# Movie Recommendation System

*Goal:*

In this assignment we will write two programs. Program 5a will import data from two files and preprocess it for 5b. Program 5b 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 5a:

1) Read file *movie-names.txt*. Print each line that contains a non-Ascii character. Print the total number of non-Ascii characters. Print the number of lines containing a non-Ascii character.

2) Once you see which non-Ascii characters are included in the file, make a map that can be used to convert each non-Ascii character to the closest Ascii character, e.g. 'é' to 'e'. Update your program to replace each non-Ascii 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) 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 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 5b:

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/hw02-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. If instead you hardcode your own directory structure for testing, make sure you update it before you submit!)

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. 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.

*Programming tips:*

A) As you load the ratings, you should check that each movie has the same number of reviewers. They may have different numbers of reviews (non-null entries), but remembering that there is a null slot when a reviewer doesn't rate a movie, each inner sublist in the ratings matrix should have the same length. If that's not true, you won't be able to do the similarity calculation.

B) Regardless of where you write and test your program, you should do your final testing on turing because that is where we will run your code.

C) Again, make sure that your files use *relative* filenames, not absolute filenames or any filename

that requires creating a new subdirectory. As a reminder, your TA is going to run your program on turing in his directory, not in yours.

*Background information:*

The slides used in class to explain the concept are in the assignment directory under *470-2016b-recom04.ppt*. The example used in class to illustrate the Pearson calculation is located in *similarity1.pdf*.

If you want further implementation suggestions, I describe my implementation of the Pearson calculation and the data structures I used in *pearson2.doc*. Feel free to follow my implementation or use different data structures if you prefer.

*FAQ:*

1. Q: Why are you asking us to convert the movie names to Ascii when Java can handle Unicode just fine?

A: The issue isn't just Java's underlying character representation (although that is important), but how the input is encoded. C++ still doesn't support Unicode and a lot of people still have files encoded with various sorts of "extended Ascii" schemes. There are also still some 7-bit mailers around that mangle 8-bit input.

2: Q: Why did you ask us to serialize one of the input files and not the other?

A: Simplicity, mainly. You could build a data structure containing all the movie names in 5a and pass that to 5b also, and that would be a good design. But we hadn't had an assignment that just wrote a simple byte-oriented file yet (i.e., what people commonly call "an Ascii file" whether it is actually used an Ascii encoding or not).