Mahidol University

ITCS 208 – Object Oriented Programming

Project 03

Project 3: NETFLICT

A simple movie recommendation system

Due: Friday April 27, 2018 11:59PM

Learning Objectives:

On the course of implementing this programming project, you will learn some of the basic concepts of object oriented programming and how to apply them to practical, real world programming applications. Specifically, upon accomplishing this project, we expect you to be able to:

1. Instantiate objects and use them.
2. Implement classes given from interfaces.
3. Learn how and when to use Set and Map data structures.
4. Read/Write simple text files.
5. Learn how to use regular expressions and simple string manipulation techniques.
6. Translate mathematical models and ideas into code.
7. Generate runnable/portable jar files.
8. Get a glimpse of how OOP can be applied to real-world problems.
9. Enjoy coding with Java.

Introduction:

Have you ever wondered how Netflix is able to guess what exactly you want to watch, or how Facebook manages to pull up your friends' posts that you have been eager to read? Oftentimes, these mind guessing tasks are translated into a recommendation problem in which items are ranked based on their relevance scores and the top items are suggested to the user. In this project, you will be implementing one of the most widely used social recommendation paradigms, called ***user-user collaborative filtering*** (well, a simple version of it). The intuition behind the algorithm is that users who have the same "taste" are likely to like to same set of things. With such an assumption, a recommendation system (a.k.a. recommender) could make a recommendation to a user by returning the list of items that other

"similar" users have expressed their liking on. For example, you would see that Facebook often displays the posts that your friends "like" on your feed. In this project, you will be implementing a version of user-user collaborative filtering algorithm for recommending movies to users.

Technical Definitions:

Movie: A movie is a tuple of (`mid`, `title`, `year`, `tags`). `mid` is the ID of the movie, always a positive integer. `title` is the String title of the movie. `year` is a positive integer representing the year in which this movie was released. `tags` is a set of String relevant tags that describe this movie.

User: A user is a tuple of (`uid`, `ratings`). `uid` is the ID of the user, always a positive integer. `ratings` is the set of ratings that this user has given. Ratings are stored in a `Map<mid, Rating>` data structure for fast lookup.

Rating: A rating is a tuple of (`uid`, `mid`, `score`, `timestamp`). `uid` and `mid` are the IDs of the rating user and the rated movie, respectively. `score` is a double numeric value. If given, a rating score can range from [0.5,5] inclusive. A rating score of 0 or negative values implies that no rating information of the user *uid* and the movie *mid* is available (the user may never have rated this movie or the rating is made unavailable). `timestamp` is a long value indicating the time at which this rating was given. What is a timestamp?[1]

User-User Collaborative Filtering for Movie Recommendation

User-user collaborative filtering is a straightforward algorithmic interpretation of the core principles of collaborative filtering. Intuitively, it finds other users whose past rating behavior is similar to a given user, and uses their past ratings to predict the rating of a movie that the current user has not rated. If a movie is predicted to have high rating from the user (i.e. 4-5 stars) then it is likely that this user will like this movie.[2]

Computing the rating prediction $p_{u,i}$:

N	: the set of all users.
$u \in N$	: the target user
$N_i \subseteq N - u$	: the set of all users who have rated the movie *i*.
I	: the set of all movies
$i \in I$	: target movie (whose rating is to be predicted)

In order to predict the rating score for i given by the user u, use the following formula:

$$p_{u,i} = \bar{r}_u + \frac{\sum_{u' \in N_i} \{s(u,u') \cdot (r_{u',i} - \bar{r}_{u'})\}}{\sum_{u' \in N_i} |s(u,u')|}$$

If the denominator is 0 or N_i is empty, then $p_{u,i} = \bar{r}_u$.

$p_{u,i} \in [0,5]$ is the predicted rating of the movie i given by the user u.

[1] See: https://en.wikipedia.org/wiki/Timestamp

[2] Knowledge-hungry students may read further on these movie recommendation techniques from (Schafer, 2007).

$\bar{r}_u$ and $\bar{r}_{u'}$ are the average rating score that the users u and u' have given, respectively. For example, if the user *u* rated three movies in the past with rating scores of 1.5, 0.5, and 4 stars, then $\bar{r}_u = 2.0$.

$r_{u',i}$ is the rating score that the user *u'* gave to the movie *i*.

$s(u, u')$ is the similarity score between the users *u* and *u'*.

Note that $|x|$ denotes the absolute value of *x*.

Computing the user similarity $s(u, v)$:

For a pair of users $u \in N$ and $v \in N$, their similarity is determined by the correlation of their past common ratings. Let I_u and I_v be the sets of movies that *u* and *v* have rated, respectively. $s(u, v)$ is defined as:

$$s(u,v) = \frac{\sum_{i \in I_u \cap I_v}\left[(r_{u,i} - \bar{r}_u) \cdot (r_{v,i} - \bar{r}_v)\right]}{\sqrt{\sum_{i \in I_u \cap I_v}(r_{u,i} - \bar{r}_u)^2}\ \sqrt{\sum_{i \in I_u \cap I_v}(r_{v,i} - \bar{r}_v)^2}}$$

$s(u, u) \in [-1,1]$. $s(u, u) = 1.0$. If the denominator is 0, then $s(u, v) = 0$.

Recommending movies:

To recommend *K* movies produced during `fromYear` to `toYear` to the user *u*, the recommender simply performs the following steps:

STEP 1: Collect all the movies released during `fromYear` to `toYear` → I^*
STEP 2: For each movie $i \in I^*$, compute $p_{u,i}$.
STEP 3: Rank the movies based on their $p_{u,i}$ scores.
STEP 4: If $|I^*| > K$, return the top *K* movies. Otherwise, return all the movies.

Movie file format:

A movie file (e.g. movies.csv) stores the meta-information about all the movies. Except for the first line which is the table header, each line in the movie file has one of the following formats:

```
<mid>,<title> (<year>),<tag_1>|<tag_2>|<tag_3>|...|<tag_n>
//title does not contain ‘,’

<mid>,"<title> (<year>)",<tag_1>|<tag_2>|<tag_3>|...|<tag_n>
//title contains ‘,’
```

For example,

```
movieId,title,genres
22,Copycat (1995),Crime|Drama|Horror|Mystery|Thriller
23,Assassins (1995),Action|Crime|Thriller
24,Powder (1995),Drama|Sci-Fi
54,"Big Green, The (1995)",Children|Comedy
25,Leaving Las Vegas (1995),Drama|Romance
```

Your recommender needs to parse this file to load the movies into the memory. The movies are stored in a Map structure where they can be looked up using their `mid`'s.

Hint: You may find regex and String.split() useful to parse this file.

User file format:

A user file (e.g. users.train.csv or users.test.csv) lists individual ratings. Rating generated by the same user can be grouped together, and stored in a Map structure of the same User object where they can be looked up using the corresponding `mid`'s. Except for the first line which contains the table header, each line in the user file has the following format:

`<uid>,<mid>,<rating>,<timestamp>`

For example,

```
userId,movieId,rating,timestamp
6,94864,4.5,1348693600
6,78039,3.0,1348693590
6,76251,3.0,1348693953
```

The recommender needs to parse the ***training user file*** to train the model, and the ***testing user file*** to test the model.

Hint: You may find the method String.split() useful when parsing a user file.

Training the recommender system:

The user-user collaborative filtering algorithm above is a learning process, and hence must be trained before used. In the training process, the recommender first loads the movies and ***the training data*** stored in the training user file (i.e. users.train.csv). Then, $s(u, v)$ is pre-computed for each pair of users and stored in a model file. These computations can be expensive, but typically needed to be computed only once. Additionally, the model file stores auxiliary information for fast lookup such as the number of users in the training data, the number of movies, and the rating matrix. A model file has ".model" extension and has the following format.

```
@NUM_USERS <num_users>
@USER_MAP {0=<uid1>, 1=<uid2>, 2=<uid3>, ...}
@NUM_MOVIES <num_movies>
@MOVIE_MAP {0=<mid1>, 2=<mid2>, ...}
@RATING_MATRIX
[R]
@USERSIM_MATRIX
[S]
```

`num_users`/`num_movies` is the number of all users/movies used to build the model, respectively.
`@USER_MAP` maps the matrix indices (starts from 0, 1, ..., `num_users-1`) to the actual `uid`'s.
`@USER_MAP` maps the matrix indices (starts from 0, 1, ..., `num_movies-1`) to the actual `mid`'s.

@RATING_MATRIX defines the rating matrix [R], which is a num_usersX(num_movies+1) matrix. Each element R(i,j) is the rating score that the user indexed by *i* gave to the movie indexed by *j*. The last field R(i,num_movies) stores the average rating of the user index *i*, which is the average of the rating scores he/she has given. Since the lowest rating is 0.5, if R(i,j) = 0, it means that the rating information of the user indexed by *i* and the movie indexed by *j* is not provided.

@USERSIM_MATRIX defines the user similarity matrix [S], which is a num_usersXnum_users matrix. Each element S(i,j) denotes the similarity score between the user indexed by *i* and the user indexed by *j*. Note that since S(i,j) = S(j,i), [S] is a symmetric matrix.

For example (model file generated from the "micro" test case. See also micro.simple.model.),

```
@NUM_USERS 3
@USER_MAP {0=1, 1=21, 2=475}
@NUM_MOVIES 19
@MOVIE_MAP {0=16, 1=19, 2=24, 3=32, 4=3018, 5=3030, 6=3033, 7=99992, 8=99996, 9=100108,
10=100326, 11=101864, 12=106489, 13=107406, 14=108188, 15=108190, 16=108729, 17=109487,
18=111360}
@RATING_MATRIX
4.0 0.0 1.5 4.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 3.1666666666666665
0.0 3.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 3.5
0.0 0.0 0.0 0.0 2.5 4.0 3.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 3.3333333333333335
@USERSIM_MATRIX
1.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
```

Testing the recommender system:

The ***test user file*** (e.g. users.test.csv) contains a small subset ground truth rating information, called the test data. This test data is disjoint from the training data and is used to determine how well the recommendation system performs. Specifically, the test data contains the ratings of the movies released during 2010 – 2015 by some users. The efficacy of the recommendation system is evaluated in two aspects: rating prediction and movie recommendation.

Evaluation of rating prediction determines how well the system predicts the rating score of a movie that a target user would give. The system predicts the rating for each (user, movie) pair in the test data, and compares the predicted rating with the actual one. Metrics used to evaluate the prediction include Root Mean Square Error (RMSE) and Correlation (CORR).

Evaluation for the recommended set of movies is done using the traditional information retrieval evaluation protocol. The evaluation is done on the top 20 recommended items using standard precision@20, recall@20, and F1@20.

Note: You are not required to implement the evaluation protocol. This functionality is implemented for you.

Instructions:

In the project package you will find the following files:

BaseMovieRecommender.java

Evaluator.java

MainController.java

Movie.java

MovieItem.java

Rating.java

User.java

lib/commons-io-2.4.jar
lib/commons-lang3-3.4.jar
lib/commons-math3-3.6.jar
lib/guava-19.0.jar

You must not modify the above java files. Your task is simply to implement class SimpleMovieRecommender in SimpleMovieRecommender.java that ***implements*** the interface BaseMovieRecommender**.** You will find the description of each method in BaseMovieRecommender.java. You may use any data structures to store your information.

You may use the functionalities in *commons-io-2.4.jar*[3], *commons-lang3-3.4.jar*[4], *commons-math3-3.6.jar*[5], and *guava-19.0.jar*[6]. You may consult the Internet and external sources for usages of these libraries. Specifically, you may find the IO library useful when dealing with files and directories, the Lang library for String manipulation and other enhancement on the base Java core classes, the Math library for statistical computation, and the Guava library for set manipulation and operations. You are not allowed to use other third-party Java libraries, besides the provided ones.

Test cases:

Two test cases are provided for you to debug your code: `micro` and `small`. To test your code, produce a runnable ***jar*** file **MainController.jar** from MainController class.

Open a Command Prompt (Windows), Terminal (Linux/Mac OS), and "cd" to the directory where MainController.jar and test cases are located.

1. Test parsing movie and user files.

```
java -jar MainController.jar parse <movie_fname> <user_fname>
```

For example,

```
>java -jar MainController.jar parse small/movies.csv small/users.train.csv
Test pasrsing data.

10311 movies loaded. The first movie is
[mid: 1:Toy Story (1995) [Adventure, Fantasy, Animation, Comedy, Children]]
668 users loaded. The first user is
[user: 1 rates 113 movies]
        [mid: 256 rating: 0.5/5 timestamp: 1217895764]
        [mid: 8961 rating: 4.0/5 timestamp: 1217896012]
        [mid: 5378 rating: 3.5/5 timestamp: 1217897078]
        [mid: 33794 rating: 4.5/5 timestamp: 1217896007]
        [mid: 260 rating: 4.5/5 timestamp: 1217895864]
        [mid: 261 rating: 1.5/5 timestamp: 1217895750]
        [mid: 1287 rating: 4.5/5 timestamp: 1217897783]
        [mid: 2571 rating: 4.5/5 timestamp: 1217895908]
        [mid: 780 rating: 3.5/5 timestamp: 1217895969]
        [mid: 54286 rating: 4.0/5 timestamp: 1217896447]
        [mid: 527 rating: 4.5/5 timestamp: 1217896341]
        [mid: 16 rating: 4.0/5 timestamp: 1217897793]
        [mid: 277 rating: 0.5/5 timestamp: 1217895772]
        [mid: 24 rating: 1.5/5 timestamp: 1217895807]
        [mid: 32 rating: 4.0/5 timestamp: 1217896246]
        [mid: 1061 rating: 4.0/5 timestamp: 1217895811]
        [mid: 296 rating: 4.0/5 timestamp: 1217896125]
```

[3] https://commons.apache.org/proper/commons-io/
[4] https://commons.apache.org/proper/commons-lang/
[5] https://commons.apache.org/proper/commons-math/
[6] https://github.com/google/guava

```
        [mid: 2858 rating: 4.0/5 timestamp: 1217896098]
        [mid: 5418 rating: 4.0/5 timestamp: 1217896980]
        [mid: 33834 rating: 4.5/5 timestamp: 1217897409]
        [mid: 1580 rating: 3.5/5 timestamp: 1217896279]
        [mid: 47 rating: 4.0/5 timestamp: 1217896556]
        [mid: 50 rating: 4.0/5 timestamp: 1217896523]
        [mid: 6711 rating: 3.5/5 timestamp: 1217896498]
        [mid: 2105 rating: 1.5/5 timestamp: 1217895783]
        [mid: 318 rating: 4.0/5 timestamp: 1217895860]
        [mid: 5952 rating: 4.5/5 timestamp: 1217896346]
        [mid: 1089 rating: 4.5/5 timestamp: 1217897621]
        [mid: 2628 rating: 3.0/5 timestamp: 1217896237]
        [mid: 5445 rating: 3.5/5 timestamp: 1217895979]
        [mid: 32587 rating: 4.5/5 timestamp: 1217896582]
        [mid: 589 rating: 3.5/5 timestamp: 1217896078]
        [mid: 590 rating: 3.5/5 timestamp: 1217896038]
        [mid: 592 rating: 2.5/5 timestamp: 1217896043]
        [mid: 593 rating: 5.0/5 timestamp: 1217895932]
        [mid: 1617 rating: 3.5/5 timestamp: 1217896636]
        [mid: 597 rating: 3.0/5 timestamp: 1217897176]
        [mid: 858 rating: 5.0/5 timestamp: 1217896428]
        [mid: 2396 rating: 4.0/5 timestamp: 1217897010]
        [mid: 349 rating: 4.5/5 timestamp: 1217897058]
        [mid: 3421 rating: 4.5/5 timestamp: 1217895728]
        [mid: 57949 rating: 0.5/5 timestamp: 1217896004]
        [mid: 608 rating: 3.5/5 timestamp: 1217896319]
        [mid: 4963 rating: 3.5/5 timestamp: 1217896313]
        [mid: 356 rating: 3.0/5 timestamp: 1217896231]
        [mid: 2407 rating: 2.5/5 timestamp: 1217895736]
        [mid: 110 rating: 4.0/5 timestamp: 1217896150]
        [mid: 1136 rating: 5.0/5 timestamp: 1217897630]
        [mid: 2161 rating: 1.5/5 timestamp: 1217895776]
        [mid: 49272 rating: 3.5/5 timestamp: 1217896576]
        [mid: 377 rating: 2.5/5 timestamp: 1217896373]
        [mid: 8825 rating: 2.5/5 timestamp: 1217896490]
        [mid: 380 rating: 3.0/5 timestamp: 1217896030]
        [mid: 45950 rating: 0.5/5 timestamp: 1217897813]
        [mid: 4993 rating: 4.5/5 timestamp: 1217895872]
        [mid: 1923 rating: 1.5/5 timestamp: 1217897087]
        [mid: 2947 rating: 3.5/5 timestamp: 1217895742]
        [mid: 4995 rating: 4.0/5 timestamp: 1217896614]
        [mid: 48516 rating: 5.0/5 timestamp: 1217896206]
        [mid: 648 rating: 3.5/5 timestamp: 1217896397]
        [mid: 48780 rating: 4.0/5 timestamp: 1217897632]
        [mid: 2959 rating: 5.0/5 timestamp: 1217896334]
        [mid: 912 rating: 5.0/5 timestamp: 1217897623]
        [mid: 2194 rating: 4.5/5 timestamp: 1217897671]
        [mid: 150 rating: 3.0/5 timestamp: 1217895940]
        [mid: 2455 rating: 2.0/5 timestamp: 1217895762]
        [mid: 6807 rating: 4.0/5 timestamp: 1217897777]
        [mid: 2716 rating: 3.5/5 timestamp: 1217896417]
        [mid: 161 rating: 4.0/5 timestamp: 1217897864]
        [mid: 2467 rating: 4.5/5 timestamp: 1217897780]
        [mid: 165 rating: 3.0/5 timestamp: 1217897135]
        [mid: 4262 rating: 5.0/5 timestamp: 1217897697]
        [mid: 2728 rating: 5.0/5 timestamp: 1217897681]
        [mid: 1961 rating: 3.0/5 timestamp: 1217897157]
        [mid: 4011 rating: 4.5/5 timestamp: 1217897657]
        [mid: 1196 rating: 4.5/5 timestamp: 1217896095]
        [mid: 1198 rating: 4.0/5 timestamp: 1217896450]
        [mid: 3256 rating: 5.0/5 timestamp: 1217895803]
        [mid: 1721 rating: 1.5/5 timestamp: 1217896210]
        [mid: 1210 rating: 4.5/5 timestamp: 1217895912]
        [mid: 4027 rating: 3.5/5 timestamp: 1217897174]
        [mid: 1213 rating: 5.0/5 timestamp: 1217897617]
        [mid: 4033 rating: 3.5/5 timestamp: 1217897841]
        [mid: 1220 rating: 4.0/5 timestamp: 1217897823]
        [mid: 1221 rating: 5.0/5 timestamp: 1217897613]
        [mid: 1222 rating: 5.0/5 timestamp: 1217897625]
        [mid: 2502 rating: 4.0/5 timestamp: 1217897700]
        [mid: 968 rating: 4.5/5 timestamp: 1217897263]
        [mid: 457 rating: 4.0/5 timestamp: 1217896015]
        [mid: 2762 rating: 3.0/5 timestamp: 1217896143]
        [mid: 204 rating: 0.5/5 timestamp: 1217895786]
        [mid: 719 rating: 0.5/5 timestamp: 1217895799]
        [mid: 1233 rating: 4.5/5 timestamp: 1217897648]
        [mid: 4306 rating: 4.0/5 timestamp: 1217895903]
        [mid: 724 rating: 3.5/5 timestamp: 1217895794]
        [mid: 33493 rating: 4.5/5 timestamp: 1217896473]
        [mid: 1243 rating: 3.5/5 timestamp: 1217897427]
        [mid: 6365 rating: 4.5/5 timestamp: 1217896386]
        [mid: 223 rating: 4.0/5 timestamp: 1217897795]
        [mid: 480 rating: 3.5/5 timestamp: 1217895972]
        [mid: 736 rating: 3.0/5 timestamp: 1217896621]
        [mid: 2021 rating: 4.0/5 timestamp: 1217895796]
        [mid: 5349 rating: 3.0/5 timestamp: 1217896286]
        [mid: 1258 rating: 4.5/5 timestamp: 1217897678]
        [mid: 2028 rating: 4.5/5 timestamp: 1217896291]
        [mid: 52973 rating: 3.5/5 timestamp: 1217897820]
        [mid: 2542 rating: 4.5/5 timestamp: 1217897667]
        [mid: 1265 rating: 3.0/5 timestamp: 1217897107]
        [mid: 7153 rating: 4.5/5 timestamp: 1217896065]
        [mid: 1267 rating: 4.0/5 timestamp: 1217895768]
        [mid: 4085 rating: 3.5/5 timestamp: 1217897858]
        [mid: 1270 rating: 3.0/5 timestamp: 1217897006]
        [mid: 3578 rating: 4.0/5 timestamp: 1217896402]

Time Used: 00:00:00.602
```

2. To train the recommender, use command

```
java -jar MainController.jar train <movie_fname> <train_user_fname> <model_fname>
```

For example,

```
>java -jar MainController.jar train small/movies.csv small/users.train.csv small/small.simple.model

Training the recommender...
@@@ Computing user rating matrix
@@@ Computing user sim matrix
@@@ Writing out model file
Time Used: 00:00:12.455
```

This will create a ***model file*** `small.simple.model` under the directory `small`.

3. To test the recommender, use command:

```
java -jar MainController.jar test <movie_filename> <train_user_filename>
<model_filename> <test_user_filename> <report_filename>
```

```
>java -jar MainController.jar test small/movies.csv small/users.train.csv small/small.simple.model
small/users.test.csv small/small.simple.result

Testing the recommender...
Evaluation Result:
RMSE = 0.8564368531005996
Correlation = 0.22470193349982165
Precision = 1.1764705882352942
Recall = 0.4321326561606352
F1 = 0.5064082952243881

Time Used: 00:00:03.204
```

This will create a ***result file*** `small.simple.result` under the directory `small`.

4. To recommend movies to a user *uid*, use command:

```
java -jar MainController.jar recommend <movie_fname> <train_user_fname> <model_fname> <user_id>
```

For example,

```
>java -jar MainController.jar recommend small/movies.csv small/users.train.csv small/small.simple.model 19

Recommendation for 19
1. Annabelle (2014)
2. Impossible, The (Imposible, Lo) (2012)
3. Dying of the Light (2014)
4. Never Sleep Again: The Elm Street Legacy (2010)
5. PK (2014)
6. The Face of an Angel (2015)
7. Robot Overlords (2014)
8. Cold Fish (Tsumetai nettaigyo) (2010)
9. Dear John (2010)
10. Young and Prodigious T.S. Spivet, The (L'extravagant voyage du jeune et prodigieux T.S. Spivet) (2013)
11. Mortdecai (2015)
12. Palo Alto (2013)
13. The Hateful Eight (2015)
```

```
14. Serbian Film, A (Srpski film) (2010)
15. Wild Card (2015)
16. Vampires Suck (2010)
17. Jupiter Ascending (2015)
18. Life as We Know It (2010)
19. 30 Days of Night: Dark Days (2010)
20. Garden of Words, The (Koto no ha no niwa) (2013)
Time Used: 00:00:02.504
```

Coding Style Guideline:

It is important that you follow this coding style guideline strictly, to help us with the grading and for your own benefit when working in groups in future courses. Failing to follow these guidelines may result in a deduction of your project scores.

1. Write your name, student ID, and section on top of every code file that you submit.
2. Comment your code, especially where you believe other people will have trouble understanding, such as purposes of each variable, loop, and chunk of code. If you implement a new method, make sure to put an overview comment at the beginning of each method that explains 1) Objective, 2) Inputs (if any), and 3) Output (if any).
3. Do not put your code in packages. Put everything in the default package.

Submission Instruction:

It is important that you follow the submission instructions to enable your code and output files to be graded by the auto grader. Failing to do so may result in a deduction and/or fault evaluation of your project.

1. Generate the model file *my.micro.simple.model*, and the result file *my.micro.simple.result* from the "micro" test case.
2. Generate the model file *my.small.simple.model*, and the test result file *my.small.simple.result* from the "small" test case.
3. Generate the model file *my.q1.simple.model*, and the result file *my.q1.simple.result* from the "q1" test case.
4. Generate the model file *my.q2.simple.model*, and the result file *my.q2.simple.result* from the "q2" test case.
5. Produce a runnable **jar** file from MainController class and name it *my.MainController.jar*.

6. Put *SimpleMovieRecommender.java*, *my.micro.simple.model*, *my.micro.simple.result*, *my.small.simple.model*, *my.small.simple.result*, *my.q1.simple.model*, *my.q1.simple.result*, *my.q2.simple.model*, *my.q2.simple.result*, *my.MainController.jar* in a folder.
7. Rename the folder to your *Student ID (e.g. 5888xxx)*.
8. Use 7-Zip program[7] to zip the folder. Make sure the extension of the zip file is .7z (E.x., `5888xxx.7z`). Note that the model files can be huge. 7-zip allows huge text files to be efficiently compressed.
9. Submit the 5888xxx.7z file on eLearning, under `Project02 Submission`, before the deadline.
10. RECHECK YOUR SUBMISSION. Always re-check what you just submitted is actually what you wanted to submit. Any submission received after the deadline will be considered late submission and will receive the same penalty as one—no excuses.

Late submission will suffer a penalty of 20% deduction of the actual scores for each late day. You can keep resubmitting your solutions, but the only latest version will be graded. *Even one second late is considered late.*

Grading:

Grade for this project will be given in percentage basis (max 100 points, without bonus). Here's the tentative breakdown of the criteria:

If you submit	5	points
Follow the submission protocol	5	points
Follow the coding style	5	points
Outputs	70	points
Specific method functionalities	20	points
Total	105	points (But only collect 100 points)

We do not grade precision, recall, and F1 values. For the recommendation tasks, we only grade users whose at least one of the recommended movies has a predicted rating of < 5 stars.

Points will be deducted if your code runs too slow, so make sure that you use appropriate data structure to store the data. Training/Testing the small, q1, q2 cases should not take longer than 30 seconds each. Small deviation (i.e. ±0.0001) in double point precision is OK. Buggy code that does not compile may

[7] http://www.7-zip.org/

result in a 0 for the output portion. Note that the grading criteria above may be changed. You can print out anything you want on the screen, we only check the model and result files, and values returned from each method.

Your method should appropriately handle corner cases such as null references, out-of-range values, NaN/Inf values, division by 0, etc.

Suggestions:

1. Though NETFLICT is a fairly simple movie recommendation system compared to other commercial ones in the market, the small details can overwhelm you during the course of implementation. Our suggestion is to incrementally implement and test one method at a time, to see if it behaves as expected. Sometimes, you may need to write your own test cases to test specific functionalities, which is much encouraged. It is discouraged to implement everything and test it all at once. Our suggestion is to implement/test the methods in the following order:

   - `public Map<Integer,Movie> loadMovies(String movieFilename);`
   - `public Map<Integer, User> loadUsers(String ratingFilename);`
   - `public void loadData(String movieFilename, String userFilename);`
   - `public Map<Integer, Movie> getAllMovies();`
   - `public Map<Integer, User> getAllUsers();`
   - `public void trainModel(String modelFilename);`
   - `public void loadModel(String modelFilename);`
   - `public double predict(Movie m, User u);`
   - `public List<MovieItem> recommend(User u, int fromYear, int toYear, int K);`
2. Understand the math equations before implementing them. Make sure you can write a pseudo-code that algorithmically represents each equation.
3. Map data structure[8] (e.g. HashMap) allows fast lookup on your data. For example, getting Movie objects using `mid`'s as keys takes only O(1) runtime complexity. You may also find Guava's HashBiMap useful for 1-to-1 mappings.
4. Use Set data structure[9] (e.g. HashSet) whenever your data can be modelled using a set representation. Guava library implements multiple set manipulation operations that you may find useful such as intersection, union, etc.
5. Though we give you ample time to complete this project, don't wait until the last week to start working on it. Start early so you have enough time to cope with unforeseen situations. Plus, it can take some time to completely understand the project. You know how it went with Project 1, this project is probably harder for some of you.

[8] https://docs.oracle.com/javase/7/docs/api/java/util/Map.html
[9] https://docs.oracle.com/javase/7/docs/api/java/util/Set.html

6. Use Java JDK 1.8. Some of the functionalities may not be available in the older versions.
7. If your program needs more memory, use flag –Xmx1G to allocate 1 Gigabyte (etc.) of heap memory for your program.

[Optional] Special Challenge (Bonus): Can you make a better movie recommender?

You may notice that our naive `SimpleMovieRecommender` has a lot of room for improvement. If you wish to pursue the bonus credit, implement `BonusMovieRecommender` in BonusMovieRecommender.java that either ***implements*** `BaseMovieRecommender` or ***extends*** `SimpleMovieRecommender`. Test `BonusMovieRecommender` on the "***small***" dataset, and collect the average RMSE value. Note that, the smaller the RMSE value, the better the prediction performance. Your bonus recommender must perform better than the SimpleMovieRecommender (i.e. RMSE < `0.8564368531005996` for "small" case). You may use any cool techniques or third party java libraries. However, the only exception is that your `BonusMovieRecommender` cannot learn any information from the test file during the training process. For this bonus problem, you may consult the Internet to study ideas and potential techniques that can be adopted in your implementation. *Be creative!*

To submit the bonus credit, send an email to punyanuch.bor@mahidol.ac.th using your official ***Mahidol email***, with the subject line "`[ITCS208] MINE PERFORMS BETTER!`" (without quotes). In the email body, write down your name, ID, section, ***RMSE value*** and a (short) ***cool name of the technique*** you use, and also attach the ***result file***. *The deadline for the bonus submission is exactly 72 hours before the deadline of the project.* You can keep submitting newer (better) version of your `BonusMovieRecommender` as long as it is before the bonus deadline.

When you submit the project on eLearning, make sure to also submit your bonus code (and the libraries that you use) and a PDF file with one paragraph explaining briefly how your bonus recommender works.

The top 20 recommenders ranked by their RMSE values will be displayed on eLearning. At the end of the competition, the i^{th} ranked recommender will receive 26-*i* bonus points for the top 20 recommenders, and 5 bonus points for the rest. Furthermore, physical awards will be given to the implementers of the top 3 bonus recommenders. *If you end up scoring more than 100 points for this project, excessive points can be transferred to fill the gap in your Project 1 score.*

Bug Report and Specs Clarification:

Though not likely, it is possible that our solutions may contain bugs. A bug in this context is not an insect, but error in the solution code that we implemented to generate the test cases. Hence, if you believe that your implementation is correct, yet yields different results, please contact us immediately so proper actions can be taken. Likewise, if the project specs contain instructions that are ambiguous, please notify us immediately.

Need help with the project?

If you have questions about the project, please first post them on the forum on eLearning, so that other students with similar questions can benefit from the discussions. The tutors can also answer some of the questions and help to debug trivial errors. If you still have questions or concerns, please make an appointment with one of the instructors. We do not debug your code via email. If you need help with debugging your code, please come see us. Consulting ideas among friends is encouraged; however, the code must be written by yourself without looking at others' code. (See next section.)

Academic Integrity

Don't get bored about these warnings yet. But please, please do your own work. Your survival in the subsequent courses heavily depends on the programming skills that you harvest in this course. Though students are allowed and encouraged to discuss ideas with others, the actual solutions must be written by themselves without ***being dictated or looking at others' code***. Collaboration in writing solutions is not allowed, as it would be unfair to other students. It is better to submit a broken program that is a result of your own effort than taking somebody else's work for your own credit! Students who know how to obtain the solutions are encouraged to help others by guiding them and teaching them the core material needed to complete the project, rather than giving away the solutions. ***You can't keep helping your friends forever, so you would do them a favor by allowing them to be better problem solvers and life-long learners.* ** If you get caught cheating, serious actions will be taken!

References

Schafer, J. B. (2007). Collaborative filtering recommender systems. In A. K. Peter Brusilovsky, *The adaptive web: Methods and Strategies of Web Personalization.* Springer Berlin Heidelberg.