In this session:

- We'll implement in python item-to-item and user-to-user recommendation.
- We'll try our implementations on a database of movie ratings.
- We'll reflect (optionally: work on) the evaluation of recommenders and scalability of our algorithms.

1 The Data

The Grouplens website collects, among other datasets, data from the movielens movie recommendation site. Check `https://grouplens.org/datasets/movielens/`.

You will see various datasets, from huge to large to smallish. we recommend that for this lab you start with the small one: `http://files.grouplens.org/datasets/movielens/ml-latest-small.zip`. Then, once you have all working, you can try with larger datasets if you want.

The dataset contains four files in csv. You can study the documentation for a description of the four, but for the purpose of this lab you can concentrate on these two:

- `movies.csv`: Contains lines of the form movieId,title,genres. For example, the first movie, with id 1, is Toy Story and looks like:

  ```
  1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
  ```

- `ratings.csv`: Contains lines of the form userId,movieId,rating,timestamp. For example, the first line

  ```
  1,1,4.0,964982703
  ```

  says that user 1 rated Toy Story with a 4.0 at a certain date and time that we do not need to decipher for this lab. According to the documentation, ratings go from 0.5 to 5 with 0.5 resolution.

2 A Recommender skeleton

We give you the skeleton of a python `Recommender` class that reads a couple of files and then can emit recommendations. The parameters of the recommendation function are:

1. A list of pairs `(movieid,rating)`, presumably the ratings of some user who may or may not be in the system.
2. an integer `k` indicating the desired number of recommendations.

The function must return a list of at most `k` movieid's ordered from "most likely to be liked by this user" to "less likely". None of these movies can be in the input list, of course – we do not recommend to a user a movie that s/he has already rated. The list can be shorter than `k` if there are few than `k` recommendations that the algorithm can or should make – you don't want to recommend movies whose predicted rating is 1.0, for example.

Note that the class contains two recommending functions (item-by-item and user-by-user). This is to help what we ask you to do later. In practice, you would have a single function, where you would put perhaps several interesting algorithms and combine their recommendations somehow.

3 To do

1. Implement the two two versions of the recommendation function one that does user-to-user CF, and another that does item-to-item CF.

   Remember: Modularity is good. A long function with complex structure such as loop within loop within loop usually indicates a mediocre programmer.

2. Create a main function that:

   - Initializes a Recommender object reading from the files you have downloaded.
   - Repeatedly, asks for a list of movies and ratings, asks the Recommender to provide recommendations given this list and prints the titles of the recommended movies plus their predicted rating.

   (The main we ask for is very basic. To use it, you open `movies.csv` with an editor next to the python screen, select movies "by eye", look at their id's, then type the list of id's into the python input line. Obviously, you can work on nicer interfaces if you like.)

3. Test your code with various lists of recommendations. Put yourself in the mind of your friend who loves sci-fi. Choose a few movies she does like, some she does not like, imagine her ratings of the movies, prepare a list reflecting this, and see what the recommender functions return. Then put yourself in the mind of your friend who loves animation movies, and repeat. And same for your friend who likes romantic comedies. And same for your grandpa who likes golden-age Hollywood musicals. Whatever.

4. See if the recommendations make sense to you, and if you see any difference in quality between user-to-user CF and item-to-item CF.

5. Optional work 1: Try writing a program that checks your methods by selecting a set of users as training set and the rest as test set, and evaluating the error on the test set. You will find a problem here that does not appear in the usual supervised Machine Learning setting – not all labels are there. Which means you may have to reorganize the class a bit.

6. Optional work 2: Analyze (big-Oh style) the time and memory costs of your recommender functions. Try them on some of the larger datasets in the grouplens set, and see if the scalability is as predicted by your analysis.

7. Optional work 3: What would be the major changes if you did not have ratings in 0..5, but just boolean "liked/disliked" ratings? Or just positive "likes"?

4 Deliverables

Rules: Same rules as in previous labs about working solo/in pairs and plagiarism.

To deliver: You must deliver a zip file contaning 1) the `Recommender.py` class that you completed (plus any other code files you wrote) 2) a report (2-3 pages max) explaining the main difficulties and choices you faced, the tests you performed (please give a few specific examples of movie lists you tried and what kind of results you got), any of the optional work you may have done, and what you think you learned in this session.

Procedure: Submit your work through the Racó as a single zipped file.

Deadline: Work must be delivered within **2 weeks** from the lab. Late deliveries risk being penalized or not accepted at all. If you anticipate problems with the deadline, tell me as soon as possible.