CHAPTER I

Introduction

Collaborative filtering is a technique of predicting the preferences and interests of users from the collections of taste information given by large amount of users. It has the basic assumption that people who agreed in the past will also agree in the future. In application, collaborative filtering can be used for recommendation system that automatically gives suggestions to users about the products he or she should like.

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 will not score all of the objects in the database, but only those which the user has touched. Furthermore, some users may score most of the objects, but there are also some users who only score a few.

Let $V \in \mathbb{R}^{n \times m}$ be a matrix which represents the collected scores in the database. In most of the cases, $V$ is sparse and unbalanced. The existing scores in $V$ work as the training data of collaborative filtering algorithms, and the destination is to predict the missing scores in the database. Let $A \in \mathbb{R}^{n \times m}$ be another sparse matrix that is all or part of the missing votes. An algorithm of collaborative filtering will aim to predict the values in $A$. The performance of collaborative filtering can be measures
by the error between the prediction values and the ground-truth, or by the difference between the predicted and true ranking of objects about the preference order for each user, regardless of the actual score values.

1.2 Related Work
CHAPTER II

Singular Value Decomposition

Singular Value Decomposition, abbreviated to SVD, is one of the factorization algorithms for collaborative filtering. This type of algorithm finds the factors, or features of users and objects, and makes the predictions based on these factors. Some of the algorithms have restrictions on the feature values or between the features of multiple users and/or objects, while Singular Value Decomposition does not have any prior 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 \( I \) is its indicator that \( I_{ij} \) is equal to 1 if object \( j \) is scored by user \( i \) and 0 if the vote is missing. Usually the data is not only sparse but also unbalanced, i.e., some users scored a large amount of objects but others scored less. The SVD algorithm finds two matrices \( U \in \mathbb{R}^{f \times n} \) and \( M \in \mathbb{R}^{f \times m} \) as the features matrix of users and objects, and \( f \) is 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 \( i \)-th user and \( j \)-th object respectively. Once \( U \) and \( M \) are found, the missing scores in \( V \) can be predicted by the prediction function.

The optimization of \( U, M \) is performed by minimizing the sum of square error
between the existing scores and their prediction values, which gives the following objective function with regularization terms:

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

where \(k_u\) and \(k_m\) are the regularization coefficients which add the Frobenius norm of \(U\) and \(M\) as penalty. The negative gradients on matrix columns \(U_i\) and \(M_j\) are:

\[
- \frac{\partial E}{\partial U_i} = \sum_{j=1}^{m} I_{ij} \left( (V_{ij} - p(U_i, M_j)) \frac{\partial p(U_i, M_j)}{\partial U_i} \right) - k_u U_i,
\]

\[
- \frac{\partial E}{\partial M_j} = \sum_{i=1}^{n} I_{ij} \left( (V_{ij} - p(U_i, M_j)) \frac{\partial p(U_i, M_j)}{\partial M_j} \right) - k_m M_j.
\]

The most common prediction function is the dot product of feature vectors, i.e., \(p(U_i, M_j) = U_i^T M_j\). But in most of the applications, the values of scores in \(V\) are bounded by an interval \([a, b]\). One way to restrict the predicted values within the interval is to scale the scores to \([0, 1]\) and use a sigmoid function \(g\) in prediction:

\[
p(U_i, M_j) = g(U_i^T M_j),
\]

where

\[
g(x) = \frac{1}{1 + e^{-x}}
\]

and its derivative is

\[
\frac{dg(x)}{dx} = g(x)(1 - g(x)).
\]

Another method is to clip the values of dot products:

\[
p(U_i, M_j) = \begin{cases} 
  a & \text{if } U_i^T M_j < 0 \\
  a + U_i^T M_j & \text{if } 0 \leq U_i^T M_j \leq b - a \\
  b & \text{if } U_i^T M_j > b - a 
\end{cases}
\]

(2.7)
This prediction function is easier to implement and usually leads to better performance than using the sigmoid function, but it is non-differentiable when $U_i^T M_j = a$ or $b$. Hence, the prediction function is not clipped during training. i.e., it is simply

$$p(U_i, M_j) = a + U_i^T M_j$$  \hspace{1cm} (2.8)

with the gradients

$$\frac{\partial p(U_i, M_j)}{\partial U_i} = M_j, \quad \frac{\partial p(U_i, M_j)}{\partial M_j} = U_i.$$  \hspace{1cm} (2.9, 2.10)

But only clip the predicted values in validation and testing, since it will always increase the performance. Another way is still to clip the values in training but using 2.9 as gradients regardless of the value of $U_i^T M_j$. Because in most of the cases, the prediction function is in the differentiable range if it is approximated to the ground-truth score in $V$. But this method does not have significant difference in performance compared with the previous one.

When using the prediction function in (2.8), the objective function and its negative gradients have the following forms:

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

$$(2.11)$$

$$-\frac{\partial E}{\partial U_i} = \sum_{j=1}^{m} I_{ij} ((V_{ij} - p(U_i, M_j)) M_j) - k_u U_i,$$

$$-\frac{\partial E}{\partial M_j} = \sum_{i=1}^{n} I_{ij} ((V_{ij} - p(U_i, M_j)) U_i) - k_m M_j.$$  \hspace{1cm} (2.12, 2.13)

The optimization of $U$ and $M$ can be performed by gradient descent.

In the fashion of incremental learning, only one user is considered at a time. The variables related to a given user $i$ 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 $i$.

The objective function about user $i$ becomes:

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

(2.14)

with the negative gradients

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

(2.15)

$$- \frac{\partial E_i}{\partial M_j} = I_{ij} ((V_{ij} - p(U_i, M_j)) U_i) - k_m I_{ij} M_j$$

$$= I_{ij} [(V_{ij} - p(U_i, M_j)) U_i] - k_m M_j].$$

(2.16)

Note that if $I_{ij} = 0$, then the gradients on $M_j$ will be 0. Hence an object which is not scored by user $i$ will not be updated. This learning approach has different meaning from batch learning, as the sum of $E_i$ over all users does not lead to the objective function $E$ given in (2.11):

$$\sum_{i=1}^{n} E_i = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} I_{ij} (V_{ij} - p(U_i, M_j))^2$$

$$+ \frac{k_u}{2} \sum_{i=1}^{n} \|U_i\|^2 + \frac{k_m}{2} \sum_{j=1}^{m} \left( \sum_{i=1}^{n} I_{ij} \right) \|M_j\|^2.$$

(2.17)

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 $j$. The object with more scores will have larger regularization coefficient in this approach.

An extreme case of incremental learning is to update the features after looking each single score, i.e. to consider the following objective function and gradients if $I_{ij} = 1$:

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

(2.18)

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

(2.19)

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

(2.20)
The overall objective function becomes the summation of $E_{ij}$ over all existing votes:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} E_{ij} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} I_{ij} (V_{ij} - p(U_i, M_j))^2 + \frac{k_u}{2} \sum_{i=1}^{n} \left( \sum_{j=1}^{m} I_{ij} \right) \|U_i\|^2 + \frac{k_m}{2} \sum_{j=1}^{m} \left( \sum_{i=1}^{n} I_{ij} \right) \|M_j\|^2. \tag{2.21}$$

Compared with (2.11) and (2.17), we can find that if we consider the partial error function of each user instead of the whole one, then the regularization term of an object in the summation of objective function has an additional coefficient, which is proportional to the number of existing scores for that object, and vice versa.

The performances of algorithms are measured by Root Mean Square Error (RMSE). Consider the prediction matrix of test set $P \in \mathbb{R}^{n \times m}$ with indicator $J$, and the ground-truth answer matrix $A \in \mathbb{R}^{n \times m}$. The RMSE between the prediction $P$ and answer $A$ is defined as

$$rmse(P, A) = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} J_{ij} (A_{ij} - P_{ij})^2} \sum_{i=1}^{n} \sum_{j=1}^{m} J_{ij}, \tag{2.22}$$

and the training RMSE is proportional to the objective function except the regularization terms:

$$rmse(P_{\text{train}}, V) = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} I_{ij} (V_{ij} - p(U_i, M_j))^2} \sum_{i=1}^{n} \sum_{j=1}^{m} I_{ij}. \tag{2.23}$$

During the update of feature values, the test RMSE will decrease at first but start to increase at some time while the training RMSE continue decreasing. This overfitting phenomenon is the stopping condition of optimization and the main problem of SVD algorithm. By using better regularization coefficients, the algorithm can slow down the overfitting and reach lower test RMSE.

### 2.2 Optimization

The values of feature matrices $U$ and $M$ are optimized by the training data $V$, using gradient descent of the objective function. Usually the objection function as well
as training RMSE is easy to be decreased, but the actual performance of algorithm is measures by the test RMSE which may suffer from overfitting. In order to decide when to stop the learning procedure, A validation set of data is often used to prevent overfitting. The validation data can be part of training data, but it should have similar distribution to the test data.

The following two algorithms describe the optimization of \( U \) and \( M \) in the SVD model. Algorithm 1 performs batch learning, and Algorithm 2 does incremental learning which updates the feature values at each single score.

**Algorithm 1 (Batch learning of Singular Value Decomposition)**

Select values of learning rate \( \mu \), and regularization coefficients \( k_u,k_m \).

1. Set the starting values of matrices \( U,M \).

2. Repeat

   (a) Compute the matrix gradients \( \nabla_U \) and \( \nabla_M \) by (2.12) and (2.13).

   (b) Set \( U \leftarrow U - \mu \nabla_U, M \leftarrow M - \mu \nabla_M \).

until validation RMSE starts to increase.

**Algorithm 2 (Incremental learning of Singular Value Decomposition)**

Select values of learning rate \( \mu \), and regularization coefficients \( k_u,k_m \).

1. Set the starting values of matrices \( U,M \).

2. Repeat

   (a) Foreach existing score \( V_{ij} \) in training data

      i. Compute the feature gradients \( \nabla_{U_i} \) and \( \nabla_{M_j} \) by (2.19) and (2.20).

      ii. Set \( U_i \leftarrow U_i - \mu \nabla_{U_i}, M_j \leftarrow M_j - \mu \nabla_{M_j} \).
until validation RMSE starts to increase.

In Algorithm 2, only the feature vector of one user and one object are updated at a time instead of the full matrices. It causes different optimization results and performances in the optimization problem. Both algorithms have similar performances if $k_u$ and $k_m$ are set to 0, but Algorithm 1 has to use an extremely small learning rate to prevent divergence and hence has unaffordable learning time. When regularization is involved, Algorithm 2 has significant improvement in performance while Algorithm 1 does not no matter what values of $k_u$ and $k_m$ are used. Recall that the objective function (2.11) of batch learning is not equal to the summation of objective functions (2.21) of incremental learning. If Algorithm 1 is modified that (2.21) is used as its objective function when computing the matrix gradients $\nabla U$ and $\nabla M$, then it can still have similar performance to Algorithm 2 under the same regularization coefficients. However, an smaller learning rate and longer learning time are still required. Although the learning speed can be increased up by adding a momentum term on gradient descent, it is still not acceptable compared with incremental learning.

Batch learning is considered not suitable for SVD algorithms on sparse and unbalanced data, since the gradients over all existing data are also unbalanced and a single value of learning rate or regularization coefficient can not work well on all of them. On the other hand, when regularization is considered, (2.21) is a better choice of objective function than (2.11). Since if we neglect the time requirement, Both batch learning and incremental learning are able to reach good performance by using (2.21).

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 directly, but a too large value may lead to divergence. In general, smaller learning rate gives better performance, but the learning time is also longer. The regu-
larization coefficients prevent the feature value from being extreme, which is the main cause of overfitting. Small regularization coefficients may not have significant effect, but large ones will slow down the learning speed and even give inferior performance.

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 it cannot be too randomized otherwise the performance will be too unstable. For previous algorithms, the starting point can be set according to the average of all existing scores $\bar{V}$:

$$U_{ij}, M_{ij} = \sqrt{\frac{\bar{V} - a}{f}} + n(r) \text{ for each } i, j.$$  \hspace{1cm} (2.24)

where $a$ is the lower bound of score, $f$ is the dimension of SVD algorithm, and $n(r)$ is a random noise with the range $[-r, r]$. By the prediction function (2.8), all of the prediction values will approximate to the average $\bar{V}$ under this starting point. The random noise is required in the setting, otherwise the features of an user or object will have the same value because they always have the same gradients during optimization. A small value of $r$ is enough to make the algorithm work without bringing too much randomness. This kind of starting point is considered to be the best for Algorithm 2 as it can give the lowest test RMSE compared with other settings. For example, the starting point can also be set by the average score of each user and object. This setting has lower training RMSE at the beginning of optimization than using the overall average, but it overfits faster and leads to higher test RMSE.

Besides Algorithm 1 and 2, there are another methods for the optimization of $U$ and $M$. Rather than considering the whole feature matrix, the optimization can be performed on one row of $U$ and $M$ at a time, denoted by $u_k$ and $m_k$ which mean the $k$-th feature values of every user and object. The features are updated in the order like the following algorithm. For incremental learning, each single vote $V_{ij}$ will effect the $k$-th feature value of user $i$ and object $j$. i.e., $U_{ki}$ and $M_{kj}$.
Algorithm 3 (Ordered incremental learning of Singular Value Decomposition)

Select values of learning rate $\mu$, and regularization coefficients $k_u, k_m$.

1. Set the starting values of matrices $U, M$.

2. For $k = 1$ to $f$

   (a) Repeat

      i. Foreach existing score $V_{ij}$ in training data

         A. Compute the feature gradients $\nabla U_{ki}$ and $\nabla M_{kj}$ by (2.19) and (2.20).

         B. Set $U_{ki} \leftarrow U_{ki} - \mu \nabla U_{ki}$, $M_{kj} \leftarrow M_{kj} - \mu \nabla M_{kj}$,

   until no significance decrease of the objective function.

Algorithm 3 is similar to Algorithm 2, but there are several things different. First, the stopping condition of learning for each feature is not from the validation RMSE, but decided by the decrement of objective function. It is because that sometimes the first several features should be updated more even if the validation RMSE starts to increase, because it is possible to go down later in this kind of optimization while it seldom does in Algorithm 2. Usually an additional parameter $\text{miniter}$ is used to restrict that the learning should process at least $\text{miniter}$ iterations for each feature. Hence, the validation data is not needed during learning but the parameter $\text{miniter}$ should be tuned. The other thing is that the starting point of $U$ and $M$ will not be set like 2.24 because it is not suitable for Algorithm 3. Instead, a simple small value like 0.1 for each $U_{ij}, M_{ij}$ is enough to lead good performance. In addition, the random noise in 2.24 is not required when the features are updated in the order.

Algorithm 3 can be speeded up by caching if the prediction function has the form of dot product like 2.8. However, it is still slower than simple incremental learning like Algorithm 2 and also has inferior performance. In summary, Algorithm 2 is the best
for optimization, in both efficiency and performance. In order to exploit the validation
data and boost the performance more, one can record the number of iterations in
Algorithm 2 when using the validation data. After that, the training and validation
data are merged as the final training data and Algorithm 2 is run again with that
number of iterations.

2.3 Variants

The basic SVD algorithm with proper optimization is able to give good perfor-
mance. However, some variants of the algorithm have the potential for making more
accurate predictions. The simplest one is to add per-user and per-object bias on the
prediction function. i.e., the prediction function is modified to

\[ p(U_i, M_j) = a + U_i^T M_j + \alpha_i + \beta_j, \]

(2.25)

where \( \alpha_i \) is the bias of \( i \)-th user and \( \beta_j \) is the bias of \( j \)-th object. Like the previous
setting, the predicted values are only clipped in validation and testing.

The biases can be updated like the feature values. For incremental learning, a single
score \( V_{ij} \) gives the following gradients of biases \( \alpha_i \) and \( \beta_j \):

\[
E_{ij} = \frac{1}{2}(V_{ij} - p(U_i, M_j))^2 + \frac{k_u}{2}||U_i||^2 + \frac{k_m}{2}||M_j||^2 + \frac{k_b}{2}(\alpha_i^2 + \beta_j^2),
\]

(2.26)

\[
-\frac{\partial E_{ij}}{\partial \alpha_i} = (V_{ij} - p(U_i, M_j)) - k_b\alpha_i,
\]

(2.27)

\[
-\frac{\partial E_{ij}}{\partial \beta_j} = (V_{ij} - p(U_i, M_j)) - k_b\beta_j,
\]

(2.28)

where \( k_b \) is the regularization coefficient of biases that has similar effect to \( k_m \) and \( k_u \).

Of course, the learning rate of biases can be different from the one for feature values.

Another variant of SVD algorithm, called constrained SVD, adds additional con-
straints to each user feature vector \( U_i \). In this algorithm, the user feature matrix
\(U \in \mathbb{R}^{f \times n}\) is replaced by a matrix \(Y \in \mathbb{R}^{f \times n}\) with the same size, and an object constraint matrix \(W \in \mathbb{R}^{f \times m}\) which shifts the values of user features:

\[
U_i = Y_i + \sum_{k=1}^{m} I_{ik} W_k,
\]

(2.29)

where \(W_k\) is the \(k\)-th column of matrix \(W\). In other words, the feature vector \(U_i\) is the user-dependent feature vector \(Y_i\) plus an offset \(\sum_{k=1}^{m} I_{ik} W_k\) which is the mean of object constraint feature vectors on the objects scored by user \(i\). Each \(W_k\) represent the effect of scoring object \(k\) on user features.

Since batch learning is not suitable for this kind of algorithm, we consider the error function of each user for constrained SVD:

\[
E_i = \frac{1}{2} \sum_{j=1}^{m} I_{ij} (V_{ij} - p(U_i, M_j))^2 + \frac{k_y}{2} \|Y_i\|^2 + \frac{k_m}{2} \sum_{j=1}^{m} I_{ij} \|M_j\|^2 + \frac{k_w}{2} \sum_{k=1}^{m} I_{ik} \|W_k\|^2,
\]

(2.30)

\[
-\frac{\partial E_i}{\partial Y_i} = \sum_{j=1}^{m} I_{ij} ((V_{ij} - p(U_i, M_j)) M_j) - k_y Y_i,
\]

(2.31)

\[
-\frac{\partial E_i}{\partial M_j} = I_{ij} ((V_{ij} - p(U_i, M_j)) U_i) - k_m I_{ij} M_j
\]

\[
= I_{ij} [((V_{ij} - p(U_i, M_j)) U_i) - k_m M_j],
\]

(2.32)

\[
-\frac{\partial E_i}{\partial W_k} = \sum_{j=1}^{m} I_{ij} \left((V_{ij} - p(U_i, M_j)) \frac{I_{ik} M_j}{\sum_{k=1}^{m} I_{ik}}\right) - k_w I_{ik} W_k
\]

\[
= I_{ik} \left[\sum_{j=1}^{m} I_{ij} \left((V_{ij} - p(U_i, M_j)) \frac{M_j}{\sum_{k=1}^{m} I_{ik}}\right) - k_w W_k\right].
\]

(2.33)

Note that \(U_i\) follows the relation given in (2.29), and \(p(U_i, M_j)\) can be the prediction function described as (2.8) or (2.25). A single score \(V_{ij}\) has contributions to the gradients on the user feature \(Y_i\), the object feature \(M_j\), and each object constraint feature \(W_k\) with \(I_{ik} = 1\), i.e., each \(W_k\) that object \(k\) is scored by user \(i\). The coefficients \(I_{ij}\) and \(I_{ik}\) in (2.32) and (2.33) show that if an object is not scored by user \(i\), then the
corresponding feature vectors in $M$ and $W$ will not be updated during the learning procedure about user $i$.

For original SVD algorithm, the extreme case of incremental learning which update the feature values at each score like Algorithm $2$ can give the best performance with the fastest speed. But in constrained SVD, pure incremental learning has a problem in updating $W$: because each single score $V_{ij}$ affects all $W_k$ that user $i$ scores object $k$, the update for all scores of user $i$ will cause enormous computation on $W$, and it cannot be simplified since the gradient of $W$ will depend on the current values of $W$ when regularization is used. In order to solve this problem, a compound algorithm will be proposed in the next section. It can efficiently raise the performance of original SVD with the same time complexity.

### 2.4 Compound SVD

Consider the combination of constraint SVD and movie/customer biases, the error function of a single score $V_{ij}$ becomes:

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

$$+ \frac{k_w}{2} \sum_{k=1}^{m} I_{ik} ||W_k||^2 + \frac{k_b}{2}(\alpha_i^2 + \beta_j^2),$$

where $U_i$ is also computed by (2.29) and the prediction function $p(U_i, M_j)$ follows (2.25). The user feature vector $Y_i$ and object feature vector $M_j$ can be updated directly by the negative gradients of (2.34):

$$-\frac{\partial E_{ij}}{\partial Y_i} = ((V_{ij} - p(U_i, M_j))M_j) - k_y Y_i,$$

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

and the biases $\alpha_i$ and $\beta_j$ are updated by (2.27) and (2.28). In other words, the values in $Y$ and $M$ as well as all biases are updated as the extreme case of incremental learning.
like Algorithm 2.

On the other hand, the values in object constraint matrix $W$ could not be updated at each single score. However, it can be updated by another way under the condition that the scores in training data are ordered by the user ID. That is, in the scanning of scores for each iteration, all scores made by the first user are looked, and the corresponding variables are updated. Then it proceeds to the second user, and the third, and so on. During the update procedure of an user $i$, the values in $W$ are temporarily fixed while the other variable are updated as described above. By (2.29), the computationally expensive term $\frac{\sum_{k=1}^{m} I_{ik}W_k}{\sum_{k=1}^{m} I_{ik}}$ is also fixed and can be precomputed. Although the update of $Y_i$ will also change the value of $U_i$, its overhead is relatively negligible. After that, each $W_k$ with $I_{ik} = 1$ are updated as the final optimization step for user $i$.

The following algorithm, called Compound SVD, use the learning method described above to perform the optimization. The key point of this algorithm is that the gradient of (2.34) on $Y_i$ and $W_k$ are proportional if the regularization coefficients become 0:

$$-\frac{\partial E_{ij}}{\partial Y_i} = (V_{ij} - p(U_i, M_j))M_j,$$ \hspace{1cm} (2.37)

$$-\frac{\partial E_{ij}}{\partial W_k} = I_{ik} \left( (V_{ij} - p(U_i, M_j)) \frac{M_j}{\sum_{k=1}^{m} I_{ik}} \right)$$
$$= \frac{I_{ik}}{\sum_{k=1}^{m} I_{ik}} (V_{ij} - p(U_i, M_j))M_j,$$ \hspace{1cm} (2.38)

and the summation of (2.38) over all $j$ that $I_{ij} = 1$ gives the gradient on $W_k$ for user $i$ as (2.33) except the regularization term:

$$\sum_{j=1}^{m} I_{ij} \left( \frac{I_{ik}}{\sum_{k=1}^{m} I_{ik}} (V_{ij} - p(U_i, M_j))M_j \right)$$
$$= I_{ik} \left[ \sum_{j=1}^{m} I_{ij} \left( (V_{ij} - p(U_i, M_j)) \frac{M_j}{\sum_{k=1}^{m} I_{ik}} \right) \right].$$ \hspace{1cm} (2.39)

Hence, the common term $(V_{ij} - p(U_i, M_j))M_j$ can be recorded when computing the gradients on $Y_i$ for each scores $V_{ij}$, and used to compute the gradients of each $W_k$ later.
Moreover, the gradients of $W_k$ without regularization term are all the same for those $k$ that $I_{ik} = 1$, since $\sum_{k=1}^{m} I_{ik}$ is the number of objects scored by user $i$. However, the regularization coefficient $k_w$ is still needed to prevent the overfitting, but it can be added easily when the values in $W$ are actually updated.

**Algorithm 4 (Compound Singular Value Decomposition)**

Select values of learning rates $\mu_v, \mu_w, \mu_b$, and regularization coefficients $k_y, k_m, k_w, k_b$.

1. Set the starting values of matrices $Y, M, W$.

2. Repeat

   (a) For user $i = 1$ to $n$

      i. Compute the user feature vector $U_i$ from $Y_i$ and $W$ by (2.29).

      ii. Initialize the gradient buffer $g^{f \times 1}$ to $0^{f \times 1}$

      iii. Foreach existing score $V_{ij}$ made by user $i$

         A. Compute the feature gradient $\nabla Y_i$ by (2.35),

         and also set $g \leftarrow g + (V_{ij} - p(U_i, M_j))M_j$.

         B. Compute the feature gradient $\nabla M_j$ by (2.36).

         C. Compute the bias gradients $\nabla \alpha_i$ and $\nabla \beta_j$ by (2.27) and (2.28).

         D. Set $Y_i \leftarrow Y_i - \mu_v \nabla Y_i$, $U_i \leftarrow U_i - \mu_v \nabla Y_i$, $M_j \leftarrow M_j - \mu_v \nabla M_j$.

         E. Set $\alpha_i \leftarrow \alpha_i - \mu_b \nabla \alpha_i$, $\beta_j \leftarrow \beta_j - \mu_b \nabla \beta_j$.

      iv. Foreach $k$ that object $k$ is scored by user $i$

         A. Compute the feature gradient $\nabla W_k$ from $g$.

         B. Set $W_k \leftarrow W_k - \mu_w \nabla W_k$.

   until validation RMSE starts to increase.
There are two special settings in Algorithm 4. First, the values of $U_i$ are precomputed but it will not temporarily fixed during optimization like $W$. When the feature values are updated, $U_i$ is shifted with the same movement as $Y_i$ to preserve the relation $2.29$. If the starting point and learning rate of $W$ are all set to 0, Algorithm 4 will be identical to Algorithm 2. Second, the gradients of $W_k$ are computed by $g$ since

$$\mathbf{g} = \sum_{j=1}^{m} I_{ij} ((V_{ij} - p(U_i, M_j))M_j)$$

(2.40)

at that time, and

$$\frac{\mathbf{g}}{\sum_{k=1}^{m} I_{ik}} - k_w W_k = \frac{\sum_{j=1}^{m} I_{ij} ((V_{ij} - p(U_i, M_j))M_j)}{\sum_{k=1}^{m} I_{ik}} - k_w W_k$$

$$= \sum_{j=1}^{m} I_{ij} \left( \frac{(V_{ij} - p(U_i, M_j)) M_j}{\sum_{k=1}^{m} I_{ik}} \right) - k_w W_k,$$

(2.41)

which is exactly $2.33$ for those $W_k$ with $I_{ik} = 1$. This property shows that Algorithm 4 updates $Y$, $M$, and the biases like Algorithm 2 but updates $W$ in a fashion of optimization that consider one user at a time as well as its error function. Since the gradients of $W$ can be computed efficiently by 2.41, Algorithm 4 is able to overcome the problem of time complexity caused by updating the object constraint matrix $W$.

Besides the original parameters in Algorithm 2, Algorithm 4 has some additional parameters since it involves more variables. The learning rates $\mu_v$ for matrix $Y$ and $M$, $\mu_w$ for matrix $W$, and $\mu_b$ for biases can all be different, but the same value of $\mu_v$ and $\mu_w$ also works well while $\mu_b$ should be smaller. The regularization coefficients can be determined by the validation data, and the starting point of $W$ is the easiest to be set: if $Y$ and $M$ are initialized by $2.24$ all values in $W$ can just be initialized to 0 and Algorithm 4 is able to give better performance than the best result of Algorithm 2 under appropriate values of parameters.