

Collaborative Filtering for the NetFlix Prize

Masahiro Ono and Yang Zhang

December 4, 2007

Contents

1	Introduction	1
2	Mixture Model	2
3	Linear Regression + EM-like alternate update	6
4	Performance Comparison	8
5	Appendix	8

1 Introduction

1.1 Project Overview

We implemented and evaluated two approaches to collaborative filtering: mixture model MLE and low-rank approximation. We analyze these approaches and evaluate them on a subset of the NetFlix Prize dataset. This report includes the theoretical basis of each approach and the empirical results of our experimental evaluation as well as complexity analysis and discussion of their feasibility for larger-scale problems.

1.2 Dataset

The NetFlix dataset is large: it contains 480,189 users, 17,770 movies, and 100,480,507 ratings. MATLAB's in-memory data management is not amenable to datasets of such scale, and working with such a large dataset incurs implementation challenges that are beyond the scope of our class project. Hence, we use a much smaller subset of the full NetFlix Prize dataset consisting of 51 movies and 892 users, chosen so that the resulting rating matrix is fully packed. We then divide the rating matrix into training and test data. Since we have the fully packed rating matrix, we can arbitrarily specify the *sparsity ratio*¹ of the training data to evaluate the dependency of the performance on the sparsity ratio.

In this report, we denote by \mathbf{R} the rating matrix whose rows correspond to users and columns correspond to movies, such that entry $r_{i,j} \in \{1, \dots, 5\}$ is the rating given by user i to movie j . We also denote by n_U and n_M the number of users and movies.

1.3 Performance Measure

The performance of each algorithm is measured by root-mean-square error,

$$\text{RMSE} = \left[\sum_{(i,j) \in I_T} (\hat{r}_{i,j} - r_{i,j})^2 \right]^{\frac{1}{2}} \quad (1)$$

where I_T is the set of indices of the ratings included in the test data and $\hat{r}_{i,j}$ is the estimated rating of i th user for j th movie.

¹ The *sparsity ratio* is the ratio of missing ratings to the size of the matrix. The original NetFlix Prize data has a sparsity ratio of 99%.

2 Mixture Model

We use a mixture model to cluster together ratings based on user types and movie types. There are various other mixture models for collaborative filtering; this model is the one presented in 6.867 lecture 15 and is known as the *flexible mixture model* [2]. A survey of various alternative models can be found in [1].

For this part of our project, we rigorously derived the E-M algorithm, implement the algorithm in MATLAB, and analyzed its results. In the following sections, we describe and justify the model, then derive the E-M algorithm and analyze its complexity. Next, we describe our implementation, and finally we evaluate its RMSE performance. Throughout these sections, we will identify key variables, but also provide complete definitions in the appendix for the reader's reference.

2.1 Overview

We say that *each user* has a distribution over user types (*e.g.*, mostly romantic, a bit of an intellectual), and each movie has a distribution over movie types (mostly horror, some action). Each pair of user type and movie type has a distribution over ratings (*e.g.*, romantics tend to give horrors low ratings). The likelihood of our model given the parameters is then

$$L(D|\theta) = \Pr(R_{1,1} = r_{1,1}, \dots, R_{n_U, n_M} = r_{n_U, n_M} | \theta) \quad (2)$$

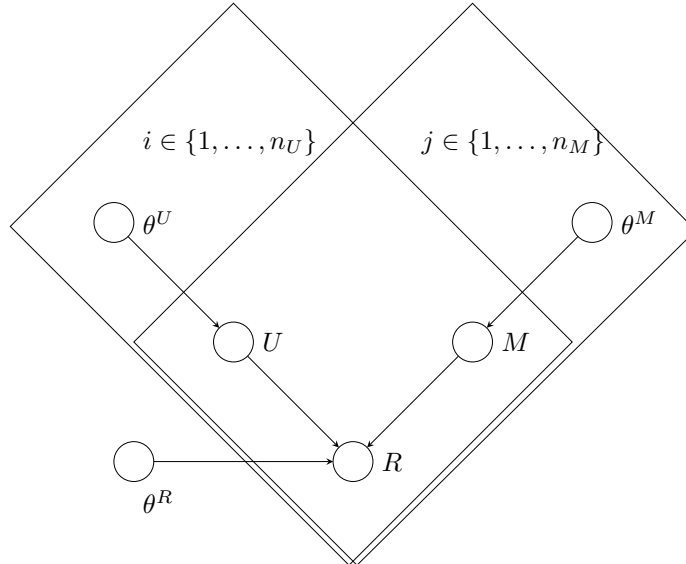
$$= \prod_{i,j} \sum_{u,m} [\Pr(R_{i,j} = r_{i,j} | U_{i,j} = u, M_{i,j} = m, \theta_{u,m}^R) \cdot \Pr(U_{i,j} = u | \theta_i^U) \cdot \Pr(M_{i,j} = m | \theta_j^M)] \quad (3)$$

$$= \prod_{i,j} \sum_{u,m} \theta_{u,m}^R(r) \cdot \theta_i^U(u) \cdot \theta_j^M(m) \quad (4)$$

We see that there are three sets of parameters, and our approach is to use an E-M algorithm to perform maximum likelihood estimation of these parameters.

Each cell has its own user type and movie type. What does all this mean? Intuitively, we can think of each cell as the result of an *act* of rating. Hence, every time a user rates a movie, we are randomly sampling from among the user's multiple personalities (*e.g.*, the romantic in the user was rating this movie). We are also sampling from among the movie's multiple aspects (the user saw this movie as a horror movie). We will loosely use the term "rating" to mean either the act of the rating or the rating value itself; hopefully the meaning remains clear from the context.

It is natural to use multinomial distributions for generating the user types for a user, and the movie types for a movie. We are also using multinomial distributions to generate ratings given a user type and movie type; this we do for simplicity. **TODO:** *have a better explanation?*



TODO: *how do i float this figure?* Figure (TODO) shows our graphical model.

TODO: *verify the following* Note that this model works for collaborative filtering for a static set of users and movies (filling in holes). New users/movies (augmenting our matrix with rows/columns that contain no data) would indeed be difficult to provide recommendations for regardless of model. However, even if these new rows/columns had

data available, our approach requires re-training the model, because each user and each movie has its own distribution over user types and movie types. **TODO:** *say something about how we can get a prior over these?*

2.2 E-M Algorithm

2.2.1 M-step with complete data

To derive the E-M algorithm, it is helpful to start by assuming that we have “complete data”—that is, for each of the (observed) ratings i, j , we know precisely which user type u and movie type m the rating belongs to. With this information, we can directly and analytically find the parameters that maximize the likelihood of the data. We can equivalently optimize the log-likelihood: $\arg \max_{\theta} L(D|\theta) = \arg \max_{\theta} \log L(D|\theta)$.

$$l(D|\theta) = \log L(D|\theta) \quad (5)$$

$$= \sum_{i,j \in I} \left[\log \Pr(R_{i,j} = r_{i,j} | U_{i,j} = u_{i,j}, M_{i,j} = m_{i,j}, \theta^R) + \log \Pr(U_{i,j} = u_{i,j} | \theta_i^U) + \log \Pr(M_{i,j} = m_{i,j} | \theta_j^M) \right] \quad (6)$$

$$= \sum_{i,j \in I} \left[\log \theta_{u_{i,j}, m_{i,j}}^R(r_{i,j}) + \log \theta_i^U(u_{i,j}) + \log \theta_j^M(m_{i,j}) \right] \quad (7)$$

$$= \sum_{r,u,m,i,j} n_{r,u,m,i,j} [\log \theta_{u,m}^R(r) + \log \theta_i^U(u) + \log \theta_j^M(m)] \quad (8)$$

Above, $n_{r,u,m,i,j}$ is 1 if rating i, j has rating value r , user type u , and movie type m , and 0 otherwise.

Implicit throughout the previous equations were a number of constraints due to the fact that we’re dealing with probabilities. In particular,

$$\sum_r \theta_{u,m}(r) = 1 \quad \sum_u \theta_i(u) = 1 \quad \sum_m \theta_j(m) = 1 \quad (9)$$

To allow ourselves to optimize the equations analytically, we can incorporate these into the equation by using LaGrange multipliers (λ s below):

$$\begin{aligned} & \sum_{r,u,m,i,j} n_{r,u,m,i,j} [\log \theta_{u,m}^R(r) + \log \theta_i^U(u) + \log \theta_j^M(m)] + \\ & \left\{ \lambda_{u,m}^R \left(1 - \sum_r \theta_{u,m}^R(r) \right) \forall u, m \right\} + \left\{ \lambda_i^U \left(1 - \sum_u \theta_i^U(u) \right) \forall i \right\} + \left\{ \lambda_j^M \left(1 - \sum_m \theta_j^M(m) \right) \forall j \right\} \end{aligned} \quad (10)$$

To find the maximizing value of a parameter, we find the zeros of the partial derivative with respect to that parameter. This turns out to be straightforward for the above expression, because the parameters are in separate terms, so that many terms in the sums and quantifiers “disappear.”

For all u, i ,

$$0 = \frac{\partial l(D|\theta)}{\partial \theta_i^U(u)} = \sum_{r,m,j} \frac{n_{r,u,m,i,j}}{\theta_i^U(u)} - \lambda_i^U \quad \Rightarrow \quad \theta_i^U(u) = \frac{\sum_{r,m,j} n_{r,u,m,i,j}}{\lambda_i^U} \quad (11)$$

$$1 = \sum_u \theta_i^U(u) = \sum_u \frac{\sum_{r,m,j} n_{r,u,m,i,j}}{\lambda_i^U} \quad \Rightarrow \quad \lambda_i^U = \sum_{r,u,m,j} n_{r,u,m,i,j} \quad (12)$$

$$\therefore \theta_i^U(u) = \frac{\sum_{r,m,j} n_{r,u,m,i,j}}{\sum_{r,u,m,j} n_{r,u,m,i,j}} = \frac{n_{u,i}}{n_M} \quad (13)$$

This makes intuitive sense, since $\theta_i^U(u) = \Pr(U_{i,j} = u | \theta_i^U)$ for any movie j . The probability that we see the romantic in a user is the number of her other ratings where it was the romantic in her speaking, over her total number of ratings.

Similarly (almost symmetrically), we find:

$$\forall m, j, \theta_j^M(m) = \frac{n_{m,j}}{n_U} \quad \forall r, u, m, \theta_j^M(m) = \frac{n_{r,u,m}}{n_{u,m}} \quad (14)$$

However, we do not have complete data—we do not have the above counts, since we do not know the user-/movie-type assignments. Since the incomplete data does not yield an analytical solution, we must use a numerical approach, namely an E-M algorithm. This algorithm iteratively updates the parameters in the M-step based on intermediately calculated posteriors in the E-step. Essentially, we use the posteriors to find the *expected* counts in the above equations.

2.2.2 E-step

For all $(i, j) \in I$, we calculate the posterior.

$$\Pr(U_{i,j} = u, M_{i,j} = m \mid R = \mathbf{R}, \theta) \quad (15)$$

$$= \Pr(U_{i,j} = u, M_{i,j} = m \mid R_{i,j} = r_{i,j}, \theta_i^U, \theta_j^M, \theta_{u,m}^R) \quad \text{independencies can be seen in graphical model} \quad (16)$$

$$= \frac{1}{Z} \left(\frac{\Pr(R_{i,j} = r_{i,j} \mid U_{i,j} = u, M_{i,j} = m, \theta_i^U, \theta_j^M, \theta_{u,m}^R)}{\Pr(U_{i,j} = u, M_{i,j} = m \mid \theta_i^U, \theta_j^M)} \right) \quad \text{by Bayes' rule; } Z \text{ is a normalizer} \quad (17)$$

$$= \frac{1}{Z} \left(\frac{\Pr(R_{i,j} = r_{i,j} \mid U_{i,j} = u, M_{i,j} = m, \theta_{u,m}^R)}{\Pr(U_{i,j} = u \mid \theta_i^U) \Pr(M_{i,j} = m \mid \theta_j^M)} \right) \quad \text{independencies in graphical model} \quad (18)$$

$$= \frac{1}{Z} \theta_{u,m}^R(r_{i,j}) \theta_i^U(u) \theta_j^M(m) \quad (19)$$

Above,

$$Z = \Pr(R_{i,j} = r_{i,j} \mid \theta) = \sum_{u,m} \Pr(U = u, M = m \mid R = r, \theta_i^U, \theta_j^M, \theta^R) \quad (20)$$

The above independencies can be verified from the moralized ancestral graphs of our model, but we do not have space to illustrate these graphs here.

For any i, j , letting $U = U_{i,j}$, $M = M_{i,j}$, $R = R_{i,j}$, we can marginalize the above joint distribution over M and U to get the distributions over U and M , respectively:

$$\Pr(U = u \mid R = r, \theta_i^U, \theta_j^M, \theta^R) = \sum_m \Pr(U = u, M = m \mid R = r, \theta_i^U, \theta_j^M, \theta^R) \quad (21)$$

$$\Pr(U = u \mid R = r, \theta_i^U, \theta_j^M, \theta^R) = \sum_m \Pr(U = u, M = m \mid R = r, \theta_i^U, \theta_j^M, \theta^R) \quad (22)$$

2.2.3 M-step

Now we can use these posterior values to find expected counts in order to update the parameters.

For all $i \in \{1, \dots, n_U\}$, for any $j \in \{1, \dots, n_M\}$,

$$\mathbb{E}[n_{u,i} \mid \mathbf{R}, \theta] = \mathbb{E} \left[\sum_{r,m,j} n_{r,u,m,i,j} \mid R = \mathbf{R}, \theta \right] \quad (23)$$

$$= \sum_{r,m,j} \mathbb{E}[n_{r,u,m,i,j} \mid R = \mathbf{R}, \theta] \quad (24)$$

$$= \sum_{r,m,j} [0 + 1 \cdot \Pr(R_{i,j} = r, U_{i,j} = u, M_{i,j} = m \mid R = \mathbf{R}, \theta)] \quad (25)$$

$$= \sum_j \Pr(U_{i,j} = u \mid R = \mathbf{R}, \theta) \quad \text{marginalized } M, \text{ observed } R \quad (26)$$

$$= \sum_j \Pr(U_{i,j} = u \mid R_{i,j} = r_{i,j}, \theta_{u,m}^R, \theta_i^U, \theta_j^M) \quad \text{independencies in graph} \quad (27)$$

$$\theta_i^{U'}(u) = \mathbb{E} \left[\frac{n_{u,i}}{n_M} \mid \theta \right] \quad \text{by Eq. 13} \quad (28)$$

$$= \frac{1}{n_M} \sum_j \Pr(U_{i,j} = u \mid R_{i,j} = r_{i,j}, \theta_{u,m}^R, \theta_i^U, \theta_j^M) \quad \text{weighted sum over the row} \quad (29)$$

Similarly, for all $j \in \{1, \dots, n_M\}$,

$$\theta_j^{M'}(m) = \frac{1}{n_U} \sum_i \Pr(M = m \mid R_{i,j} = r_{i,j}, \theta_{u,m}^R, \theta_i^U, \theta_j^M) \quad (30)$$

Finally, for all $u \in \{1, \dots, k_U\}, m \in \{1, \dots, k_M\}$,

$$\theta_{u,m}^R(r) = \mathbb{E} \left[\frac{n_{r,u,m}}{n_{u,m}} \mid \theta \right] = \frac{\mathbb{E} [n_{r,u,m} \mid \theta]}{\mathbb{E} [n_{u,m} \mid \theta]} \text{TODO: how to justify this?} \quad (31)$$

$$= \frac{\sum_{i,j:r_{i,j}=r} \Pr(U_{i,j} = u, M_{i,j} = m \mid R_{i,j} = r_{i,j}, \theta_{u,m}^R, \theta_i^U, \theta_j^M)}{\sum_{i,j} \Pr(U_{i,j} = u, M_{i,j} = m \mid R_{i,j} = r_{i,j}, \theta_{u,m}^R, \theta_i^U, \theta_j^M)} \quad (32)$$

$$(33)$$

2.3 Complexity Analysis

TODO: fix this; our algo actually does not build complete tables, only what is necessary given the data; $r_{i,j}$, not all r

Here is a break-down of various dimensions of our model and algorithm. Each parameter involves “weighted sums” over some number of components of the joint posterior over U, M given the data R ; the third column is the order of the number of terms in these summations (the cost of the summations). The fourth column is the total complexity, the product of the second and third columns. Note that in all of the following, we are providing a loose upper bound; the expense of this algorithm is more tightly bounded by the number of observed ratings in the matrix I , rather than the matrix’s dimensions n_U and n_M .

component	no. parameters	no. posterior components	complexity
θ_R	$(k_R - 1) \cdot k_U \cdot k_M$	$\mathcal{O}(n_U \cdot n_M)$	$\mathcal{O}(k_R \cdot k_U \cdot k_M \cdot n_U \cdot n_M)$
θ_U	$(k_U - 1) \cdot n_U$	$\mathcal{O}(k_M \cdot n_M)$	$\mathcal{O}(k_U \cdot n_U \cdot k_M \cdot n_M)$
θ_M	$(k_M - 1) \cdot n_M$	$\mathcal{O}(k_U \cdot n_U)$	$\mathcal{O}(k_M \cdot n_M \cdot k_U \cdot n_U)$

(A multinomial of k possible outcomes needs only $k - 1$ parameters.)

The posterior joint distribution over U, M given the data R is in Eq. 19. This table is of size $k_U \cdot k_M \cdot k_R \cdot n_U \cdot n_M$.

TODO: what to say here? The number of iterations is less straightforward to calculate.

2.4 Experimental Evaluation

We implemented the E-M algorithm in MATLAB. The MATLAB execution speed was poor when run on even our smaller dataset, but it allowed us to rapidly prototype a working implementation.

We partitioned the cells of \mathbf{R} into a training set matrix and a test set matrix using a sparsity ratio of **TODO: sparsity ratio**. The EM algorithm was repeated **TODO: iterations** times, and we took the result parameters that yielded the largest likelihood. The reason for this is that EM does not guarantee it will find the global maximum; where it ends up depends on how the parameters were initialized. The general way to deal with this is to randomly initialize the likelihood parameters, so we randomly initialize θ^R . Note that no two components therein must be equal, otherwise they would have the exact same likelihood of generating any of the points (i.e., $\forall u, m, r, r', \theta_{u,m}^R(r) \neq \theta_{u,m}^R(r')$). θ^U and θ^M are uniformly initialized; we are essentially starting with an “unbiased” prior (from a Bayesian perspective, we are doing the same thing as finding the maximum *a posteriori* parameters assuming a uniform prior).

In choosing the number of user types and movie types, an approach would be to try some different values and see how well they optimize some metric, settling with the value that best optimizes this metric. **TODO: really need to check the following!** However, if we simply let the metric be the likelihood, then we can simply blow up the number of parameters: have as many user types as there are users, and as many movie types as there and number of movie types will equal to the number of cell types.

One common and easy-to-use metric is the *Bayesian information criterion* (BIC), also known as the *Schwarz information criterion* (SIC). The formula for the BIC is

$$-2 \cdot \log L(R = \mathbf{R} \mid \hat{\theta}) + k \log n \quad (34)$$

where k is the dimensionality of the model space and $n = |I|$ is the number of rating observations. The model with the lower value of BIC is the one to be preferred: increasing the log-likelihood decreases the BIC, but increasing the number of parameters increases (penalizes) the BIC.

In our model, we have

$$l(R = \mathbf{R} | \theta) = \sum_{i,j \in I} \left[\log \Pr(R_{i,j} = r_{i,j} | U_{i,j} = u_{i,j}, M_{i,j} = m_{i,j}, \theta^R) + \right. \\ \left. \log \Pr(U_{i,j} = u_{i,j} | \theta_i^U) + \log \Pr(M_{i,j} = m_{i,j} | \theta_j^M) \right] \quad (35)$$

$$= \sum_{i,j \in I} \left[\log \theta_{u_{i,j}, m_{i,j}}^R(r_{i,j}) + \log \theta_i^U(u_{i,j}) + \log \theta_j^M(m_{i,j}) \right] \quad (36)$$

$$(37)$$

TODO: RESULTS

Finally, we computed the RMSE. These results will be discussed in section **TODO: reference the comparison section**

3 Linear Regression + EM-like alternate update

3.1 Overview

The goal of the algorithm is to find a low-rank approximation of the sparse rating matrix

$$R \approx MU \quad (38)$$

where M is a n_M -by- d movie property matrix, U is a d -by- n_U user property matrix, and d is the dimension of the low-rank approximation. Due to the missing entries of R , M and U cannot be obtained from singular value decomposition. Instead, the algorithm learns M and U using linear regression and an EM-like alternate update algorithm.

The time and space complexity of the algorithm is proportional to the data size $(n_U + n_M)$, so it is tractable even for the huge data set of Netflix Prize. However, it badly overfits to the training data especially when the rating matrix is sparse.

3.2 Learning Algorithm

At the beginning of the algorithm, M is randomly initialized and fixed. Then the i th column of U is learned by regular linear regression as follows;

$$\mathbf{u}_i = (\check{M}^T \check{M})^{-1} \check{M}^T \check{\mathbf{r}}_i \quad (39)$$

where $\check{\mathbf{r}}_i$ is a column vector constructed from the i th column of the rating matrix R , by excluding the missing elements from it. For example, if the i th column of R is $[2 \quad \circ \quad 4 \quad \circ \quad \circ \quad 5 \quad \circ]^T$ where “ \circ ” means missing element, then $\check{\mathbf{r}}_i = [2 \quad 4 \quad 5]^T$. \check{M} is a matrix with the corresponding rows of M .

Then in the next step, U is fixed and each row of M is updated in the exactly same manner. These steps are repeated and U and M are alternatively updated until the training RMSE converges, just as EM algorithm.

The important fact is that, by repeating this iteration, the training RMSE monotonically decreases, and eventually converges to the (local) optimum. Linear regression finds the parameter which minimizes the log likelihood given the following normal distribution with arbitrary fixed variance σ^2 ;

$$r_{i,j} \sim N(\vec{m}_i \cdot \mathbf{u}_j, \sigma^2) \quad (40)$$

where \vec{m}_i is the i th row of M and \mathbf{u}_j is j th column of U .

Thus, just as EM algorithm, the log likelihood of the training data monotonically decreases in each iteration. With $\sigma = 1/\sqrt{2}$, the RMSE is exactly the same as the log likelihood. Thus the training RMSE monotonically decreases by iterations.

3.3 Estimation

Given the trained matrices U and M , the maximum likelihood estimation of the ratings can be easily obtained as follows;

$$\hat{r}_{i,j} = \vec{m}_i \cdot \mathbf{u}_j. \quad (41)$$

Although the actual ratings are given as integer numbers from one to five, the estimations by this algorithm are real numbers. We use the real number outputs without rounding them. However, since ratings are from one to five, the estimations above five are turned into five, and those below one are turned into one.

3.4 Result

The algorithm is implemented with MATLAB and worked successfully.

Since the initial value of M is randomly chosen, the result is stochastic. In fact it appears that the result is very sensitive to the initial value of M . Figure 1 shows the typical result of the training RMSE and the test RMSE as well as the RMSE when the estimation is simply the average rating of each movie (i.e. zero-order estimation). The training RMSE monotonically decreases over iterations as we expected, but test RMSE does not necessarily monotonically decreases. There is a large gap between training RMSE and test RMSE, indicating that the algorithm badly overfits to the training data.

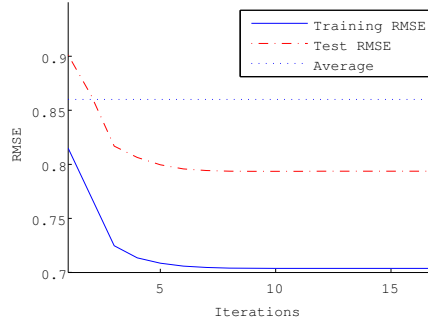


Figure 1: Typical result when $d = 2$ and (Ratio of missing element)=0.5. The dotted line shows the RMSE when the estimation is simply the average rating of each movie (i.e. zero-order estimation)

Table 1 compares the average RMSE of ten runs with different dimensions of the low-rank approximation d and different ratio of the missing element in the rating matrix. Looking at the table in column-wise, it is found that the algorithm works better for the raining data with less missing ratings, which is an obvious result. An interesting result can be found by looking at the table in row-wise; larger dimension results in the larger RMSE when the rating matrix is sparse, while it results in the smaller RMSE when the rating matrix is mostly filled. It is because the algorithm

badly overfits to the training data when the number of free parameters are large while the number of the available training data is small.

Table 1: RMSE with different dimensions d and different ratio of the missing element. The values are the average of ten runs.

		Ratio of missing element in R		
		0.7	0.5	0.2
d	1	0.808	0.795	0.789
	2	0.859	0.790	0.772
	3	0.933	0.808	0.772

3.5 Complexity Analysis

Time complexity The most time consuming part of the algorithm is the d by d matrix inversion $(\check{M}^T \check{M})^{-1}$ in Eq. 39. In practice, Gauss Elimination, which has a complexity of $O(n^3)$, is used to find the solution instead of complete matrix inversion. The inverse of matrix is computed $(n_U + n_M)$ times in each iteration. Thus the time complexity of the algorithm is

$$O(d^3 N(n_U + n_M)) \quad (42)$$

where N is the number of iterations.

Space complexity Only U and M are stored during computation. Thus the space complexity is

$$O(d(n_U + n_M)). \quad (43)$$

3.6 Estimated performance on the full Netflix Prize data

Computational cost It took about 15 seconds for 100 iteration with $n_U = 892$, $n_M = 51$, and $d = 3$ on Intel Celeron 2.0 GHz CPU. Thus for the full data set of Netflix Prize where $n_U = 480,189$ and $n_M = 17,770$, the computation for 100 iterations would take about 2 hours with $d = 3$. Required memory space would be about 11 MByte. Thus this is a tractable algorithm for the full Netflix Prize data.

RMSE In the actual Netflix prize data 99.9% elements of the rating matrix are missing. With the small data set where $n_U = 892$, $n_M = 51$, the algorithm cannot run with 99.9% sparsity since $(\check{M}^T \check{M})^{-1}$ in Eq. 39 become singular for most of the rows due to the lack of data. One thing we can say for sure from Table 1 is that RMSE would be worse than 0.808 for the full Netflix prize data. Trying the algorithm on full data is the future work.

4 Performance Comparison

4.1 RMSE Error

4.2 Comutation Time and Memory Use

4.3 Feasibility for the large-scale problem

5 Appendix

5.1 Mixture Models

Here are the variables we're using for the mixture models:

- θ : the entire set of parameters. TODO explain the sharing (but not here)
- $\theta_i^U(u)$: the probability distribution over user types u for a user i (think of this as a table). This is shared by all user-movie ratings for user i .

- $\theta_j^M(m)$: the probability distribution over movie types m for a movie j . This is shared by all user-movie ratings for movie j .
- $\theta_{u,m}^R(r)$: the probability distributions over ratings r for user type u and movie type m . This is shared by all user-movie ratings i, j .
- Dimensions and sets
 - n_U : number of users
 - n_M : number of movies
 - k_U : number of user types
 - k_M : number of movie types
 - I : the set of indices i, j of the ratings in the training set
 - I_T : the set of indices i, j of the ratings in the test set
- Indexes. This allows us to omit ranges in sums and quantifiers.
 - i : generally used to index over users
 - j : generally used to index over movies
 - u : generally used to index over user types
 - m : generally used to index over movie types
- Random variables
 - U_i : the user type of user i
 - M_j : the movie type of movie j
 - $R_{i,j}$: the rating user i gave for movie j

References

- [1] C. Z. Rong Jin, Luo Si. A study of mixture models for collaborative filtering. URL <http://www.cs.purdue.edu/homes/lsi/publications.htm>.
- [2] L. Si and R. Jin. A flexible mixture model for collaborative filtering, 2003. URL citeseer.ist.psu.edu/si03flexible.html.