.1 Data Sets

The *Movielens* data set is provided by GroupLens [Konstan et al., 1997], a research group which collects and supplies several collaborative filtering data in different domains. The data set contains $1,000,209$ scores for $6,040$ users and $3,706$ movies. Each user gives at least 20 scores. Since the data set does not come with a training and test split, we randomly select three scores from each user as test data. Hence the ratio of training and test sets is similar to the *Netflix* data set.

The other data set is given by the Netflix Prize as mentioned in Section 1. There are $100,480,507$ scores for $480,189$ users and $17,770$ movies. Each user has at least 20 scores in the beginning, but the competition organizers intentionally perturb the data set so some users may have less than 20 scores. The test data used for experiments are exactly the probe set provided by the Netflix Prize, which includes $1,408,395$ scores. The training data are the remaining $99,072,112$ scores.

Besides the above two data sets, we sample three small data sets with different attributes from the *Netflix* data set. We merge the training and test sets of *Netflix*, and randomly sample 1% of the users and objects as the first data set *small1*. Then we remove the users and objects which have no scores in *small1*, and select 138 scores as a test set such that the training/test ratios of *small1* and *Netflix* are similar. The test scores have a uniform distribution on users, which is the same as the formulations in *Movielens* and *Netflix* data sets. The second data set *small2* consists of the users and objects with 1% highest number of scores in *Netflix*, hence it is denser than *small1* and *Netflix*. The third data set *small3* contains the same objects as those in *small2*. However, it contains the users with the 56 highest number of scores in *small2*, so the numbers of scores in *small3* and *small1* are close. We also select 138 scores as the test data of *small2* and *small3* by the same way. Table 1 lists the statistics of data sets.

Table 1: Statistics of data sets

| Dataset | # user | # object | # training score | # test score | density |
|---|---|---|---|---|---|
| *Movielens* | 6,040 | 3,706 | 982,089 | 18,120 | 4.61% |
| *Netflix* | 480,189 | 17,770 | 99,072,112 | 1,408,395 | 1.18% |
| *small1* | 2,917 | 167 | 9,734 | 138 | 2.00% |
| *small2* | 4,802 | 178 | 702,956 | 138 | 82.24% |
| *small3* | 56 | 178 | 9,809 | 138 | 98.40% |

## 3.2 Experimental Results

We implement the SVD algorithms in C, since they are the most time-consuming in computation. We use MATLAB for data preprocessing and postprocessing as it is convenient for matrix manipulation. For different SVD algorithms, the test RMSEs are plotted against the training time. Algorithms used for comparison are listed below:

**AVGB**: A simple baseline predictor that predicts the score $P_{ij}$ for user $i$ and object $j$ as $\mu_j + b_i$, where $\mu_j$ is the mean score of object $j$ in training data, and $b_i$ is the average bias of user $i$ computed as:

$$b_i = \frac{\sum_{j=1}^{m} I_{ij}(V_{ij} - \mu_j)}{\sum_{j=1}^{m} I_{ij}}. \tag{23}$$

**SVD**: Algorithm 3, which uses incremental learning with regularization terms.
**SVDNR**: It is the same as algorithm **SVD** but has the regularization coefficients set to 0. This algorithm works as another baseline.
**SVDUSER**: The SVD algorithm which has the same gradients as **SVD**, but uses incomplete incremental learning which updates the feature value after scanning all scores of a user instead of each single score.
**SVDBATCH**: Algorithm 4, which use batch learning with a learning momentum 0.9.
**CSVD**: A compound SVD algorithm which incorporates per-user and per-object biases and the constraints on feature vectors. Implementation details are described in [Ma, 2008].

First, we give a comparison between batch learning and incremental learning. We apply **SVDNR** and **SVDBATCH** on the data sets, where the regularization coefficients of **SVDBATCH** are set to 0. The test RMSEs with respect to training time for the four data sets *small1*, *small2*, *small3*, and *Movielens* are shown in Figure 1. The learning rate in **SVDNR** is 0.005 for each data set. For the algorithm **SVDBATCH**, the learning rates are $0.0005, 0.0003, 0.001$ and 0.0002 for *small1*, *small2*, *small3*, and *Movielens*, respectively. These values give the lowest RMSEs in the shortest time without making the algorithm diverge.

Batch learning with a high learning momentum has similar performances to incremental learning in small data sets, but the decrease of RMSE is usually unstable. Batch learning is considered inefficient in medium or large data sets like *Movielens* or *Netflix*. In figure 1, **SVDBATCH** needs more training time to give the same performance for *Movielens* data set. Moreover, **SVDBATCH** requires over $10,000$ seconds to get the lowest RMSE for *Netflix* data set, while **SVDNR** takes about 300 seconds to reach a similar RMSE.

Then, we show the performances of **SVDNR**, **SVD**, **SVDUSER**, and **CSVD** for two larger data sets *Movielens* and *Netflix*. For each data set, the matrices $M, U$ in **SVDNR**, **SVD** and **SVDUSER** as well as $M, Y$ in **CSVD** are initialized by (7). The values of the constraint matrix $W$ and biases $\alpha, \beta$ in **CSVD** are all initialized as 0. As parameters, the learning rate $\mu_v$ in **SVDNR** and **SVD** is 0.003 for the *Movielens* data set and 0.001 for the *Netflix* data set, but **SVDUSER** uses various learning rates for comparison. The regularization coefficients $k_u, k_m$ are 0 in **SVDNR**, while **SVD** and **SVDUSER** use regularization coefficients leading to the best performances, which are both 0.05

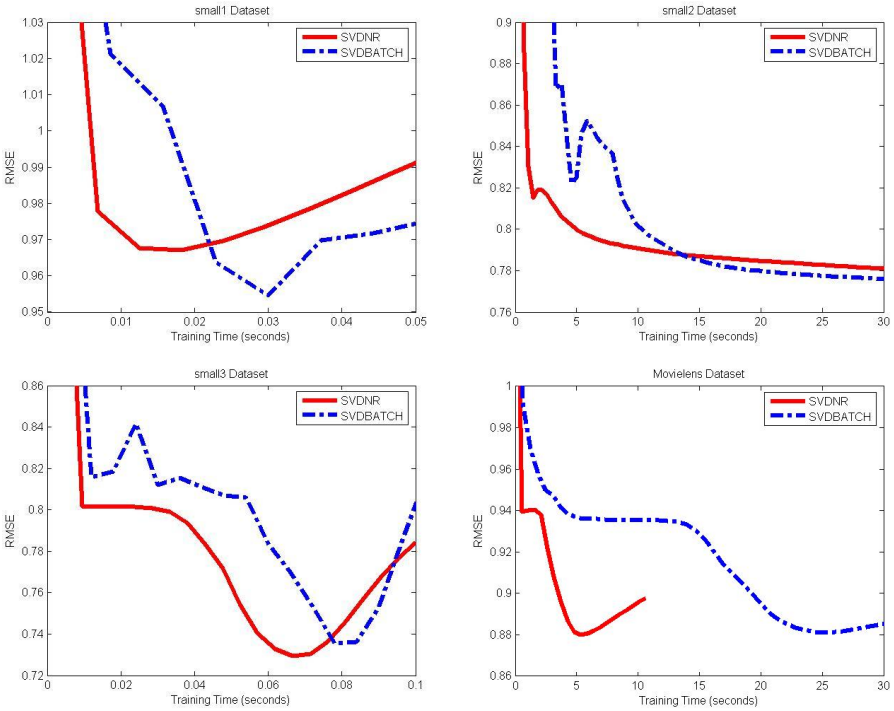


Figure 1: Plots of RMSE on batch and incremental learning

Table 2: Best RMSE achieved by each algorithm

| Dataset | **AVGB** | **SVDNR** | **SVD** | **CSVD** |
|---|---|---|---|---|
| *Movielens* | 0.9313 | 0.8796 | 0.8713 | 0.8706 |
| *Netflix* | 0.9879 | 0.9280 | 0.9229 | 0.9178 |

for *Movielens* and both 0.005 for *Netflix*. The algorithm **CSVD** has more parameters than other algorithms, but its learning rates $\mu_v$ and $\mu_w$ are the same as $\mu_v$ in **SVDNR** and **SVD**, and its first two regularization coefficients $k_y, k_m$ are also equal to $k_u, k_m$ in **SVD**. The other regularization coefficient $k_w$ for the constraint matrix $W$ is 0.02 for *Movielens* and 0.001 for *Netflix*. For the biases in **CSVD**, the learning rate and regularization coefficient $(\mu_b, k_b)$ are $(0.0005, 0.05)$ for *Movielens* and $(0.0002, 0.005)$ for *Netflix*. All these SVD-like algorithms have the dimension $f = 10$.

Table 2 shows the best RMSEs achieved by algorithms **AVGB**, **SVDNR**, **SVD**, and **CSVD**. That is, the lowest test RMSEs given by those algorithms before overfitting. We show the RMSEs versus training time in Figure 2, including algorithm **SVDUSER** with different learning rates. **SVDNR** suffers from overfitting quickly as it does not have a regularization term, while **SVD** gives a better performance with the same learning rate. **CSVD** is the best algorithm for both data sets, but it does not outperform **SVD** too much in *Movielens*

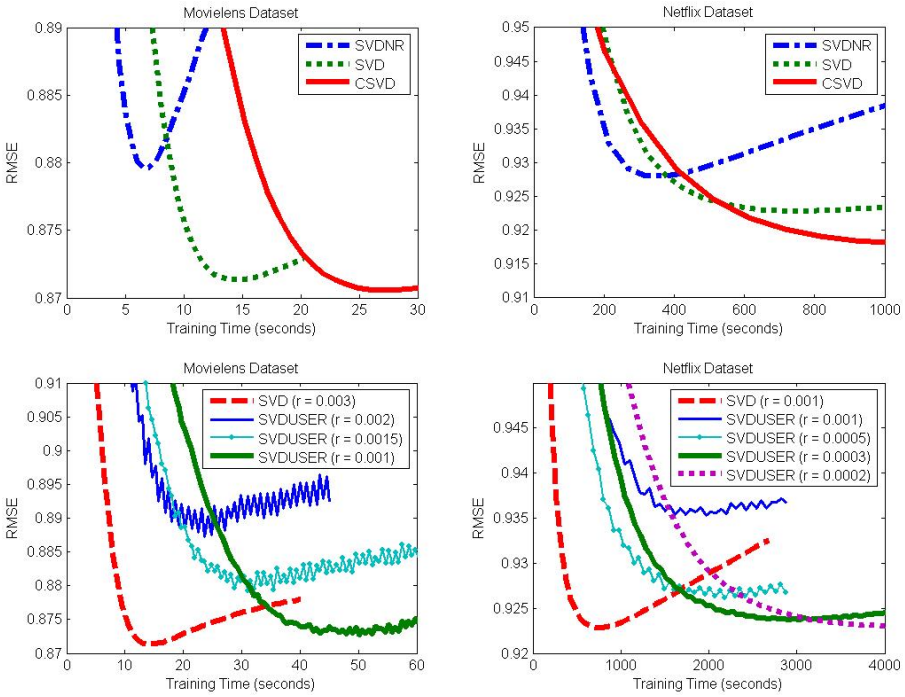


Figure 2: Plots of RMSE on the SVD-like algorithms

as its assumption on user features are not effective in a denser data set. The behavior of **SVDUSER** shows that incomplete incremental learning is a bad choice. It can still lead to similar performances if the same gradients are used, but a smaller learning rate and more training time are needed. If the learning rate is not small enough, the decrease of RMSE is even unstable, so it is hard to decide the stopping condition of an algorithm. Batch learning also has the same problem as incremental learning does. In our experiments, batch learning requires an extremely small learning rate to reach the same performance as complete incremental learning, and it even takes several days to complete the optimization for *Netflix*. In conclusion, complete incremental learning is better in the optimization of SVD algorithms for collaborative filtering, and the compound SVD algorithm can further boost the performances of conventional SVD algorithms for a sparse data set like *Netflix*.

At last, we show the effect of increasing the dimension of SVD. We use the algorithm **SVD** with dimension $f = 20, 30, 50$ and 100. The regularization coefficients $k_u$ and $k_m$ are both 0.015, which is suitable for each number of dimension in the experiments. The results are shown in Figure 3. The increase of dimension does raise the performance, but the training time is also proportional to the number of dimension.

In the Netflix Prize contest, the dimensions of SVD-like algorithms are usually set to a large value for lower RMSE. On the other hand, the training data of *Netflix* in our experiments contain only $99,072,112$ scores as we need a validation set, but it is better to run the algorithm with the full training data with $100,480,507$ scores, using the best parameters found by the training/validation

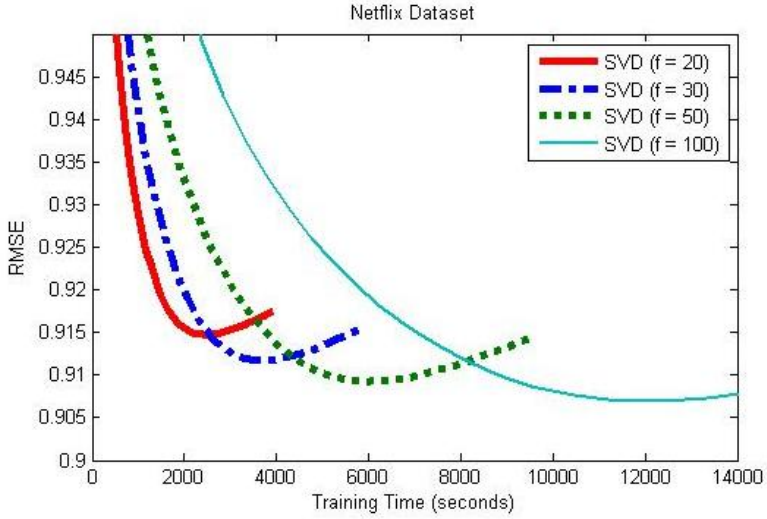


Figure 3: Plots of RMSE on the SVD algorithms with different dimensions

split. This approach decreases the test RMSE by about $0.005 - 0.01$. For the Netflix Prize competition, we use the algorithm **CSVD** with dimension $f = 256$, followed by a post processing method described in [Ma, 2008]. The RMSE of our best result is 0.8868.

# 4 Conclusions

A careful implementation of Singular Value Decomposition is effective for collaborative filtering. Our experiments show that, in general, batch learning or incomplete incremental learning requires a smaller learning rate, and has an unstable performance than complete incremental learning. We find that incremental learning, especially completely incremental learning which updates values after looking at a single training score, is the best choice for collaborative filtering with millions of training instances.

# References

R. Bell, Y. Koren, and C. Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 95–104, New York, NY, USA, 2007a. ACM. ISBN 978-1-59593-609-7. doi: http://doi.acm.org/10.1145/1281192.1281206.

R. Bell, Y. Koren, and C. Volinsky. The BellKor solution to the Netflix Prize. 2007b. URL `http://www.research.att.com/~volinsky/netflix/ProgressPrize2007BellKorSolution.pdf`.

J. Bennett and S. Lanning. The Netflix Prize. *Proceedings of KDD Cup and Workshop*, 2007.

J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):77–87, 1997. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/245108.245126.

C.-C. Ma. Large-scale collaborative filtering algorithms. Master's thesis, National Taiwan University, 2008. URL `http://www.csie.ntu.edu.tw/~r95007/thesis/svdnetflix/thesis.pdf`.

B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl. Movielens unplugged: experiences with an occasionally connected recommender system. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces*, pages 263–266, New York, NY, USA, 2003. ACM. ISBN 1-58113-586-6. doi: http://doi.acm.org/10.1145/604045.604094.

A. Paterek. Improving regularized Singular Value Decomposition for collaborative filtering. *Proceedings of KDD Cup and Workshop*, 2007.

R. Salakhutdinov and A. Mnih. Probabilistic Matrix Factorization. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1257–1264. MIT Press, Cambridge, MA, 2008.

R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann Machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: http://doi.acm.org/10.1145/1273496.1273596.

S. Zhang, W. Wang, J. Ford, F. Makedon, and J. Pearlman. Using Singular Value Decomposition approximation for collaborative filtering. *Seventh IEEE International Conference on E-Commerce Technology, 2005. CEC 2005.*, pages 257–264, July 2005. ISSN 1530-1354. doi: 10.1109/ICECT.2005.102.