Programming assignment 1: implement a collaborative-filtering based recommender system

In this programming assignment, you are asked to implement an item-based CF algorithm. You also need to implement some metrics to evaluate your algorithms.

Due time:

Wednesday, Oct 21, 14:59pm.

Here is the instruction:

1. You are provided with two files, one of which has the training data (matrix) and the other has the testing data (matrix). These two files follow the format as we discussed in the class. You need to first read in these two files and construct the CSR data representations from the files (10 points). Please note that in the input files, the indexing follows Matlab style, not C style, that is, the first index is 1, not 0.
2. Implement the item-based CF algorithm
    a. For each item, find its **K** most similar items in the training data.
        i. You need to use the cosine similarity function (10 points) to measure item similarities. In order to calculate cosine similarities, you need to first transpose the training matrix (20 points) as it is provided in the training file as a row-majored representation. After the transpose, you can calculate the similarities along its rows. Note that in the given files, all the non-zero values are rating values (i.e., 1, 2, 3, 4, 5 stars). You need to treat them as binary values (i.e., for all the ratings, just consider them as 1s) when you use cosine similarities.
        ii. **K** is a parameter and you have to allow the users of your recommender system to provide certain K values. That is, you have to implement a user interface (5 points) via a command line or a configuration file where the users can input the K values. You need to consider the special cases when an item has only P (P<K) similar items (items with non-zero similarity values).
        iii. You need to **implement** (not using an existing one from libraries) a sorting algorithm (20 points) so that given a set of items (item IDs/indices) and their similarities values with respect to a certain item, the sorting algorithm can find the top K items whose similarities are the largest, and their IDs/indices. Note that you just need to the TOP-K items.
        iv. You need to think about the data structures you use to save the results (i.e, item indices and their similarities with respect to a certain item).
    b. For each user u_i in the training data
        i. Look at all her purchased/rated items { item_j }. For each item_j, find its K most similar un-purchased/un-rated items (10 points) based on your calculation in (a). Assume item_j's K most similar items are item_{j, k}, k = 1, …, K
        ii. For all u_i's rated items {item_j}, put their K most similar items item_{j, k} together into a candidate set A. Then for each item c in A, it is a candidate item for recommendation. Calculate the ranking score of a candidate item c (20

points) in A as the sum of its similarities with respect to u_i's all rated items. That is,

score(c, u_i) = sum_j cosine(item_j, c),

where u_i rated item_j.

You need to think about how to implement ii) efficiently and what data structures you can use. You can implement i) and ii) simultaneously.

iii. Sort all the candidate items based on their scores in decreasing order. You should be able to re-use the sorting algorithm implemented in step (a) to do so. Then find the top **N** ranked items and output them as recommendations for user u_i. Here again N should be a parameter (similar as K) that you should get from the users of your recommender systems via the user interface. You should output the recommendations into an output file (10 points) following this format:

User_ID: item_ID_1 ranking_score_1 item_ID_2 ranking_score_2 …

That is, in each line, the first number is the user id (use Matlab style indexing, that is, the first user has ID 1, the second user has ID 2, …). In the rest of the line, item_ID_1 is the ID (again, Matlab style indexing) of the first (the most similar) recommended item, and ranking_score_1 is its ranking score as you calculated in step ii).

Like K and N, the name of the output file should be an input from the user interface.

c. Implement HR (5 points) and ARHR (5 points) metrics for all the users
   i. For each user, compare your recommendations with the item in the testing data. If the testing item is in your recommended items, then hits++. HR is calculated as the #hits/# users (as in the testing data, each user has only one testing item). You should calculate ARHR correspondingly.
d. Free all the memories in the end of your program (5 points).
e. Bonus credits (20 points):
   i. If you can implement an item-item similarity metric which gives better HR and ARHR
f. Submission:
   i. Source code
   ii. A README.txt file containing following information, each piece in a paragraph
      1. How to compile and run your code (3 points)
      2. Timing for step a) and b), in ms with 2 digits after decimal point
      3. Provide the name of your sorting algorithm and its place in your source code
      4. A description on how you complete step a i) (e.g., computational complexity, algorithms, etc.) and the data structure you used to save the item similarities in step a i) (3 points)

5. The algorithm and data structures you used to find the candidate items and calculate their ranking scores (3 points)
6. If you do the bonus credits, describe your new item-item similarity function in a formula. (3 points)

iii. A results.txt file containing the following information (3 points). If you do bonus credits, a results_bonus.txt file with the same information in the same format.
Information:
For K = 3, 5, 10, 20,

For N = 5, 10, 20

HR value and ARHR value

That is, you need to run your recommendation algorithm for K = 3, 5, 10, 20, and N = 5, 10, 20, and print out the HR and ARHR values in the following format

3 5 HR ARHR

3 10 HR ARHR

3 20 HR ARHR

5 5 HR ARHR

5 10 HR ARHR

5 20 HR ARHR

10 5 HR ARHR

…