Movie Recommendation Using Latent Factor Models

Mustafa Onur Eken
Bogazici University - CmpE 462

May 16, 2016

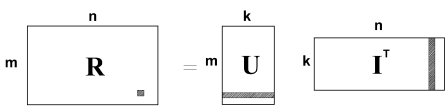
Abstract

Recommender systems are nearly obligatory in every platform where a type of content is served to users as long as conversion rates are concerned in perspective of bussiness owners. In like manner customers also benefit from since they discover interesting content at a faster rate. There have been different proposals to tackle problem of recommendation. In this project, we will implement a recent technique namely, latent factor models, go through the results we obtained and discuss strengths and weaknesses of it.

1 Introduction

Content-based filtering and collaborative filtering are two main approaches used in practice nowadays to overcome the problem of item recommendation to users. In spite of content-based filtering, collaborative filtering does not make use of any domain-specific features about items and users, instead it attempts to find similarities between users and items only by examining the given ratings.

Latent factor models assume that there are a number of characteristics of items and users. If characteristics of a user and an item are aligned then a user tend to like that item. Building on this assumption, we can discover the characteristics of users and items through factorizing the rating matrix, R, into two other matrices U_{mxk} and I_{nxk} such that $R = UI^T$.



Where k is the number of latent factors.

Since matrix R has missing entries computing SVD with common methods is not possible. There has been several proposals to compute this decomposition [1, 2]. In this project we are going to implement computation of this factorization via 1) Stochastic gradient descent 2) Alternating least squares. In the following section we will briefly tell some important points in implementation of these algorithms. On the third section we will discuss the results we have obtained. Afterwards we will review strengths, weaknesses of matrix factorization methods and by point our some possible improvements. You can find the full source code in the appendices section.

2 Approach

2.1 Data

In this project we use the well known *MovieLens 100k* dataset where 943 people, 1682 movies and 100,000 ratings exist. Size of training data is 80,000 and the size of test data is 20,000. The rating density corresponds to 5.04% when training data is considered. Although dataset contains item and

user specific information such as occupation of a user, genre of a movie etc. we are not going to exploit them as they are irrelevant to basic version of latent factor models.

2.2 SGD

One way to find matrices U and I is by applying stochastic gradient descent. We initialize the matrices U and I randomly. We compute error based on one instance at a time. Lets denote ith row of matrix U as p_i and adjoint of jth row of matrix I as q_j. In addition denote the difference between desired output and prediction as $d_{ij} = r_{ij} - \hat{r}_{ij}$

2.2.1 Plain

In this model, prediction, error and update are calculated as follows:

$$\hat{r}_{ij} = p_i q_j$$

$$d_{ij} = r_{ij} - p_i q_j$$

$$e_{ij} = (d_{ij})^2 = (r_{ij} - p_i q_j)^2$$

$$\frac{\partial e_{ij}}{\partial p_i} = -2 d_{ij} q_j$$

$$\frac{\partial e_{ij}}{\partial q_j} = -2 d_{ij} p_i$$

$$p'_i = p_i + \eta d_{ij} q_j$$

$$q'_j = q_j + \eta d_{ij} p_i$$

2.2.2 Plain + Regularization

In this model, prediction, error and update are calculated as follows:

$$\hat{r}_{ij} = p_i q_j$$

$$d_{ij} = r_{ij} - p_i q_j$$

$$e_{ij} = (d_{ij})^2 + \lambda(\|p_i\|^2 + \|q_j\|^2)$$

$$\frac{\partial e_{ij}}{\partial p_i} = -2 d_{ij} q_j + 2\lambda p_i$$

$$\frac{\partial e_{ij}}{\partial q_j} = -2 d_{ij} p_i + 2\lambda q_j$$

$$p'_i = p_i + \eta(d_{ij} q_j - \lambda p_i)$$

$$q'_j = q_j + \eta(d_{ij} p_i - \lambda q_j)$$

2.2.3 Plain + Biases

In this model, we take user and movie specific biases into account to enhance the accuracy furthermore [2]. It's well known that some users tend to rate high on average while some other users rate. In like manner some movies are rated higher than others. In this sense bias of a movie represent how much users liked it. Here, μ is the mean of all ratings, u_i is the bias of ith user and m_j is the bias of jth movie. Prediction, error and update are calculated as follows:

$$\hat{r}_{ij} = \mu + u_i + m_j + p_i q_j$$

$$d_{ij} = r_{ij} - \mu - + u_i - m_j - p_i q_j$$

$$e_{ij} = (r_{ij} - \mu - + u_i - m_j - p_i q_j)^2$$

$$\frac{\partial e_{ij}}{\partial p_i} = -2 d_{ij} q_j$$

$$\frac{\partial e_{ij}}{\partial q_j} = -2 d_{ij} p_i$$

$$\frac{\partial e_{ij}}{\partial u_i} = -2 d_{ij}$$

$$\frac{\partial e_{ij}}{\partial m_j} = -2 d_{ij}$$

$$p'_i = p_i + \eta d_{ij} q_j$$

$$q'_j = q_j + \eta d_{ij} p_i$$

$$u'_i = u_i + \eta d_{ij}$$

$$m'_j = m_j + \eta d_{ij}$$

2.2.4 Plain + Regularization + Biases

If we combine all, we get the following definitions for prediction, error and update:

$$\hat{r}_{ij} = \mu + u_i + m_j + p_i q_j$$

$$d_{ij} = r_{ij} - \mu - + u_i - m_j - p_i q_j$$

$$e_{ij} = (d_{ij})^2 + \lambda(\|p_i\|^2 + \|q_j\|^2 + u_i^2 + m_j^2)$$

$$\frac{\partial e_{ij}}{\partial p_i} = -2 d_{ij} q_j + 2\lambda p_i$$

$$\frac{\partial e_{ij}}{\partial q_j} = -2 d_{ij} p_i + 2\lambda q_j$$

$$\frac{\partial e_{ij}}{\partial u_i} = -2d_{ij} + 2\lambda u_i$$

$$\frac{\partial e_{ij}}{\partial m_j} = -2d_{ij} + 2\lambda m_j$$

$$p_i' = p_i + \eta(d_{ij}q_j - \lambda p_i)$$

$$q_j' = q_j + \eta(d_{ij}p_i - \lambda q_j)$$

$$u_i' = u_i + \eta(d_{ij} - \lambda u_i)$$

$$m_j' = m_j + \eta(d_{ij} - \lambda m_j)$$

2.3 ALS

The key idea in alternating least squares somewhat similar to SGD. Instead of adjusting matrices U and I instances by instance, we fix one of them and solve least squares for the other one.

$$UI^T = R$$

Fix U and solve for I^T. Then fix I^T then solve for U. Repeat this until convergence. If one desires to use MATLAB's $\backslash$ operator for solving least squares, a re-formulation of matrices U and I^T is needed. An example of a toy set up would be as following:

$$\begin{bmatrix} x_{11} & ? \\ x_{21} & x_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$\begin{bmatrix} x_{11} \\ x_{21} \\ x_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & 0 & a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} b_{11} \\ b_{21} \\ b_{12} \\ b_{22} \end{bmatrix}$$

Here we find b values by solving $x = A_m b$. (I.e. $A_m \backslash x$ in MATLAB) The same formulation must be done on the second matrix as well. ALS execution time is unfavorable [2] compared to SGD unless

- Parallelization is possible.
- Data is not sparse.

3 Results & Discussion

We have implemented both algorithms (biases and regularization terms are not included in ALS) and executed on 1)*Handcrafted toy data* 2)*MovieLens 100k data.* Find output plots and interpretation of them in the following pages.

Although both ALS and SGD worked just fine on toy data, ALS became infeasible when MovieLens data is given as input. Therefore, plots associated with MovieLens data are generated as a result of SGD algorithm only.

For illustration purposes we first run the algorithms on "full" toy data (i.e. no missing entries) assuming two latent factors. Since the rating matrix is full we also executed SVD algorithm to validate results of other algorithms, SGD and ALS. When we plot the values of latent factors for movies and users we see nice clusters in accordance with the generation of the toy data. This vaguely confirms that the model can indeed find hidden patterns in the data. We also trained the model using one latent factor to see if clusters are distinguishable. Unfortunately there were no apparent separate clusters. It indicates that one factor is not enough to explain/approximate the given data.

We used a small, fixed learning rate in SGD instead of an adaptive learning rate because the dataset was not gigantic and execution times were reasonable in nearly all settings of parameters.

4 Conclusion

As seen on the plots and tables, latent factor models are rather accurate in finding recommendations. When data density is high (see tests on toy data) the results are nearly perfect. When it is sparse (see tests on MovieLens data) we can say that it is satisfactory. It is neat that when data can be explained with a few latent factors we can visualize it and expose clusters.

This project can be improved by attack-

ing the problem of finding optimal values for parameters such as number of latent factors, regularization coefficient in order to minimize error. Furthermore, it could have been nice to implement ALS to work on a parallel platform and see the training results on the real data.

References

[1] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. Citeseer, 2011.

[2] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems, 2009.

Tests on Toy Data

		Serious	Funny	Funny	Funny	Serious	Funny	Funny	Funny	Serious	Serious	Funny	Serious	Serious	Serious	Serious	Funny	Funny	Serious	Funny	Serious
		Jon	Robb	Arya	Olly	Daenerys	Eddard	Sansa	Ramsay	Melisandre	Sandor	Joffrey	Tyrion	Petyr	Bran	Margaery	Tommen	Gilly	Jaime	Cersei	Varys
Serious	The Shawshank Redemption	5	3	2	1	5	3	1	1	5	5	1	3	5	3	4	1	1	4	1	5
Serious	The Green Mile	5	2	1	1	4	3	1	5	5	5	1	5	4	4	4	3	1	3	4	5
Serious	Forrest Gump	4	2	4	2	5	3	3	1	4	5	1	4	3	3	5	2	2	5	2	3
Serious	Inception	5	3	3	3	5	3	2	3	4	4	1	5	5	3	4	3	1	4	1	5
Serious	Léon	5	2	3	3	5	3	3	2	4	5	2	3	4	3	5	1	1	5	2	4
Serious	Knockin' on Heaven's Door	4	3	2	2	4	2	1	3	5	5	1	4	3	4	5	2	1	5	3	4
Serious	The Imitation Game	4	3	2	2	5	1	1	1	4	3	1	3	5	4	5	1	2	5	4	5
Serious	The Curious Case of Benjamin Button	4	3	3	2	5	1	2	3	3	3	1	4	4	3	5	1	3	5	2	5
Serious	Mr. Nobody	5	3	5	3	5	1	2	4	5	5	1	3	3	4	5	1	1	4	1	5
Serious	The Truman Show	4	5	4	3	5	2	3	5	4	2	1	4	5	4	4	2	4	3	1	5
Funny	The Mask	3	5	3	3	4	3	4	3	3	1	5	2	2	3	3	3	3	2	5	5
Funny	Yes Man	2	5	3	2	3	4	4	5	3	1	5	1	3	2	2	4	5	3	5	3
Funny	Dumb & Dumber	2	5	4	2	3	4	5	4	1	1	5	1	2	1	1	5	4	2	3	2
Funny	Deadpool	2	5	5	4	3	5	5	5	2	1	5	3	2	2	1	5	3	1	5	1
Funny	Who Framed Roger Rabbit	1	4	4	4	2	5	5	4	3	1	4	2	2	1	2	4	4	1	4	1
Funny	Intouchables	4	5	5	5	4	4	5	3	3	1	5	3	1	1	3	5	4	1	5	1
Funny	Back to the Future	3	5	5	4	3	5	4	3	1	1	5	2	2	1	1	2	4	4	4	1
Funny	Ice Age	3	5	3	5	3	4	5	3	1	1	5	1	3	1	2	5	5	4	5	2
Funny	Interstate 60	2	4	4	4	2	5	4	5	3	1	5	1	2	2	1	5	5	2	5	3
Funny	Click	2	5	3	3	1	4	5	4	1	1	5	1	1	2	1	5	5	1	4	1

Figure 1: The toy data we hand crafted. Movies and users are chacterized as funny or serious. The ratings are given accordingly with a slight noise. This is the input data used to generate plots [2-3].

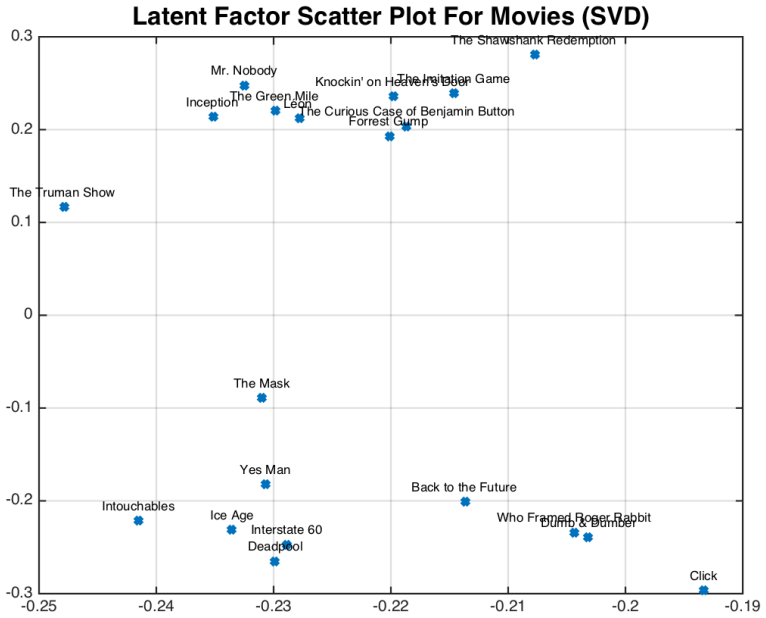


Figure 2: Toy data clustering illustration. No missing entries in the rating matrix. Number of latent factors is 2. Solved with SVD. This is the latent factor values for all movies. Two clusters are present as expected.

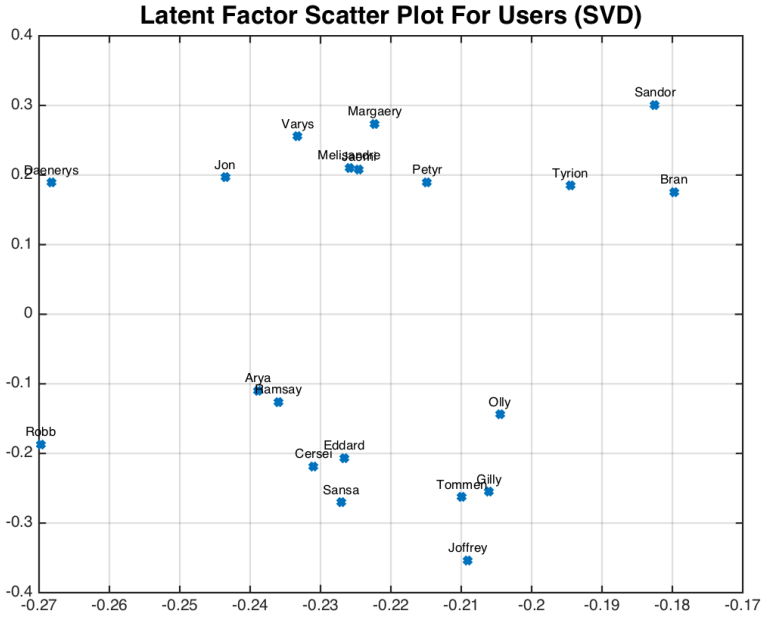


Figure 3: Toy data clustering illustration. No missing entries in the rating matrix. Number of latent factors is 2. Solved with SVD. This is the latent factor values for all users. Two clusters are present as expected.

		Serious	Funny	Funny	Funny	Serious	Funny	Funny	Funny	Serious	Serious	Funny	Serious	Serious	Serious	Serious	Funny	Funny	Serious	Funny	Serious
		Jon	Robb	Arya	Olly	Daenerys	Eddard	Sansa	Ramsay	Melisandre	Sandor	Joffrey	Tyrion	Petyr	Bran	Margaery	Tommen	Gilly	Jaime	Cersei	Varys
Serious	The Shawshank Redemption	NaN	NaN	4	NaN	5	NaN	4	NaN	NaN	4	NaN	2	2	NaN	1	4	NaN	NaN	NaN	2
Serious	The Green Mile	3	2	2	3	NaN	3	3	NaN	3	NaN	NaN	5	NaN	NaN	4	5	5	NaN	NaN	NaN
Serious	Forrest Gump	2	1	NaN	NaN	3	2	2	NaN	5	NaN	3	3	4	NaN	NaN	5	5	NaN	4	NaN
Serious	Inception	NaN	1	2	3	3	NaN	2	2	3	NaN	3	2	NaN	4	NaN	NaN	NaN	5	NaN	NaN
Serious	Léon	NaN	4	5	5	5	NaN	5	5	NaN	5	4	3	NaN	3	NaN	NaN	NaN	3	NaN	NaN
Serious	Knockin' on Heaven's Door	3	NaN	3	3	3	NaN	NaN	NaN	NaN	NaN	NaN	4	4	5	NaN	4	5	4	5	NaN
Serious	The Imitation Game	NaN	1	3	NaN	3	1	NaN	NaN	NaN	3	NaN	4	NaN	5	NaN	5	4	5	NaN	5
Serious	The Curious Case of Benjamin Button	1	NaN	1	NaN	NaN	3	1	3	4	NaN	3	NaN	4	5	4	NaN	NaN	NaN	5	4
Serious	Mr. Nobody	NaN	NaN	4	NaN	4	5	4	3	5	4	3	3	1	NaN	NaN	3	1	NaN	3	1
Serious	The Truman Show	NaN	5	5	4	NaN	5	3	NaN	5	2	1	NaN	1	1	NaN	NaN	1	1	1	NaN
Funny	The Mask	NaN	1	NaN	NaN	NaN	1	NaN	NaN	1	1	NaN	5	NaN	NaN	4	5	NaN	5	NaN	5
Funny	Yes Man	NaN	5	4	NaN	3	NaN	3	4	3	4	2	NaN	1	3	NaN	NaN	2	1	NaN	1
Funny	Dumb & Dumber	5	4	3	5	4	3	5	4	3	NaN	NaN	NaN	2	NaN	NaN	NaN	2	3	NaN	1
Funny	Deadpool	3	4	3	NaN	3	4	4	NaN	4	4	NaN	2	NaN	2	1	1	1	NaN	2	2
Funny	Who Framed Roger Rabbit	4	NaN	5	4	5	5	NaN	5	5	NaN	3	NaN	1	1	2	3	1	2	1	1
Funny	Intouchables	1	3	2	NaN	1	2	1	1	NaN	NaN	3	4	5	NaN	4	NaN	NaN	NaN	5	NaN
Funny	Back to the Future	1	1	2	1	1	NaN	2	3	NaN	NaN	3	5	NaN	3	4	NaN	4	5	NaN	5
Funny	Ice Age	NaN	3	NaN	4	NaN	NaN	5	5	4	3	2	3	2	NaN	1	NaN	4	NaN	2	NaN
Funny	Interstate 60	1	NaN	2	NaN	2	3	4	2	1	NaN	5	NaN	NaN	5	4	5	4	NaN	NaN	NaN
Funny	Click	5	5	NaN	NaN	NaN	NaN	5	5	5	NaN	5	3	2	1	NaN	NaN	NaN	NaN	NaN	1

Figure 4: The toy data we hand crafted. Movies and users are chacterized as funny or serious. The ratings are given accordingly with a slight noise. Some entries are hidden. This is the input data used to generate plots [5-18].

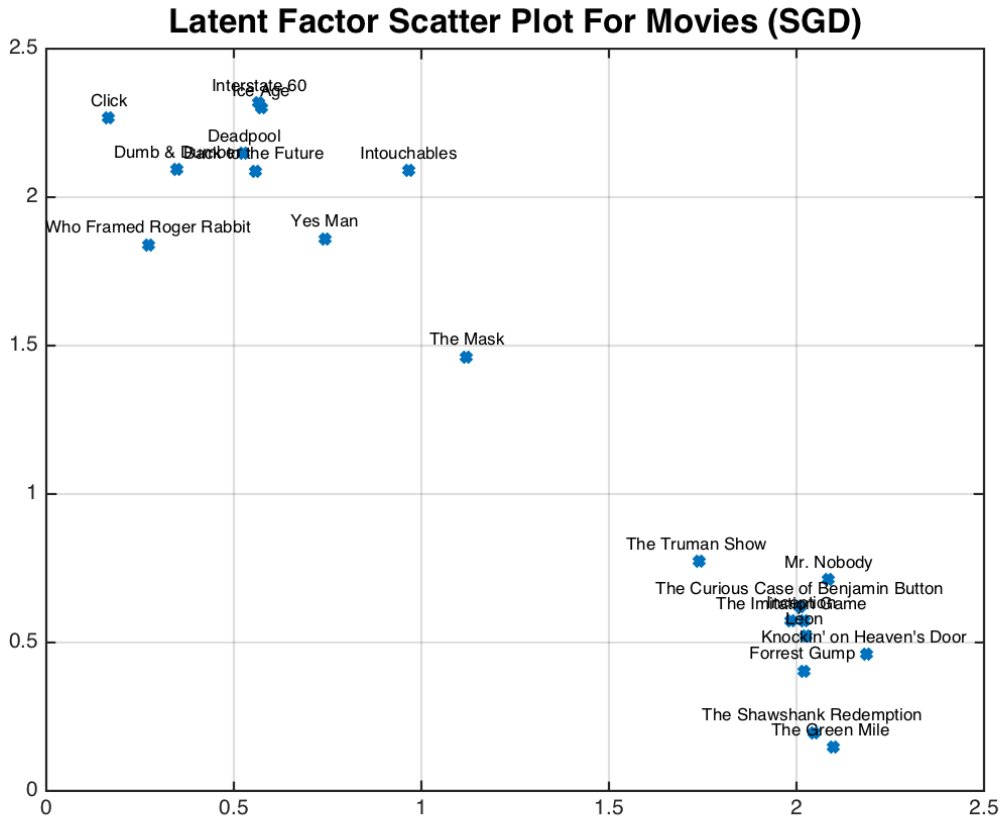


Figure 5: Toy data clustring illustration. Half of the entries are missing in the rating matrix. Number of latent factors is 2. Trained with SGD, used plain model. This is the latent factor values for all movies. Two clusters are present as expected. The amount of data is enough to spot the patterns.

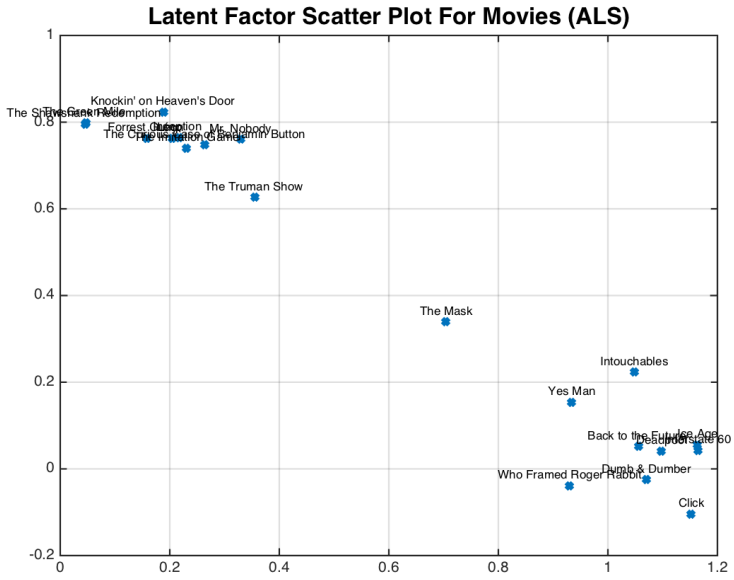


Figure 6: Toy data clustring illustration. Half of the entries are missing in the rating matrix. Number of latent factors is 2. Trained with ALS, used plain model. This is the latent factor values for all movies. Two clusters are present as expected. The amount of data is enough to spot the patterns. Notice that we cannot call Matlab's **svd()** because there are missing entries.

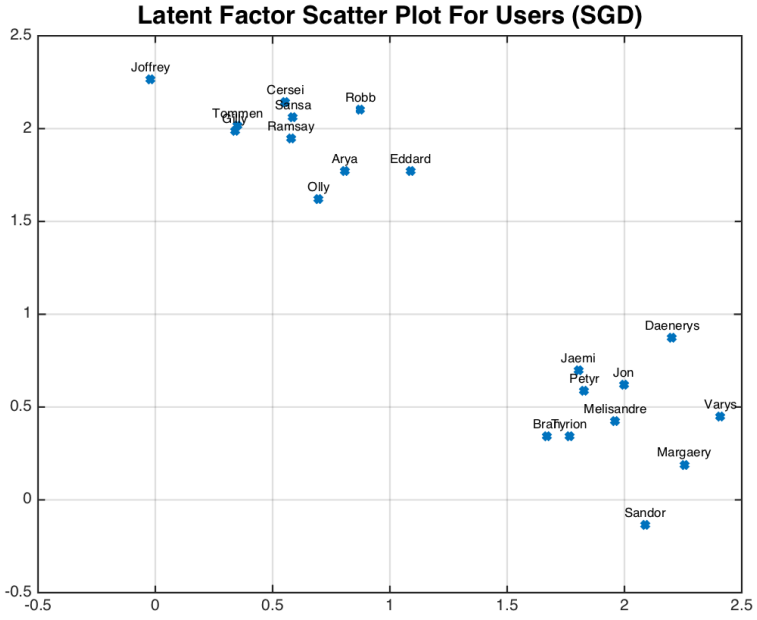


Figure 7: Toy data clustring illustration. Half of the entries are missing in the rating matrix. Number of latent factors is 2. Trained with SGD, used plain model. This is the latent factor values for all users. Two clusters are present as expected. The amount of data is enough to spot the patterns.

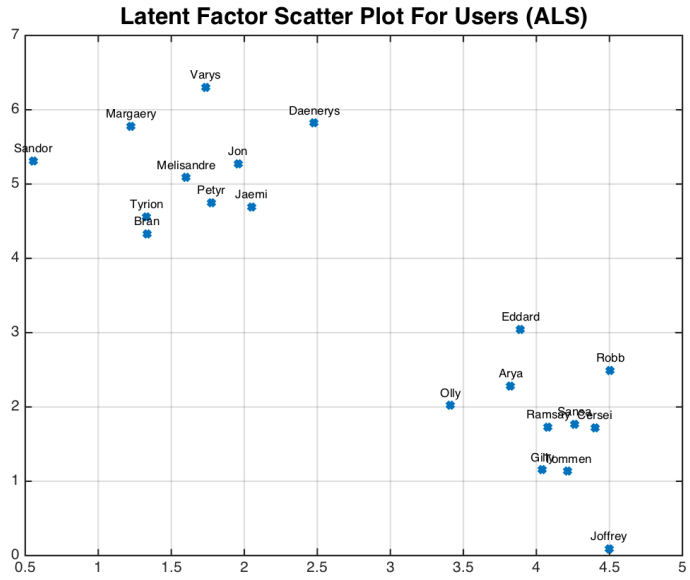


Figure 8: Toy data clustring illustration. Half of the entries are missing in the rating matrix. Number of latent factors is 2. Trained with ALS, used plain model. This is the latent factor values for all users. Two clusters are present as expected. The amount of data is enough to spot the patterns.

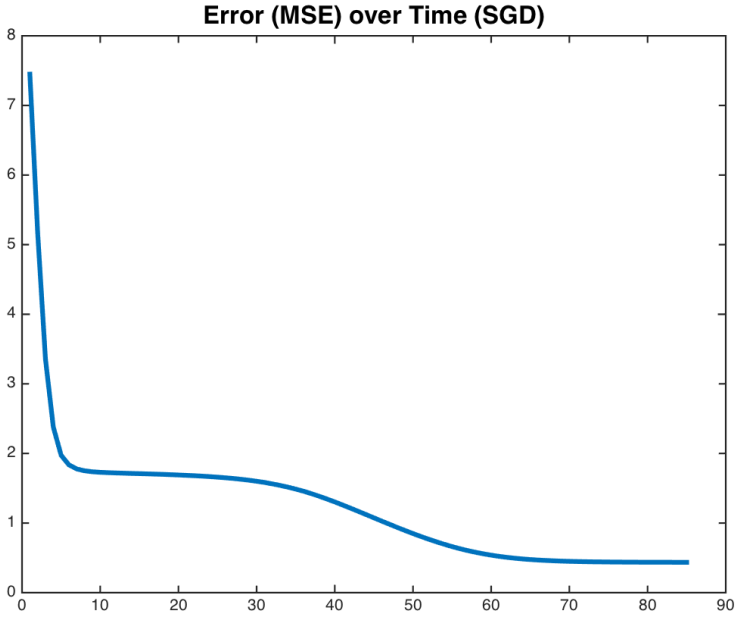


Figure 9: Toy data illustration. Half of the entries are missing in the rating matrix. Plot of mean square error versus epoch using SGD. Where epoch is one iteration through whole dataset. Notice that a larger learning rate might have cause an oscillation at the error level of 2. Time elapsed: 0.1603 sec

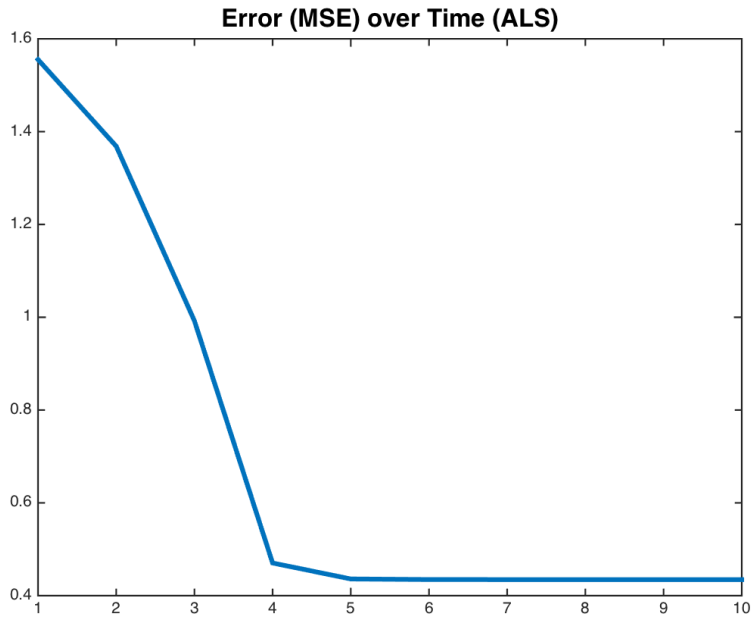


Figure 10: Toy data illustration. Half of the entries are missing in the rating matrix. Plot of mean square error versus iteration. One iteration is defined as solving least squares once both for U and I_T. Time elapsed: 0.0313 sec. Converges faster than SGD. As expected, ALS has good performance when the data is not sparse.

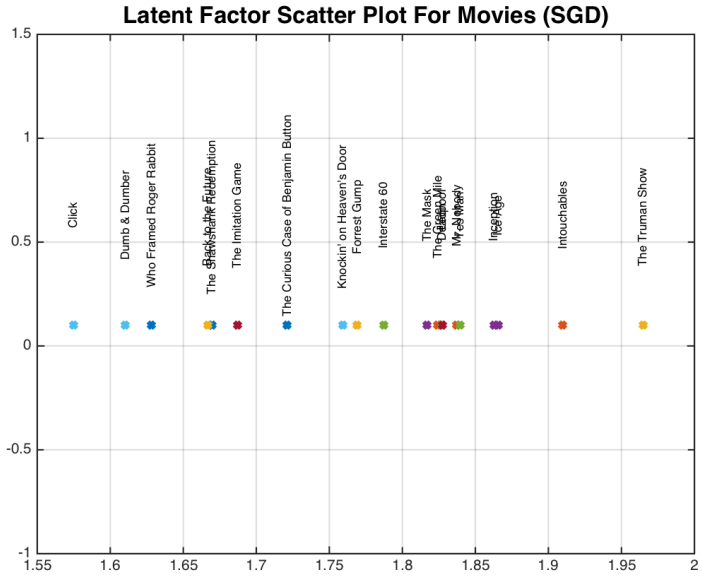


Figure 11: Toy data clustring illustration. Half of the entries are missing in the rating matrix. Number of latent factors is 1. Trained with SGD, used plain model. This is the latent factor values for all movies. No clusters can be spotted in spite of the case where number of latent factors were two. This points out that one factor is too small to capture the characteristics.

Figure 12: Toy data clustring illustration. Half of the entries are missing in the rating matrix. Number of latent factors is 1. Trained with ALS, used plain model. This is the latent factor values for all movies. No clusters can be spotted in spite of the case where number of latent factors were two. This points out that one factor is too small to capture the characteristics.

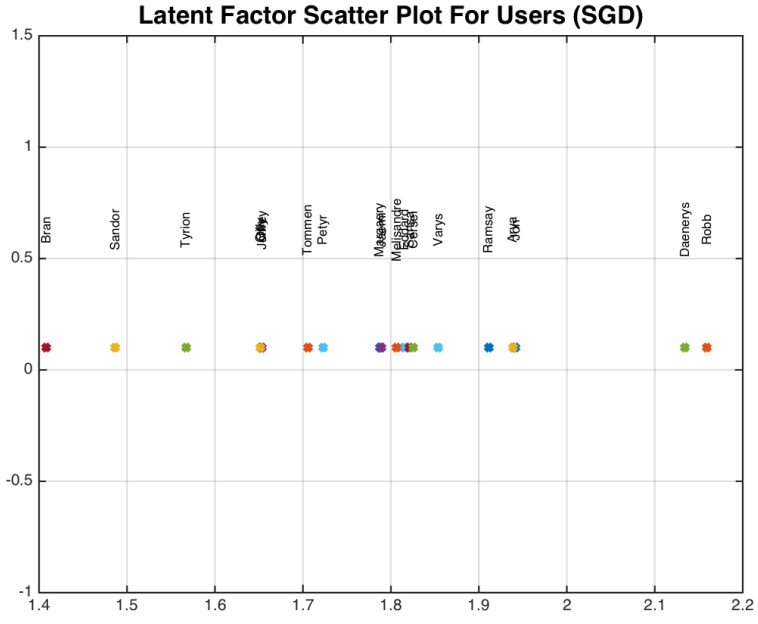


Figure 13: Toy data clustring illustration. Half of the entries are missing in the rating matrix. Number of latent factors is 1. Trained with SGD, used plain model. This is the latent factor values for all users. No clusters can be spotted in spite of the case where number of latent factors were two. This points out that one factor is too small to capture the characteristics.

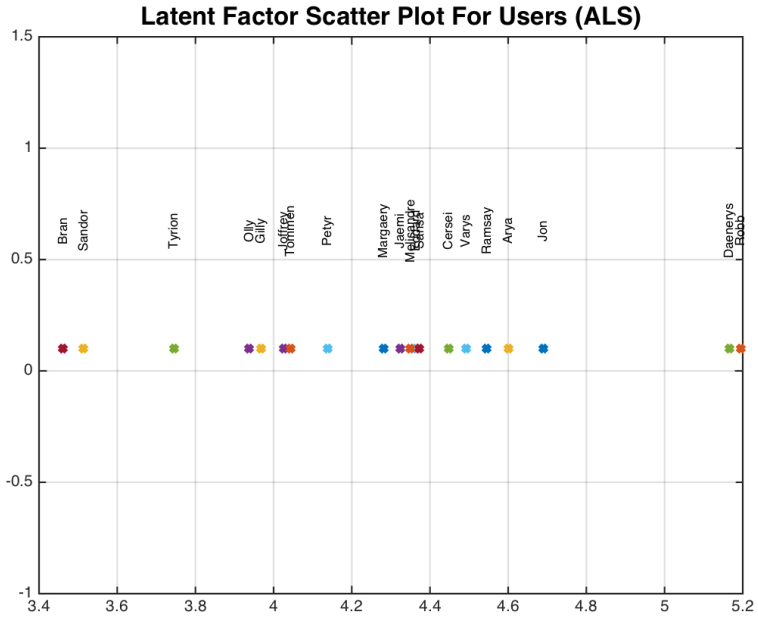


Figure 14: Toy data clustring illustration. Half of the entries are missing in the rating matrix. Number of latent factors is 1. Trained with ALS, used plain model. This is the latent factor values for all movies. No clusters can be spotted in spite of the case where number of latent factors were two. This points out that one factor is too small to capture the characteristics.

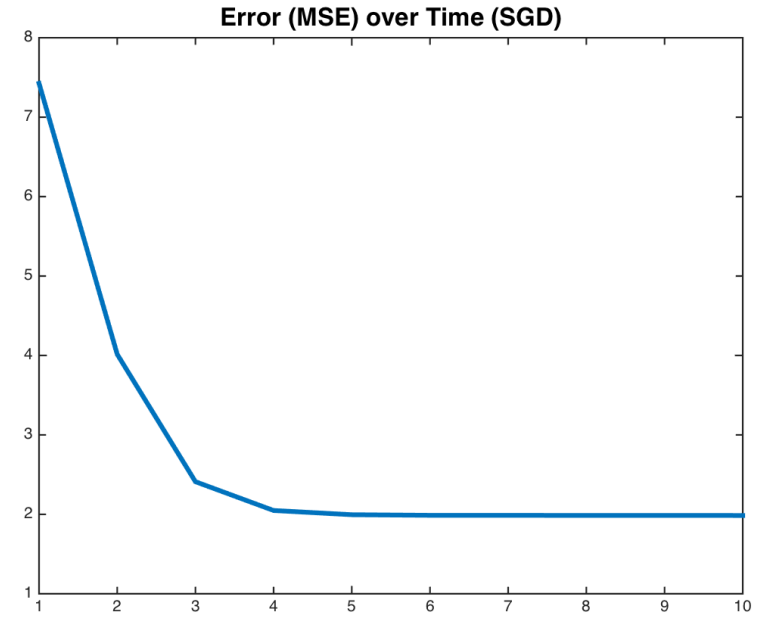


Figure 15: Toy data illustration. Half of the entries are missing in the rating matrix. Plot of mean square error versus epoch using SGD. Where epoch is one iteration through whole dataset. Time elapsed: 0.0541 sec

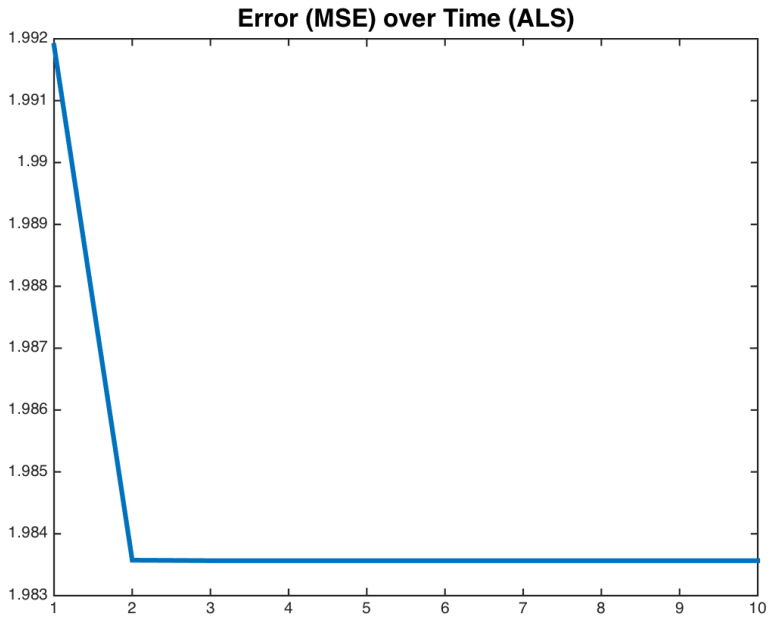


Figure 16: Toy data illustration. Half of the entries are missing in the rating matrix. Plot of mean square error versus iteration. One iteration is defined as solving least squares once both for U and I_T. Time elapsed: 0.0408 sec. Converges faster than SGD. As expected, ALS has good performance when the data is not sparse.

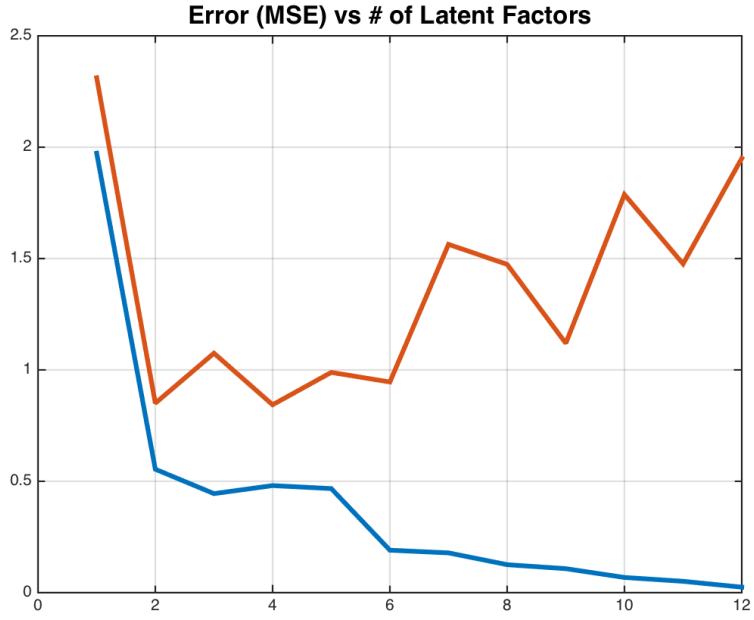


Figure 17: Toy data illustration. Half of the entries are missing in the rating matrix. This plot demonstrates how error on training (blue) and test (red) data changes according to number of latent factors when the model is trained with SGD.

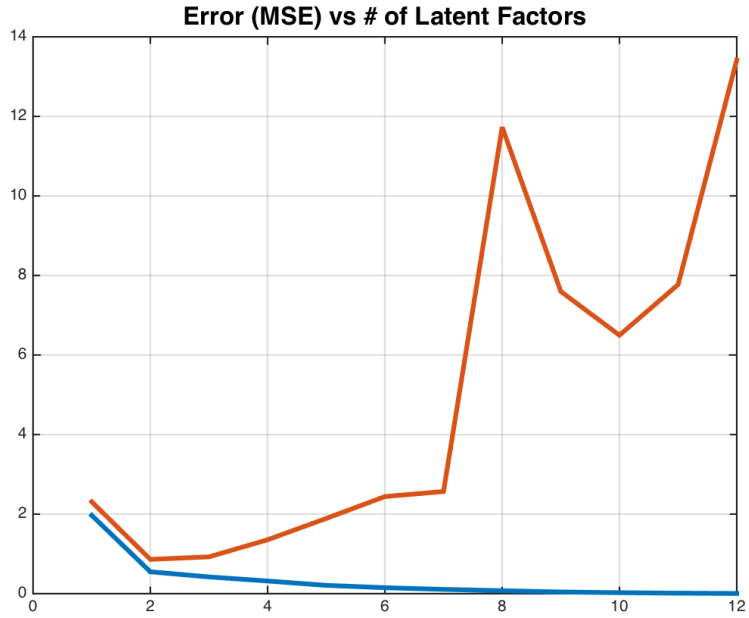


Figure 18: Toy data illustration. Half of the entries are missing in the rating matrix. This plot demonstrates how error on training (blue) and test (red) data changes according to number of latent factors when the model is trained with ALS.

Tests on MovieLens Data

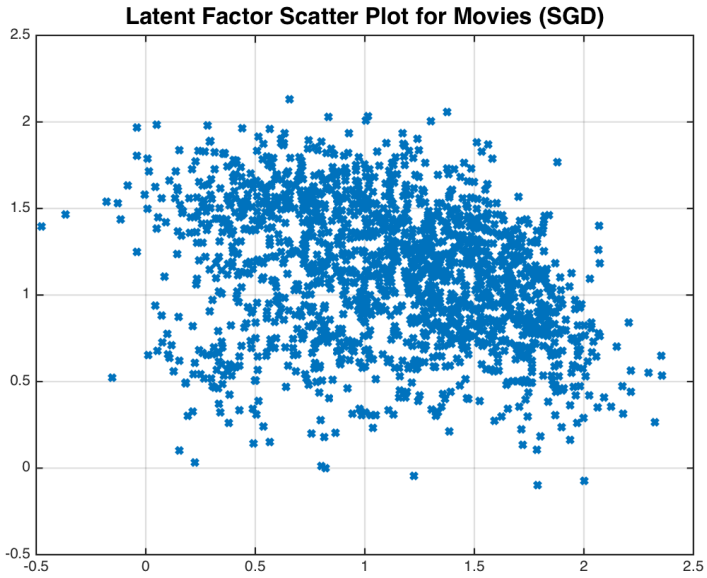


Figure 19: MovieLens data clustering illustration. Number of latent factors is 2. Trained with SGD, used plain model. This is the latent factor values for all 1682 movies. No clusters can be spotted. This points out that 2 factors are too small to capture the characteristics.

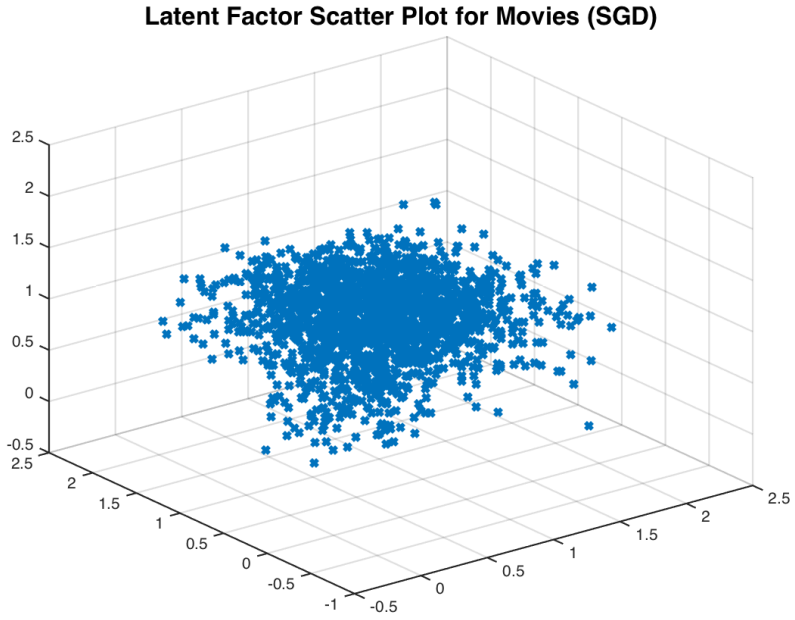


Figure 20: MovieLens data clustering illustration. Number of latent factors is 3. Trained with SGD, used plain model. This is the latent factor values for all 1682 movies. No clusters can be spotted. This points out that 3 factors are too small to capture the characteristics.

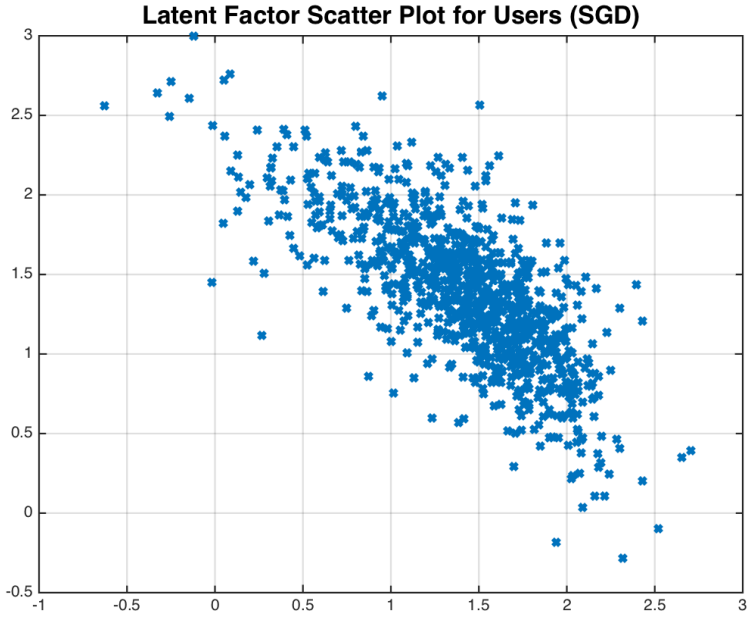


Figure 21: MovieLens data clustering illustration. Number of latent factors is 2. Trained with SGD, used plain model. This is the latent factor values for all 943 users. No clusters can be spotted. This points out that 2 factors are too small to capture the characteristics.

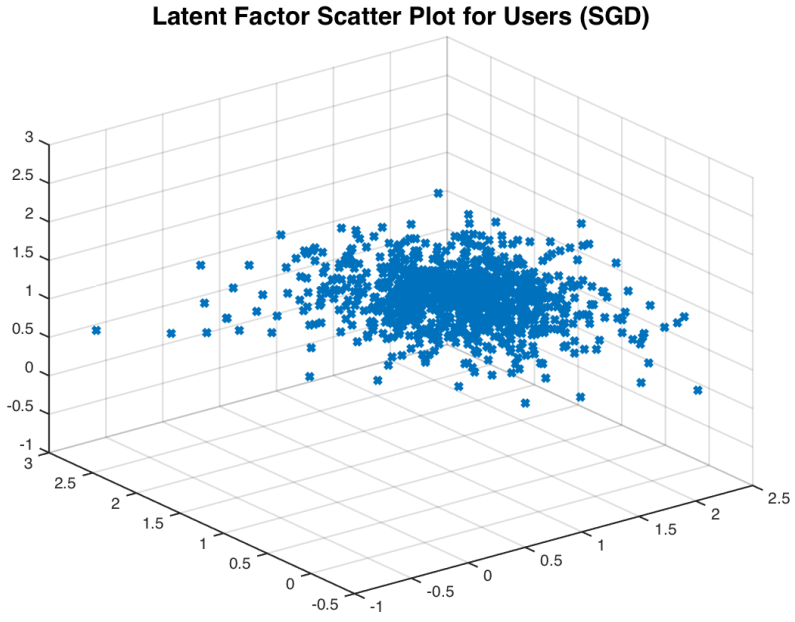


Figure 22: MovieLens data clustering illustration. Number of latent factors is 3. Trained with SGD, used plain model. This is the latent factor values for all 943 users. No clusters can be spotted. This points out that 3 factors are too small to capture the characteristics.

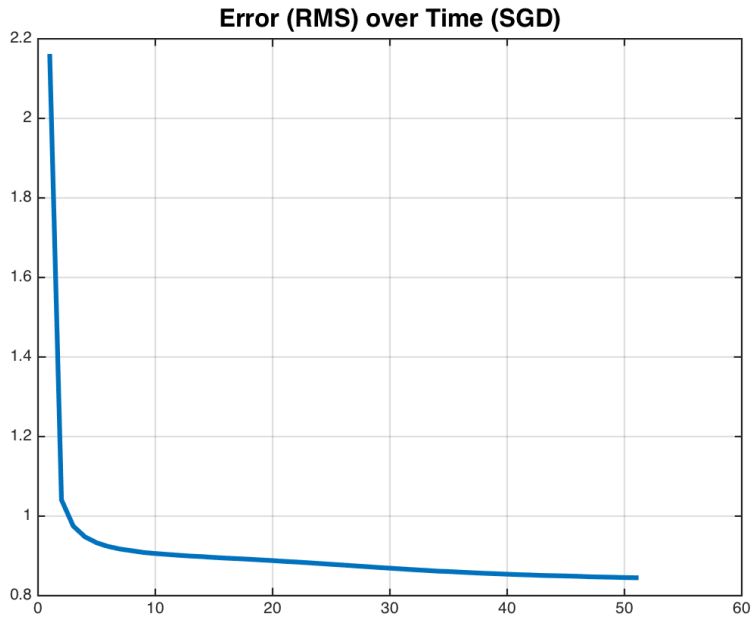


Figure 23: MovieLens data illustration. Plot of root mean square errors versus epochs using SGD, used plain model. Where epoch is one iteration through whole dataset.

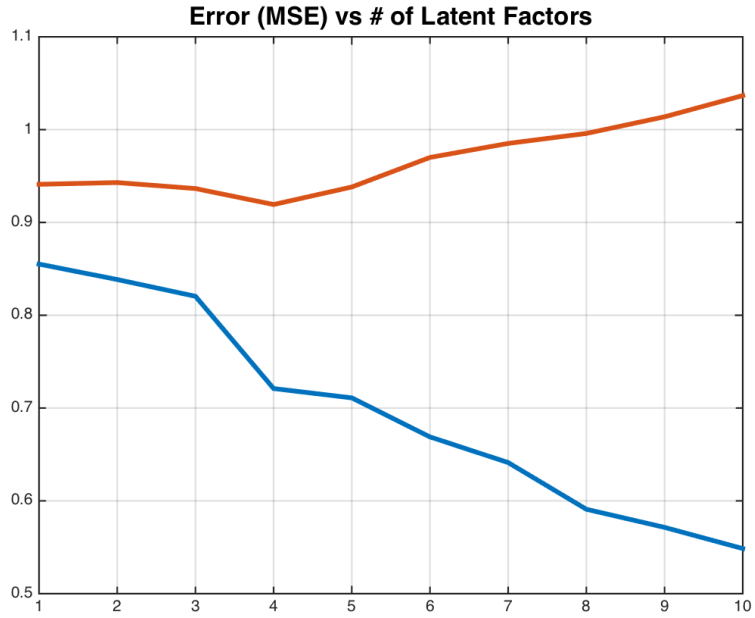


Figure 24: MovieLens data illustration. This plot demonstrates how error on training (blue) and test (red) data changes according to number of latent factors when the model is trained with SGD.

		BIAS ON							BIAS OFF						
		0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.0	0.1	0.2	0.3	0.4	0.5	0.6
RMS Error on Training Data	1	0.90	0.91	0.92	0.93	0.94	0.94	0.95	0.92	0.93	0.95	0.98	1.01	1.06	1.10
	2	0.90	0.91	0.92	0.93	0.94	0.94	0.95	0.92	0.93	0.94	0.97	1.01	1.05	1.10
	3	0.88	0.90	0.92	0.93	0.93	0.94	0.95	0.90	0.92	0.94	0.97	1.00	1.06	1.10
	4	0.88	0.90	0.91	0.92	0.93	0.94	0.95	0.90	0.92	0.94	0.97	1.01	1.05	1.10
	5	0.87	0.90	0.91	0.92	0.93	0.94	0.95	0.89	0.92	0.94	0.97	1.00	1.05	1.10
	6	0.83	0.89	0.91	0.92	0.93	0.94	0.95	0.88	0.91	0.94	0.97	1.01	1.05	1.11
	7	0.83	0.89	0.91	0.92	0.93	0.94	0.95	0.87	0.91	0.94	0.97	1.00	1.05	1.11
	8	0.81	0.89	0.91	0.92	0.93	0.94	0.95	0.86	0.91	0.94	0.97	1.01	1.05	1.10
	9	0.78	0.89	0.91	0.92	0.93	0.94	0.95	0.82	0.90	0.94	0.97	1.01	1.04	1.10
	10	0.76	0.88	0.91	0.92	0.93	0.94	0.95	0.77	0.90	0.93	0.96	1.00	1.05	1.10
RMS Error on Test Data	1	0.96	0.96	0.97	0.97	0.98	0.98	0.99	0.97	0.98	0.99	1.02	1.05	1.09	1.14
	2	0.96	0.96	0.96	0.97	0.98	0.98	0.99	0.97	0.97	0.99	1.01	1.05	1.09	1.14
	3	0.96	0.96	0.96	0.97	0.97	0.98	0.99	0.97	0.97	0.99	1.01	1.04	1.09	1.13
	4	0.97	0.96	0.96	0.97	0.98	0.98	0.99	0.97	0.97	0.99	1.01	1.05	1.08	1.13
	5	0.97	0.96	0.96	0.97	0.97	0.98	0.99	0.97	0.97	0.99	1.00	1.04	1.09	1.14
	6	0.97	0.96	0.96	0.97	0.97	0.98	0.98	0.98	0.97	0.98	1.01	1.04	1.08	1.14
	7	0.99	0.96	0.96	0.97	0.97	0.98	0.99	0.98	0.97	0.99	1.01	1.04	1.09	1.14
	8	0.98	0.96	0.96	0.97	0.97	0.98	0.99	0.98	0.97	0.98	1.01	1.04	1.08	1.13
	9	0.99	0.96	0.96	0.97	0.97	0.98	0.98	0.99	0.97	0.98	1.01	1.04	1.08	1.13
	10	1.00	0.96	0.96	0.97	0.97	0.98	0.98	1.00	0.97	0.98	1.00	1.04	1.08	1.14

Figure 25: MovieLens data illustration. Table of root mean square errors on training and test data according to chosen number of latent factor [1-10], regularization coefficient [0.0-0.6] and bias condition 0,1. When bias is set on the bias terms *(described in "Approach" section)* are included in the training of the model and vice versa. Bias terms in the model indicate a slight enhancement.

Index	Liked	6	12	14	44	60	64	81	82	86	91	96	100	108	113	114	129	150	154	170	171	174	175	177	183	190	196	202	208	221	224	228	235	242	253	258
	Disliked	74	78	103	104	112	120	140	143	219	243	254	259	260	266																					
Reel	Liked	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	Disliked	1	1	1	1	1	1	1	1	1	1	1	1	1	1																					
Estimated	Liked	3,81	4,76	4,39	3,19	4,78	4,61	3,86	3,20	4,34	3,87	3,94	4,75	3,42	4,09	4,50	4,16	4,20	4,41	4,17	4,54	4,22	4,47	4,48	4,61	4,49	3,96	3,88	4,13	4,01	4,03	3,82	3,05	4,55	4,07	4,05
	Disliked	3,64	1,45	2,08	2,26	1,72	1,84	2,80	3,61	3,27	1,78	1,86	2,22	2,40	2,66																					

Figure 26: In this table we show the estimated ratings for favourite (rated five stars) and disliked (rated one star) movies of a random user. Index rows contain movie ids rather than names.

Unknown Disliked	Unrecommended	Unknown Liked	Recommended
'Faster Pussycat! Kill! Kill! (1965)'	'Kazaam (1996)'	'Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)'	'Pather Panchali (1955)'
'Free Willy (1993)'	'Island of Dr. Moreau, The (1996)'	'Usual Suspects, The (1995)'	'Usual Suspects, The (1995)'
'All Dogs Go to Heaven 2 (1996)'	'Celtic Pride (1996)'	'Postino, Il (1994)'	'Silence of the Lambs, The (1991)'
'Theodore Rex (1995)'	'Free Willy (1993)'	'Dolores Claiborne (1994)'	'Shawshank Redemption, The (1994)'
'Flipper (1996)'	'Flipper (1996)'	'Three Colors: Blue (1993)'	'Pulp Fiction (1994)'
'Striptease (1996)'	'Spy Hard (1996)'	'Shawshank Redemption, The (1994)'	'Schindler''s List (1993)'
'Homeward Bound: The Incredible Journey (19	'Cabin Boy (1994)'	'Hudsucker Proxy, The (1994)'	'Saint of Fort Washington, The (1993)'
'Sound of Music, The (1965)'	'Meet Wally Sparks (1997)'	'Jurassic Park (1993)'	'Yankee Zulu (1994)'
'Nightmare on Elm Street, A (1984)'	'Amityville 3-D (1983)'	'Remains of the Day, The (1993)'	'Psycho (1960)'
'Jungle2Jungle (1997)'	'Striptease (1996)'	'Nightmare Before Christmas, The (1993)'	'Close Shave, A (1995)'
'Batman & Robin (1997)'	'Drop Dead Fred (1991)'	'Terminator 2: Judgment Day (1991)'	'High Noon (1952)'
'George of the Jungle (1997)'	'Man of the House (1995)'	'Fargo (1996)'	'Apocalypse Now (1979)'
'Event Horizon (1997)'	'Leave It to Beaver (1997)'	'Kids in the Hall: Brain Candy (1996)'	'One Flew Over the Cuckoo''s Nest (1975)'
'Kull the Conqueror (1997)'	'Dangerous Ground (1997)'	'Horseman on the Roof, The (Hussard sur le toit, Le) (1995)'	'Aiqing wansui (1994)'
	'Vampire in Brooklyn (1995)'	'Wallace & Gromit: The Best of Aardman Animation (1996)'	'Rear Window (1954)'
	'Sunset Park (1996)'	'Bound (1996)'	'Paradise Lost: The Child Murders at Robin Hood Hills (1996)'
	'Beautician and the Beast, The (1997)'	'Swingers (1996)'	'Citizen Kane (1941)'
	'Amityville Curse, The (1990)'	'Monty Python''s Life of Brian (1979)'	'Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb
	'Gordy (1995)'	'Cinema Paradiso (1988)'	'Crossfire (1947)'
	'Ed (1996)'	'Delicatessen (1991)'	'Damsel in Distress, A (1937)'
	'That Darn Cat! (1997)'	'Raiders of the Lost Ark (1981)'	'Some Mother''s Son (1996)'
	'Super Mario Bros. (1993)'	'Brazil (1985)'	'Casablanca (1942)'
	'Anaconda (1997)'	'Good, The Bad and The Ugly, The (1966)'	'Raiders of the Lost Ark (1981)'
	'Wishmaster (1997)'	'Alien (1979)'	'Lady of Burlesque (1943)'
	'Keys to Tulsa (1997)'	'Henry V (1989)'	'North by Northwest (1959)'
	'White Man''s Burden (1995)'	'Dead Poets Society (1989)'	'Good Will Hunting (1997)'
	'Pocahontas (1995)'	'Groundhog Day (1993)'	'When We Were Kings (1996)'
	'Mortal Kombat: Annihilation (1997)'	'Young Frankenstein (1974)'	'Boot, Das (1981)'
	'Lawnmower Man 2: Beyond Cyberspace (1996)'	'Breaking the Waves (1996)'	'Henry V (1989)'
	'Land Before Time III: The Time of the Great Giving (1995) (V)'	'Ridicule (1996)'	'Paris Was a Woman (1995)'
	'Jury Duty (1995)'	'Star Trek: The Wrath of Khan (1982)'	'Butcher Boy, The (1998)'
	'Double Team (1997)'	'Mars Attacks! (1996)'	'Some Folks Call It a Sling Blade (1993)'
	'Big Bully (1996)'	'Kolya (1996)'	'Taxi Driver (1976)'
	'Poison Ivy II (1995)'	'Pillow Book, The (1995)'	'Big Bang Theory, The (1994)'
	'Jingle All the Way (1996)'	'Contact (1997)'	'Sunset Blvd. (1950)'
	'Ready to Wear (Pret-A-Porter) (1994)'		'39 Steps, The (1935)'
	'Richie Rich (1994)'		'Wallace & Gromit: The Best of Aardman Animation (1996)'
	'Getting Even with Dad (1994)'		'Paths of Glory (1957)'
	'In the Army Now (1994)'		'Wild Bunch, The (1969)'
	'Speed 2: Cruise Control (1997)'		'Alien (1979)'
	'Amityville 1992: It''s About Time (1992)'		'As Good As It Gets (1997)'
	'B*A*P*S (1997)'		'Duck Soup (1933)'
	'Stupids, The (1996)'		'Fargo (1996)'
	'Amityville II: The Possession (1982)'		'L.A. Confidential (1997)'
	'Congo (1995)'		'Thin Man, The (1934)'

Figure 27: In this table **Unkown Disliked** column contains the movies given one star by a random user. They are *unkown* to system becuse movies are from test set. Similarly, **Unknown Liked** column contains the movies given five stars by that user. **Recommended** column contains the movies that we estimated highest ratings for. In like manner, **Unrecommended** column contains the movies that we estimated lowest ratings for.

5 Appendices

See the source code online at `https://github.com/oeken/cathar`

script_1.m

```
close all;
clc;
clear;
format bank;
rng(12345);
%%
%read the excel file
% filename='toy_movies.xlsx';
% [num,txt,raw]=xlsread(filename);
% R_true = num';
% R_trai = R_true;
% R_test = R_true;
% [u,i] = size(R_true);
% trai_size = 0.9;
% test = randperm(u*i,u*i*trai_size);
% trai = setdiff(1:u*i,test);
% R_trai(trai) = nan;
% R_test(test) = nan;
% instances_trai = to_instances(R_trai);
% instances_test = to_instances(R_test);

%% READ TRAINING AND TEST DATA

filename = '../data/ml-100k/movs';
movie_names = importdata(filename);

% ===== Training Data =====
filename = '../data/ml-100k/u1.base';
A = importdata(filename);
instances_trai = A(:,1:3);
R_trai = to_matrix(instances_trai,1);
% ===== Test Data =====
filename = '../data/ml-100k/u1.test';
A = importdata(filename);
instances_test = A(:,1:3);
R_test = to_matrix(instances_test,1);

%% TRAIN & PLOT

acc = 0.001;     % convergence condition
err = 'rms';     % error calc method
l   = 3;         % latent factors
reg = 0.1;       % regulatization weight
bia = 1;

tic
[L_U, L_I, B_U, B_I, mu, iteration, e_all] = sgd(R_trai, ...
                                                 instances_trai, ...
                                                 l, ...
                                                 reg, ...
                                                 acc, ...
                                                 err, ...
                                                 bia);
```

```
54 toc
55
56
57 %% LATENT & REGULARIZATION VS ERROR
58 % bia = 0
59 % bia_off_stats = zeros(10,7,2);
60 % for j=1:10
61 %     j
62 %     for k=1:7
63 %         k
64 %         tic
65 %         [L_U, L_I, B_U, B_I, mu, iteration, e_all] = sgd(R_trai, ...
66 %                                                          instances_trai, ...
67 %                                                          j, ...
68 %                                                          (k-1)/10, ...
69 %                                                          acc, ...
70 %                                                          err, ...
71 %                                                          bia);
72 %         toc
73 %         R_hat = mu + B_U + B_I + L_U * L_I';
74 %         bia_off_stats(j,k,1) = compute_error(R_trai, R_hat, err);
75 %         bia_off_stats(j,k,2) = compute_error(R_test, R_hat, err);
76 %     end
77 % end
78 %
79 % bia = 1
80 % bia_on_stats = zeros(10,7,2);
81 % for j=1:10
82 %     j
83 %     for k=1:7
84 %         k
85 %         tic
86 %         [L_U, L_I, B_U, B_I, mu, iteration, e_all] = sgd(R_trai,
87 %                                                          instances_trai, ...
88 %                                                          j, ...
89 %                                                          (k-1)/10, ...
90 %                                                          acc, ...
91 %                                                          err, ...
92 %                                                          bia);
93 %         toc
94 %         R_hat = mu + B_U + B_I + L_U * L_I';
95 %         bia_on_stats(j,k,1) = compute_error(R_trai, R_hat, err);
96 %         bia_on_stats(j,k,2) = compute_error(R_test, R_hat, err);
97 %     end
98 % end
99
100 %% COMPUTE ERROR
101
102 % R_hat1 = mu1 + B_U1 + B_I1 + L_U1 * L_I1';
103 % e_1 = compute_error(R_trai, R_hat1, err);
104 % e_2 = compute_error(R_test, R_hat1, err);
105
106 %% RECOMMENDATION
107 R_hat = mu + B_U + B_I + L_U * L_I';
108 u = 1;
109 liked = find(R_test(u,:) == 5);
110 disliked = find(R_test(u,:) == 1);
111 est_liked = R_hat(u,liked);
112 est_disliked = R_hat(u,disliked);
113
```

```
liked = find(R_test(u,:) == 5);
disliked = find(R_test(u,:) == 1);
unknown_liked_movies = movie_names(liked);
unknown_disliked_movies = movie_names(disliked);

liked = find(R_trai(u,:) == 5);
disliked = find(R_trai(u,:) == 1);
known_liked_movies = movie_names(liked);
known_disliked_movies = movie_names(disliked);

known = find(~isnan(R_trai(u,:)));
lk = length(known);
est = R_hat(u,:);
est(known) = nan;
[v, index] = sort(est,'descend');
recx = 50;
recommended = movie_names(index(lk+1:lk+recx+1));
unrecommended = movie_names(index(end-recx+1:end));

% opinion_on_recommended = R_test(index(lk+1:lk+recx+1));
% opinion_on_unrecommended = R_test(index(end-recx+1:end));

good_guess_on_like = intersect(recommended,unknown_liked_movies);
good_guess_on_dislike = intersect(unrecommended,unknown_disliked_movies);


%% PLOTS

ts = 16;
figure()
plot(1:length(e_all),e_all,'LineWidth',3);
title('Error (RMS) over Time (SGD)','FontSize',ts)
grid on
saveas(gcf, 'buff/1', 'png')

%%% 1 %%%
% figure()
% % plot(L_U1(:,1),L_U1(:,2),'x','LineWidth',3);
% scatter3(L_U1(:,1),L_U1(:,2),L_U1(:,3),'x','LineWidth',3)
% title('Latent Factor Scatter Plot for Users (SGD)','FontSize',ts)
% grid on
% saveas(gcf, 'buff/users', 'png')

%%% 2 %%%
% figure()
% % plot(L_I1(:,1),L_I1(:,2),'x','LineWidth',3);
% scatter3(L_I1(:,1),L_I1(:,2),L_I1(:,3),'x','LineWidth',3)
% title('Latent Factor Scatter Plot for Movies (SGD)','FontSize',ts)
% grid on
% saveas(gcf, 'buff/movs', 'png')



% %%% 2 %%%
% figure()
% % len_e1 = length(e_all1);
% plot(1:max_l,e_als_trai,'LineWidth',3);
% hold on;
% plot(1:max_l,e_als_test,'LineWidth',3);
```

```
% title('Error (MSE) vs # of Latent Factors','FontSize',ts)
% grid on
% saveas(gcf, 'buff/l_als', 'png')

%%% 2 %%%
% figure()
% len_e2 = length(e_all2);
% plot(1:len_e2,e_all2,'LineWidth',3);
% title('Error (MSE) over Time (ALS)','FontSize',ts)
% saveas(gcf, 'buff/6', 'png')

%% FIN
disp('␣')
disp('All␣done')
```

sgd.m

```
function [L_U, L_I, B_U, B_I, mu, iteration, e_all] = sgd(R, ...
                                                          instances, ...
                                                          l, ...
                                                          reg, ...
                                                          acc, ...
                                                          err, ...
                                                          bia)

    N = length(instances);
    u = max(instances(:,1));
    i = max(instances(:,2));

    L_U = rand(u,l);
    L_I = rand(i,l);
    if bia
        B_U = rand(u,1);
        B_I = rand(i,1);
        mu = mean(instances(:,3));  % average of all ratings
    else
        B_U = zeros(u,1);
        B_I = zeros(i,1);
        mu = 0;
    end

    % loss fn = (r - b_u - b_i - mu - qp)^2
    %           + lambda * (q^2 + p^2 + b_u^2 + b_i^2)

    eta = 0.01;  % learning rate
    lambda = reg;  % weight of regularization

    b_size = 5;
    iteration = 1;
    R_hat = mu + repmat(B_U,1,i) + repmat(B_I,1,u)' + L_U * L_I';
    e = compute_error(R, R_hat, err);
    e_all = e;

    while true
        I = speye(N);
        P = I(randperm(N),:);  % permutation matrix
        shuffled_instances = P * instances;  % shuffle instances
        for j=1:N
            current_instance = shuffled_instances(j,:);
```

```
43              user = current_instance(1);
44              item = current_instance(2);
45              r = current_instance(3);  % desired output
46
47              user_fact = L_U(user,:);
48              item_fact = L_I(item,:);
49              user_bias = B_U(user);
50              item_bias = B_I(item);
51
52              y = mu + user_bias + item_bias + user_fact * item_fact';  % predicted output
53              e = r - y;  % error
54
55              update_user_fact = eta * (e * item_fact - lambda * user_fact);
56              update_item_fact = eta * (e * user_fact - lambda * item_fact);
57
58              if bia
59                  update_user_bias = eta * (e - lambda * user_bias);
60                  update_item_bias = eta * (e - lambda * item_bias);
61              end
62
63              L_U(user,:) = user_fact + update_user_fact;
64              L_I(item,:) = item_fact + update_item_fact;
65
66              if bia
67                  B_U(user) = user_bias + update_user_bias;
68                  B_I(item) = item_bias + update_item_bias;
69              end
70
71          end
72
73          R_hat = mu + repmat(B_U,1,i) + repmat(B_I,1,u)' + L_U * L_I';
74          e = compute_error(R, R_hat, err);
75          e_all = [e_all e];
76
77          converged = 0;
78          if mod(iteration,b_size) == 0
79              e_buffer = e_all(end-b_size+1:end)';  % last <b_size> elements
80              converged = has_converged(e_buffer, 2, acc);
81          end
82          if converged
83              break;
84          end
85          iteration = iteration + 1;
86      end
87      B_U = repmat(B_U,1,i);
88      B_I = repmat(B_I,1,u)';
89  end
```

als.m

```
function [L_U, L_I, iteration, e_all] = als(R, instances, l, acc)

    N = length(instances);
    u = max(instances(:,1));
    i = max(instances(:,2));

    L_U = rand(u,l);
    L_I = rand(i,l);

    A = L_U;
    B = L_I';

    b_size = 5;
    iteration = 1;
    e_all = [];

    while true
        b_mat = zeros(N,u*l);
        for k=1:N
            temp = instances(k,1:2);
            x = temp(1);
            y = temp(2);
            locations = 1+(x-1)*l : x*l;
            values = B(:,y);
            b_mat(k,locations) = values;
        end

        A = reshape(b_mat\instances(:,3),[l,u]);
        A = A';

        a_mat = zeros(N,i*l);
        for k=1:N
            temp = instances(k,1:2);
            x = temp(1);
            y = temp(2);
            locations = 1+(y-1)*l : y*l;
            values = A(x,:);
            a_mat(k,locations) = values;
        end
        B = reshape(a_mat\instances(:,3),[l,i]);

        e = compute_error(R, A*B, 'mse');
        e_all = [e_all e];

        converged = 0;
        if mod(iteration,b_size) == 0
            e_buffer = e_all(end-b_size+1:end)';  % last <b_size> elements
            converged = has_converged(e_buffer, 2, acc);
        end
        if converged
            break;
        end
        iteration = iteration + 1;

    end
    L_U = A;
    L_I = B';
end
```

has_converged.m

```
function r = has_converged(buffer, smooth, threshold)
    val = tsmovavg(buffer,'s',smooth,1);  % moving average
    val = val(smooth:end); % remove NaNs
    change = abs(diff(val) ./ val(1:end-1));  % relative differences
    good = change < threshold;  % if they are all less then threshold
    r = min(good);  % then output 1 else 0
end
```

compute_error.m

```
function e = compute_error(R, R_hat, method)
    E = R - R_hat;
    E_vec = E(:);
    e = nanmean(E_vec.^2);
    if strcmp(method,'rms')
        e = sqrt(e);
    end
end
```

to_instances.m

```
function instances = to_instances(R)
    [u,i] = find(~isnan(R));
    s = length(u);
    instances = [u i zeros(s,1)];
    for j=1:s
        instances(j,3) = R(u(j),i(j));
    end
end
```

to_matrix.m

```
function R = to_matrix(instances,huge)
    u = 943;
    i = 1682;
    if ~huge
        u = max(instances(:,1));
        i = max(instances(:,2));
    end
    R = nan(u,i);
    N = length(instances);
    for j=1:N
        u_j = instances(j,1);
        i_j = instances(j,2);
        R(u_j,i_j) = instances(j,3);
    end
end
```