Aziz Amino, Andrew Park (mason id: aamino / apark19, miner id:5-3AAA\_2Z / johndoe1, rank: N/A)

CS 484

Prof. Lin

HW#4 report

**Pre – Processing**

Sheer volume in data had created an urgency within our team to optimize for efficiency and avoid costly algorithms. Attempts at this include utilization of the multiprocessing. Pool library as well as exploring the Cython language. Ultimately, this prioritization had cost more overall time than using costly algorithms, which led our team unable to produce a miner submission.

Libraries used to read and extract the information from the csv and dat files include standard csv and pandas. While pandas dataframes were useful in reading data from columns of the dat files, using these objects had made creating utility and item matrices difficult and been extremely costly. The most simplistic and efficient way of exporting the utility and item matrices was for us to use the read\_csv() and writerow() methods, allowing for creation of a matrix by writing lists as rows.

Major issues had appeared when reading and comparing strings from the files due to character encoding, switching the open functions encoding parameter to various key words such as utf-8, utf\_escape, and ascii. In part, this was due to code portability for python 2.7 and python 3.7 (constantly switching to evaluate which version had supported our source files the best). One of our machines had also experienced problems reading the large file sizes of both item and utility csv files, necessitating reliance one as opposed to both of us being able to test the predictor algorithm.

Attribute selection was based on all unique director names gathered form the movie\_directors dat file, all genres that appeared in the movie\_genres file, and actors with rank three or lower (originally using all actors however, later on realizing the purpose of rank). The attribute selection had reflected in the production of the item and utility matrices having ~17916 columns. Using tag weights was not attempted due to potential added complexity when creating an evaluation metric (though time permitting, it would have been considered for incorporation into the matrices).

The data was used to create binary values, depending on whether the actor, director, or genre had appeared in the data. Upon reflection, the matrices had appeared very sparse and some method in aggregating the data may have reduced overall runtime.

**Methods**

The general idea for the predictor algorithm was to create a local K-NN evaluator based on user history (the attempt at this implementation was performed the source file rater.py). The user-movie pair would be collected from the test.dat file, inserted into the predictors list, and looped through before being passed to the predict\_rate() function. Within each iteration of the loop through the predictors list, the user\_his() function is called to collect the user history vectors from the utility matrix and passed to the predict\_rate() function along with the current user movie pair.

The predict\_rate() function was intended to process the user history and fitted to a sklearn NearestNeighbors object with K chosen by obtaining the square root of N samples from the user history rounded down. The nearest neighbor indexes would then have been collected and extracted from the user\_his() return value, which is a list. The ratings of these nearest neighbors would be averaged and returned as the predicted rating for the user-movie pair in the test set.

**Results**

No results to miner were submitted, due to incomplete code. The run-times for the matrices had taken several hours, however, was significantly reduced when the number of attributes was reduced from ~98,000 to 17916.

|  |  |  |
| --- | --- | --- |
|  | 98160 columns | 17916 columns (reduced) |
| Item Profiles | ~ 16 minutes | ~ 13 minutes |
| Utility Matrix | ~ 5000 sec per 50,000 chunk (size of utility matrix was too large to process normally) | ~ 5 hours |