# Lab 5: Collaborative Filtering and Recommender Systems

**Due date:** Tuseday, May 19, midnight.

## Overview

In this assignment you will build a simple recommender system based on a *stable* (i.e., non-changing) collection of ratings made available to you. You will implement your recommender system using **memory-based collaborative filtering** techniques. 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 **tasteless**, **sexist**, **inapropriate** or just plain **stupid**. While each team will work with the texts of the jokes as well as with the numeric data, please be aware, 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 wiki to post links to the files you will need for this assignment:

http://wiki.csc.calpoly.edu/csc466-2009/wiki/Lab5

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 wiki 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.

## Lab Assignment

You will build a simple recommender system using the data provided to you. The recommender system consist of two parts:

1. **Off-line recommendation computation component.** A standalone program that takes as input the ratings data and computes item-based recommendations. The computed recommendations are then stored in persistent storage and used by the second component of the system.
2. **On-line recommendation display component.** Also known as the **front-end** of the system, this component will display a list of available jokes. It will allow the user to
   (a) Receive *"More like this"*-style joke recommendations for a selected joke.
   (b) View the list of users from the dataset who would be recommended a selected joke.

### Recommendation computation component.

The **off-line** or **back-end** component of your system will work as follows.

**General notes.** The off-line component can be written in any programming language. For simplicity, we refer to it as being a Java program named `computeRecs.java`.

**Input.** Your off-line component will take as input the file name of the data file containing the ratings matrix (for example the `jester-data-1CSV.csv` file available on the wiki).

**Program flow.** Your program shall work as follows.

1. **Input handling.** The input file will contain information about 23-24 thousand users, each with at least 36 ratings on **Jester**. The data needs to be handled graciously. In particular, note the following:

Your program will be computing a list of jokes, and a list of users for each of the 100 jokes in the dataset. This needs to be done in a way that minimizes the number of times the data is scanned. The order in which the loops need be executed is: **foreach row** first (outer loop), **foreach column** next (inner loop).

2. **Item-based item recommendations.** For each joke in the dataset, your program will compute 10 **most similar** jokes according to the existing user ratings. You can use *Pearson correlation* and/or *cosine* similarity measures.

3. **Lists of users.** For each joke in the dataset, your program will find 10 users with the **highest predicted rating value**. You can use any *memory-based prediction approach* discussed in class. You can also extrapolate and/or experiment with other *memory-based prediction approaches*. Note, that in order to add user $x$ to the list for joke $y$, there must not be a rating for $y$ from $x$.

**Output.** Your prgram shall output the lists its computed. It shall create two files, `jokesRecs.dat` and `userRecs.dat` to store the lists of similar jokes and the lists of users respectively.

The format of the `.dat` files is left up to you. Given the expected size of each file: 100 rows of up to 11 columns each (one column for the joke Id, 10 columns for similar joke Ids or users), you may choose plain `CSV` format. You may also elect to represent data in binary form for ease of retrieval.

Your front-end program will be responsible for reading and correctly interpreting the contents of `jokeRecs.dat` and `userRecs.dat`.

### Recommendation display component

This is the **on-line**, **interactive front-end** of your system.

**General notes.** The **recommendation display component** is a simple graphical user interface that reads in the information about the jokes (`Jokes.xml` file), and the information produced by the **recommendation computation component** and displays it upon user request.

**Starting point.** For the **recommendation display component** we provide you a collection of Java files. Some of the methods inside these files are stubbed. You are asked to **complete** this component (rather than to write it from scratch) by writing full versions of the stubbed methods.

Note, that because code is provided for you for this component, we expect you to use Java. Because the on-line and off-line components are independent, this does not affect your choice of the programming language for the off-line component of the system.

**Available code.** To get the code provided to you, download the `Lab5-Student-sources.zip` from

http://wiki.csc.calpoly.edu/csc466-2009/wiki/Lab5

This archive contains the following files:

`recommender.java` The "main" classes for the **recommendation display component**. This file contains the methods which populate the jokes into the main GUI window. All methods that deal exclusively with GUI and with the `Jokes.xml` files are provide (e.g. of such a method is the method that finds the text of a selected joke in the `Jokes.xml` file and places it in the text box.

`RecToPerson.java` This file contains the GUI for the pop-up window that is used to display the list of users with the highest predicted ratings for a selected joke. You probably do not have to edit this code at all. While you are only asked to provide the list of users with the highest predicted ratings (who your system recommend the joke for), the GUI and the API is designed to also support display of the "don not recommend to" list of users with the *lowest predicted ratings*.

`SparseVector.java` This class implements a simple sparse vector. Feel free to use it or to ignore it (you may have similar classes created from previous assignments in this course.)

**GUI.** Figure 1 shows the starting state for the display component (`recommender.java` program). The GUI consists of two list boxes (left and right), two text boxes (left and right) and three buttons: Recommend To and More Like This and Exit.

The lefthand side of the GUI is used to display the full list of available jokes (in the list box, at the top). Once a joke is selected, its text appears in the text box below.

The righthand side of the GUI is used to display the list of jokes similar (according to the user ratings) to the selected joke. Figure 2 shows the righthand side of the GUI populated after the More Like This button is pressed. The list of similar jokes is at the top, and the text of the joke selected from the list is in the text box at the bottom.

Pressing the Recommend To button causes a pop-up window to open. Figure 3 shows the pop-up window. The pop-up window contains a text box at the top, displaying the currently selected joke. It also contains two list boxes. The Recommend For list box shows the list of 10 users with the highest predicted scores and the Don't Recommend For list box shows the **optional** (you get extra credit for displaying it) list of 10 users with the lowest predicted scores.

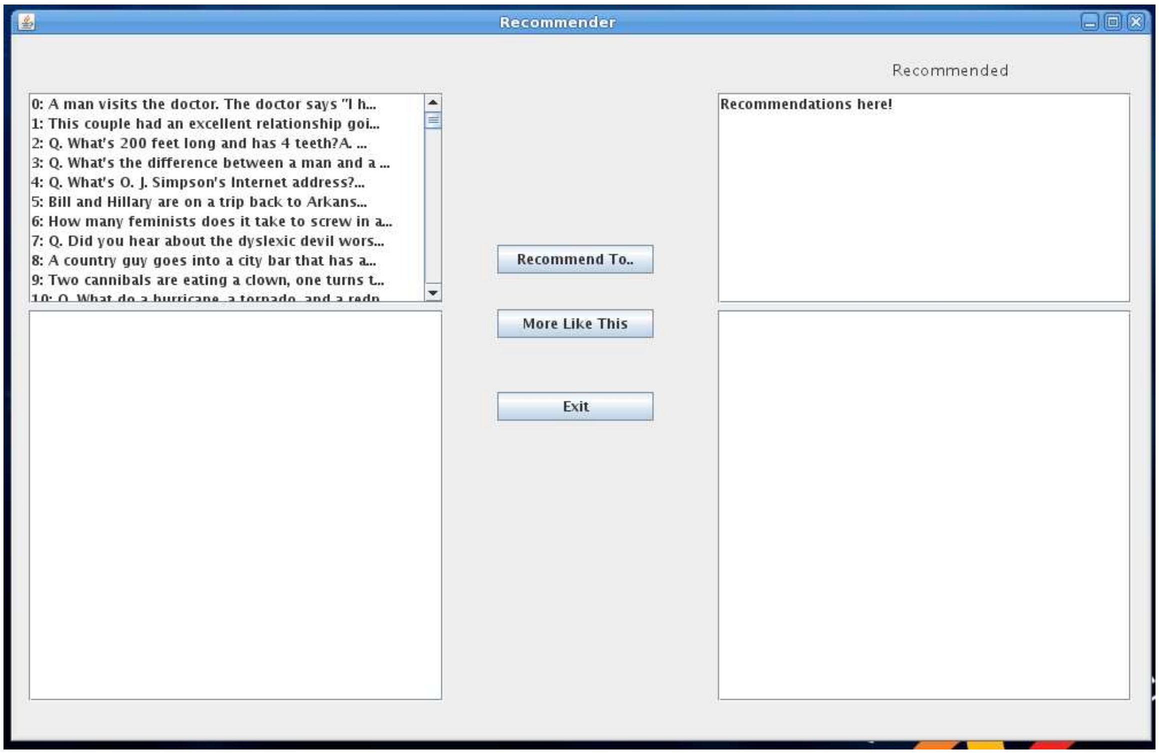


Figure 1: Recommendations display component. Main GUI.

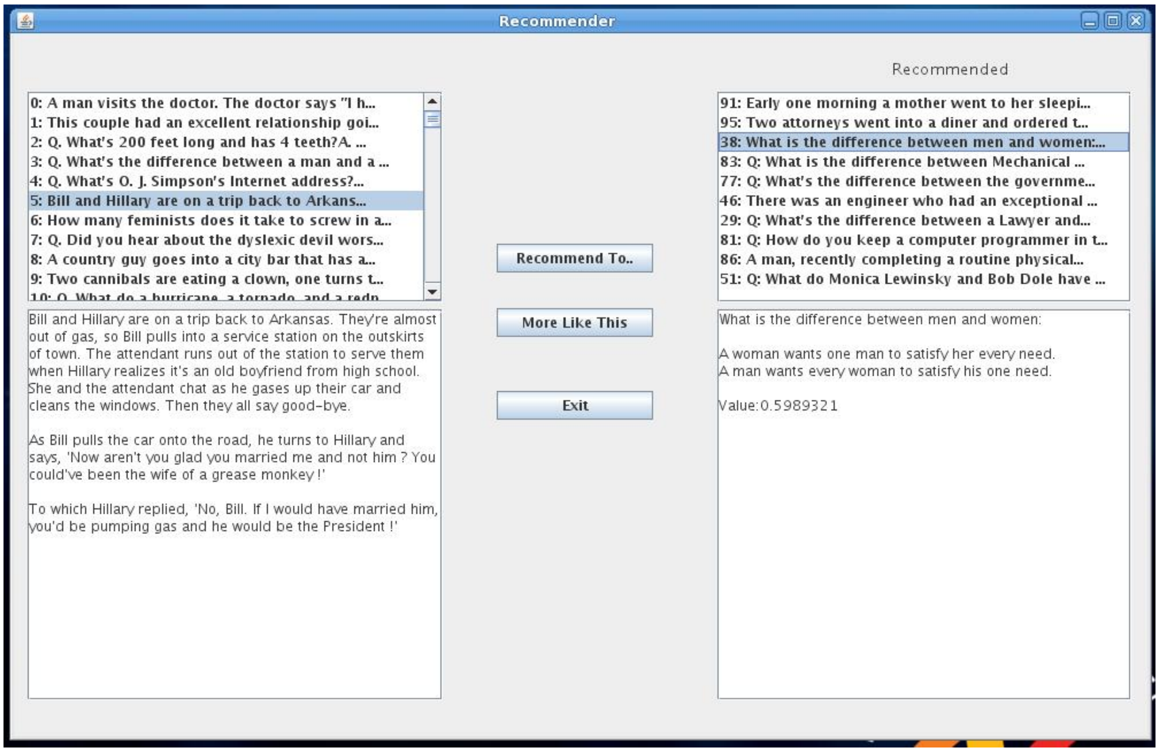


Figure 2: Recommendations display component. After More Like This button is pressed.

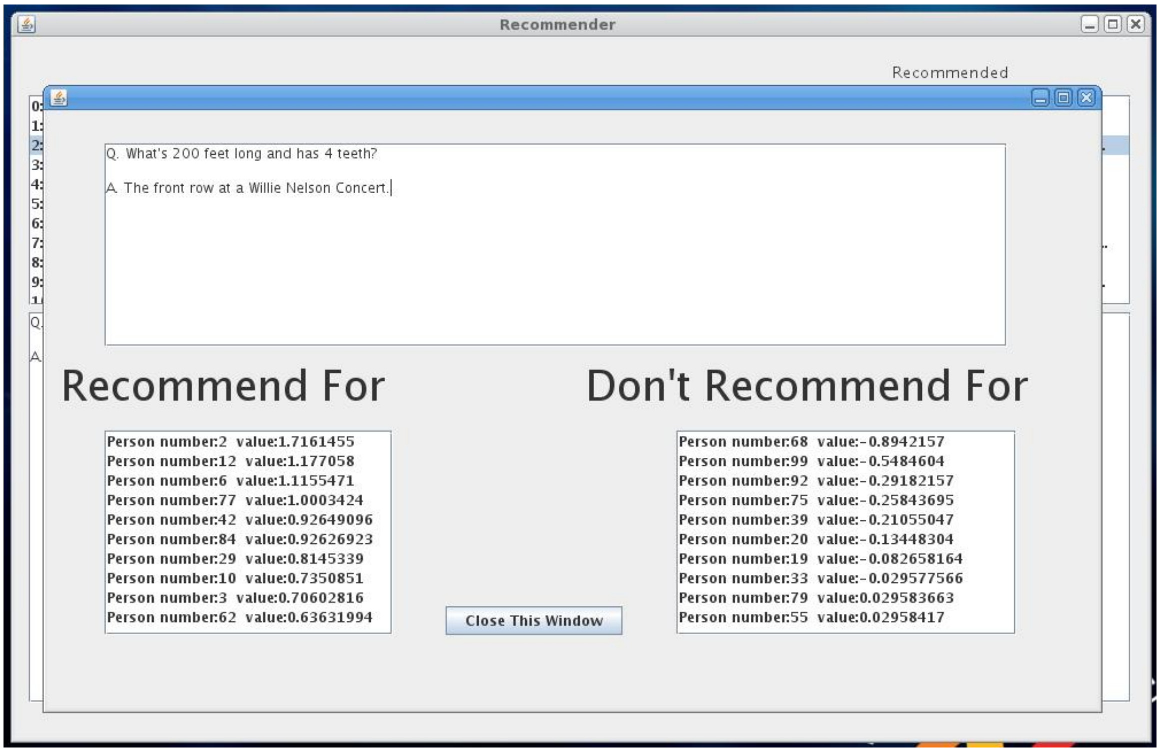


Figure 3: Recommendations display component. After Recommend To button is pressed.

**Stubs.** The following methods contained in the code provided to you are stubs that you need to write.

```
//This method is for clicking the "more like this" button
//below is code showing you how to populate the top right list
//with the jokes, you need to figure out what to place in the list.
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

    ArrayList<String> result = new ArrayList<String>();
    result.add("hey                                ");
    result.add("this is a test                       ");
    result.add("Add recommended jokes like this");
    populateL2(result);
    result.clear();
    //add your code here! to populate jList2 with recommendations!
}


//In this method, you will be calculating the scores for who to recommend the joke to
//and who not to recommend the joke to
//below is the skeleton code, it is up to you to populate the ArrayLists.
//recToP is the second GUI that appears when the button is clicked and displays
//the joke and two lists, one of people who would like the joke, and one who wouldnt
```

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {

    ArrayList<String> forList = new ArrayList<String>();
    ArrayList<String> againstList = new ArrayList<String>();
    forList.clear();
    againstList.clear();
    //add your code here! populate the recommended for and against lists!

     recToP.set_for(forList);
     recToP.set_against(againstList);
     recToP.setVisible(true);

}

//NOT PROVIDED FOR YOU BUT RECOMMENDED
//create a method which takes in 2 sparsevectors, and computes the pearson correlation
private float pearson(SparseVector q, SparseVector v)
{

}

//NOT PROVIDED FOR YOU BUT RECOMMENDED
//create a method which takes in a sparsevector and computes its average
private float avg(SparseVector v)
{
}
```

## Deliverables and submission instructions

This lab has only electronic deliverables. Submit the following:

- Source code for both components of the recommender system.
- README file describing the following:
  - names of all team members;
  - any non-Java components of your submissions;
  - which similarity score(s) and which memory-based recommendation method(s) are implemented;
  - any compile/runtime instructions for the TA;
  - any extra credit claims;

Submit all electronic deliverables as a single zip of gzipped tar archive (lab05.zip or lab05.tar.gz). Use the following command

```
$ handin dekhtyar-grader lab05 lab05.<ext>
```

## 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/.