Practical work N°1

Recommender Systems

General remarks

1- This lab is an opportunity to review some Pandas library functionalities and discover one way to build a simple recommender system.
2- A report is required to show obtained results with supporting comments.

Part 1: Manipulating Data with Pandas library

Let's consider the database of *movies_metadata.csv* file.

- Create a small DataFrame *small_df* that only has the following features: *title*, *release_date*, *budget*, *revenue*, *runtime*, and *genres*.
- revenue, runtime, and genres.
- Check out the data types of the different features. What is the type of "*budget*"?
- Convert the *budget* feature to *float*. What is the result?
- Create a function *to_float(number)* to convert every number field in budget to *float* and, if that fails, convert it to *NaN*.
- Apply the *to_float* function to all values in the *budget* column, then try converting to float using *pandas astype*. Check out your result by getting the data types for all features.
- Define a new feature called "year" that represents the year of release by using the *datetime* functionality from *pandas* as following:
    - Convert *release_date* into *pandas datetime* format.
    - Extract *year* from the *datetime*.
- What are the oldest movies available in this dataset?
- What are the most successful (*revenue* feature) movies of all time?
- Create a new DataFrame *new* of only movies that earned more than $1 billion.
- Create another new DataFrame *new2* of only movies earned more than $1 billion, but where the budget less than $150 million.

Part 2: Building a simple recommender

Building the simple recommender is straightforward. The steps are as follows:

1. Choose a metric (or score) to rate the movies on.
2. Decide on the prerequisites for the movie to be featured on the chart.
3. Calculate the score for every movie that satisfies the conditions.
4. Output the list of movies in decreasing order of their scores.

The metric

The metric is the numeric quantity based on which we rank movies. A movie is considered to be better than another movie if it has a higher metric score than the other movie.
One of the simplest metrics that can be used is the movie rating. However, this suffers from a disadvantage. The movie rating does not take the popularity of a movie into consideration. So, a movie rated 9 by 100,000 users will be placed below a movie rated 9.5 by 100 users which is not desirable.
Therefore, what we need is a metric that can take into account the movie rating and the number of votes it has garnered. This would give a greater preference to a blockbuster movie rated 8 by 100,000 users over an art house movie rated 9 by 100 users.
One of the metrics taking into consideration the weight, can be expressed as following:

$$\text{Weighted Rating (WR)} = (\frac{v}{v+m} \times R) + (\frac{m}{v+m} \times C)$$

With:

- v is the number of votes garnered by the movie
- m is the minimum number of votes required for the movie to be in the chart
- R is the mean rating of the movie
- C is the mean rating of all the movies in the dataset

The prerequisties

The *movies_metadata.csv* database already has the values for v and R for every movie in the form of the *vote_count* and *vote_average* features respectively.

The variable *m* is in place to make sure that only movies that are above a certain threshold of popularity are considered for the rankings.

The choice of the value of *m* is arbitrary. In other words, there is no right value for *m*. It is a good idea to experiment with different values of *m* and then choose the one that you (and your audience) think gives the best recommendations.

For our recommender, we will use, firstly, the number of votes garnered by the 80th percentile movie as our value for *m*. In other words, for a movie to be considered in the rankings, it must have garnered more votes than at least 80% of the movies present in our dataset.

Indication: use *quantile()* method from python

Another prerequisite that we want in place is the runtime. Only consider movies that are greater than 45 minutes and less than 300 minutes in length.

Calculating the score and output

In addition to *v*, *R*, and *m*:

1. The last value to calculate is *C* which is the mean rating for all the movies in the dataset.
2. Calculate the score.
3. Add the score as a feature to the dataframe, by using *Apply* function.
4. Sort the dataframe on the basis of the score just computed.

5. Output the list of top movies.

Part 3: Implement a user-based collaborative filtering system

Now that you are comfortable with data and its manipulation, you can build more complex recommender systems.

1. Using the data in *ratings_small.csv*, construct the user rating matrix (where the rows represent the users, and the columns the movies).
2. This matrix will be sparse (lot of unknown values), explain why. Propose a way to fill in the missing values and justify it. Implement it to have a matrix without missing values.
3. Calculate the cosine similarity for each pair of users, in order to obtain a user-user similarity matrix.
4. Implement a function to return the *n* nearest neighbors of a given user in the sense of the calculated similarity.
5. Using the preceding function, implement a function that recommends *k* movies—at most—for a given user. Explain how this function works.
6. Propose a procedure to find which value of *n* is appropriate to determine the recommendations for this dataset.