Username: gtrivell

Rank: 3

RMSE score: 0.88

**Approach**

My first step was to create the movie profiles matrix and the user profiles matrix. To make these, I went through all the movie\_ prefix files, and for each new index, I created a profile, adding actors, directors, tags, and genres. This process was simple enough bar for two issues:

* The movie\_ data files did not have the same number of movies. Some movies didn’t have tags or actors for example. I didn’t know this when creating my program, so when making the index array, some movies are missing because I processed movie\_actors.dat first.
* Dealing with actor rankings and tag frequency. I had the initial thought that I could add an actors name to the profile by the ranking (If the rank of tom\_hanks was 5, 5 instances of ‘tom\_hanks’ was appended). I thought that TFIDF could handle this by recognizing that these actors that show up often are not relevant enough to be impactful on data. However, this was a time complexity issue because when adding actors when there are 50+ of them, profiles tend to get large. I decided to only include the top 5 actors to simplify things.

User profiles are essentially class labels for movies. A 0 if the user has no rating for it, otherwise the rating. Again, I had two issues:

* Continuing off the first issue above, since some movie IDs were not in my database (I am not sure whether they were present in the other files or just missing), I ran in to an issue where my loop would keep parsing the movie indexes until the end, adding thousands of extra 0’s. I had to add a check to ensure that the movie was present, but this unfortunately increased the time it took to create this matrix from 5 minutes to about 20.
* When attempting to use classifiers, I kept running in to the same error: ValueError: setting an array element with a sequence. It took me a while to figure out that once my loop found the movie index of the last rating in the training file, it didn’t fill out the rest of the user’s profile with 0’s, meaning that user’s profile had less than the total needed features. A simple fix was added after the loop to fill out the last profile.

From there, to make the profiles usable by packages, I recreated it to be an array of numpy arrays, where each numpy array is a profile of doubles.

**Methodology/Features**

I ended up with 10174 movie profiles, each profile being a string that looked like a description (ex. ‘kate\_winslet kathy\_bates leonardo\_di\_caprio michael\_shannon sam\_mendes Drama Romance chuck\_palahniuk’) and 2113 user profiles (ex. [0.0 , 0.0 , 3.1, …]) with the number of movies being the feature count. I had 3 packages that I attempted to use. Non-Negative Matrix Factorization (NMF), Stochastic Gradient Descent (SGDClassifier), and Multi-layer Perceptron regressor (MLPRegressor).

**What Worked and What Didn’t**

First off, I couldn’t format my data correctly to work without throwing an error with SGDClassifier. I quickly gave up on using that option. I tried to use NMF, however, the run time seemed like it would take days. I changed the parameters a bit so that it would take only an hour and a half, but the array resulting from it seemed to be incorrect. Too many values ended up as 0 and it resulted in last place on miner.

Luckily, the final package I tried seemed to work correctly, however, without parameters, it would also take days. I tweaked the parameters to still be accurate but take way less time though still far from desirable. It takes about 7 hours to finish every line in the test file.

Additionally, since I ran out of time dealing with packages that don’t work and that take obscene amounts of time, I couldn’t fix the issue with movies that aren’t present in the database. Ideally, if a new movie or new user is found, my program would find a similar movie/user through clustering, KNN, etc. and get the rating from there. As it is, it just adds an average rating of 3.0 instead.

With MLPRegressor taking as long as it does, I would assume that it just isn’t the correct algorithm for this problem. Using clustering exclusively would likely result in a much, much faster program with decent accuracy. It could even be using in two ways, once by clustering on similar users and once by clustering on similar movies.