Junxu Chen, ID: 467525 repository: yihenghuang

Nikki Xie, ID: 448385

Shuang Wei, ID: 467487

Yiheng Huang 452257

milestone 2 (20%)

- project management (*every* team member is required to *participate* in the discussion)
    - conceptual understanding (5%)
    - understanding of roles and individual responsibilities (5%)
- technical (*check project instructions for more details*)
    - data preprocessing and other steps completed (10%)

Problem 2
(a).
Distinct items: 1701
Distinct Users: 27555

(b).
estimated average overlap of items for users:

```
File1 = sc.textFile("file:/home/cloudera/netflix_subset/TestingRatings.txt")
movie_counts=File1.map(lambda x:(x.split(",")[1],1)).reduceByKey(lambda v1,v2:v1+v2)

File2 = sc.textFile("file:/home/cloudera/netflix_subset/TrainingRatings.txt")
rating_counts=File2.map(lambda x:(x.split(",")[0],1)).reduceByKey(lambda v1,v2:v1+v2)

final=File1.map(lambda x:(x.split(",")[0],x.split(",")[1])).join(rating_counts).values().reduceByKey(lambda v1,v2:v1+v2).join(movie_counts).mapValues(lambda v:v[0]/v[1])

final.take(5)
```

Junxu Chen, ID: 467525 repository: yihenghuang

Nikki Xie, ID: 448385

Shuang Wei, ID: 467487

Yiheng Huang 452257

[(u'1014928', 7599), (u'1942739', 10055), (u'1006655', 19085), (u'2065448', 12451), (u'2490923', 8278)]

estimated average overlap of users for items:

```
File1 = sc.textFile("file:/home/cloudera/netflix_subset/TestingRatings.txt")
user_counts=File1.map(lambda x:(x.split(",")[0],1)).reduceByKey(lambda v1,v2:v1+v2)

File2 = sc.textFile("file:/home/cloudera/netflix_subset/TrainingRatings.txt")
rating_counts=File2.map(lambda x:(x.split(",")[1],1)).reduceByKey(lambda v1,v2:v1+v2)

final=File1.map(lambda x:(x.split(",")[1],x.split(",")[0])).join(rating_counts).values().reduceByKey(lambda v1,v2:v1+v2).join(user_counts).mapValues(lambda v:v[0]/v[1])

final.take(5)
[(u'12482', 150), (u'11829', 132), (u'8307', 195), (u'5656', 118), (u'14079', 622)]
```

The average overlap of items rated by the users in the training set for users in the test set is higher because base on the data of question a, the number of distinct items is much smaller than the number of distinct users.

(c).

For the given evaluation task, we have larger user set and smaller item set.

Quality-wise:

-The item-item based recommendation method will have better quality. For the user-based method, when the data for rated movies is sparse, we can

hardly find similar users, and thus is likely to recommend wrong items to users. However, for item-based method, for a given item, it calculates the similarity(the possibility of co-occurrence of two items) between two items, so even when the data for rated movies are sparse, or when recommending movies to a user with distinct taste, we can still manage to recommend the right movie. It searches users who watch this item and based on these users preferences to filter the items that other users may like. This method will give the better prediction for users.

Efficiency-wise:

-The latter method will be more efficient. Since it is item-item similarity with number of query users>>number of items. Number of reduce tasks and work to shuffle and sort will be smaller comparing to the former method.

-item-based recommendation is more efficient because the item-item similarity can be calculated offline. The movies are less dynamic and don't change as much as users' states, so we can calculate the item-item similarity table when offline, thus saves time and space for calculating the similarity and increase the efficiency. And the scalability of this approach makes it easy to recommend items to new users.

(d).

```
File1 = sc.textFile("file:/home/cloudera/netflix_subset/TestingRatings.txt")
Rating_count = File1.map(lambda x:(x.split(",")[0],1)).reduceByKey(lambda v1,v2:v1+v2)
Total_rating = File1.map(lambda x:(x.split(",")[0],x.split(",")[2])).reduceByKey(lambda v1,v2:v1+v2)
Avg_rating = total_rating.join(rating_count).mapValues(lambda v: v[0]/v[1])
```

We get (movie_ID, average rating of this movie)