CSC 212 Project Phase II
Developing a Ratings Query Application

College of Computer and Information Sciences
King Saud University

Fall 2016

1 Introduction

In this phase, your goal is to make the access to the data faster and add some extra functionalities.

From a performance viewpoint, we want that access to ratings is done in logarithmic time in average. If n is the number of users, m is the number of items, then

- The method *getUserRatings(i)* must be $O(\log n + k_i)$, where k_i is the number of items rated by user i.

- The method *getItemRatings(j)* and *getAverageItemRating(j)* must be $O(\log m + k_j)$, where k_j is the number of ratings for item j.

- The method *getRating(i, j)* (a new method that returns the rating given by user i to item j) must be $O(\log n + \log k_i)$, where k_i is the number of items rated by user i.

In term of functionality, we want to estimate the rating r_{ij} that user i will give to item j [1]. This is achieved as follows:

1. First, we create the list U_j of all users who rated item j.

2. We compute the distance between every user $u \in U_j$ and i. The distance between two users reflects how much they are dissimilar: if the distance is small, then the two users give similar ratings to the same items. If the distance is large, then the two users have very different tastes.

3. We select the k users that are nearest to i, that is, having the smallest distance to i (this set is also called the set of k nearest neighbors). The parameter k is an input.

4. The estimated rating r_{ij} will be the average rating given by these k nearest neighbors to item j.

[1] This is how websites decide what items to recommend to a given user: They estimate the ratings that the user would give to items and then select the items with the highest estimated ratings.

To compute the distance $d_{u,i}$ between two users u and i:

1. We create the list of items $I_{i,u}$ of items that are rated by both users i and u.
2. If $I_{i,u} = \emptyset$, then $d_{u,i} = +\infty$.
3. Else:

$$d_{i,u} = \frac{1}{n_{i,u}} \sqrt{\sum_{j \in I_{i,u}} \left(r_{i,j} - r_{u,j}\right)^2},$$

   where $n_{i,u}$ is the number of elements in $I_{i,u}$.

Example 1. *Consider the following ratings given by 6 users (rows) to 5 items (columns):*

	1	2	3	4	5
1		3	?	2	5
2	2	5			1
3	2		4		
4	1		4	3	
5			1	3	4
6	2	2	1	2	

We want to estimate $r_{1,3}$ using $k = 2$ neighbors:

1. *The users who rated item 3 are $\{3, 4, 5, 6\}$.*
2. *The distances between these users and user 1 are:*

$$\begin{aligned}
d_{1,3} &= +\infty, \\
d_{1,4} &= \sqrt{(2-3)^2}/1 = 1, \\
d_{1,5} &= \sqrt{(2-3)^2 + (5-4)^2}/2 = 0.707, \\
d_{1,6} &= \sqrt{(3-2)^2 + (2-2)^2}/2 = 0.5.
\end{aligned}$$

3. *The 2 nearest neighbors to user 1 are users 6 and 5.*
4. *The estimated rating is*

$$r_{1,3} = (r_{6,3} + r_{5,3})/2 = (1+1)/2 = 1.$$

2 Requirements

In this phase, you are required to implement the following methods:

```
public class RatingManager {
    // Constructor
    public RatingManager();
```

```
// Read ratings from a file and create a RatingManager object that stores these
    ratings. The ratings must be inserted in their order of appearance in the
    file.
public static RatingManager read(String fileName);

// Add a rating
public void addRating(Rating rating);

// Return all ratings given by user i.
public LinkedList<Rating> getUserRatings(int i);

// Return all ratings given to item j
public LinkedList<Rating> getItemRatings(int j);

// Return the list of highest rated items
public LinkedList<Integer> getHighestRatedItems();

// Return the average rating of item j. If i has no ratings, -1 is returned
public double getAverageItemRating(int j);

// Return the average rating given by user i. If i has no ratings, -1 is
    returned
public double getAverageUserRating(int i);
//************************************************************************

// Return the rating of user i for item j. If there is no rating, -1 is returned
    .
public int getRating(int i, int j);

// Return the number of keys to compare with in order to find the rating of user
     i for item j.
public int nbComp(int i, int j);

// Compute the distance between the two users ui and uj. If ui and uj have no
    common item in their ratings, then Double.POSITIVE_INFINITY is returned.
public double getDist(int ui, int uj);

// Return a list of at most k nearest neighbors to user i from a list of users.
    User i and users at infinite distance should not be included (the number of
    users returned can therefore be less than k).
public LinkedList<Integer> kNNUsers(int i, LinkedList<Integer> users, int k);

// Return the average rating given to item j by a list of users. If the list
    users is empty or non of the users it contains rated item j, then the global
     average rating of item j (as computed by getAverageItemRating(j)) is
    returned.
public double getAverageRating(int j, LinkedList<Integer> users);

// Return an estimation of the rating given by user i for item j using k nearest
     neighbor users.
public double getEstimatedRating(int i, int j, int k)   int r = getRating(i, j);
  if (r != -1) {
     return r;
  }
  LinkedList<Rating> ratings = getItemRatings(j);
```

```
        LinkedList<Integer> users = new LinkedList<Integer>();
        if ((ratings != null) && !ratings.empty()) {
            ratings.findFirst();
            while (!ratings.last()) {
                users.insert(ratings.retrieve().getUserId());
                ratings.findNext();
            }
            users.insert(ratings.retrieve().getUserId());
        }
        LinkedList<Integer> knn = kNNUsers(i, users, k);
        return getAverageRating(j, knn);
    }
}
```

3 Deliverables and rules

You must deliver:

1. A report written using the provided template.
2. Source code submission to Web-CAT.

The submission **deadline** is: **22/12/2016**.

You have to read and follow the following rules:

1. All data structures used in this project **must be implemented** by the students. The use of Java collections or any other library is strictly forbidden.
2. This project is to be conducted by groups of **three** students. Groups of more than three students are not accepted. Groups of two students are strongly discouraged and can only be accepted with a special permission from the course instructor.
3. All the members of a group must have the **same course instructor**.
4. All students must **submit** the list of their **group members** within one week of the announcement of this project. Once the groups are chosen, no student can change the group (even if some group members have dropped the course).
5. **Every member** of the group must participate in **all parts of the project**: designing the software, programming and writing the report. Members of the same group may receive different marks according to their participation in the project.
6. The submitted software will be evaluated automatically and/or in a demonstration to which all the group members must attend.
7. Any member of the group who fails to **attend the demonstration** without a proper excuse (consult the university and college regulations) shall receive the **mark 0** in the project.
8. In accordance with the university regulation, **cheating** in the project will be sanctioned by the **grade F**.