Recommender System using Collaborative Filtering

z5049624 Sida Zhang

1.Introduction

Collaborative filtering filters information by using the recommendations of other people. It is based on the idea that people who agreed in their evaluation of certain items in the past are likely to agree again in the future. A person who wants to see a movie for example, might ask for recommendations from friends. The recommendations of some friends who have similar interests are trusted more than recommendations from others. This information is used in the decision on which movie to see. Basically, there are three typical types of collaborative filtering, which are memory-based, model-based, and hybrid. The goal of this project is to use Biased Singular Value Decomposition, a model-based approach improved upon Simon Funk's SVD algorithm (2006), to build a movie recommender system using the 100K dataset from MovieLens. Features of this project include:

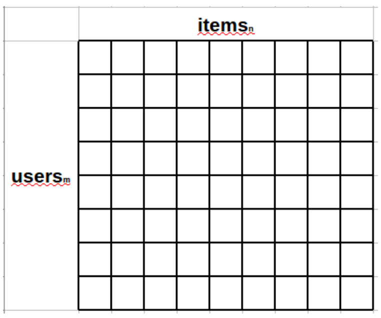
(1) Predict rating of any movie that a user has not given.
(2) Give recommendation movie list of any user according to predicted ratings. (extra feature)
(3) Evaluate the system using metrics of RMSE and MAE
(4) Provide an interactive interface (CLI) to perform the above functionalities (extra feature)

2. Method

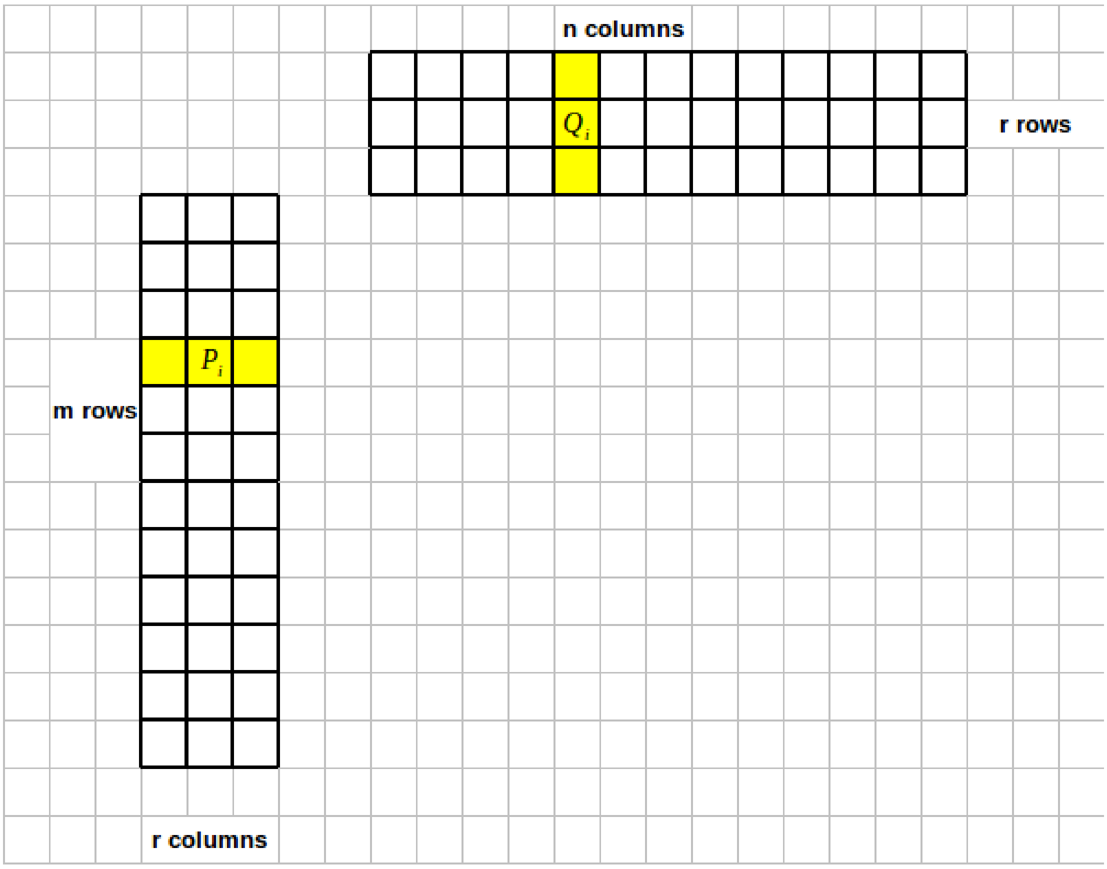
Since only rating scores and user identifiers are given, how much a user likes an item can be evaluated by this user's taste on other items or views of other users who share the same interest. A common disadvantage of the later intuition is the fact that provided datasets are always sparse, and it would be difficult for each user to join a group of the same taste with sufficient members. Besides, a group of people prefer one kind of items but others can also be desired for each individual. This poses a challenge to selecting an appropriate group of users to represent a user's interest on the given item. Hence, item-based approach is preferable.

But, meanwhile, we will be facing a problem that people make decisions according to some of its attributes in fact, which are not explicitly exposed in original data. Thereby, the core of the recommender system becomes latent factor analysis, finding what "attributes" (factors) can determinate the ratings. Alternatively, in a mathematical way, the machine learns a linear combination of what factors can best represent the score that a user would give to an item. If we represent user i as a vector of preferences for a few factors P_i, and represent item j as a vector Q_j where each element expresses how much the item exhibits that factor, the production of P_i and Q_j could be an estimate of the user's rating.

Consider a user-item matrix where each cell is a rating that a user gives to an item,



it is expected that the matrix can be decomposed into two low-rank matrices that $P_i . Q_j$ corresponds to user i's rating on item j in the original matrix above, and r can be seen as the number of factors of user and item.



However, r cannot be guaranteed to achieve for both P and Q. As preferences of users must correspond to factors of items, the production of P and Q in such a manner can be an approximation to the user-item matrix, and we expect the error is minimized. If we use the traditional SVD, the matrix will be decomposed into three ones and this process will be computational expensive. More importantly, it does not suit the sparse matrix in our case.
Simon Funk introduced a method to break these barriers, that minimizing the cost function to achieve the decomposition rather than the inverse way. And it is easy to implement in computer program.

$$\underset{P_i, Q_j}{argmin} \sum_{i,j} (m_{i,j} - P_i . Q_j)^2 \quad ,$$

P_i *is a row vector* , Q_j *is a column vector* , $m_{i,j}$ *is rating of item*$_j$ *given by user*$_i$

In practice, we apply regularization to avoid overfitting.

$$J = \sum_{i,j} (m_{i,j} - P_i . Q_j)^2 + \lambda (\|p_i\|_2^2 + \|q_j\|_2^2)$$

Take derivatives to obtain gradients:

$$\frac{\partial J}{\partial P_i} = -2(m_{ij} - P_i . Q_j) Q_j + 2\lambda P_i ,$$

$$\frac{\partial J}{\partial Q_j} = -2(m_{ij} - P_i . Q_j) P_i + 2\lambda Q_j$$

Then the updating rules are:

$$P_i = P_i + \alpha((m_{ij} - P_i . Q_j) Q_j - \lambda P_i),$$
$$Q_j = Q_j + \alpha((m_{ij} - P_i . Q_j) P_i - \lambda Q_j),$$
$$\alpha \; is \; learning \; rate$$

The above illustrates the Funk SVD algorithm. An improvement to it is adding a biased item $\mu+b_i+b_u$ where

b_u is the user bias, which measures the user's tendency to rate the items and

b_i is the item bias, which measures the tendency of the item's ratings in the dataset, μ is average rating of all elements in the user-item matrix. Then the cost function accordingly becomes

$$J=\sum_{i,j}(m_{ij}-\mu-b_i-b_u-P_i.Q_j)^2+\lambda(\|P_i\|_2^2+\|Q_j\|_2^2+\|b_i\|_2^2+\|b_j\|_2^2)$$

and the corresponding updating rules are

$$P_i=P_i+\alpha((m_{ij}-\mu-b_i-b_u-P_i.Q_j)Q_j-\lambda P_i),$$
$$Q_j=Q_j+\alpha((m_{ij}-\mu-b_i-b_u-P_i.Q_j)P_i-\lambda Q_j),$$
$$b_i=b_i+\alpha((m_{ij}-\mu-b_i-b_u-P_i.Q_j)-\lambda b_i),$$
$$b_u=b_u+\alpha((m_{ij}-\mu-b_i-b_u-P_i.Q_j)-\lambda b_u)$$

To predict a new item's rating, simply recover it from the trained model P,Q.

$$m_{ij}=\mu+b_i+b_u+P_i.Q_j$$

3. Experiment

RMSE is used to indicate the number of iterations to stop the gradient descent in the experiment. As each iteration of the gradient descent takes around 15s which is fairly long for the whole process, a limit to the number of iterations (**MaxIter**), instead of the error change, is set to **20** according to average performance on all the 5 CV sets. Besides, to ensure the accuracy of converge, the learning rate will decline 10% after each iteration (**learningRate *= 0.9**). And the **number of latent factors**, which is the number of columns in P (and the row number in Q), is set to **10** which has the best performance in a range from 5 to 50. The last important parameter is **L2 penalty coefficient**, which gives the best results when it is **0.05**.
(graphs to derive these values are in appendix)

4. Result

4.1 Running result

The recommender system can receive a user's id as input and predict the ratings that this user would give to not-yet-rated movies. And it is noticeable that the predicted ratings are float numbers, ranging from 0.0 to 5.0. Then, in such a manner, recommended movie list can be given, including movie titles and corresponding sorted ratings. Returning top-k recommendations is also supported.

4.2 Performance evaluation

Accuracy of the prediction is assessed using RMSE and MAE which are defined as:

$$RMSE=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(predict_i-groundTruth_i)^2}$$

$$MAE=\frac{1}{n}\sum_{i=1}^{n}|predict_i-groundTruth_i|$$

For the 5-fold cross validation of the Movielens 100K dataset, the best result achieved in the experiment is shown as follow:

	u1	u2	u3	u4	u5	avg
RMSE	0.919698253396208	0.9132460483363246	0.9063472176375376	0.908711352078286	0.9095967927705968	0.9115318039252122
MAE	0.7204997928764971	0.7141578579506594	0.7103402739888629	0.7115254135143657	0.718575437861617	0.7150197552384004

This performance is close to mymedialite's benchmark (BiasedMatrixFactorization), and further works on adjusting Max Iteration, regularization coefficient, and number of factors could give a better result since they were optimized individually while the combination of them determines the performance.

MovieLens 100k

5-fold crossvalidation with `--random-seed=1`

Method	`--recommender-options`	RMSE	MAE
BipolarSlopeOne		0.96754	0.74462
UserItemBaseline	`reg_u=5 reg_i=2`	0.94192	0.74503
SlopeOne		0.93978	0.74038
UserKNNCosine	`k=40 reg_u=12 reg_i=1`	0.937	0.737
UserKNNPearson	`k=60 shrinkage=25 reg_u=12 reg_i=1`	0.92971	0.72805
ItemKNNCosine	`k=40 reg_u=12 reg_i=1`	0.924	0.727
FactorWiseMatrixFactorization	`num_factors=5 num_iter=5 shrinkage=150`	0.9212	0.7252
BiasedMatrixFactorization	`num_factors=5 bias_reg=0.1 reg_u=0.1 reg_i=0.1 learn_rate=0.07 num_iter=100 bold_driver=true`	0.91678	0.72289
BiasedMatrixFactorization	`num_factors=10 bias_reg=0.1 reg_u=0.1 reg_i=0.12 learn_rate=0.07 num_iter=100 bold_driver=true`	0.91496	0.72209
SVDPlusPlus	`num_factors=4 regularization=0.1 bias_reg=0.005 learn_rate=0.01 bias_learn_rate=0.007 num_iter=50`	0.9138	0.71836
ItemKNNPearson	`k=40 shrinkage=2500 reg_u=12 reg_i=1`	0.91327	0.7144
BiasedMatrixFactorization	`num_factors=40 bias_reg=0.1 reg_u=1.0 reg_i=1.2 learn_rate=0.07 num_iter=100 frequency_regularization=true bold_driver=true`	0.90764	0.71722
BiasedMatrixFactorization	`num_factors=80 bias_reg=0.003 reg_u=0.09 reg_i=0.1 learn_rate=0.07 num_iter=100 bold_driver=true`	0.91153	0.72013
SVDPlusPlus	`num_factors=10 regularization=0.1 bias_reg=0.005 learn_rate=0.01 bias_learn_rate=0.007 num_iter=50`	0.91096	0.7152
BiasedMatrixFactorization	`num_factors=320 bias_reg=0.007 reg_u=0.1 reg_i=0.1 learn_rate=0.07 num_iter=500 bold_driver=true`	0.91073	0.72053
BiasedMatrixFactorization	`num_factors=160 bias_reg=0.003 reg_u=0.08 reg_i=0.1 learn_rate=0.07 num_iter=100 bold_driver=true`	0.91047	0.71944

(http://www.mymedialite.net/examples/datasets.html)

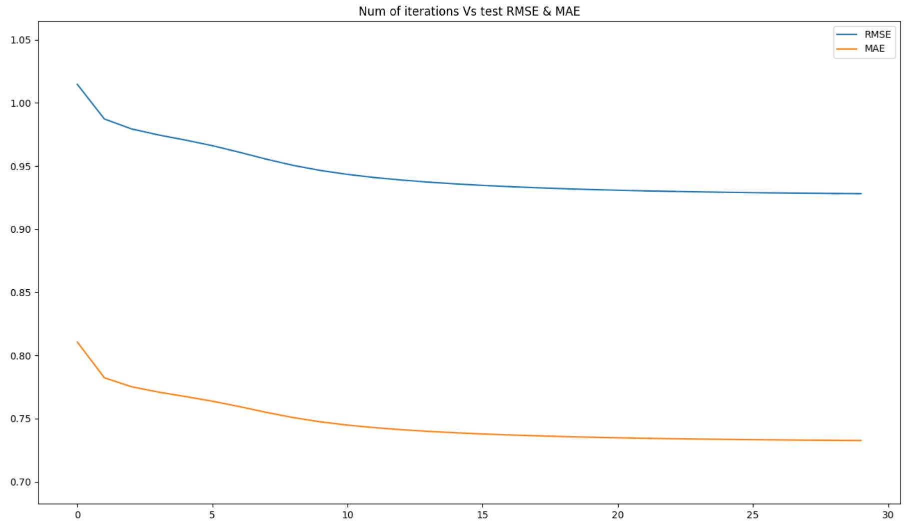
Name	Call Count	Time (ms)
SVD	1	46453 98.8%

(It takes 46s to train 80K records – e.g. u1.base)

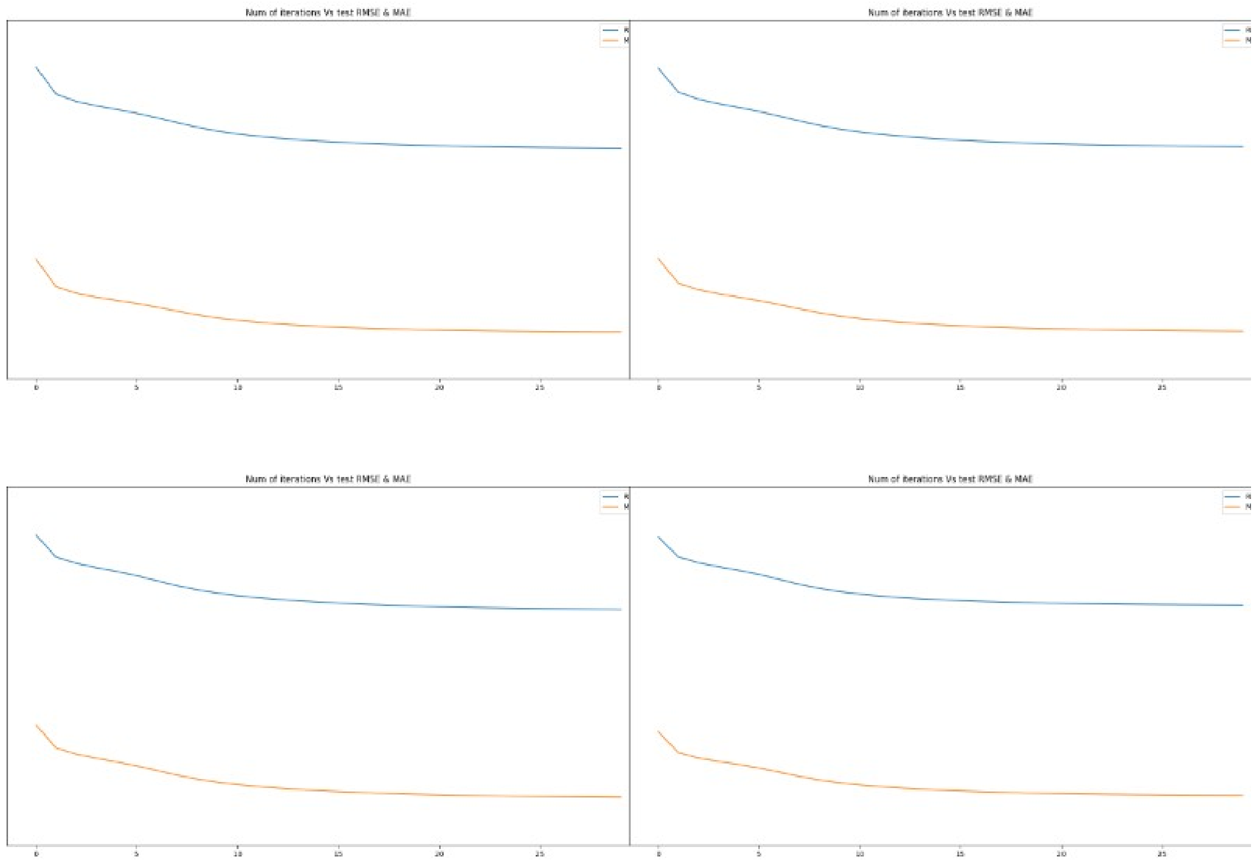
Predict	1	7046 91.4%

(and 7s to predict – e.g. u1.test)

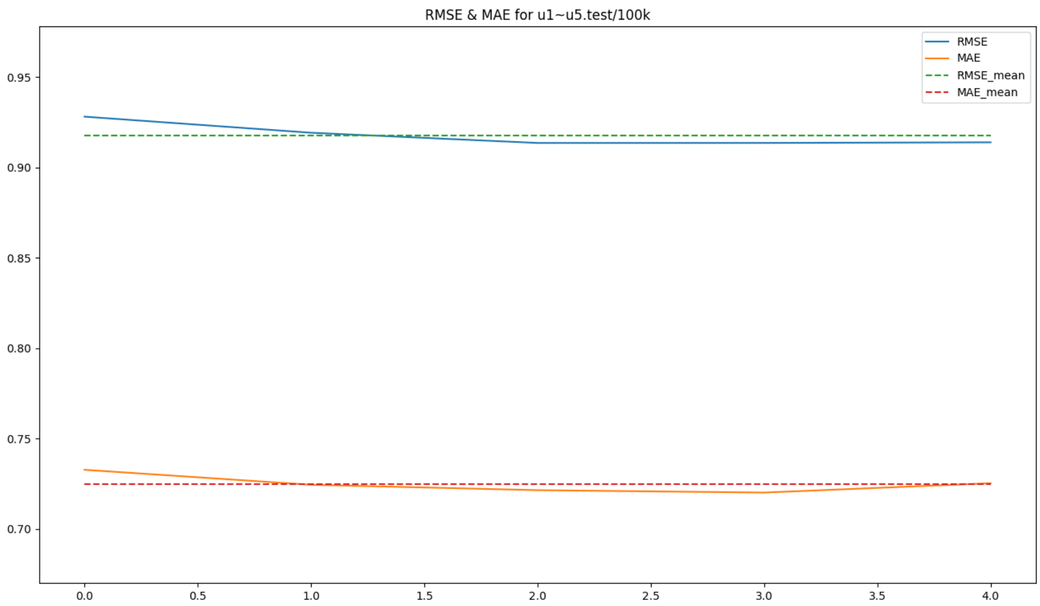
5. Appendix



(RMSE and MAE curves on u1.test with MaxIteration number from 1 to 29)



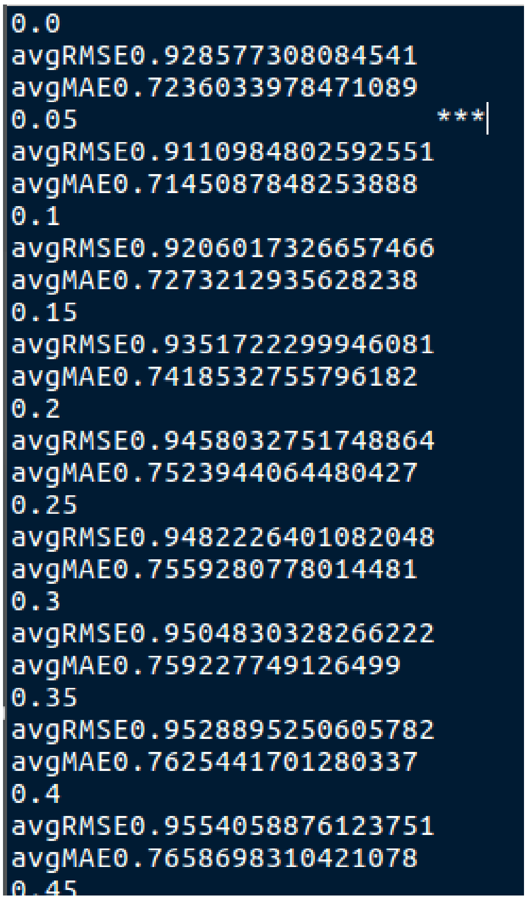
(Similar shapes on u2 ~ u5)



(RMSE and MAE on the 5 sets)

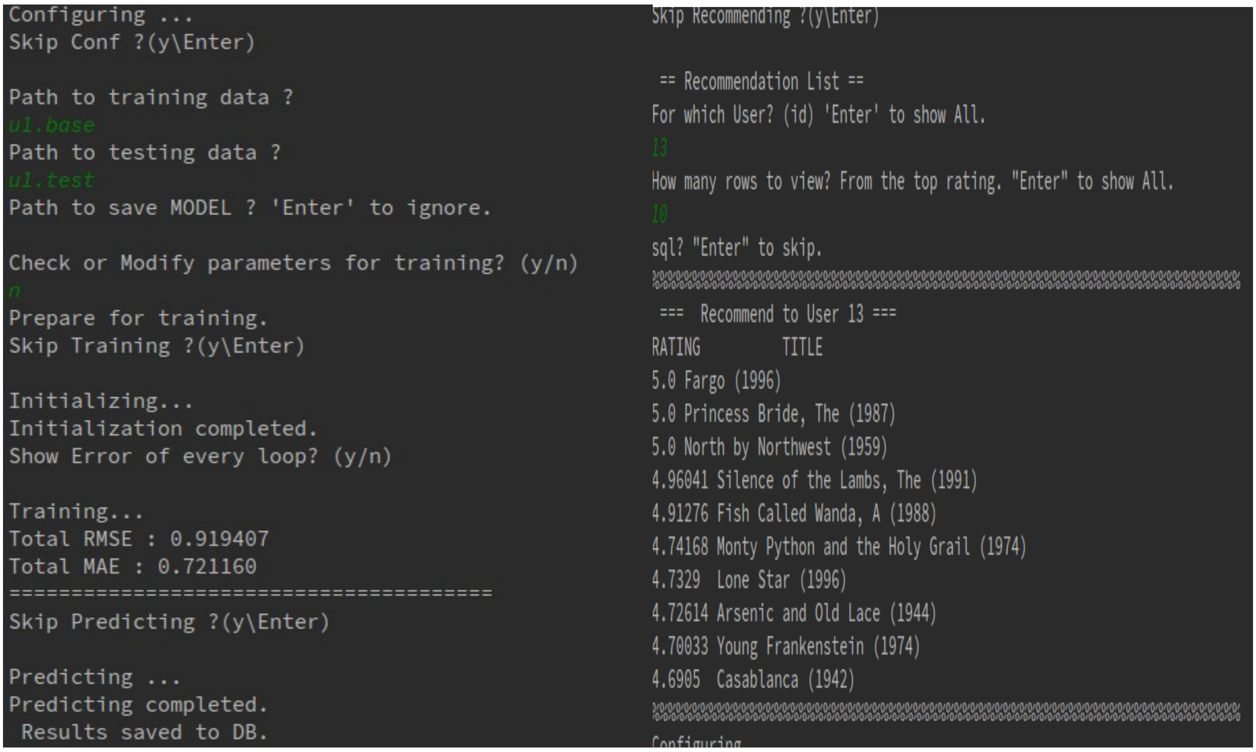
#Factors	RMSE	MAE	
5	0.921054430731856	0.7276799351051431	
6	0.9208890317628455	0.7274626719963752	
7	0.9208870421543842	0.7275392068563215	
8	0.9213644621154286	0.7278965132362223	
9	0.9208666267703857	0.7275139317775654	
10	0.9201111776456024	0.7269030420597229	***
11	0.9205978624490095	0.727300737763004	
12	0.9209702992430115	0.7276337914072017	
13	0.9206935639687901	0.727340143186229	
14	0.9207304799858461	0.727438374745595	
15	0.9208728824591506	0.7275020525394659	
16	0.9207296513773902	0.727428080365217	
17	0.9205168451595235	0.7272292464484525	
18	0.9205900982454656	0.7273032352573309	
19	0.9205646991893468	0.727305973901702	

(NumOfFactors=10, full list in ‘FactorNumSelect.txt’)



```
0.0
avgRMSE0.928577308084541
avgMAE0.7236033978471089
0.05                          ***
avgRMSE0.9110984802592551
avgMAE0.7145087848253888
0.1
avgRMSE0.9206017326657466
avgMAE0.7273212935628238
0.15
avgRMSE0.9351722299946081
avgMAE0.7418532755796182
0.2
avgRMSE0.9458032751748864
avgMAE0.7523944064480427
0.25
avgRMSE0.9482226401082048
avgMAE0.7559280778014481
0.3
avgRMSE0.9504830328266222
avgMAE0.759227749126499
0.35
avgRMSE0.9528895250605782
avgMAE0.7625441701280337
0.4
avgRMSE0.9554058876123751
avgMAE0.7658698310421078
0.45
```

(regularization coefficient = 0.05, full list in ‘PenaltySelect.txt’)



```
Configuring ...
Skip Conf ?(y\Enter)

Path to training data ?
u1.base
Path to testing data ?
u1.test
Path to save MODEL ? 'Enter' to ignore.

Check or Modify parameters for training? (y/n)
n
Prepare for training.
Skip Training ?(y\Enter)

Initializing...
Initialization completed.
Show Error of every loop? (y/n)

Training...
Total RMSE : 0.919407
Total MAE : 0.721160
======================================
Skip Predicting ?(y\Enter)

Predicting ...
Predicting completed.
 Results saved to DB.
```

```
Skip Recommending ?(y\Enter)

 == Recommendation List ==
For which User? (id) 'Enter' to show All.
13
How many rows to view? From the top rating. "Enter" to show All.
10
sql? "Enter" to skip.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 ===  Recommend to User 13 ===
RATING        TITLE
5.0 Fargo (1996)
5.0 Princess Bride, The (1987)
5.0 North by Northwest (1959)
4.96041 Silence of the Lambs, The (1991)
4.91276 Fish Called Wanda, A (1988)
4.74168 Monty Python and the Holy Grail (1974)
4.7329  Lone Star (1996)
4.72614 Arsenic and Old Lace (1944)
4.70033 Young Frankenstein (1974)
4.6905  Casablanca (1942)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Configuring ...
```

(Interaction example)

References

1. Zhao, Q. and Dong, Z., Attribute-Biased-SVD in Recommeder System.
2. Paterek, A., 2007, August. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop* (Vol. 2007, pp. 5-8).
3. Lévy, P., 1997. *Collective intelligence*. New York: Plenum/Harper Collins.
4. Simon, F., 2006. Netflix Update: Try This at Home