CS 333 2021 Homework [5]

Net ID: [JET48]
Name: [John Thomas]

Problem 1

a

(a) What is the highest rated movie that has been rated by at least 100 users? You can just give the movie id, but you can also look up the title in the movies.txt file. What is its average rating?
The movie ID of the highest rated movie that has been rated by at least 100 users is 318. The movie is Schindler's List, and it has an average rating of 4.507177033492823

b

(b) What is the lowest rated movie that has been rated by at least 100 users? You can just give the movie id, but you can also look up the title in themovies.txt file. What is its average rating?
The lowest rated movie that has been rated by at least 100 users in Batman Returns, it has an average rating of 2.6601941747572817

c

(c) What is the average number of movies that a user has rated? Call this value m; we will reference it in subsequent problems.
average movie rates per user = 3.90625

Problem 2

a

I was able to implement this efficient algorithm in my code by performing preprocessing upon one iteration through the ratings file. Here are all the maps I created with their definitions written above
static ArrayList<String> movies = new ArrayList();
static ArrayList<String> users = new ArrayList();
static HashMap<String, Integer> u1u2bothratedcount = new HashMap();
//ratings stores list of all ratings from ratings.csv

```
static ArrayList<String> ratings = new ArrayList();
//testratings stores list of all ratings from testratingscsv
static ArrayList<String> testratings = new ArrayList();
//moviesover100ratings = list of movies rated over 100 times in training data
static ArrayList<String> moviesOver100ratings = new ArrayList();
//avgratings stores key = movieid, value = average rating for that movie across the training
testratings file
static HashMap<String, Double> avgratings = new HashMap();
//totalratings stores key = movieid, value = sum of all ratings of that movie
static HashMap<String, Integer> totalratings = new HashMap();
//movieratingsCount stores key = movieid, value = count of ratings for that movie
static HashMap<String, Double> movieratingsCount = new HashMap();
//useratingscount stores key = userid, value = ratingscount
static HashMap<String, Double> userratingsCount = new HashMap();
//user2moviesrated stores key = userid, value = list of movies rated
static HashMap<String, ArrayList<String>> user2moviesrated = new HashMap();
//movie2usersrated stores key = movieid, value = list of users who rated that movie
static HashMap<String, ArrayList<String>> movie2usersrated = new HashMap();
//useridmovieid2rating stores key = userid + movieid, value = rating that user gave that
movie
static HashMap<String, Integer> useridmovieid2rating = new HashMap<String, Integer>();
```

Problem 3

b

```
movie: 484—Maltese Falcon, The (1941)—01-Jan-1941——http://us.imdb.com/M/title-exact?Maltesepred
4.7
movie: 654—Chinatown (1974)—01-Jan-1974——http://us.imdb.com/M/title-exact?Chinatownprediction
4.7
movie: 408—Close Shave, A (1995)—28-Apr-1996——http://us.imdb.com/M/title-exact?Closeprediction:
4.7
movie: 56—Pulp Fiction (1994)—01-Jan-1994——http://us.imdb.com/M/title-exact?Pulpprediction:
4.653846153846154
movie: 480—North by Northwest (1959)—01-Jan-1959——http://us.imdb.com/M/title-exact?Northpredic
4.642857142857143
```

c

movie: 169—Wrong Trousers, The (1993)—01-Jan-1993——http://us.imdb.com/M/title-exact?Wrongprediction: 4.7
movie: 64—Shawshank Redemption, The (1994)—01-Jan-1994——http://us.imdb.com/M/title-exact?Shawshankprediction: 4.694444444444445
movie: 483—Casablanca (1942)—01-Jan-1942——http://us.imdb.com/M/title-exact?Casablancapredictio
4.642857142857143
movie: 318—Schindler's List (1993)—01-Jan-1993——http://us.imdb.com/M/title-exact?Schindler'spredic
4.633333333333334
movie: 523—Cool Hand Luke (1967)—01-Jan-1967——http://us.imdb.com/M/title-exact?Coolprediction:
4.625

d

The k variable definitely has a greater effect on runtime because the only thing we use the r for is to trim the return list at the end of the algorithm.

e

The smooth prediction includes the baseline user rating of 3.5 into the average for moviej
When we calculate the smoothed prediction for moviej, if moviej isn't in N_j then the rating for moviej should be 3.5

Problem 4

a

RMSE for these predictions = 1.05
Took 120 min to compute

b

for user-item pairs in testratings.csv
RMSE = 1.1713240371477085

Baseline RMSE = 1.1713240371477085

C

To improve the performance of the model, in general we want to increase the size of the k-neighbors while also giving an increased weight to the ratings of the k-neighbors with the higher CRD, as the closer neighbors should have more of a relative impact on the prediction outcome