Coding Assignment 1

Due: 11:59PM, March 3, 2025

Overall objectives Understand the structure of a real-world recommendation dataset by creating train/testing split and conducting basic analyses; Implement similarity functions and memory-based recommendations; Evaluate recommendation performance with retrieval metrics.

Expected submissions You will need to submit a zipped file to BrightSpace, which contains the following files as your solution to this coding assignment:

- **Python files (.py)**. 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 and that they output the results aligned with your answers. This document shall be submitted to BrighSpace as part of the grading process.
- **Answer sheet (.pdf)**. This file documents your 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. This document shall be submitted to GradeScope as part of the grading process.

Important notes:

- Late submission will not be graded.
- Add reference for any code or discussions involved in completing this assignment.

Problem 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 *userID=2025*.

(c) ***(5pt)*** 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*.

Problem 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:

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

Compute the Jaccard Similarity of the following movies:

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

(b) ***(5pt)*** 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. Compute the Cosine Similarity of the same movie pairs in (a).

(c) ***(5pt)*** Implement the *pearson_similarity* function according to this formula:

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

where $\overline{R_i}$ is the average rating for movie i across all user ratings (that are non-zero). Compute the Pearson Similarity of the same movie pairs in (a).

Problem 3

Now we can start to implement our first recommender. A file called *knn_recomender.py* is given as a template class file and you will need to fill in the missing functions. You might reuse certain functions implemented in the previous problems. In this problem, we will generate recommendations for a particular user, **userID=2025**, with a nearest-neighbor based collaborative filtering algorithm. In the case of rating prediction with the same scores for two different movies, choose either of them shall be fine.

(a)*(10pt)* Use $CosSim$ and $PearsonSim$ in Problem 2 as the similarity functions to predict ratings based on the following:

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

Generate top-10 recommendations for userID=2025 and the corresponding predicted rating scores $\hat{r}_{ui}$. Compare the results from **Cosine Similarity** and **Pearson Similarity**, which similarity function gives better recommendation by eye-checking? (*Hint: you will need to predict the ratings scores for all movies that the user has not watched yet, rank them from highest to lowest and select the top-10 as recommendations.*)

(b)*(10pt)* Examine how would the top-10 recommendations change if we use **Cosine Similarity** as the simulation function but with the *user-based* collaborative filtering:

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

Generate top-10 recommendations for userID=2025 and the corresponding predicted rating scores $\hat{r}_{ui}$. Compare the results from **user-based** and **item-based**, which gives better recommendation by eye-checking?

(c)*(20pt)* Reuse the user-based collaborative filtering implemented from (b), but change the user set U_i to the top-k nearest neighbors in U_i according to $CosSim(u,v)$, where $k = \{10, 100, 1000\}$ (if $|U_i| < k$ just use U_i). Use sklearn.neighbors.NearestNeighbors to implement the k-nearest-neighbor search.
(1) Try brute force and KDTree-based search by tuning the *algorithm* parameter. Draw a table comparing the time efficiency of different k under brute force and KDTree-based nearest neighbor search.
(2) Generate top-10 recommendations for userID=2025 and the corresponding predicted rating scores $\hat{r}_{ui}$ with different k.

Problem 4

Now we want to understand how good are the recommendations from our KNN recommender and quantify the performance using evaluation metrics introduced in the lecture.

(a)*(10pt)* Implement different train/test splits of the Movielens-1M dataset:

- **Leave-One-Last**: Order the dataset by time recency and split it into train/validation/test sets by holding out each user's **last interaction** as the test set. You shall use the second-to-last item as validation set, and the remaining data as the training set.
- **Temporal Global**: Order the dataset by time recency and split the dataset by a specific timestamp cutoff. You shall use the last 10% of interactions from the training split as validation set. In this problem, use the time cutoff of **2002-1-1:00:00:00 GMT**. (*Hint: you will need to transform the Unix timestamps from the dataset to human readable dates, or vice versa.*)

Complete the following table using the above two data splits. You may refer to tab.2 from this paper (`https://arxiv.org/pdf/2007.13237`).

Data split	# Users	# Items	# Interactions (train/valid/test)
Leave-One-Last			
Temporal Global			

Table 1: Statistics of the dataset splits.

(b) ***(10pt)*** Now we would like to try the Approximate Neareast Neighbor search and see how it can speed up the experiments with the ANNOY library. We are going to tune the two main parameters: the number of trees n_trees and the number of nodes to inspect during searching $search\_k$.
(1) fix $n\_trees = 1$, tune $search\_k = \{10, 100, 1000\}$. Compare the time efficiency and top-10 recommendations with Problem 3c.
(2) use the best $search\_k$ from the last question, try $n\_trees = \{10, 20, 50\}$. Compare the time efficiency and top-10 recommendations with Problem 4b.

(c) ***(10pt)*** Using the **Leave-One-Last** data split, fit your KNN recommender on the training set and tune your model (e.g., # of neighbors, user or item based, similarity functions) on the validation set. Report Root Mean Squared Error (RMSE) on the test set R_{test} achieved by your best model and the corresponding model settings.

$$RMSE = \sqrt{\frac{1}{|R_{test}|} \sum_{r_{ui} \in R_{test}} (\hat{r}_{ui} - r_{ui})^2}$$