In this session:

- We will explore how machine learning techniques are applied to recommender systems and analyze the behavior and characteristics that these algorithms impart to the recommenders.
- We will complete programs to build recommender systems using collaborative filtering techniques: user-based, item-based and machine learning-based.
- We will compare the proposed recommendation systems with a trivial classifier.
- We will compare the four methods using a heuristic validation based on genres.

1 Understanding the input data

Within this assignment, we will use the **MovieLens dataset**. The raw data is available through the following link. Note: we recommend using the educational and development dataset, specifically the smallest one: ml-latest-small.zip, with a size of 1 MB). In detail, the compressed file contains the following raw data files:

1. movies.csv: Contains a list of movies identified by a unique movieId, a title, and the various genres of the film.
2. ratings.csv: Contains user ratings for a specific movie, ranging from 1 to 5, where 1 indicates that the user did not like the movie and 5 indicates that the user enjoyed the movie.
3. links.csv: Contains a mapping between a movieId and references to IMDb and TMDb databases.
4. tags.csv: Contains tags for different movies. For instance, the film Toy Story has tags like 'pixar,' 'fun,' and 'fantasy,' while the film Dead Man Walking has the tag 'death penalty'.

The first task of this assignment is to understand and provide a statistical description of the data provided. For example, you can examine the distribution of rating, number of unique users or identify the most frequently watched genres historically. To achieve this, we have already prepared a utility function to load the dataset (utils.load_dataset_from_source).

2 Building a trivial recommender

The first recommender system to build in this session is a trivial system that we use as a baseline. In this case, the trivial recommender system will be based on making recommendations by proposing films with the highest ratings. Films with better rankings will be recommended to the users. We ask you to fill in the code for this recommender in the script **trivial_recommender.py**. What is the time complexity of this recommender system? Could you envision the current limitation of this recommender system ?

3 Building an user-to-user recommender

In this section, we ask you to build a user-based collaborative filtering recommendation system. The recommendation engine will estimate the potential interest of a user *a* on an item *s* based on be based on of other user in a set *U* and the degree of similarity between user *a* and users in *U.* In particular, that interest is defined as

$$interest(a,s) = \bar{r}_a + \sum_b w(a,b)\big(r_{b,s} - \bar{r}_b\big), \qquad (1)$$

where $w(a,b)$ is the normalized similarity between user *a* and *b*, $r_{b,s}$ is the rating of movie *s* by user *b* and $\bar{r}_a$ is the average rating of user *a*.

You have to decide how to define *U* and justify it in your report.

Here we ask to you to fill the script **user_based_recommender.py** following the next steps:

- Generate training and validation partitions using the split_users() function. This function will enable you to split the dataset and fold a set of movies for each user as a validation partition (we will work with this partition during all the sections).
- Complete the code in generate_m(). This function should return a data structure M, such that M[user][movie] yields the rating for a user and a movie.
- Complete the similarity function in similarity.py. This function should compute the similarity between two lists. You should determine which metric is better for the proposed problem.
- Complete the user_based_recommender() function. (1) Determine, for the target user, the most similar users using the similarity metric that you proposed during the preliminary activities. (2) Determine the unseen movies by the target user. (3) Generate recommendations for unrated movies based on user similarity and ratings.

4 Building an item-to-item recommender

In this section, we ask you to build an item-based collaborative filtering recommendation system. This recommendation system has a similar behavior to a user-based recommender mentioned before. The recommendation algorithm will predict the rating of a user *a* on an item *s* based on finding similarity between item *s* and other items *s* rated by user *a* in a set *U* and the ratings of these items rated by user *a* in *U*. That prediction is defined as

$$rating(a,s) = \frac{\sum_j rating(a,j) * w(s,j)}{\sum_j w(s,j)}, \qquad (1)$$

where $w(s,j)$ is the normalized similarity between item *s* and *j*, $rating(a,j)$ is the rating of item *j* that user *a* has rated before.

You have to decide how to define *U* and justify it in your report. You may use the same approach as the previous method.

Here we ask to you to fill the script **item_based_recommender.py** following the next steps:

- Generate training and validation partitions using the split_users() function.
- Complete the code in generate_m(). This function should return a data structure M, such that M[movie][user] yields the rating for a user and a movie.
- Complete the similarity function for this recommender in similarity.py. This function may be the same algorithm that you used in the computation of user similarities; it's depended on your decision.
- Complete the item_based_recommender() function. (1) Determine, for the target user, the greatest similarity between rated and unrated items using the similarity metric that you proposed during the preliminary activities. (2) Determine the unseen movies by the target user. (3) Generate recommendations for unrated movies based on items similarity and ratings.

5 Building a K-NN hybrid recommender

The K-Nearest Neighbor algorithm is a popular machine learning technique that utilizes the similarity between data points to make predictions. In the context of recommender systems, both user-based and item-based recommenders employ the K-NN algorithm to compute predictions by using the similarity between users or items, respectively.

The parameter K can be set to include all users or items to improve performance because in recommender systems, obtaining more information about users or items could significantly increase precision. However, you can experiment with different values of this parameter.

In this section, we ask you to build a K-NN hybrid recommender that combines the prediction from the recommendations obtained by both user-based and item-based recommender, using some metrics or simple operations like (sum, multiplication, etc). In particular, we propose a metric defined as

$$metric(s, u2u, i2i) = (u2u_s * i2i_s) * \sqrt{(u2u_s - i2i_s)^2}, \qquad (1)$$

Here we ask to you to fill the script **knn_hybrid-based_ recommender.py** following the next steps:

- Generate training and validation partitions using the split_users() function.
- Determine, for the target user, the recommendations computed by the user_based and item_based recommenders.
- Complete the knn_hybrid_based_recommender() function. (1) Generate new recommendations for unrated movies from user_based and item_based recommenders using the combination metric.

6 Validation based on genres

In this section, we request that you decide which recommender is more suitable based on accuracy and complexity constraints using **validation.py** script. For this purpose, we suggest comparing the top *k* movies retrieved from the four recommenders (rec1, rec2, rec3, rec4) against the validation set that we have folded for each user. To ease this comparison, we have prepared for you the function matrix_ genres() located in the **utils.py** script, which contains the relationship between movies and genres. Given a target user, we ask you to evaluate each recommender system according to the resemblance between the frequencies of each genre in the top *k* movies recommended by the system and the frequencies of each genre for the movies of the target user in the validation set. Please provide a set of experiments with different users; we do not expect an exhaustive evaluation for each user but the evaluation should be statistically robust.

7 Deliverables

*Rules:*1. You should solve the problem with one other person, we discourage solo projects, but if you are not able to find a partner it is ok. 2. No plagiarism; do not discuss your work with others, except your teammate if you are solving the problem in two; if in doubt about what is allowed, ask us. 3. If you feel you are spending much more time than the rest of the group, ask us for help. Questions can be asked either in person or by email, and you'll never be penalized by asking questions, no matter how stupid they look in retrospect.

To deliver: You must deliver a brief report describing your results and the main difficulties/choices you had while implementing this lab session's work. You also have to hand in the source code of your implementations. If the report is done in pairs, only one needs to submit but both names have to be clearly stated in the report.

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

Late submissions risk being penalized or not accepted at all. If you anticipate problems with the deadline, tell me as soon as possible.