ASSIGNMENT #4
Movie Similarity
100 points

Topics covered:
Java file handling: output, regular expressions, serialization, Unicode characters
Data structures: String, map, sort with comparator function
Review of input, arrays and arrayList

Goal:

In this assignment we will write two programs. Program 4a will import data from two files and preprocess it for 4b. Program 4b will ask the user for a movie number and display the 20 movies in the list that are most similar to it. It will measure similarity by using the Pearson *r* coefficient among same reviewers.

Specs:

Program 4a: (35 points)

1) Make a map that can be used to convert non-Ascii characters to the closest Ascii character, e.g. ‘é’ to ‘e’. (You only need to include the non-Ascii characters used in the file.)

2) Read file *movie-names.txt*. Use a regular expression to separate the movie number, the vertical bar and the movie name. Print each line that contains a non-Ascii character. Then replace that character by its replacement.

3) Use a regular expression to separate the movie number, the vertical bar and the movie name. Write out a revised file where each record contains the movie number as a four-digit Ascii string representing an integer followed by an Ascii representation of the movie name (with no vertical bar). Call your output file *movie-names2.txt*.

4) After reading the entire file, print the total number of non-Ascii characters and the number of lines containing a non-Ascii character.

5) Read file *movie-matrix.txt*. Expand the data into a non-sparse format, e.g., a 2-D array. Write the resulting data structure as a one-record serialized file under the name *movie-matrix2.ser*.

When you write your files, make sure to use *relative* filenames, i.e., your file name should not include your z-id or any other hard-coded directory information. Remember that your TA is going to run your program on turing in his directory, not in yours.

.

Program 4b: (65 points)

1) Read the files *movie-matrix.ser* and *movie-names2.txt* and store the data.

2) Prompt the user for a movie number. Repeat the following algorithm until the user replies ‘q’ or ‘quit’.

If the reply is ‘q’ or ‘quit’, end the program.

If the reply is non-numeric or out of range, display a message and ask again.

Print the movie number and name.

3) Compare the target movie to all of the movies in the matrix, including itself, as follows:

   a. Find all people who have rated both movies. If there are less than 10, don’t use this movie for comparison purposes (i.e., skip the rest of this loop).

   b. Compute Pearson *r* between the target movie and the comparison movie. (Make a function for this. It should call smaller functions you have written for the pieces of the calculation.)

   c. If there are less than 20 comparison movies, print “Insufficient comparison movies” and return to the prompt for a new movie.

4) Sort the comparison movies by decreasing Pearson *r* value.

5) Print the top 20 comparison movies with their Pearson *r* value. Print headers on your list and number the entries. Make sure that the *r* values are decimal-aligned and that the movie numbers are zero-suppressed and right-aligned.

Data:

The files *movie-matrix.txt* and *movie-names.txt* are available in *d470/dhw/hw4-movies*. When running on turing, your program should open these files, not your own copies of them.

(If you are testing on a machine with a different directory structure, I suggest you use an optional command line parameter that defaults to the file structure on turing. In that fashion we can run your code without any parameters and you can use a command line parameter to match the directory structure on your machine.)

Each record in *movie-matrix.txt* represents people’s ratings of one movie, i.e., the first reviewer’s rating, the second reviewer’s rating, etc. Each rating is followed by a ‘;’. When you see two semicolons in a row, that is effectively a null rating followed by a semicolon, i.e., it indicates that the given reviewer didn’t rate the movie. This type of layout is frequently used to save space in a sparse matrix, i.e., a file with a lot of zeroes.

The *movie-names.txt* file gives the names of the movies, i.e, the first record gives the name of the first movie in *movie-matrix.txt*, etc.

Testing:

Test with a few movies that you liked or think you might like. Each movie you try should have a Pearson *r* comparison to itself of 1; however, due to roundoff error, you may get a number like 0.997. The other movies with high *r* values will be those that the same reviewers gave similar scores to as they did to the target movie. See if you think they are movies you might like.

In addition, I will post my output.