Recommender system

~ Implement movie recommender system by matrix factorization ~

Professor: Sang-Wook Kim
Department of Computer Science
2012003716 Kwangil Cho
2017. 06. 13

0. Introduction

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 have become increasingly popular in recent years, and are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general. There are also recommender systems for experts, collaborators, jokes, restaurants, garments, financial services, life insurance, romantic partners (online dating), and Twitter pages.

1. Objectives and Environment

The goal of this project is to create a recommendation system based on the training data and to guess the new incoming data and to give the closest movie rating. There are many methods to implement the recommendation system. I applied the model based technique, the matrix factorization.

I have been working on the following environment:

- Operating System: Ubuntu 14.05 LTS 64-bit
- Compilation: g++ compiler
- Language: C/C++ with C++ standard 11
- Source code editor: Vim
- Source code version control: Git
- The movie rating data source: movielens(http://grouplens.org/datasets/movielens/)
- Followed HYU coding convention faithfully

2. Program Structures

(0) The program expects 3 arguments (executable file, training data file, test data file) for executing recommender system. When the arguments are successfully ready, input handling routine works. First, input handler sets paths of input files. Training data file path, test data file path, root directory path (based on training data) and prediction data(output data) file path(based on training data) are set here.

(1) When the paths of input and output files are ready, the first thing to do is to get number of user and items. handler reads number information from .info file and initialize the NUM_USER and NUM_ITEM constants. After that, get the rating data and initialize the rating matrix with input rating data. The rating matrix is NUM_USER * NUM_ITEM size, and usually it is very sparse. I didn't use the timestamp information because the matrix factorization uses rating data only.

(2) After the rating matrix has been initialized, matrix factorization should work.
Here are the steps of matrix factorization:

A. The matrix R can be divided into multiplication of P and Q where the P is NUM_USER * K(here K is latent value) matrix and Q is K * NUM_ITEM matrix.

B. Initialize all elements of P and Q with random values in range 0.00001 ~ 5.00000. If the value range is too big, the error fixing(future job) can be hard to be converged. This job is not identical because every time the program executes, the elements are initialized with different random values.

C. For all elements, evaluate the difference between original element and new element which is made by multiplication of P[i][k] * Q[k][j].

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

D. Adjust the elements in P and Q to reduce the difference. parameter alpha determines the rate of approaching the minimum. Usually alpha takes a small value like 0.0002. This is because if it is too large a step towards the minimum may run into the risk of missing the minimum and end up oscillating around the minimum.

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

E. The above algorithm is a very basic algorithm for factorizing matrix. 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:

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

F. However, when I evaluated the effectiveness of beta by experiment, the parameter wasn't that critical or helpful, so I just used the very basic one although I implemented the parameter beta to avoid overfitting.

G. Repeat the above algorithm until the difference(error) becomes satisfactory or the limit iterations. Here, I limited the iteration 5000 times.

H. Multiply P and Q which are well-tempered now. Get new R by P * Q. The result matrix becomes the estimation matrix which determines future rating.

(3) If the rating matrix has been re-generated well, then the predict all test data from test file. Make a rating by referencing the rating matrix and print it into output file.

(4) Evaluate the accuracy with output file and test file by RMSE(root mean square error).

3. Execution

```
ki@ubuntu:~/DataMining/assignment4$ make
g++ -g -Wall -std=c++11 -I./include/ -o ./bin/recommender src/constant.cc src/matrixfact.cc src
/rate.cc src/recommender.cc  -L./lib/ -lpthread
ki@ubuntu:~/DataMining/assignment4$ bash run.sh
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Press [Enter] key to start u1 test...
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

(1) Build and execute the program with prepared test script.

```
run.sh
#!/bin/bash

DATASET=(
u1
u2
u3
u4
u5
)

for data in ${DATASET[@]}; do
    echo -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
    echo "Press [Enter] key to start ${data} test..."
    echo -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
    read
    ./bin/recommender ./data/${data}.base ./data/${data}.test
done

```

(2) Test script runs 5 tests iteratively with the prepared training data and test data.

```
ki@ubuntu:~/DataMining/assignment4/data$ ls
allbut.pl               u2.base                 u4.base                 ua.base  u.info
mku.sh                  u2.base_prediction.txt  u4.base_prediction.txt  ua.test  u.item
README                  u2.test                 u4.test                 ub.base  u.occupation
u1.base                 u3.base                 u5.base                 ub.test  u.user
u1.base_prediction.txt  u3.base_prediction.txt  u5.base_prediction.txt  u.data
u1.test                 u3.test                 u5.test                 u.genre
ki@ubuntu:~/DataMining/assignment4/data$
```

(3) There are 3 types of data in data directory. the training data, test data, and evaluated(predicted) data. If the program completes, the output file (postfix with '.base_prediction.txt') has been generated.

```
C:₩Users₩KwangilCho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제4₩test>PA4 u1
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 0.8335766

C:₩Users₩KwangilCho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제4₩test>PA4 u2
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 0.832256

C:₩Users₩KwangilCho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제4₩test>PA4 u3
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 0.8300602

C:₩Users₩KwangilCho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제4₩test>PA4 u4
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 0.8311438

C:₩Users₩KwangilCho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제4₩test>PA4 u5
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 0.8345059

C:₩Users₩KwangilCho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제4₩test>
```

(4) Test with TA's testing program with 5 test files. The program successfully executes the tests with **0.83 average RMSE**.

4. Analysis

In addition to matrix factorization, several experiments were run with a naive approach for comparison.

```
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 32
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 0.9701319

C:₩Users₩Kwangil Cho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제4₩test>PA4.exe u1
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 0.9704896

C:₩Users₩Kwangil Cho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제4₩test>
```

(1) Rating by movie's previous rating average: **0.970**

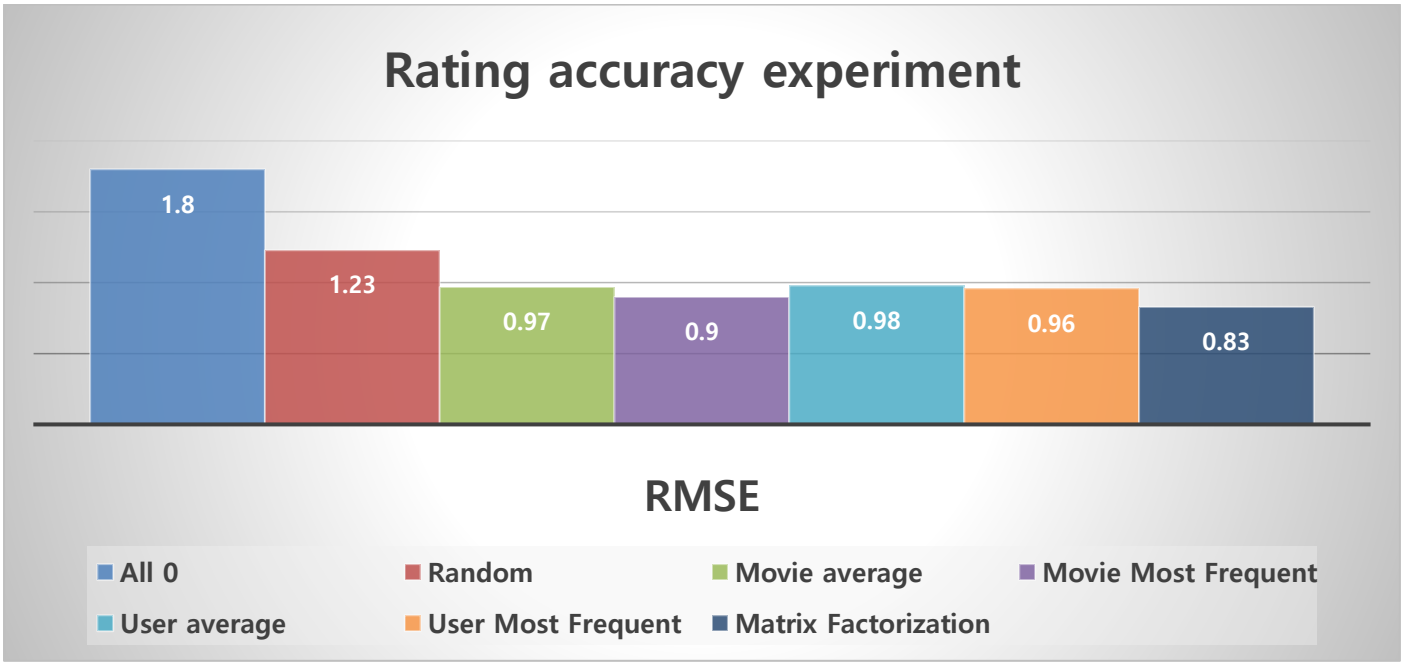
```
C:₩Users₩Kwangil Cho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제4₩test>PA4.exe u1
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 0.9093954

C:₩Users₩Kwangil Cho₩OneDrive - 한양대학교₩1. 수업₩1. 데이터마이닝₩과제4₩test>
```

(2) Rating by movie's the most previous rating: **0.909**

Like these, I experimented with adjusting various variables. The graph chart below is result.



Through the experiment, the matrix factorization recommender system which is the model based shows the highest accuracy.

5. Epilogue

Through this project, I was able to design and implement the recommender system with the matrix

factorization model. It was very interesting job to evaluate the accuracy of my recommender system and I was trying to make it better. Through the project, there was no comparison data so I had to implement all the various approaches for comparison data. If there was more time to do the project, I would have tried the content-based approach and trust map approach. Thanks to professor and assistants who prepared for the class and practices.

6. Reference

- Wikipedia: Recommender System (https://en.wikipedia.org/wiki/Recommender_system)
- Matrix Factorization simple tutorial