Junxu Chen 467525 **Nikki Xie 448385** **Shuang Wei 467487** **Yiheng Huang 452257**
repository: yihenghuang

Problem 1:

We are planning to use Pearson Correlation with weighted normalized ratings. Item-item approach is more efficient in our opinion, due to the fact that we have much more users than items, and the number of similar users should be set to 5.

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). We expect the average overlap of items to be higher because for this particular question, we are given a larger pool for users than for items. So it is possible for us to get a lot of clustering between user's preferences, making overlap for users high.

estimated average overlap of items for users - commands:

if we only use a small data set, then for the next step, wo copy the first 5 lines of the data set to a new txt file and import the new txt.

```
//import the small testing rating file
File1 = sc.textFile("file:/home/cloudera/netflix_subset/TestingRatings.txt")
```

//count the number of movies one user has rated, the result will be [user1, n], where n is how many movies the user has rated
movie_counts=File1.map(lambda x:(x.split(“,”)[1],1)).reduceByKey(lambda v1,v2:v1+v2)

//we import the training ratings
File2 = sc.textFile(“file:/home/cloudera/netflix_subset/TrainingRatings.txt”)

//similarly, we count the frequency of one particular movie has been rated, the result will be [movie1, n] where n is the frequency of this movie being rated.
rating_counts=File2.map(lambda x:(x.split(“,”)[0],1)).reduceByKey(lambda v1,v2:v1+v2)
//we join the two tables and calculate the overlap with sum of ratings / number of movies user1 rated.
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])

//take the first 5 results, the format is [user1, overlap]
final.take(5)
(u'1014928', 7599),
(u'1942739', 10055),
(u'1006655', 19085),
(u'2065448', 12451),
(u'2490923', 8278)

The average overlap of the five picked users are 11493.6

estimated average overlap of users for items:

if we only use a small data set, then for the next step, wo copy the first 5 lines of the data set to a new txt file and import the new txt.

//import the testing rating file.

```
File1 = sc.textFile(“file:/home/cloudera/netflix_subset/TestingRatings.txt”)

//count the number of movies user1 has rated
user_counts=File1.map(lambda x:(x.split(“,”)[0],1)).reduceByKey(lambda
v1,v2:v1+v2)

//import training rating file
File2 = sc.textFile(“file:/home/cloudera/netflix_subset/TrainingRatings.txt”)

//count the frequency of one movie being rated by different users
rating_counts=File2.map(lambda x:(x.split(“,”)[1],1)).reduceByKey(lambda
v1,v2:v1+v2)

//join two tables and calculate the overlap for users
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])

//the results will be in the format [movie1, overlap]
final.take(5)
```

```
(u'12482', 150),
(u'11829', 132),
(u'8307', 195),
(u'5656', 118),
(u'14079', 622)
```

The average overlap of the picked 5 users for items is 243.4

The average overlap of items is higher because 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 user-based recommendation method will generally have more accurate recommendation because as the data shows, there's a higher overlap for users. And for this case, the users may contain much more features that we can learn from than from the items. So we can recommend more accurately to users.

Efficiency-wise:
-The latter method will be more efficient because the queries of users is way larger than the pool for items. Therefore, it takes much more time to calculate the user similarity than item similarity. Number of reduce tasks and work to shuffle and sort for item-item similarity calculation will be smaller comparing to user-based method
-item-based recommendation is more efficient also because the item-item similarity can be calculated offline when we have streaming data. 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).
We decide using normalized ratings. We use the following commands to preprocess our data.

```
//import the original data
File1 = sc.textFile("file:/home/cloudera/netflix_subset/TestingRatings.txt")

//calculate the number of ratings by each user
Rating_count = File1.map(lambda x:(x.split(",")[0],1)).reduceByKey(lambda
v1,v2:v1+v2)

//calculate the total total rating for each user
Total_rating = File1.map(lambda
x:(x.split(",")[0],x.split(",")[2])).reduceByKey(lambda v1,v2:v1+v2)
```

Junxu Chen 467525 **Nikki Xie 448385** **Shuang Wei 467487** **Yiheng Huang 452257**
repository: yihenghuang

```
//calculate the average rating
Avg_rating = total_rating.join(rating_count).mapValues(lambda v: v[0]/v[1])
```

We get (movie_ID, average rating of this movie)