CALCULATING THE PEARSON CORRELATION

Calculating r

The Pearson correlation, *r*, is a number representing how similar two matched sets of data are.

For example, suppose that you have two movies and six reviewers saw each movie.

Ratings for movie 1: 5, 3, 1, 5, 1, 3
Ratings for movie 2: 3, 2, 1, 3, 1, 2

Although the values are different (movie 1 was way more popular than movie 2), the pattern of reviewers' opinions was similar; people who were above average in their opinion of movie 1 were also above average in their opinion of movie 2, even though movie 2's average was lower.

You can use the following steps to calculate *r*:

1) Calculate the average rating for each movie.

Movie 1 average = 18/6 = 3
Movie 2 average = 12/6 = 2

2) Calculate the standard deviation for each movie. The s.d. is a measure of dispersion, i.e., there is a difference between a movie that has an average score of 3 because everyone gave it a 3, and a movie that has an average score of 3 because half of the people liked it and half hated it.

You can calculate the s.d. as follows:

a) For each movie, subtract the average for that movie from each review in the list:

Movie 1 differences: 2, 0, -2, 2, -2, 0
Movie 2 differences: 1, 0, -1, 1, -1, 0

b) Square those differences:

Movie 1 squares: 4, 0, 4, 4, 4, 0
Movie 2 squares: 1, 0, 1, 1, 1, 0

c) Get a total for each movie:

Movie 1 total: 16
Movie 2 total: 4

d) Divide by one less than the number of reviews:

Movie 1: 16/5 = 3.2
Movie 2: 4/5 = 0.8

e) Take the square root of the result:

Movie 1: sqrt(3.2) = 1.79 (all values approximate after this).
Movie 2: sqrt(0.8) = 0.89

3. Convert the reviews from absolute scores to s.d. units from the mean. (These are *z*-scores if you are a statistician.)

In other words, for each review, subtract the average for that movie from the review. Then divide by the s.d. for that review.

That will tell you how far from the average for that movie a given reviewer is, in multiples of the s.d.

Overall, movie 1 may be have better scores than movie 2, but for purposes of finding the correlation between them, we don't care about that. What we care about is how the pattern of reviews by the same people goes up and down. This step normalizes the effect of the absolute scores.

For example, the normalized score for movie 1 reviewer 1 will be (5-3)/1.79 = 1.12.
The normalized score for movie 2 reviewer 1 will be (3-2)/.89 = 1.12.
That indicates that reviewer 1 was among the more positive reviewers for each movie, even though movie 1 was way more popular.

4. Multiply the normalized scores for the same reviewer together

product 1 = (movie 1 reviewer 1)(movie 2 reviewer 1) = 1.12 * 1.12 = 1.25.
etc.

5. Add up the products for all the reviewers.

6. Divide by 1 less than the number of reviewers. The result is *r*.

In this case the result is 1.0, meaning that the two movies are perfectly correlated, i.e., everyone who likes movie 1 will like movie 2, and vice versa. That rarely happens in real life. If you look at the original data, you can see that it was set up to show exactly that case.

A high positive number indicates a strong but realistic relationship.

A correlation of 0 means that there is no relation between people's opinions of the two movies.

A correlation of -1.0 means that the movies are complete opposites; anyone who likes movie 1 will hate movie 2, and vice versa.

A correlation of 0 means that there is no relation between people's opinions of the two movies.

Note: This calculation is the same as the one in the *similarity1.pdf* handout, except that that document is calculating similarity between two reviewers, and this one is calculating the relationship between two movies. (In that example, the effect of converting to *z*-scores is to account for the fact that one reviewer grades harder than the other. We don't want that to affect our opinion of which movies did well.)

Possible implementations

a) I implemented the calculation above by writing functions for sum, average, s.d. and r.

b) Remember that in order to do this calculation, you need to extract from the original data only those scores where both reviewers saw the movie. Otherwise you won't have the pairs of numbers required for this algorithm.

There are at least two ways you can get these pairs:

1) Set up two data structures (1-D arrays, vectors, lists, etc.) to store the common reviews, one for each movie. Go through each of the two rows of the movie matrix, checking to see if both movies have a non-zero rating for a given reviewer. If yes, copy the two scores to the respective data structures.

2) Run through the two rows in the same way, but instead of copying the scores, just make one data structure giving the numbers of the reviewers whose scores are to be used when calculating *r*.

You can see some sample data for this step in the sample output on turing, for movie 1 compared to itself and for movie 1 compared to movie 2. (A movie compared to itself should always give a value of 1, so this calculation is just there to verify that your code is correct. In the real world you would not suggest to a person that if they liked a certain movie, they should just see it again.)

Once you have this step correct, then you can use your new vectors to calculate *r*. The sample output gives you the value of *r* as well as some of the intermediate values for a number of movie pairs.

c) Finally, once you have calculated the correlation (*r*-value) between the target movie chosen by the user and all the other movies, you need to choose the movies with the 20 highest correlations (i.e., the 20 movies with the most similar pattern), sort and print them.

There are many ways to do this. One way is to use an ArrayList to store the data because ArrayList has a *sort* method. I created an object to store the fields I needed to print, and created an ArrayList of that object.

One detail that needs to be observed is that you cannot sort the objects directly because Java does not know which field you want to sort them on. So just as in C++, you need to write a comparison function for your object. This function will return 1, 0, or -1 depending on whether the first object is greater, equal or less than the second object, based on the field you have chosen.

The example used in class is on turing in d470 under ch. 16, figs. 8 and 9. (If you'd like to start with simpler examples, look at figs. 6 and 7 in the same directory first.)

d) I also found it convenient to use an arrayList to store the movie names.