CSC385 Data Structures and Algorithms Semester Project (100 points)

Due Sunday May 5th, 11:59 pm.

Introduction

For the semester project, you will be implementing an item-based recommendation system. Recommendation systems are widely used on the web for recommending products and services to users based on their past actions and interaction with the system. Recommendation systems have important applications in several areas, such as:

- *Product recommendation.* Amazon and ebay provide recommendations to you based on your purchase history. Facebook recommends friends. Dating websites recommend dating partners, etc.
- *Movie recommendation:* Netflix offers its customers recommendations of movies they might like. These recommendations are based on ratings provided by users. In fact, the importance of predicting ratings accurately is so high, that Netflix offered a prize of $1M to the first algorithm that could beat its own recommendation system by 10%.
- *News Articles:* news services recommends news articles to the readers based on the articles that they have read in the past.

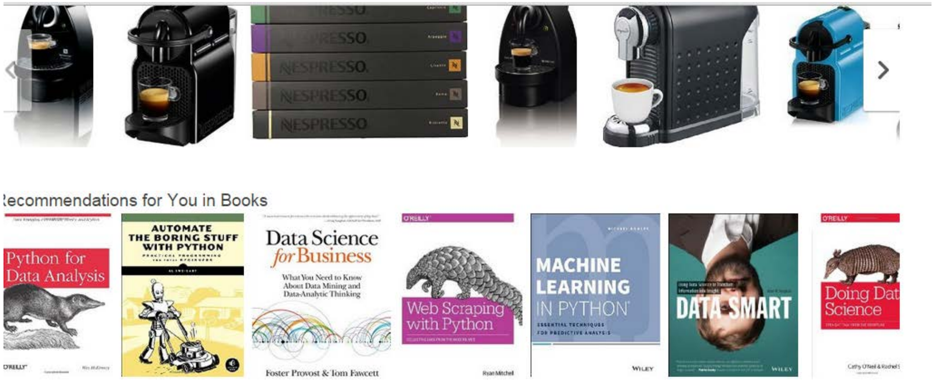

ecommendations for You in Books
Python for Data Analysis
AUTOMATE THE BORING STUFF WITH PYTHON
Data Science for Business
Web Scraping with Python
MACHINE LEARNING IN PYTHON
DATA SMART
Doing Data Science

Figure 1-My recommendation on Amazon—I have recently searched for nespresso and also bought a book in data science with Python. Check out my recommendations!!

What is the input to recommender system?

The input to your recommender system is a dataset consisting of 100,000 ratings from about 1000 users on about 1700 movies. The dataset is extracted from here: http://files.grouplens.org/datasets/movielens/ml-100k.zip

There are two input files attached to this assignment:

1. **movies.dat:** Each line in the movies.dat represents a unique movie and has the following format:

```
movie id | movie title | release date | video release date |etc.
```

For this project, we only need to parse the first two attributes, movie_id, and movie_name. You can extract these attributes using the String. You can use the **split** method in the String class in java standard library to extract the fields that are separated by “\\|” delimiter.

2. **ratings.dat**: All ratings are contained in the file ratings.dat. Each line of ratings.dat has the following format (the fields are separated by tab):

```
UserID     MovieID    Rating     Timestamp
```

Where:
-UserIDs: the id of the user who gave rating
- MovieIDs: the id of the movie for which the user gave rating
- Ratings: a number in the scale (1-5) given to movie_ID by user_ID.

For example, the following record means user id 1 gave rating 5 to the item id 1193 (the last entry is the time stamp which is not used in this project)

```
1     1193  5     978300760
```

What should be the output of recommender system?

Your recommender system will predict the ratings that the user will give to the movies he/she has not rated yet. The output of your recommender system should be a file that contains the top 5 movie recommendations for every user in a descending order of their predicted ratings (i.e., the first movie recommendation should have the highest predicted rating). The format of the output file should be as follows:

```
UserID     MovieTitle1::predicted rating |  MovieTitle2::predicted
rating | MovieTitle3::predicted rating | MovieTitle4::predicted rating
| MovieTitle5::predicted rating
```

For example,

```
user ID: 1 top 5 recommendations: Cyclo (1995)::4.382758930148253| Office Killer
(1997)::4.24082725472752| Little City (1998)::4.234925942215377| Death in Brunswick
(1991)::4.224473123463171| Mamma Roma (1962)::4.178130372636395|
```

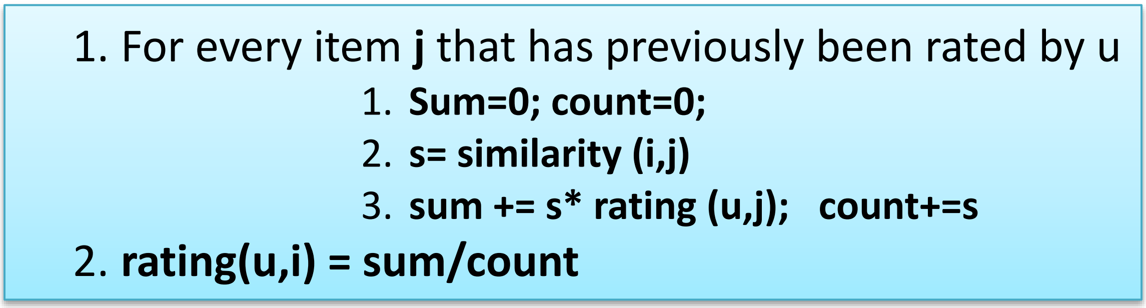
This means that the top 5 movie recommendations for user ID 1 are: cyclo (1995) with predicted rating 4.38, office killer (1997) with predicted rating 4.2, etc.

I have attached a copy of the output I get when I run my program. You can compare your output against mine to test your program.

Predicting movie ratings using an item-based recommender system?

An item based recommender system predicts a rating that userId (u) will give to itemId (i) based on the ratings that the user has previously given to items that are similar to i. More specifically, to predict rating(u,i), an item-based recommender iterates through all items that have been previously rated by user u and takes the weighted average of the ratings that the user gave to such items . The rating of item j is weighted by the similarity of item j to item i. That means the more similar item j is to the target item i, the more its rating weighs in predicting the rating of item i.

This is explained in the following algorithm:



```
1. For every item **j** that has previously been rated by u
        1. Sum=0; count=0;
        2. s= similarity (i,j)
        3. sum += s* rating (u,j);   count+=s
2. rating(u,i) = sum/count
```

Hence,

$$\boldsymbol{rating(u,i)} = \frac{\sum_j \boldsymbol{rating(u,j)*similarity(i,j)}}{\sum_j \boldsymbol{similarity(i,j)}} \quad \textbf{(1)}$$

There are various metrics to measure the similarity between two items. The one that we use for this project is the *Cosine similarity* between the two item vectors.

Suppose that we represent each item as a vector of all the ratings for that item, that is:

$$item_i = (r_{1i}, r_{2i}, r_{3i}, \dots, r_{ni})$$

Where r_{ui} is the rating given by user u to items i and n is the number of unique users in the system. If user u has not rated item i, then the entry $r_{ui} = 0$.

Then the *cosine similarity of* $item_i$ and *of* $item_j$ is calculated as follows:

$$similarity(item_i, item_j) = \frac{item_i.item_j}{\|item_i\|\|item_j\|} = \frac{\sum_{u \in N} r_{ui} * r_{uj}}{\sqrt{\sum_{u \in N} {r_{ui}}^2} * \sqrt{\sum_{u \in N} {r_{uj}}^2}} \quad (2)$$

Where N is the set of all unique users.

To demonstrate the above formula, let us consider a smaller scale example. Suppose you only have 10 movies and 5 user and the ratings form the following table:

	item1	Item2	Item3	Item4	Item5	Item6	Item7	Item8	Item9	Item 10
User1	5		2	3	1		5	4		3
User2		4	3	4		5		5	1	
User3	1	3	5		3		2	4	3	2
User4	3	2		5	2	4	3		1	
User5		3		4	1	5	2			4

Table 1: An example of user-item rating matrix

Note that the matrix is sparse (meaning we have a lot of cells with missing values) as users typically don't rate all the items (movies). Before computing pair-wise item similarity, we replace the missing values with 0.

Now, let us for example compute the similarity between items 4 and 5.

Item 4 and item 5 form the following rating vector (just take the corresponding column)

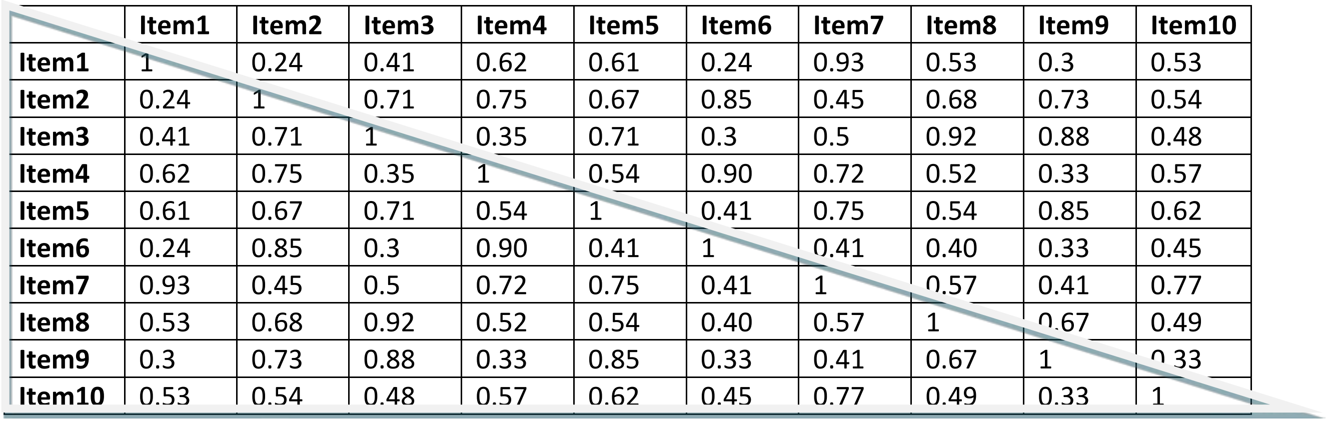
Item4=<3,4,0,5,4>

Item5=<1,0,3,2,1>

Now we can apply equation 2 to compute the similarity between item4 and item5:

Similarity <item4, item5> = $\frac{3*1+4*0+0*3+5*2+4*1}{\sqrt{3^2+4^2+0+5^2+4^2}*\sqrt{1^2+0+3^2+2^2+1}}=\frac{17}{8.12*3.87}=0.54$

In the same way, one can compute the pair-wise similarity between every two items:



	Item1	**Item2**	**Item3**	**Item4**	**Item5**	**Item6**	**Item7**	**Item8**	**Item9**	**Item10**
Item1	1	0.24	0.41	0.62	0.61	0.24	0.93	0.53	0.3	0.53
Item2	0.24	1	0.71	0.75	0.67	0.85	0.45	0.68	0.73	0.54
Item3	0.41	0.71	1	0.35	0.71	0.3	0.5	0.92	0.88	0.48
Item4	0.62	0.75	0.35	1	0.54	0.90	0.72	0.52	0.33	0.57
Item5	0.61	0.67	0.71	0.54	1	0.41	0.75	0.54	0.85	0.62
Item6	0.24	0.85	0.3	0.90	0.41	1	0.41	0.40	0.33	0.45
Item7	0.93	0.45	0.5	0.72	0.75	0.41	1	0.57	0.41	0.77
Item8	0.53	0.68	0.92	0.52	0.54	0.40	0.57	1	0.67	0.49
Item9	0.3	0.73	0.88	0.33	0.85	0.33	0.41	0.67	1	0.33
Item10	0.53	0.54	0.48	0.57	0.62	0.45	0.77	0.49	0.33	1

Table 2—pairwise item similarities for user-item matrix in table 1

Note that the pair-wise similarity is symmetric, that means similarity(itemi, itemj)= similarity(itemj,itemi). So we only need to compute the similarity for half of the table and the other half (highlighted in blue) can be derived from the first half.

Now let us find the top two recommendations for user5. User 5 has not rated item1, item3, item 8 and item9. So let us predict the rating that user5 will give to these items using equation 1:

R(u5=user5, i1= item1) =

$$\frac{r(u5,i2)*sim(i1,i2)+ r(u5,i4)*sim(i1,i4)+ r(u5,i5)*sim(i1,i5)+r(u5,i6)*sim(i1,i6)+r(u5,i7)*sim(i1,i7)+r(u5,i10)*sim(i1,i10)}{sim(i1,i2)+sim(i1,i4)+sim(i1,i5)+sim(i1,i6)+sim(i1,i7)+sim(i1,i10)} =$$

$$\frac{3*0.24+4*0.62+1*0.61+5*0.24+2*0.93+4*0.53}{0.24+0.62+0.61+0.24+0.93+0.53} = 2.83$$

R(u5=user5, i3= item3) =

$$\frac{r(u5,i2)*sim(i3,i2)+ r(u5,i4)*sim(i3,i4)+ r(u5,i5)*sim(i3,i5)+r(u5,i6)*sim(i3,i6)+r(u5,i7)*sim(i3,i7)+r(u5,i10)*sim(i3,i10)}{sim(i3,i2)+sim(i3,i4)+sim(i3,i5)+sim(i3,i6)+sim(i3,i7)+sim(i3,i10)} =$$

$$\frac{3*0.71+4*0.35+1*0.71+5*0.3+2*0.5+4*0.48}{0.71+0.35+0.71+0.3+0.5+0.48} = 2.83$$

R(u5=user5, i8= item8) =

$$\frac{r(u5,i2)*sim(i8,i2)+ r(u5,i4)*sim(i8,i4)+ r(u5,i5)*sim(i8,i5)+r(u5,i6)*sim(i8,i6)+r(u5,i7)*sim(i8,i7)+r(u5,i10)*sim(i8,i10)}{sim(i8,i2)+sim(i8,i4)+sim(i8,i5)+sim(i8,i6)+sim(i8,i7)+sim(i8,i10)} =$$

$$\frac{3*0.68+4*0.52+1*0.54+5*0.4+2*0.57+4*0.49}{0.68+0.52+0.54+0.4+0.57+0.49} = 3.05$$

R(u5=user5, i8= item9) =

$$\frac{r(u5,i2)*sim(i9,i2)+ r(u5,i4)*sim(i9,i4)+ r(u5,i5)*sim(i9,i5)+r(u5,i6)*sim(i9,i6)+r(u5,i7)*sim(i9,i7)+r(u5,i10)*sim(i9,i10)}{sim(i9,i2)+sim(i9,i4)+sim(i9,i5)+sim(i9,i6)+sim(i9,i7)+sim(i9,i10)} =$$

$$\frac{3*0.73+4*0.33+1*0.85+5*0.33+2*0.41+4*0.33}{0.73+0.33+0.85+0.33+0.41+0.33} = 2.73$$

So the top 2 recommendation for user 5 would be first item 8 and then either item 1 or item3.

Notes:

You many not hard-code the number of movies and the number of users. This information must be read from the input file.

Team Work:

You may form a team of two and do the do the final project together. If you like to team up with another classmate, please read the following instructions carefully:

1. **You may use our class slack channel to find another classmate and form a team**
2. **Once you choose your team email me the name and github id of the team members.**
3. **Every team must collaborate and submit their work on github. Submitting on github is mandatory for each team. This is a good practice to experience industry standard collaborative software development.**
4. **I will create github repository and a private slack channel for each team. I will integrate your github repository with your slack channel so you use the channel for private team communication and get notifications of changes on your repository (commit, push, etc.)**
5. **Each team is responsible for distributing the work evenly. There is no guarantee that both team members will receive the same grade. If one team member does all the work and commits everything to the repository, then he/she will not get the same grade as the other**

member who did not commit anything. With every commit, you should include a description of the changes you made in your code

What you need to submit:

1- All your java files. Please make sure to include comments in your code to make it understandable

2- A readme file including a general description of your files.

3- A document explaining the following:

   a. data structures that you used to store movie information and ratings and the algorithm that you used to compute the top 5 recommendation.

   b. Analysis of the order of magnitude efficiency (big –Oh) of your algorithm in terms of the number of users, and movies. Briefly explain how you derived the big-Oh.

If you work in a team, you must submit your files on github and then make an empty submission on blackboard with your github user_ids in the submission ext. If you decide to work individually and not in a team, you have the option to submit your work on blackboard.

Hints to get you started:

Here is a logical flow I would follow to solve this problem:

1- Download and parse the files movies.dat and ratings.dat and store them in appropriate data structures.
(Note the data structure you use should be efficient both in terms of access time and memory usage. A simple 2D array for storing ratings will have a quick access time to each rating but it is very space inefficient because it wastes a lot of memory for unrated movies and hence, it is not an appropriate data structure to store ratings)

2- Compute the similarity between every pair of movies and store the similarity values in a 2D array.

3- For each user u,

   a. Find the set of movies that u has already rated.

   b. For each movie i that u has not rated yet:

      - Use formula (1) to predict the rating that user u will give to movie i, based on the ratings that u has previously given to other movies.

   c. Find the top-5 movies with highest predicted rating for u. (Note, you don't need to sort the entire array of predicted ratings for u to get the top 5)

I strongly suggest that you first test your program on the small example of 5 -users and 10 items provided in table 1. For your convenience I have also provided input/output for this small example.

Grading Rubric:

Program produces correct output	40
Efficient data structures **(both in terms of access time and memory usage)** are used to store user ratings	20
Data is loaded correctly from movies.dat and ratings.dat and The number of movies and users are not hard-coded in the code	10
Similarity table is computed correctly and Only half of the pair-wise item similarities are actually computed (the other half is symmetric)	10
Correct analysis of the running time of the program	10
Top 5 predicted ratings are retrieved efficiently and without sorting the predicted ratings	5
A document is included which explains the data structures and the algorithms used in the code.	5
Total	100