Fall 2018	**CSC 466:** Knowledge Discovery from Data	Alexander Dekhtyar

Lab 6: Collaborative Filtering and Recommender Systems

Due date: Wednesday, Nov 28, 11:59pm.

Overview

In this assignment you will implement a number of *memory-based collaborative filtering techniques* and will test their accuracy on a *stable* (i.e., non-changing) collection of ratings made available to you. You can use techniques studied in class, or any extensions you can find in literature or come up with yourselves.

Assignment Preparation

This is a pair programming assignment. Each student teams up with a partner. Each team submits only one copy of the assignment deliverables.

Data

You will be using *joke ratings data* from the **Jester** project, run by Professor Ken Goldberg at UC Berkeley. **Jester**[1] is an on-line joke recommender system available at

http://shadow.ieor.berkeley.edu/humor/

Disclaimer. Please read this note before proceeding! Jester has a database consisting of 100 jokes. The jokes are shown to the user, and the user's reaction to them is measured on a continuous scale. **Please be aware** that the jokes available through `Jester` may contain some examples that you personally will find objectionable. While you will only be working with numeric data, and do not have to deal with the text of the jokes in this assignment, please know, that *it is not my intent to offend anyone's*

sensibilities (and neither is it the intent of the authors of the dataset and **Jester**).

The Dataset

I have created a page on the course website to post links to the files you will need for this assignment:

http://users.csc.calpoly.edu/ dekhtyar/466-Fall2018/labs/lab06.html

The full **Jester** dataset is available at

http://www.ieor.berkeley.edu/∼goldberg/jester-data/

Ratings Data. The full dataset contains three data files of roughly equal size. Of the three files, you will be using only the first one, `jester-data-1`. The file is available for download as a `zip archive` from the **Jester** page above. For your convenience, a `CSV` version of the file is available for download from our web page.

The **Jester** dataset web page describes the format of the data as follows[2]:

"Format:

1. 3 Data files contain anonymous ratings data from 73,421 users.
2. Data files are in .zip format, when unzipped, they are in Excel (.xls) format
3. Ratings are real values ranging from -10.00 to +10.00 (the value "99" corresponds to "null" = "not rated").
4. One row per user
5. The first column gives the number of jokes rated by that user. The next 100 columns give the ratings for jokes 01 - 100.
6. The sub-matrix including only columns 5, 7, 8, 13, 15, 16, 17, 18, 19, 20 is dense. Almost all users have rated those jokes (see discussion of "universal queries" in the above paper)."

Jokes. The list of jokes is available from the following direct url:

http://eigentaste.berkeley.edu/jester-data/jester-joke-texts.zip

The archive contains the list of jokes in `html` format. For simplicity, a single XML file `Jokes.xml` listing the jokes is available from the course wiki page. The structure of the file is:

```
<jokes>
  <joke>
```

```
    text of joke goes here ...
  </joke>
  <joke> ... </joke>
  ...
</jokes>
```

The jokes are not numbered internally in the XML file, but their order corresponds to the joke ids in the ratings data file. Note, that the assignment does not require you to process this XML document. It is provided for your convenience, as you may find knowing the specific joke whose rating you are trying to predict.

Lab Assignment

In this lab you will implement and test four or more different collaborative filtering techniques and perform an accuracy study. The lab consists of the following components.

Parser and data processor. Your program shall parse the input CSV file (for the `jester-data-1` subset — feel free to hardcode that). The format of the CSV file was described above. While parsing, your program shall create and maintain the data structures necessary for collecting and maintaining all information needed for your collaborative filtering techniques. In particular, your program shall manage the data structures for the following:

1. The complete ratings matrix. The full matrix can be stored as either sparse or dense matrix and either in main memory or in persistant storage. If storing the entire data structure in main memory, make certain your program is allocated sufficient amount of memory at runtime. Note that your program will spend a lot of time looping through the ratings matrix, so the overall running time of your methods will mostly depend on how fast you can scan the ratings matrix. While the running time of your methods is not being evaluated in this assignment, slow programs present debugging challenges.
2. Any item-related statistics, such as average item rating or inverse user frequency of an item, *as needed by your collaborative filtering methods.*
3. Any user-related statistics such as average user rating or number of rated items, *as needed by your collaborative filtering methods.*

Note that for the sake of efficiency, your programs are expected to compute various parameters used in collaborative filtering methods *off-line*, i.e., before a collaborative filtering query is issued.

Collaborative Filtering techniques. You shall implement at least four different memory-based collaborative filtering techniques. You can implement any techniques discussed in class. You can also implement any other

techniques/extensions that you can find in the literature *as long as you acquire understanding of how the technique works*. Techniques can differ from each other by the type of similarity score used (e.g., *Pearson correlation* vs. *cosine similarity*), the formula used to compute the predicted rating, ratings transformations and adjustments and so on. *At least one of your implemented techniques must use the N Nearest Neighbors approach.*

Each technique shall be implemented as a separate function/method (or class). It should take as input, or, otherwise, have direct access to, the ratings matrix and to all the statistics you are maintaining.

Additionally, each method shall take as input two parameters: `int UserId` and `int ItemId`. The pair `(UserId, ItemId)` is the pair of values for which the method will make the prediction of the ratings score. Your implementation of each collaborative filtering technique shall work as follows:

- If the ratings matrix has value `99` at the `(UserId, ItemId)` location, then your method shall compute the predicted rating using the entire ratings matrix.
- If the ratings matrix has a value between -10.00 and $+10.00$, then your method shall replace this value in the ratings matrix with `99` and proceed to predict the user rating for the item using the updated ratings matrix[1].

The predicted rating your methods output should be in the same range (from -10 to $+10$) as ratings in the dataset.

Evaluation. You will evaluate all methods you implemented using two metrics: the MAE (Mean Absolute Error) metric, and the predictive accuracy metric described below. To facilitate the evaluation, you will create two evaluation methods. Both methods will take as input the following parameters:

- `int method.` This parameter specifies which of your collaborative filtering techniques will be tested.
- `int size.` This parameter specifies the number of predictions you should evaluate.
- *int repeats.* This parameter specifies the number of times the test is to be repeated.

The first method, **random sampling** will do the following:

- randomly generate `size` test cases. *Note:* a test case is a pair *(userID, itemID)*, such that $rating(userID, itemID) \neq 99$ (i.e., each test case represents prediction of a score actually found in the matrix).

[1] Please make sure that this replacement is temporary and that the ratings matrix is restored back to its original form after the prediction is ready.

- for each generated test case, use the selected collaborative filtering method to predict the rating;
- compare the predicted rating with the actual rating;
- output the results;
- compute and output the Mean Absolute Error (MAE) achieved in your test.
- repeat the entire testing process `size` times, report the MAE for each run, as well as the mean and the standard deviation for the MAE.

The second method, **user-specified test** will take as input a list of *(userID, itemID)* pairs and do the following:

- for each test case (i.e., the *(userID, itemID)* pair, determine whether it is valid, i.e., whether $rating(userID, itemID) \neq 99$.
- for each valid test case, compute the predicted rating using the specified method.
- compare the predicted rating with the actual rating;
- output the result;
- compute and output the Mean Absolute Error (MAE) achieved in the test.

This test shall ignore the `size` parameter.

Both methods should provide exactly the same output. For each test case, a single line (in a CSV format) shall be printed in the following format:

```
userID, itemID, Actual_Rating, Predicted_Rating, Delta_Rating
```

The columns are:

`userID`:	id of the user (line in the ratings table)
`itemID`:	id of the joke (column in the ratings table)
`Actual_Rating`:	User rating for the joke from the Jester dataset
`Predicted_Rating`:	Rating predicted by your method
`Delta_Rating`:	the difference between `Actual_Rating` and `Predicted_Rating`

At the end of the list, your output should contain the MAE measure, as well as any other measures your want to keep track of. (*Note:* only the MAE measure will be used in grading. However, you can use other measures to conduct your analysis).

Evaluating Recommendation Accuracy. For this project, we establish the following recommendation criterion:

> *Recommend a joke j to user i if your predicted score $\hat{u_{ij}} \geq 5$.*

The ground truth is

> *Joke j should have been recommended to user i if the actual score $uij \geq 5$.*

Using your predictions, and the true values of the user ratings for the jokes, you then shall compute the following information, and report it in your output:

- **Confusion Matrix**. You have two possibilities "recommend a joke" or "do not recommend a joke". Report the confusion matrix for them.
- **Precision, Recall, and F-1 measure.** Using the "recommend a joke" as the target, report the method's precision, recall, and F-1 measure.
- **Overall Accuracy**. Report the overall accuracy of your method (this includes the accuracy of NOT recommending jokes).

Programs. Using the methods described above, write two programs: `EvaluateCFRandom.java` and `EvaluateCFList.java`.

EvaluateCFRandom.java is, essentially, a wrapper around the first evaluation method discussed above. When run with no parameters, it shall print a help message indicating the list of collaborative filtering methods implemented and their Ids. Otherwise, the program shall take two parameters:

```
> java EvaluateCFRandom Method Size Repeats
```

Here, `Method` is the collaborative filtering method and `size` is the number of test cases to generate. The program shall execute the first evaluation method and print the output.

EvaluateCFList.java uses the second evaluation method to predict the ratings for a collection of test cases contained in a file. When run without parameters, the program shall print a help message indicating the list of collaborative filtering methods implemented and their Ids. Otherwise, it shall take two parameters:

```
> java EvaluateCFList Method Filename
```

where `Method` is the Id of the collaborative filtering method to be used in evaluation and `Filename` is the name of a file containing the list of test cases. The format of the file is straightforward: each line will contain two comma-separated integers: `UserID` and `ItemID`. Here is an example of a test file `SimpleTest.csv`:

```
1, 50
75, 34
100, 8
25, 20
1010, 33
```

The program outputs the result of running the second evaluation method on the data from the input file.

Report. Each team will submit a report detailing the methods implemented and the results obtained. The report, submitted as a single document, shall contain the following information:

- Names of team members.
- List of methods implemented and studied. If you implemented any methods, not mentioned in lecture notes, please provide a brief overview of the method and an academic citation (where you found it from).
- Research questions. What are you trying to study? (Essentially, you are trying to figure out which method works better, but I want to you formulate the precise question/questions you are going to give the answer to).
- List of your experiments. Sepcify what experiments you conducted in order to answer your questions.
- Results: show the results of your experiments. Provide visualizations (graphs, charts, etc..) where appropriate.
- Conclusions. State what you have discovered. In particular, specify, which of the methods you studied, you believe to be the best and what accuracy rates you have observed for this method.

Deliverables and submission instructions

The lab has both electronic and hardcopy deliverables. Submit electronically:

- All source code for all programs you create for this assignment.
- `README` file describing the following:

  - names of all team members;
  - names of all executable programs and information on how to compile/run them (if necessary).
  - list of implemented methods

- The soft copy of your report.

Submit all electronic deliverables except for the report and the `README` file as a single zip of gzipped tar archive (`lab06.zip` or `lab06.tar.gz`). Submit the report and the `README` file as separate files. Use the following command

```
$ handin dekhtyar lab06-466 <files>
```

References

[1] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval*, 4(2), 133-151. July 2001.

[2] Ken Goldberg, Anonymous Ratings Data from the Jester Online Joke Recommender System, http://www.ieor.berkeley.edu/∼goldberg/jester-data/.