Cameron Cho
CS166b

Synopsis: PA's-Movies

Links

GitHub: https://github.com/ctcho/PAs-Movies

CodeClimate: https://codeclimate.com/github/ctcho/PAs-Movies/code

Discussion: The Prediction Algorithm

My algorithm is incredibly simple. It takes a user and a movie the user has never seen before, then it takes a list of users that are most similar to the given user (as an extra note, the list of users considered "most similar" are those that share a similarity rating of 4.0 or higher, based on both what movies both have seen and the rating each gave them). Then, for every user in the list, it checks to see if the user has seen the given movie. If they have, it obtains the rating they gave the movie and adds it to a running total. At the end, the predictor returns the average of the ratings divided by the number of users that saw the movie, but if no one in the list saw the movie, the predictor gives a 3 as a safe guess.

The advantage of such an algorithm is that it only considers the opinions of those that really matter, users that can truly be considered closest to the one in question. However, it does provide a comparably small list of users available to make a prediction, which in turn narrows the chance of actually getting a true estimate from the movie in the (somewhat common) event that no one in the list has seen it.

Algorithm Accuracy

After running through the 5 tests provided by ml-100k, I have determined a general pattern in the statistical data that appears at the end of every test. On average, my error is around 0.31, my standard deviation is about 0.41 and my RMS error is about 0.51. I get the impression that these are not the best statistics to have (this seems to imply that my algorithm usually gets around +1 or -1 for a range, but is usually mostly correct), but they are not wildly inaccurate.

Benchmarking

Using the Time.now command to benchmark the progress of the algorithm, it seems that for every 1,000 predictions the algorithm makes, it takes anywhere between 30 to 50 seconds to complete them (less if the algorithm runs over the same data in consecutive prediction runs). On average, the algorithm takes 12 minutes to complete when tasked with the full 20,000 lines in the tests. This is incredibly slow, and certainly not scalable. If the number of lines increased by a factor of 100, it would take 20 hours for the algorithm to complete the predictions. If the number of lines increased by a factor of 1,000, it would take five days before you could expect a result. Again, I reiterate that this algorithm is not scalable.