CptS 315: Introduction to Data Mining

Homework 2

(Due date: Feb 28th in class)

# Instructions

* Please use a word processing software (e.g., Microsoft word) to write your answers and submit a printed copy to me at the beginning of the class on Feb 28. The rationale is that it is sometimes hard to read and understand the hand-written answers.
* All homeworks should be done individually.

# Analytical Part (40 points)

**Q1.** Consider the following ratings matrix with three users and six items. Ratings are on a

1-5 star scale. Compute the following from data of this matrix: (20 points)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Item 1 | Item 2 | Item 3 | Item 4 | Item 5 | Item 6 |
| User 1 | 4 | 5 |  | 5 | 1 |  |
| User 2 |  | 3 | 4 | 3 | 1 | 2 |
| User 3 | 2 |  | 1 | 3 |  | 4 |

Table 1: Data of ratings from three users for six items.

1. Treat missing values as 0. Compute the jaccard similarity between each pair of users.

|  |  |  |
| --- | --- | --- |
| 1 & 2 | 1 & 3 | 2 & 3 |
|  |  |  |

1. Treat missing values as 0. Compute the cosine similarity between each pair of users.

1 & 2: , , .

1 & 3: , , .

2 & 3: , , .

1. Normalize the matrix by subtracting from each non-zero rating, the average value for its user. Show the normalized matrix.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Item 1 | Item 2 | Item 3 | Item 4 | Item 5 | Item 6 |
| User 1 (3.75) | 0.25 | 1.25 | 0 | 1.25 | -2.75 | 0 |
| User 2  (2.6) | 0 | 0.4 | 1.4 | 0.4 | -1.6 | -0.6 |
| User 3  (2.5) | -0.5 | 0 | -1.5 | 0.5 | 0 | 1.5 |

1. Compute the (centered) cosine similarity between each pair of users using the above normalized matrix.

1 & 2: , , .

1 & 3: , , .

2 & 3: , , .

**Q2.** Please read the following two papers and write a brief summary of the main points in at most TWO pages. (20 points)

Brent Smith, Greg Linden: Two Decades of Recommender Systems at Amazon.com. IEEE Internet Computing 21(3): 12-18 (2017)

<https://www.computer.org/csdl/mags/ic/2017/03/mic2017030012.pdf>

This paper talks about the success and evolution of item-item filtering for recommendation systems (from the perspective of Amazon). I thought it was interesting to read after having implemented my own (much simpler) item-based recommendation system. While I implemented probably the simplest possible system, this paper discussed a lot of details that I hadn’t considered. While I had considered separating recommendations by category rather than purely by combined score, I had not considered the time element that was mentioned in the article. Of course people’s tastes and desires change over time, so possibly the recommendation system could take into account more recent ratings vs the collection of all past ratings.

The paper also mentioned some of the shortcomings of the algorithm, which is the idea that the algorithm can overfit your tastes and keep recommending only things in the same vein, and so the author suggested to add some capability to provide alternate recommendations that are less mathematically ideal in order to diversify the recommendations, which can be done in a variety of ways. The paper mentioned recommending some things based on recent preferences and others based on long-term trends, or recommending items related to recent items purchased rather than overall related to the purchaser.

Finally, the paper discusses some long-term goals and mentions algorithms that are on the horizon that could potentially replace item-item filtering or otherwise disrupt the “status quo”. Machine learning and AI driven approaches are mentioned, though the latter extremely vaguely (digital assistants such as alexa were hinted at). I may have misinterpreted these mentions, though. Perhaps this amounts to a different interface for the results and not a completely new algorithm.

Greg Linden, Brent Smith, Jeremy York: Industry Report: Amazon.com Recommendations: Item-to-Item Collaborative Filtering. IEEE Distributed Systems Online 4(1) (2003) <https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf>

This second paper compares collaborative filtering against other algorithms and explains some of the shortcomings of the competition. Collaborative filtering, like the competition, becomes very computationally expensive as the size of the item set (or users if doing user-item comparison) expands. However, item-item filtering can be done offline. The computation for recommendations is (comparatively) fast, so it can be returned on request. Other methods to speed up ratings, such as clustering users or items, can reduce recommendation quality dramatically since large portions of the data set have been sliced off for performance reasons. Another method treats the user’s purchase history as a search query and returns similar items. This has the benefit of easing a cold-start, but once the user’s history grows, it’s effectiveness declines since the search would be useless if it tried to compare against all items and a subset of the items eliminates valuable datapoints. The paper concludes that item-item comparison provides higher quality recommendations than other algorithms, particularly at scale. Even though the similarity matrix calculations are computationally expensive compared to other recommendation algorithms, these can be done offline, long before they’re needed.

# Programming and Experimental Part (60 points)

**Movie Recommendations via Item-Item Collaborative Filtering.** You are provided with real-data (Movie-Lens dataset) of user ratings for different movies. There is a *readme* file that describes the data format. In this project, you will implement the *item-item collaborative filtering* algorithm that we discussed in the class. The high-level steps are as follows:

1. Construct the profile of each item (i.e., movie). At the minimum, you should use the ratings given by each user for a given item (i.e., movie). Optionally, you can use other information (e.g., genre information for each movie and tag information given by user for each movie) creatively. If you use this additional information, you should explain your methodology in the submitted report.
2. Compute similarity score for all item-item (i.e., movie-movie) pairs. You will employ the *centered cosine* similarity metric that we discussed in class.
3. Compute the neighborhood set *N* for each item (i.e. movie). You will select the movies that have highest similarity score for the given movie. Please employ a neigborhood of size 5. Break ties using lexicographic ordering over movie-ids.
4. Estimate the ratings of other users who didn’t rate this item (i.e., movie) using the neighborhood set. Repeat for each item (i.e., movie).
5. Compute the recommended items (movies) for each user. Pick the top-5 movies with highest estimated ratings. Break ties using lexicographic ordering over movie-ids.

Your program should output top-5 recommendations for each user.

**Instructions for Code Submission and Output Format.**

Please follow the below instructions. It will help us in grading your programming part of the homework. We will provide a dropbox folder link for code submission.

* + Supported programming languages: Python, Java, C++
  + Store all the relevant files in a folder and submit the corresponding zipfile named after your student-id, e.g., 114513209.zip
  + This folder should have a script file named run\_code.sh

Executing this script should do all the necessary steps required for executing the code including compiling, linking, and execution

* + Assume relative file paths in your code. Some examples:

‘‘./filename.txt’’ or ‘‘../hw1/filename.txt’’

* + The output of your program should be dumped in a file named “output.txt” in the following format. One line for each user.

User-id1 movie-id1 movie-id2 movie-id3 movie-id4 movie-id5

User-id2 movie-id1 movie-id2 movie-id3 movie-id4 movie-id5

···

···

**Explanation.**

* + - Line 1 should have the first user-id followed by the movie-ids of recommended movies.
    - Line 2 should have the second user-id followed by the movie-ids of recommended movies.
* Make sure the output.txt file is dumped when you execute the script run\_code.sh
* Zip the entire folder and submit it as

<student\_id>.zip

# Grading Rubric

Each question in the students work will be assigned a letter grade of either A,B,C,D, or F by the Instructor and TAs. This five-point (discrete) scale is described as follows:

* **A) Exemplary (=100%)**.

Solution presented solves the problem stated correctly and meets all requirements of the problem.

Solution is clearly presented.

Assumptions made are reasonable and are explicitly stated in the solution.

Solution represents an elegant and effective way to solve the problem and is not overly complicated than is necessary.

* **B) Capable (=75%)**.

Solution is mostly correct, satisfying most of the above criteria under the exemplary category, but contains some minor pitfalls, errors/flaws or limitations.

* **C) Needs Improvement (=50%)**.

Solution demonstrates a viable approach toward solving the problem but contains some major pitfalls, errors/flaws or limitations.

* **D) Unsatisfactory (=25%)**

Critical elements of the solution are missing or significantly flawed.

Solution does not demonstrate sufficient understanding of the problem and/or any reasonable directions to solve the problem.

* **F) Not attempted (=0%)**

No solution provided.

The points on a given homework question will be equal to the percentage assigned (given by the letter grades shown above) multiplied by the maximum number of possible points worth for that question. For example, if a question is worth 6 points and the answer is awarded a *B* grade, then that implies 4.5 points out of 6.