EE 219
Large-Scale Data Mining: Models and Algorithms

Project 3
Collaborative Filtering
Winter 2017

Guanqun Mao, Jianing Liu
204777289 804759999

February 27th 2017

Part A

1. Introduction

In this project we implement a movie recommendation system based on user ratings by using collaborative filtering, methods of predicting a user's opinion on a n entity using other users' opinions. These methods are based on the notion that there exist other users with similar behaviors as the target user and finding them and observing their actions will reveal information that we could use to predict the behavior of the target user.

We first process the dataset and create a rating matrix. Then we implement different matrix factorization algorithms to retrieve two factor matrices and produce the prediction matrix. The prediction result is measured with a 10-fold cross validation and the trade-off curve between precision and recall values. Eventually, we evaluate the performance of our recommendation system by changing the number of movies we want to recommend.

2. Data Processing

Our data comes from the MovieLens dataset, which is collected by the University of Minnesota as part of the GroupLens Research Project. MovieLens has 100,000 ratings on a scale of 5 from 943 users on 1682 movies. We use the Import Data tool in MATLAB to transfer raw data into a 100,000 * 4 matrix. The four columns are userId, itemId, rating and timestamp, respectively. With the first three columns, we can get a 943 * 1682 matrix R. $R(i,j)$ represents the rating of user i and item j.

Part B

1. Weighted Non-Negative Matrix Factorization

There are only 100,000 ratings in the dataset, therefore there will be many missing ratings in the matrix R and the corresponding vacancies will be filled by NaN values. To predict these values, we use non-negative matrix factorization to get matrices U, V such that $R_{mxn} = U_{mxn}V_{kxn}$. It is necessary to calculate the least square error and minimize it.

This can be done by putting 0s where the data is missing and creating a weight matrix to calculate the squared error. We assume that the weight matrix W_{mxn} contains 1 in entries where there are known data points and 0 in entries where the data points are missing. We can formulate the above problem as:

$$min \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij}(r_{ij} - (UV)_{ij})^2$$

We use the *wnmfrule* function in the Matrix Factorization Toolbox in MATLAB to implement this factorization. We choose k equal to 10, 50 and 100 and the total least squared error is shown in Table 1. We find that under different iterations, there may be different performances. The total least squared error becomes smaller as k and iteration rise.

k	10	50	100
Iteration = 50	779009.6637	658208.4462	577943.3807
Iteration = 100	770111.7632	646603.1885	559361.5626
Iteration = 200	761850.525	632464.3254	540979.9606
Iteration = 500	759211.7307	621074.8462	526822.2296
Iteration = 1000	758681.5707	616553.679	521024.2361
Iteration = 2000	757257.3947	615400.5248	516149.0126

Table 1 The Least Squared Error with Different Ks and Factorization Iteration

2. Testing with 10-Fold Cross Validation

To test the recommendation system, we use a 10-fold cross validation. We divide the 100, 000 records into 10 folds exclusively. Each time, we use 9 folds as training data and the remaining 1 fold as testing data. This time, we calculate the average absolute error over testing data among all entries. We choose k to be 100 and set factorization to be 50, 100, 200, 500, 1000 and 2000 to get average absolute error of the testing data for each entry of all ten tests, the highest and lowest average absolute errors of testing data for each entry. The result is shown in Table 2.

	Average	Highest	Lowest
Iteration = 50	0.85233	0.86175	0.84658
Iteration = 100	0.91325	0.92713	0.89898
Iteration = 200	0.97922	1.0637	0.95163
Iteration = 500	1.1554	1.9479	1.0251
Iteration = 1000	363.4908	3144.9163	1.1196
Iteration = 2000	375.7967	2092.4011	1.2071

Table 2 Absolute Error of Testing Data Under Different Iteration

We can conclude that we should choose a suitable iteration to get the best absolute error. In order to demonstrate this phenomenon, we try to calculate the absolute error within low iterations. The result is shown in Table 3. It appears that when $k = 100$, we should not use too high iterations for matrix factorization.

	Average	Highest	Lowest
Iteration = 10	0.80119	0.8122	0.79183
Iteration = 20	0.80963	0.8172	0.80418
Iteration = 30	0.82505	0.83003	0.81658
Iteration = 40	0.84042	0.84949	0.83135
Iteration = 50	0.85381	0.86337	0.84895
Iteration = 60	0.86493	0.87947	0.8557

Table 3 Absolute Error of Testing Data Under Low Iteration

3. Precision Over Recall

Now we assume that if a user has rated a movie 3 or lower, then it can be concluded that they did not like the movie; and if a user has rated a movie 4 or higher, they have liked it. Based on the estimated values in the R matrix, we can use a threshold on the estimated entry to predict whether a user will like a certain movie or not.

We first find out the number of entries where our system predicted the user would like the movie. Then we calculate for what percentage the users actually liked the movie. Secondly we find the entries in the testing data where the user did actually like the movie and out of these entries for what percentage our system predicted the user would like the movie.

Out of all the predicted entries in which a user likes the movie, the percentage of the users that actually liked the movie is precision. Out of all the entries in which users actually liked the movie, the percentage entries we have predicted successfully is recall.

Both k and iterations have impact on the system prediction performance. In Figure 1, 2 and 3, we show this relation and corresponding precisions and recall values. Since we only have a limited number of data points, it would be better to use a small k and fewer iterations.

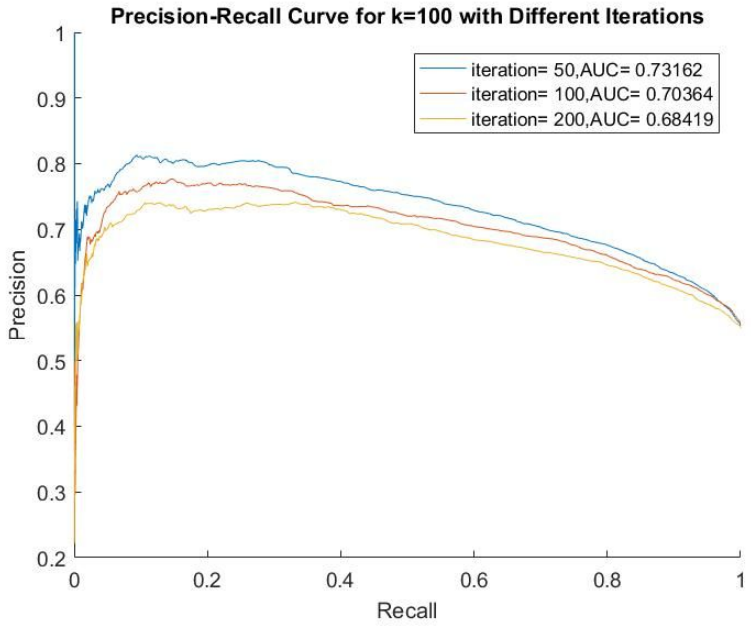

Figure 1 Precision-Recall Curve for k = 100 Under Different Iterations

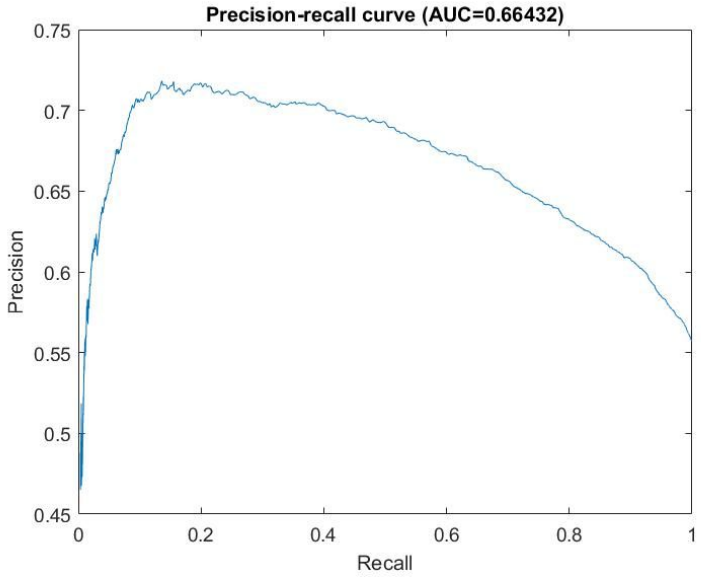

Figure 2 Precision vs. Recall

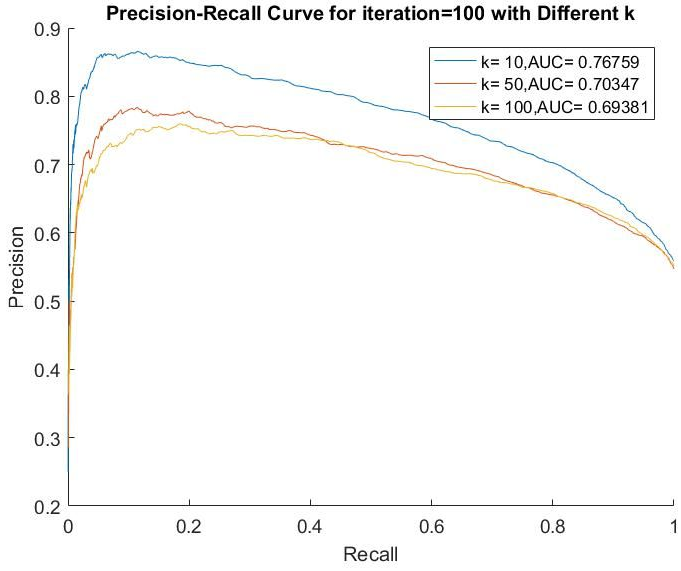

Figure 3 Precision-Recall Curve for Iteration = 100 Under Different ks

4. Weighted Non-Negative Matrix Factorization with Regularization

In the previous section, we make our recommendation system based on the weighted non-negative matrix factorization. In this part, we replace the rating matrix with the weighted matrix and vice versa. However, without any regularization parameter, the prediction matrix

would all be 1. Therefore, we need to add some regularization parameter to the cost function. This new cost function is:

$$min \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}(r_{ij} - (UV)_{ij})^2 + \lambda(\sum_{i=1}^{m}\sum_{j=1}^{k} u_{ij}^2 + \sum_{i=1}^{k}\sum_{j=1}^{n} v_{ij}^2)$$

Reason Behind Regularization

In the first part of weighted non-negative matrix factorization with regularization, we consider the problem in which we use the rating matrix as the weight and set R to a 0-1 matrix with no regularization applied. Formation of this problem is:

$$min \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}(r_{ij} - (UV)_{ij})^2$$

The algorithm to solve it is almost identical to the one used in the previous problem. The only difference is that here we exchange the placement of R and W. Thus, we are actually constructing a 0-1 matrix. We calculate the total squared error to evaluate the performance of this algorithm under 3 different k values: 10, 50 and 100. The total squared error is much smaller than before. This is mainly because the reconstructed matrix is different. Another difference is that when k gets larger the total squared error also becomes larger, which is exactly the opposite of the previous result. The optimal mechanism is to have a matrix with all 1 entries. Actually when k is small, there are fewer entries in each matrix and the constraints between them are rather loose. Thus, it is easier to get an result close to the optimal one, which comes along with a better total squared error. However, this does not mean the prediction is better, since what we want to achieve is a matrix whose element value is positively correlated to the rating, which cannot be done by introducing the regularization terms.

Regularized Version of Alternating Least Squares

In the alternating least squares algorithm, we need to construct a binary matrix P such that when $R > 0$, $P = 1$ and when $R = 0$, $P = 1$.

Then we want to factorize P into X and Y such that $P \approx XY^T$. The recommendations are the largest values in XY^T. Since optimizing X and Y simultaneously is non-convex, alternating least squares algorithm is used. This is because if X or Y is fixed, it will only be a system of linear equations, which is convex and easy to solve. The solving process is:

- Initialize Y with random values
- Solve for X
- Fix X, solve for Y
- Repeat the above process until it converges

Let's define the regularization weights $c_{ui} = r_{ui}$, where u stands for the user and i stands for the movie. We also define C_u as the diagonal matrix of c_u. Then the update equation for X is:

$$x_u = (Y^T C_u Y + \lambda I)^{-1} Y^T C_u p_u$$

We use the same evaluation methods to test the results of the regularized ALS. The Precision vs. Recall curves with k = 10, 50 and 100 are shown in Figure 4, 5 and 6 respectively.

Iteration = 200:

k	10	50	100
MSQE	52.7604185551	260.623635512	245.687335281

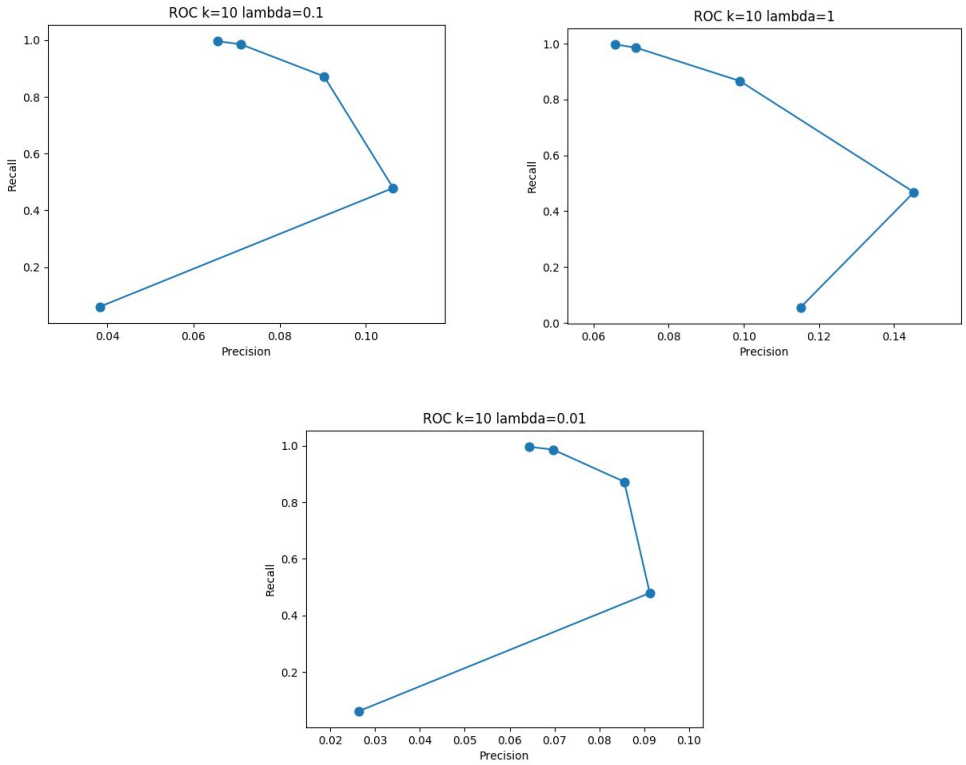

Figure 4 Precision-Recall Curve with k = 10

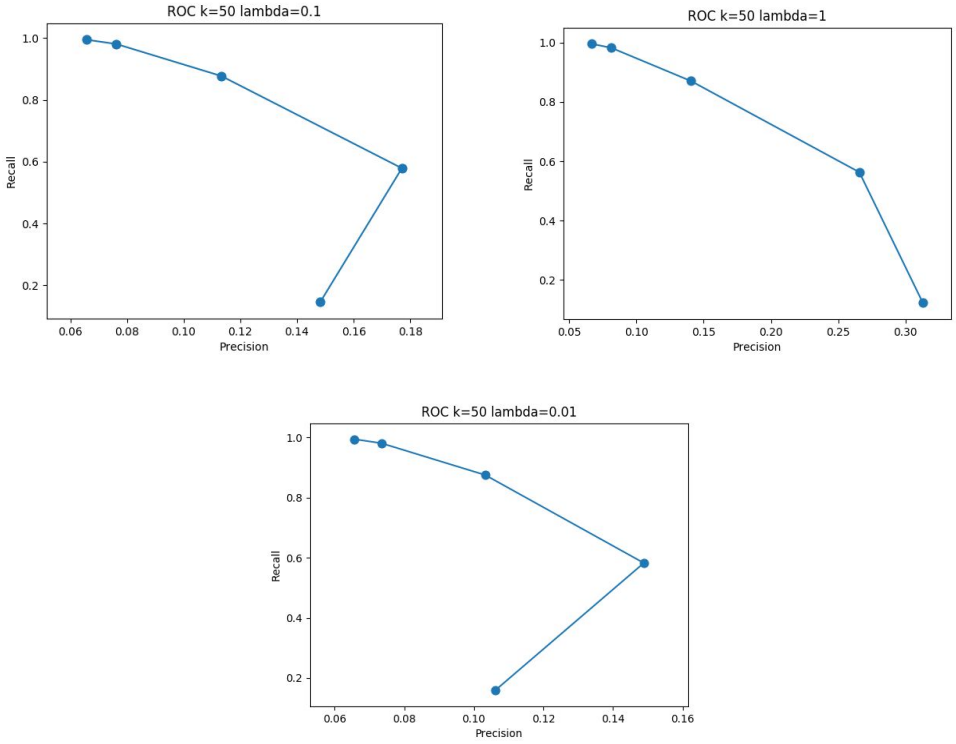

Figure 5 Precision-Recall Curve with k = 50

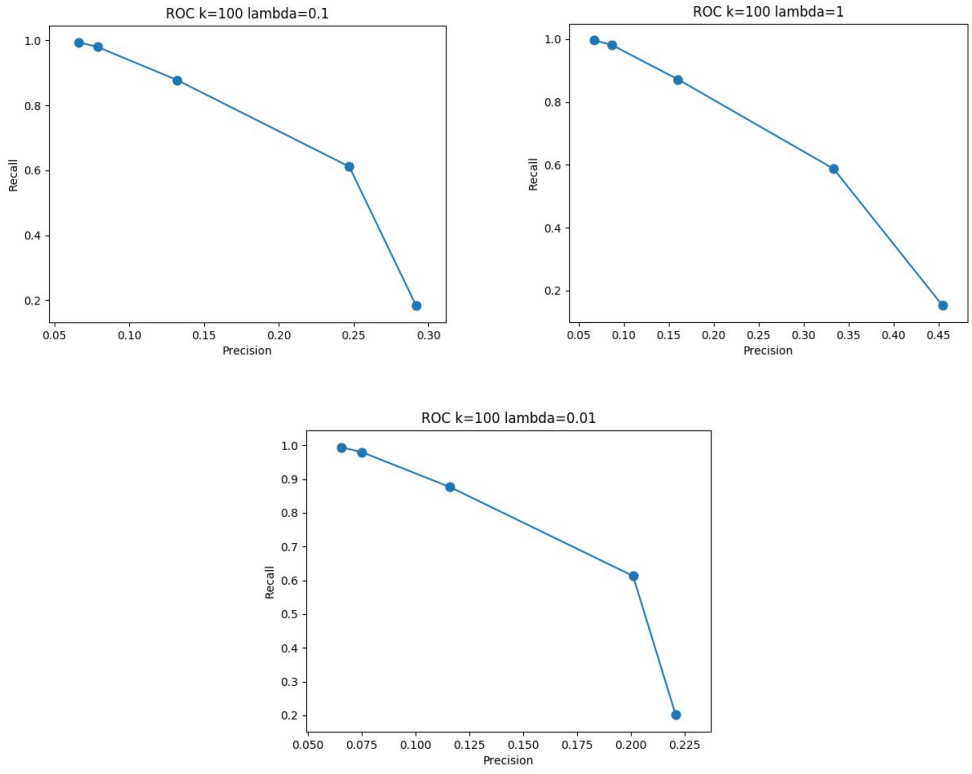

Figure 6 Precision-Recall Curve with k = 100

k	lambda	MSQE
10	0.01	60350.827379
10	0.1	60258.0025798
10	1	60755.4408029
50	0.01	30808.3073779
50	0.1	31264.625736
50	1	32241.2211674
100	0.01	17932.4625627
100	0.1	18650.98405
100	1	20389.2599445

In each figure three curves which correspond to λ=0.01,0.1 and 1 are plotted. We can see the tradeoff between precision and recall. In each graph, the precision reaches its maximum when recall is around 0 and then it gradually decreases while recall approaches 1.

5. Recommendation System

The precision of the recommendation system depends on the prediction matrix P and how many movies we want to recommend. When the top five movies are chosen, the precision in the 10-fold cross validation is shown in Figure 7 and the average precision is 84.08%.

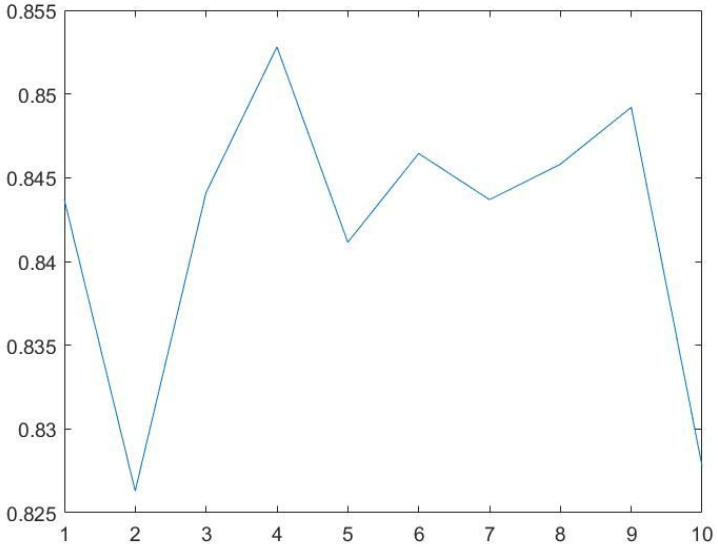

Figure 7 The Precision Over 10-Fold Cross Validation

The hit rate and false alarm rate varies with different numbers of recommendations. The result is shown in Figure 8.

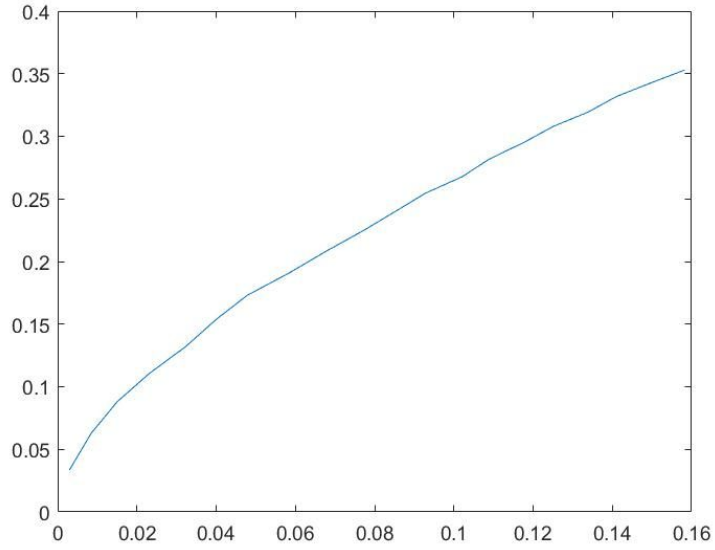

Figure 8 Hit Rate vs. False Alarm Rate

From Figure 8, we can see that the hit rate and false alarm rate are highly correlated. As we lower the threshold, we expect to see a higher fraction of titles liked by the user chosen by the

recommendation system. Similarly, as the threshold decreases, we allow for more movies that were disliked by the user to be counted as recommendations by the system. Therefore, the false alarm rate increases as we decrease the threshold, because this would decrease the criteria for an item to be classified as “liked.”

Appendix

iteration: 50
k = 10, total least squared error: 779009.6637
k = 50, total least squared error: 658208.4462
k = 100, total least squared error: 577943.3807
iteration: 100
k = 10, total least squared error: 770111.7632
k = 50, total least squared error: 646603.1885
k = 100, total least squared error: 559361.5626
iteration: 200
k = 10, total least squared error: 761850.525
k = 50, total least squared error: 632464.3254
k = 100, total least squared error: 540979.9606
iteration: 500
k = 10, total least squared error: 759211.7307
k = 50, total least squared error: 621074.8462
k = 100, total least squared error: 526822.2296
iteration: 1000
k = 10, total least squared error: 758681.5707
k = 50, total least squared error: 616553.679
k = 100, total least squared error: 521024.2361
iteration: 2000
k = 10, total least squared error: 757257.3947
k = 50, total least squared error: 615400.5248
k = 100, total least squared error: 516149.0126

iteration: 10
average absolute error: 0.80119
highest average absolute error: 0.8122
lowest average absolute error: 0.79183
iteration: 20
average absolute error: 0.80963

highest average absolute error: 0.8172
lowest average absolute error: 0.80418
iteration: 30
average absolute error: 0.82505
highest average absolute error: 0.83003
lowest average absolute error: 0.81658
iteration: 40
average absolute error: 0.84042
highest average absolute error: 0.84949
lowest average absolute error: 0.83135
iteration: 50
average absolute error: 0.85381
highest average absolute error: 0.86337
lowest average absolute error: 0.84895
iteration: 60
average absolute error: 0.86493
highest average absolute error: 0.87947
lowest average absolute error: 0.8557

iteration: 50
average absolute error: 0.85233
highest average absolute error: 0.86175
lowest average absolute error: 0.84658
iteration: 100
average absolute error: 0.91325
highest average absolute error: 0.92713
lowest average absolute error: 0.89898
iteration: 200
average absolute error: 0.97922
highest average absolute error: 1.0637
lowest average absolute error: 0.95163
iteration: 500
average absolute error: 1.1554
highest average absolute error: 1.9479
lowest average absolute error: 1.0251
iteration: 1000
average absolute error: 363.4908
highest average absolute error: 3144.9163

lowest average absolute error: 1.1196
iteration: 2000
average absolute error: 375.7967
highest average absolute error: 2092.4011
lowest average absolute error: 1.2071

Mutiple update rules based NMF successes!
 # of iterations is 1000.
 The final residual is 7.1133e+02.
The average precision for test 1 is 0.828420
Mutiple update rules based NMF successes!
 # of iterations is 1000.
 The final residual is 7.0911e+02.
The average precision for test 2 is 0.842630
Mutiple update rules based NMF successes!
 # of iterations is 1000.
 The final residual is 7.1147e+02.
The average precision for test 3 is 0.843054
Mutiple update rules based NMF successes!
 # of iterations is 1000.
 The final residual is 7.0898e+02.
The average precision for test 4 is 0.844539
Mutiple update rules based NMF successes!
 # of iterations is 1000.
 The final residual is 7.1073e+02.
The average precision for test 5 is 0.840933
Mutiple update rules based NMF successes!
 # of iterations is 1000.
 The final residual is 7.1016e+02.
The average precision for test 6 is 0.848568
Mutiple update rules based NMF successes!
 # of iterations is 1000.
 The final residual is 7.0871e+02.
The average precision for test 7 is 0.851962
Mutiple update rules based NMF successes!
 # of iterations is 1000.
 The final residual is 7.1032e+02.
The average precision for test 8 is 0.837752

Mutiple update rules based NMF successes!
 # of iterations is 1000.
 The final residual is 7.1005e+02.
The average precision for test 9 is 0.825027
Mutiple update rules based NMF successes!
 # of iterations is 1000.
 The final residual is 7.0988e+02.
The average precision for test 10 is 0.844751
The total average precision is 0.840764

Starting parallel pool (parpool) using the 'local' profile ... connected to 2 workers.
calculate hit and false alarm rate for top 14 movies...
calculate hit and false alarm rate for top 13 movies...
calculate hit and false alarm rate for top 7 movies...
calculate hit and false alarm rate for top 6 movies...
calculate hit and false alarm rate for top 12 movies...
calculate hit and false alarm rate for top 11 movies...
calculate hit and false alarm rate for top 5 movies...
calculate hit and false alarm rate for top 4 movies...
calculate hit and false alarm rate for top 10 movies...
calculate hit and false alarm rate for top 9 movies...
calculate hit and false alarm rate for top 3 movies...
calculate hit and false alarm rate for top 2 movies...
calculate hit and false alarm rate for top 8 movies...
calculate hit and false alarm rate for top 1 movies...
calculate hit and false alarm rate for top 17 movies...
calculate hit and false alarm rate for top 16 movies...
calculate hit and false alarm rate for top 15 movies...
calculate hit and false alarm rate for top 19 movies...
calculate hit and false alarm rate for top 18 movies...
calculate hit and false alarm rate for top 20 movies...

k = 10, iteration = 200
Matrix Factorization MSQE: 55.3789761985
k = 50, iteration = 200
Matrix Factorization MSQE: 262.829071059
k = 100, iteration = 200
Matrix Factorization MSQE: 246.65084427

k = 10, lambda = 0.01
Matrix Factorization MSQE: 60350.827379
k = 10, lambda = 0.1
Matrix Factorization MSQE: 60258.0025798
k = 10, lambda = 1
Matrix Factorization MSQE: 60755.4408029
k = 50, lambda = 0.01
Matrix Factorization MSQE: 30808.3073779
k = 50, lambda = 0.1
Matrix Factorization MSQE: 31264.625736
k = 50, lambda = 1
Matrix Factorization MSQE: 32241.2211674
k = 100, lambda = 0.01
Matrix Factorization MSQE: 17932.4625627
k = 100, lambda = 0.1
Matrix Factorization MSQE: 18650.98405
k = 100, lambda = 1
Matrix Factorization MSQE: 20389.2599445