The Experiment Report of Machine Learning

SCHOOL : SCHOOL OF SOFTWARE ENGINEERING

SUBJECT : SOFTWARE ENGINEERING

Author:
Hu Lin, Fengchao Wang and
Jiaqi Zhong

Supervisor:
Qingyao Wu

Student ID：
201721045497, 201720145013
and 201720144948

Grade:
Graduate

December 25, 2017

Recommender System Based on Matrix Decomposition

Abstract—This experiment of machine learning is about recommender system based on matrix decomposition, and the experiment utilized MovieLens-100k dataset. The purpose of the experiment is to Explore the construction of recommended system, and understand the principle of matrix decomposition, and be familiar to the use of gradient descent.

I. Introduction

This experiment of machine learning is about recommender system based on matrix decomposition, and the experiment utilized MovieLens-100k dataset. The purpose of the experiment is to Explore the construction of recommended system, and understand the principle of matrix decomposition, and be familiar to the use of gradient descent, and construct a recommendation system under small-scale dataset, cultivate engineering ability.

II. Methods and Theory

A. Recommender System

A recommender system or a recommendation system (sometimes replacing "system" with a synonym such as platform or engine) is a subclass of information filtering system that seeks to predict the "rating" or "preference" that a user would give to an item.

Recommender systems typically produce a list of recommendations in one of two ways – through collaborative filtering or through content-based filtering (also known as the personality-based approach). Collaborative filtering approaches build a model from a user's past behaviour (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users. This model is then used to predict items (or ratings for items) that the user may have an interest in. Content-based filtering approaches utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties. These approaches are often combined (see Hybrid Recommender Systems).

The differences between collaborative and content-based filtering can be demonstrated by comparing two popular music recommender systems – Last.fm and Pandora Radio.

Last.fm creates a "station" of recommended songs by observing what bands and individual tracks the user has listened to on a regular basis and comparing those against the listening behavior of other users. Last.fm will play tracks that do not appear in the user's library, but are often played by other users with similar interests. As this approach leverages the behavior of users, it is an example of a collaborative filtering technique.

Pandora uses the properties of a song or artist (a subset of the 400 attributes provided by the Music Genome Project) to seed a "station" that plays music with similar properties. User feedback is used to refine the station's results, deemphasizing certain attributes when a user "dislikes" a particular song and emphasizing other attributes when a user "likes" a song. This is an example of a content-based approach.

Each type of system has its strengths and weaknesses. In the above example, Last.fm requires a large amount of information about a user to make accurate recommendations. This is an example of the cold start problem, and is common in collaborative filtering systems. Whereas Pandora needs very little information to start, it is far more limited in scope (for example, it can only make recommendations that are similar to the original seed).

Recommender systems are a useful alternative to search algorithms since they help users discover items they might not have found otherwise. Of note, recommender systems are often implemented using search engines indexing non-traditional data.

Recommender systems were first mentioned in a technical report as a "digital bookshelf" in 1990 by Jussi Karlgren at Columbia University, and implemented at scale and worked through in technical reports and publications from 1994 onwards by Jussi Karlgren, then at SICS, and research groups led by Pattie Maes at MIT, Will Hill at Bellcore, and Paul Resnick, also at MIT whose work with GroupLens was awarded the 2010 ACM Software Systems Award.

Montaner provided the first overview of recommender systems from an intelligent agent perspective. Adomavicius provided a new, alternate overview of recommender systems. Herlocker provides an additional overview of evaluation techniques for recommender systems, and Beel et al. discussed the problems of offline evaluations. Beel et al. have also provided literature surveys on available research paper recommender systems and existing challenges.

Recommender systems have been the focus of several granted patents.

B. Matrix Factorization

Just as its name suggests, matrix factorization is to, obviously, factorize a matrix, i.e. to find out two (or more) matrices such that when you multiply them you will get back the original matrix.

From an application point of view, matrix factorization can be used to discover latent features underlying the interactions between two different kinds of entities. (Of course, you can consider more than two kinds of entities and you will be dealing with tensor factorization, which would be more complicated.)

And one obvious application is to predict ratings in collaborative filtering.

Having discussed the intuition behind matrix factorization, we can now go on to work on the mathematics. Firstly, we have a set U of users, and a set D of items. Let R of size |U| × |D| be the matrix that contains all the ratings that the users have assigned to the items. Also, we assume that we would like to discover K latent features. Our task, then, is to find two matrics matrices P (a |U| × K matrix) and Q (a |D| × K matrix) such that their product approximates R:

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}}$$

In this way, each row of P would represent the strength of the associations between a user and the features. Similarly, each row of Q would represent the strength of the associations between an item and the features. To get the prediction of a rating of an item d_j by u_i, we can calculate the dot product of the two vectors corresponding to u_i and d_j:

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$$

Now, we have to find a way to obtain P and Q. One way to approach this problem is the first intialize the two matrices with some values, calculate how "different" their product is to M, and then try to minimize this difference iteratively. Such a method is called gradient descent, aiming at finding a local minimum of the difference.

The difference here, usually called the error between the estimated rating and the real rating, can be calculated by the following equation for each user-item pair:

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

Here we consider the squared error because the estimated rating can be either higher or lower than the real rating.

To minimize the error, we have to know in which direction we have to modify the values of p_{ik} and q_{kj}. In other words, we need to know the gradient at the current values, and therefore we differentiate the above equation with respect to these two variables separately:

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij} q_{kj}$$
$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij} p_{ik}$$

Having obtained the gradient, we can now formulate the update rules for both p_{ik} and q_{kj}:

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$
$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij} p_{ik}$$

Here, α is a constant whose value determines the rate of approaching the minimum. Usually we will choose a small value for α, say 0.0002. This is because if we make too large a step towards the minimum we may run into the risk of missing the minimum and end up oscillating around the minimum.

A question might have come to your mind by now: if we find two matrices P and Q such that P×Q approximates R, isn't that our predictions of all the unseen ratings will all be zeros? In fact, we are not really trying to come up with P and Q such that we can reproduce R exactly. Instead, we will only try to minimise the errors of the observed user-item pairs. In other words, if we let T be a set of tuples, each of which is in the form of (u_i, d_j, r_{ij}), such that T contains all the observed user-item pairs together with the associated ratings, we are only trying to minimise every e_{ij} for (u_i, d_j, r_{ij}) ∈ T. (In other words, T is our set of training data.) As for the rest of the unknowns, we will be able to determine their values once the associations between the users, items and features have been learnt.

Using the above update rules, we can then iteratively perform the operation until the error converges to its minimum. We can check the overall error as calculated using the following equation and determine when we should stop the process.

$$E = \sum_{(u_i,d_j,r_{ij})\in T} e_{ij} = \sum_{(u_i,d_j,r_{ij})\in T} (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

The above algorithm is a very basic algorithm for factorizing a matrix. There are a lot of methods to make things look more complicated. A common extension to this basic algorithm is to introduce regularization to avoid overfitting. This is done by adding a parameter \beta and modify the squared error as follows:

$$e_{ij}^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2 + \frac{\beta}{2} \sum_{k=1}^{K} (||P||^2 + ||Q||^2)$$

In other words, the new parameter β is used to control the magnitudes of the user-feature and item-feature vectors such that P and Q would give a good approximation of R without having to contain large numbers. In practice, β is set to some values in the range of 0.02. The new update rules for this squared error can be obtained by a procedure similar to the one described above. The new update rules are as follows.

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + \alpha(2e_{ij} q_{kj} - \beta p_{ik})$$
$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + \alpha(2e_{ij} p_{ik} - \beta q_{kj})$$

C. Stochastic gradient descent

Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example x(i) and the label y(i):

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online. SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily.

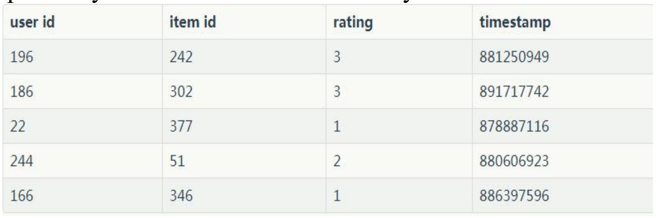
While batch gradient descent converges to the minimum of the basin the parameters are placed in, SGD's fluctuation, on the one hand, enables it to jump to new and potentially better local minima. On the other hand, this ultimately complicates convergence to the exact minimum, as SGD will keep overshooting. However, it has been shown that when we slowly decrease the learning rate, SGD shows the same convergence behaviour as batch gradient descent, almost certainly converging to a local or the global minimum for non-convex and convex optimization respectively.

III. Experiment

A. Dataset

(1) Utilizing MovieLens-100k dataset.

(2) u.data -- Consisting 10,000 comments from 943 users out of 1682 movies. At least, each user comment 20 videos. Users and movies are numbered consecutively from number 1 respectively. The data is sorted randomly



user id	item id	rating	timestamp
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596

(3) u1.base/u1.test are train set and validation set respectively, seperated from dataset u.data with proportion of 80% and 20%. It also make sense to train set and validation set from u1.base/ u1.test to u5.base / u5.test.

B. Implementation

Using stochastic gradient descent method(SGD):

1.Read the data set and divide it (or use u1.base / u1.test to u5.base / u5.test directly). Populate the original scoring matrix R $_{n\_users,\ n\_items}$ against the raw data, and fill 0 for null values.

2.Initialize the user factor matrix P $_{n\_users,\ K}$ and the item (movie) factor matrix Q $_{n\_items,\ K}$, where K is the number of potential features.

3.Determine the loss function and hyperparameter learning rate η and the penalty factor λ.

4.Use the stochastic gradient descent method to decompose the sparse user score matrix, get the user factor matrix and item (movie) factor matrix:

4.1.Select a sample from scoring matrix randomly;

4.2.Calculate this sample's loss gradient of specific row(column) of user factor matrix and item factor matrix;

4.3.Use SGD to update the specific row(column) of P $_{n\_users,\ K}$ and Q $_{n\_items,\ K}$;

4.4.Calculate the $L_{validation}$ on the validation set, comparing with the $L_{validation}$ of the previous iteration to determine if it has converged.

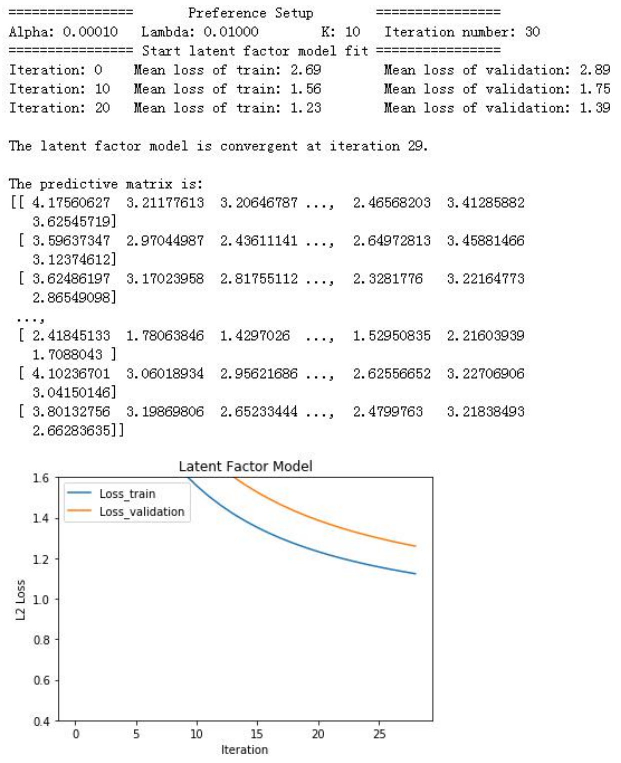
5.Repeat step 4. several times, get a satisfactory user factor matrix P and an item factor matrix Q, Draw a $L_{validation}$ curve with varying iterations.

6.The final score prediction matrix R $_{n\_users,\ n\_items}$ is obtained by multiplying the user factor matrix P $_{n\_users,\ K}$ and the transpose of the item factor matrix Q $_{n\_items,\ K}$.

When the penalty factor is 0.01, the experimental results of different learning rates are shown in Fig. 1, Fig. 2, Fig. 3. and Fig. 4.

When the learning rate is 0.01, the experimental results of different penalty factors are shown in Fig. 5, Fig. 6 and Fig. 7.

From this figures, we can see that the algorithm works better when the learning rate is 0.01 and penalty factor is 0.01.

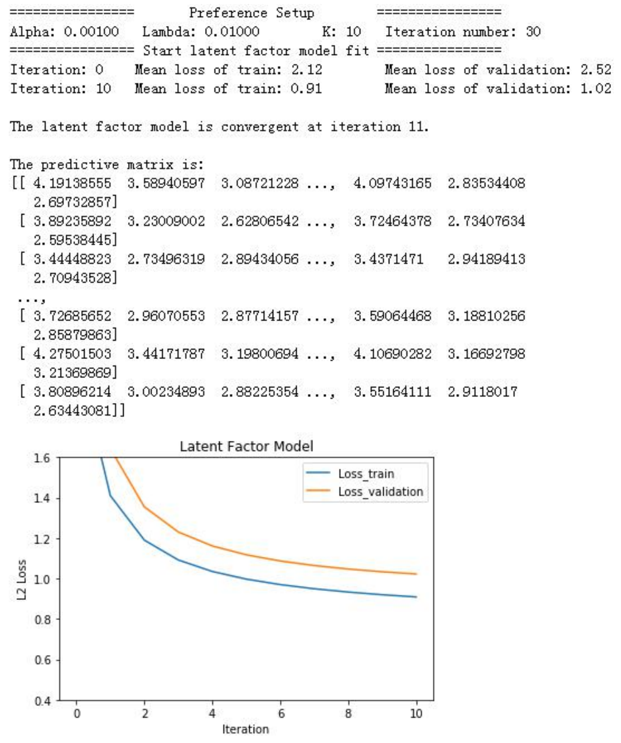


```
================        Preference Setup        ================
Alpha: 0.00010   Lambda: 0.01000       K: 10   Iteration number: 30
================ Start latent factor model fit ================
Iteration: 0    Mean loss of train: 2.69        Mean loss of validation: 2.89
Iteration: 10   Mean loss of train: 1.56        Mean loss of validation: 1.75
Iteration: 20   Mean loss of train: 1.23        Mean loss of validation: 1.39

The latent factor model is convergent at iteration 29.

The predictive matrix is:
[[ 4.17560627  3.21177613  3.20646787 ...,  2.46568203  3.41285882
   3.62545719]
 [ 3.59637347  2.97044987  2.43611141 ...,  2.64972813  3.45881466
   3.12374612]
 [ 3.62486197  3.17023958  2.81755112 ...,  2.3281776   3.22164773
   2.86549098]
 ...,
 [ 2.41845133  1.78063846  1.4297026  ...,  1.52950835  2.21603939
   1.7088043 ]
 [ 4.10236701  3.06018934  2.95621686 ...,  2.62556652  3.22706906
   3.04150146]
 [ 3.80132756  3.19869806  2.65233444 ...,  2.4799763   3.21838493
   2.66283635]]
```

Fig. 1. the result of learning rate is 0.0001 and penalty factor is 0.01



```
================        Preference Setup        ================
Alpha: 0.00100   Lambda: 0.01000       K: 10   Iteration number: 30
================ Start latent factor model fit ================
Iteration: 0    Mean loss of train: 2.12        Mean loss of validation: 2.52
Iteration: 10   Mean loss of train: 0.91        Mean loss of validation: 1.02

The latent factor model is convergent at iteration 11.

The predictive matrix is:
[[ 4.19138555  3.58940597  3.08721228 ...,  4.09743165  2.83534408
   2.69732857]
 [ 3.89235892  3.23009002  2.62806542 ...,  3.72464378  2.73407634
   2.59538445]
 [ 3.44448823  2.73496319  2.89434056 ...,  3.4371471   2.94189413
   2.70943528]
 ...,
 [ 3.72685652  2.96070553  2.87714157 ...,  3.59064468  3.18810256
   2.85879863]
 [ 4.27501503  3.44171787  3.19800694 ...,  4.10690282  3.16692798
   3.21369869]
 [ 3.80896214  3.00234893  2.88225354 ...,  3.55164111  2.9118017
   2.63443081]]
```

Fig. 2. the result of learning rate is 0.001 and penalty factor is 0.01

```
================        Preference Setup        ================
Alpha: 0.0100    Lambda: 0.0100  K: 10   Iteration number: 30
================ Start latent factor model fit ================
Iteration: 0    Mean loss of train: 1.18        Mean loss of validation: 1.48
Iteration: 10   Mean loss of train: 0.75        Mean loss of validation: 0.96

The latent factor model is convergent at iteration 18.

The predictive matrix is:
[[ 4.0492511   3.74599149  3.51297543 ...,  2.47381711  2.60602546
   2.83571496]
 [ 3.68415561  2.86928032  2.90803359 ...,  3.23108724  2.96913144
   3.23691106]
 [ 3.40943382  2.8932639   2.55696745 ...,  2.76777825  2.4310867
   2.99577626]
 ...,
 [ 4.31576054  3.72034593  3.72385085 ...,  3.445134    3.42037737
   3.0987499 ]
 [ 4.52200397  3.9747209   3.53186152 ...,  3.42267828  3.45761095
   3.28789593]
 [ 3.60475997  3.43081158  2.58525509 ...,  2.22680502  2.65388394
   3.14932803]]
```

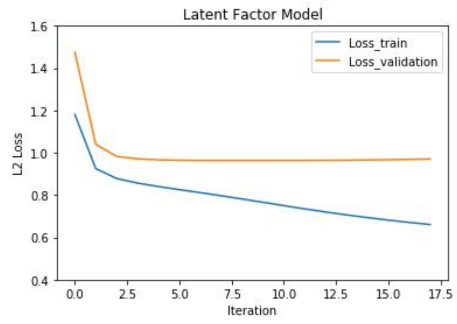


Fig. 3. the result of learning rate is 0.01 and penalty factor is 0.01

```
================        Preference Setup        ================
Alpha: 0.01000   Lambda: 0.00010        K: 10   Iteration number: 30
================ Start latent factor model fit ================
Iteration: 0    Mean loss of train: 1.18        Mean loss of validation: 1.44
Iteration: 10   Mean loss of train: 0.76        Mean loss of validation: 0.97

The latent factor model is convergent at iteration 18.

The predictive matrix is:
[[ 3.61262228  3.4818417   3.00342572 ...,  2.23139304  3.09857475
   3.78226348]
 [ 3.3682725   3.39681407  3.61118325 ...,  2.30316982  2.98049741
   3.33803349]
 [ 3.54053298  3.11460026  2.7589201  ...,  2.31644825  3.1554977
   2.67608867]
 ...,
 [ 4.76483461  3.25014033  3.4206269  ...,  2.60015669  3.64555071
   3.7045786 ]
 [ 4.80484414  4.66320712  3.53918781 ...,  2.85804908  4.64714963
   3.39345678]
 [ 3.90565146  3.54731433  3.32301793 ...,  1.63832328  3.20687911
   3.4727799 ]]
```

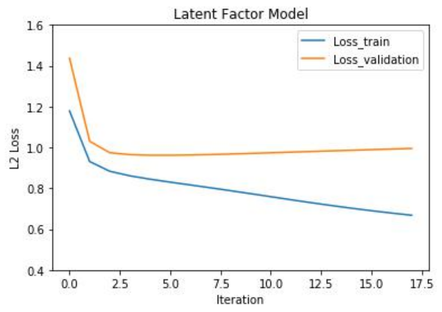


Fig. 5. the result of learning rate is 0.01 and penalty factor is 0.0001

```
================        Preference Setup        ================
Alpha: 0.10000   Lambda: 0.01000        K: 10   Iteration number: 30
================ Start latent factor model fit ================
Iteration: 0    Mean loss of train: 1.36        Mean loss of validation: 1.46

The latent factor model is convergent at iteration 6.

The predictive matrix is:
[[ 5.25825638  4.56214368  5.45918508 ...,  3.93417329  4.38896078
   4.06834167]
 [ 4.07009061  3.65637776  1.30166347 ...,  2.73824024  4.16475386
   3.36118032]
 [ 4.27909026  3.17132512  3.18382242 ...,  3.04628212  3.41616178
   3.3086557 ]
 ...,
 [ 4.87263337  4.71777653  6.69940485 ...,  3.98097773  3.72147096
   4.31041536]
 [ 4.80946796  4.94881107  3.89537781 ...,  3.60553722  4.2936939
   3.81164289]
 [ 4.805991    4.35198595  2.77255969 ...,  2.82061047  4.10654427
   3.59607746]]
```

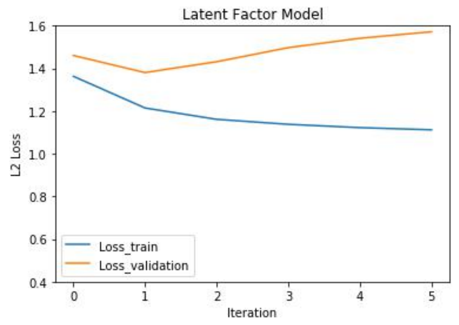


Fig. 4. the result of learning rate is 0.1 and penalty factor is 0.01

```
================        Preference Setup        ================
Alpha: 0.01000   Lambda: 0.05000        K: 10   Iteration number: 30
================ Start latent factor model fit ================
Iteration: 0    Mean loss of train: 1.17        Mean loss of validation: 1.42

The latent factor model is convergent at iteration 6.

The predictive matrix is:
[[ 3.78768108  3.59485326  3.1193614  ...,  3.85662527  3.27119135
   2.51407271]
 [ 3.94784797  3.24480211  3.23524184 ...,  3.3873075   3.4911625
   2.42160316]
 [ 3.55440941  3.12261369  2.73119027 ...,  3.33135743  3.54381767
   2.85854891]
 ...,
 [ 3.95276452  4.04941617  3.50002477 ...,  4.12456125  3.58671249
   2.74311207]
 [ 4.21829962  3.73166927  3.47819443 ...,  3.92141332  3.37750494
   2.65735313]
 [ 3.62621083  3.24959041  2.8642104  ...,  3.55180685  3.35274026
   2.64009781]]
```

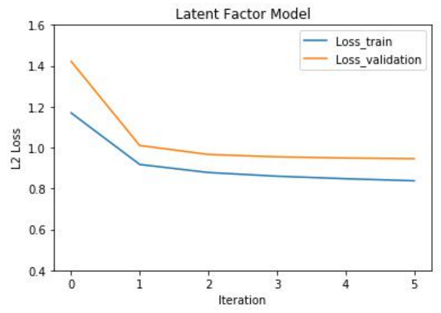
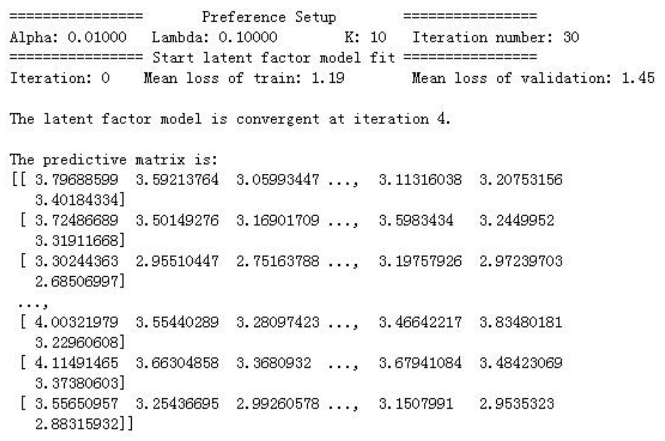


Fig. 6. the result of learning rate is 0.01 and penalty factor is 0.05



```
================      Preference Setup       ================
Alpha: 0.01000   Lambda: 0.10000       K: 10   Iteration number: 30
================ Start latent factor model fit ================
Iteration: 0    Mean loss of train: 1.19       Mean loss of validation: 1.45

The latent factor model is convergent at iteration 4.

The predictive matrix is:
[[ 3.79688599  3.59213764  3.05993447 ...,  3.11316038  3.20753156
   3.40184334]
 [ 3.72486689  3.50149276  3.16901709 ...,  3.5983434   3.2449952
   3.31911668]
 [ 3.30244363  2.95510447  2.75163788 ...,  3.19757926  2.97239703
   2.68506997]
 ...,
 [ 4.00321979  3.55440289  3.28097423 ...,  3.46642217  3.83480181
   3.22960608]
 [ 4.11491465  3.66304858  3.3680932  ...,  3.67941084  3.48423069
   3.37380603]
 [ 3.55650957  3.25436695  2.99260578 ...,  3.1507991   2.9535323
   2.88315932]]
```

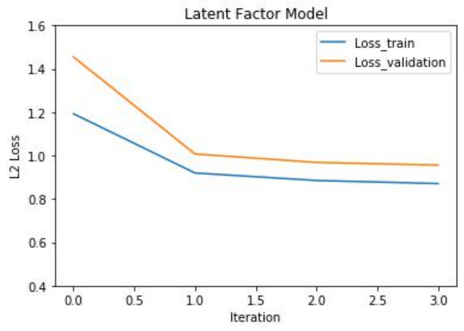


Fig. 7. the result of learning rate is 0.01 and penalty factor is 0.1

IV. CONCLUSION

This experiment of machine learning is about recommender system based on matrix decomposition, and the experiment utilized MovieLens-100k dataset.

After this experiment, we have further understood of the construction of recommended system, and understood the principle of matrix decomposition, and be familiar to the use of gradient descent, and learned to construct a recommendation system under small-scale dataset. Moreover, the experiment cultivated our engineering ability. Also, we all realized the process of optimization and adjusting parameters is very important and crucial.

In addition, This is a team experiment, and this team experiment is good for exercising our teamwork skills. This is a interesting and meaningful experience.