Ryan Johnson
COSI 166B
PA 2: Movies 2

Github Repo URL: https://github.com/rjohnson465/movies-2
Code Climate URL: https://codeclimate.com/github/rjohnson465/movies-2/issues

The Algorithm

On a high level, the chain of operations this algorithm uses goes like:
- run_test(k)
    - predict(u, m) (k times)
        - most_similar(u)
            - similarity(u1, u2) (for many users u2 with u1 = u)

Breaking it down a little further:

A MovieData object is made with a specified dataset (i.e. :u1). Data is loaded into some hashes, training sets and test sets. Run_test is called by a MovieData object, with optional parameter k. K or all 100,000 lines (not recommended) are processed within run_test. Each one is parsed and given to predict(u,m).

Predict(u,m) finds a set of users similar to u (by calling most_similar(u)) and a set of users who have seen movie m. It then cross references these sets; for each user s in the set users who have seen movie m, and if there is a similarity score given for s (with respect to u), that similarity score is added as weight to a running total_weight. Additionally, an adjusted_rating is made by multiplying the weight by the rating s gave m, then added to a running total adjusted_ratings. After all users in the set of users who have seen movie m have been processed like this, a prediction is returned, equal to all the adjusted_ratings / total_weight.

Diving deeper, most_similar(u) actually includes two possible approaches, APPROACH ONE and APPROACH TWO. APPROACH TWO is commented out currently. It is faster than APPROACH ONE, but less accurate. Both approaches take the same form.

- Compile some set of users
- Get a similarity score of each user s in that set with user u by calling similarity(u, s)
- Store all these scores in a hash s.t. {key=user_id val=similarity(u,s)}
- Return hash

APPROACH ONE uses every single user in the training set that is not u. For dataset u1.base, this is 943 other users -- that means similarity(u,s) is run 943 times to generate a single similarity hash. Bear in mind this is run every time a prediction is made. As such, this approach is very very slow -- each prediction takes around ~1200 ms on my system.

APPROACH two attempts to fix this a bit by limiting the number of similarity() calls there are to 100. Ideally, the set used in most_similar(u) is the set of users that are most likely to be similar to u. The idea was to take the top 100 users with the most amount of similar movies to user u and use that as the set. This worked, in the sense that it cut computation time down by nearly 2/3 (each call to most_similar took ~400 ms) but the accuracy plummeted. As such, this approach is commented out.

A full Big O analysis of The Algorithm is produced below.

Ryan Johnson
COSI 166B
PA 2: Movies 2

The Analysis

Big O Analysis of The Algorithm - Starting from the lowest method to the highest (see High Level structure in The Algorithm section)

- Similarity(u1, u2): $O(n^m)$
n = |smaller set of movies (u1 or u2's)|
m = |larger set of movies (which user n's set is not of)|
For every movie o in n, comparisons of ratings are made to every movie p in m, leading to exponential time

- Most_similar(u): O(l)
l = |set of users used for similarity hash construction|
l is either |train_set_users| (APPROACH ONE) or 100 (APPROACH TWO)

- Predict(u, m): O(2p)
p = |all users who have seen m|
p is used twice to format a hash called seen_m, that allows for cross referencing with similarity scores (see explanation of predict(u, m) in The Algorithm section

- Run_test(k) O(k)
k = constant, given, number of calls to predict(u,m)

When these are put together as they work together in The Algorithm, the final time complexity is

$O(k(ln^m + 2p))$

Benchmarking

Using APPROACH ONE for most_similar(u), a single prediction takes ~1200 ms (k = 1 for run_test(k)). Clearly, the scaling on this is bad. Running 1000 predictions would take twenty minutes. Bad. APPROACH ONE, with 100 tests, results in a Mean Error of .76 and a Root Mean Square Error of .92.

APPROACH TWO for most_similar(u) yields slightly better temporal results, taking ~400 ms (lower "l" value in the Big O time complexity formula). However, this still scales sort of poorly. 1,000 tests would still take over six minutes to compute. APPROACH TWO, with 100 tests, results in a Mean Error of .84 and a Root Mean Square Error of 1.07.

Ways to fix this would be to decrease the "l" value in the Big O time complexity formula, or rewrite similarity(u1,u2) to run in a different time complexity, perhaps something linear. In the time I had, I did not devise such a revision, but I imagine it's very doable. While running similarity(u1, u2) currently takes between .5 and 5 ms (depending on the "n" and "m" values), running this "l" times for many "l" is still giving bad computation time.