Coding Assignment 1

Due date: Feb 26, 2023

Overall objectives. Understand the structure of a real-world recommendation dataset and conduct basic analyses; Implement several similarity functions and build a simple memory-based recommender.

Expected submissions. You will need to submit a zipped file to BrightSpace (preferably in .zip formate) which contains the following files as your solution to this coding assignment:

- **Python files**. For each question, you will have a corresponding python file template to solve the problem. You should double check to make sure that these python files are executable with no errors (e.g. by running 'python example.py' using the command line tool') and that they output the results you expect.
- **An answer sheet (in PDF)**. This file documents your full answers to all questions. You can use Latex, Word or any of your favorite tools to generate this PDF. In certain questions that ask for a plot, you shall also put the figures generated from your code in this document. You will also use this to document your answers to the open questions.

Question 1

We will start by exploring a well-known recommendation dataset, MovieLens-1M and calculate some useful statistics from the data. Take a look at *README.txt* would help you understand the details about this dataset. A file called *rec_dataset.py* is given as a template class file and you will need to fill in the missing functions. An example of how these functions will be tested is given at the end of this file, and your code shall work in general for other test examples as well.

(a)*(5pt)* Implement the *describe* function that generates a description of the dataset in terms of (1) the number of unique users and items (2) total ratings (3) minimum and maximum number of ratings for any item.

(b)*(5pt)* Implement the *query_user* function that takes an user ID as input and outputs (1) the number of ratings of the target user and (2) the averaged ratings score by this user. For this question, please write down the answers for the two users: *ID=100* and *ID=381*.

(c)*(10pt)* Now we want to look closer into the rating behaviors of different age groups. Implement the *dist_by_age_groups* function that describes the distribution of (1) number of ratings and (2) averaged ratings scores for the *seven* age groups ('Under 18' to '56+'). Use a bar plot to illustrate the two distributions with the X-axis showing the seven age groups of users and the Y-axis showing the number of ratings/averaged rating scores for each group. For this question you will need to use both *ratings.dat* and *users.dat*.

Question 2

By far we have some basic understanding of the dataset, we now continue to implement a few similarity functions introduced from our lecture. A file called *similarity.py* is given as a template class file and you will need to fill in the missing functions.

(a) ***(5pt)*** Implement the *jaccard_similarity* function according to this formula:

$$Jaccard(i,j) = \frac{|U_i \cap U_j|}{|U_i \cup U_j|}$$

What would be the Jaccard Similarity of the following movies:

- ID=1 and ID=2 (Toy Story 1 and Jumanji)
- ID=1 and ID=3114 (Toy Story 1 and Toy Story 2)

(b) ***(10pt)*** Implement the *cosine_similarity* function according to this formula:

$$CosSim(i,j) = \frac{R_i \cdot R_j}{|R_i| \cdot |R_j|}$$

You can simply use the raw user ratings to transform it into the vector representation of R_i and R_j. Similarly, compute the Cosine Similarity of the same movie pairs in (a).

(c) ***(10pt)*** Implement the *pearson_similarity* function according to this formula:

$$PearsonSim(i,j) = \frac{\sum_{u \in U_i \cap U_j}(R_{u,i} - \overline{R_i})(R_{u,j} - \overline{R_j})}{\sqrt{\sum_{u \in U_i \cap U_j}(R_{u,i} - \overline{R_i})^2}\sqrt{\sum_{u \in U_i \cap U_j}(R_{u,j} - \overline{R_j})^2}}$$

Again, compute the Pearson Similarity of the same movie pairs in (a).

(d) ***(5pt)*** Open question: which of these three functions works best for the above two pairs of movies?

Question 3

Now we can start to implement our first intuitive recommender which hopefully can provide some good recommendations to these users. A file called *memory_recomender.py* is given as a template class file and you will need to fill in the missing functions. You might also need to use certain functions implemented in the previous questions.

(a) ***(20pt)*** Use the similarity functions implemented in Question 2 to predict ratings scores and perform top-k recommendations based on the following formula:

$$r(u,i) = \frac{\sum_{j \in I_u \setminus \{i\}} R_{u,j} \cdot Sim(i,j)}{\sum_{j \in I_u \setminus \{i\}} Sim(i,j)}$$

We will generate **top-5** recommendations for user ID=381. Please try both **Cosine Similarity** and **Pearson Similarity** as the similarity function. (*Hint: you will need to first predict the ratings scores for all movies that the user has not watched yet, rank them from highest to lowest and select the top-k*).

(b) ***(20pt)*** Examine how would the top-5 recommendations change if we use *Cosine Similarity* as the simulation function but with the *user-based* weighting strategy:

$$r(u,i) = \frac{\sum_{v \in U_i \setminus \{u\}} R_{v,i} \cdot CosSim(u,v)}{\sum_{v \in U_i \setminus \{u\}} CosSim(u,v)}$$

(c) ***(10pt)*** Open question: Evaluate the recommendation lists generated by different weighting strategy (*item-based* or *user-based*) and the choice of similarity functions (*jaccard, cosine, or pearson*). You can pull this user's rated movies and compare them with the recommendations to determine which setup is the best one based on your own judgment.