Content-Based Recommendation System

Donici Ionut Bogdan
Student ID: 34257A

The task is to implement from scratch a system recommending movies to actors. You are free to choose the technique at the basis of the recommender system, as well as the strategy to be used in order to populate the utility matrix, provided that you implement at least one of the methods explained during the course.

1 Introduction

For the implementation of this project, I chose to implement the **content-based system**. In a content-based system, we have to construct for each item a profile, which can be represented by a record or a collection of records representing the characteristics of that given item.

An item profile is nothing more than a representation of the salient features of an item in a given context (in our case, movie recommendation). The features should be chosen carefully, giving reasons for the choices, as they have a great impact on the outcome of the recommendation. In our case, it involves recommending movies with similar characteristics to movies that the user has already seen.

2 Dataset

The dataset used was `letterboxd`, with the last download and successful execution on 26/06/2024.

The dataset consists of several tables each of which has an "id" column representing the identifier of a particular movie. Among the various tables we find the following:

- `movies.csv`: represents the films main features, such as
  - `name`: name of the film
  - `minute`: duration of the film in minutes
  - `rating`: the rating that the film has recived
  - `...`

- `genres.csv`: represents the assigned genres to a film, it has only two columns: `id, genre`
- `countries.csv`: represents the countries where the film was made, only two columns: `id, country`
- ...

3 Data Organization & Pre-processing

The data was organized into `DataFrames` and dictionaries, with movies as the primary focus. Additional information such as genres and countries was merged with the main movie table, after being grouped by id, to enrich the features of the film.

The following pre-processing techniques were applied:

- **Filling empty cells for `minute` and `rating`:** Missing values in the `minute` and `rating` columns were filled based on the following idea: Group data by genres, then for each genre calculate the mean of `minute` and `rating`, if there are genres that doesn't have enough data to compute the mean, so they are `NaN`, compute the global mean for `minute` and `rating` among all genres and use it to fill those empty spaces. Once we have for each genre those means, fill the empty cells of movies, according to the calculated mean by genres.
- **Scaling `minute` and `rating`:** The `minute` and `rating` features were scaled using standardization method called `MinMaxScaler()`. This to normalize data.
- **One-hot encoding on categorical features `genres` and `countries`:** The categorical variables `genres` and `countries` were transformed into binary vectors using one-hot encoding.

4 Algorithms and Implementations

Once the data were processed, I proceeded in the following manner.

Each row, basically represents an item profile, so once the `id` and `name` columns were excluded, I proceeded to apply cosine similarity between the profiles.

I have chose cosine similarity because if there are `NaN` values or equal to 0 they do not interfere with the similarity calculation. The matrix obtained is a square matrix in which we can observe how similar two films are to each other. If we have a value close to 1 the items are similar, otherwise if we have values close to -1 it means that they are opposite to each other.

After that I proceeded to import the actors table, interpreting it as users. It consists of two columns, one is the `id` (always referring to the movie) and one with the `name` (in this case, the user name). The key takeaway for this table is: for each row we simply have the id of the movie and the user who watched it.

Of course, it is with this table that I built the matrix utility, which itself is not a `DataFrame` but rather a `csr_matrix`. A `csr_matrix`, or **Compressed Sparse Row** matrix, is a type of sparse matrix that is particularly efficient for storing and manipulating large, sparse datasets where most of the elements are zero. The `csr_matrix` compresses the data by only storing non-zero elements along with their row indices and column index pointers. This reduces memory usage and speeds up matrix operations, which is especially useful for large-scale recommendation systems where the interaction matrix between users and items is mostly sparse.

Small note: the sparse matrix has no indexes, so I had to create two dictionaries that would index the position in the matrix of movies and users.

Therefore, I used the sparse matrix to make sure that I got all the movies associated with a user.

Regarding the final recommendation step, the process is quite simple, given a list of titles, for each title I go and extract from the cosine matrix its score and save it, for subsequent titles I simply sum over it the new scores, in such a way as to obtain a score that relates to the user's global level preferences. Once all titles are processed, I convert the total scores into a list of tuples, each containing a movie index and its aggregated similarity score. This list is then sorted in descending order of similarity scores, ensuring that the most similar movies come first. Next, I extract the indices of the top N similar movies, excluding the input movies themselves. Finally, I return the rows of these top recommendations from the movies `DataFrame`.

5 Scalability

As the dataset grows, processing the entire similarity matrix can become computationally expensive and time-consuming. To address this, I implemented a chunking strategy. By breaking down the dataset into smaller chunks, so I can process each chunk separately and then combine the results. This approach helps to manage memory usage more effectively and allows for parallel processing, significantly reducing the computation time.

Additionally, I made extensive use of dictionaries to optimize lookups. Instead of searching through large arrays or `DataFrames`, using dictionaries for storing and retrieving movie indices and similarity scores ensures that these operations are performed in constant time. This not only speeds up the recommendation process but also keeps the system responsive and scalable, even as the number of movies and users increases.

Certainly there are techniques that could be used to further improve code scalability, such as using Spark in such a way as to make the computation distributed and further increase speed. But this was intended to be a simple project that did the job.

6 Results

Here is an example of a user profile with only one title watched. In Fig. 1 we can see the film and its features, while in Fig. 2 we can see the recommendation output. It is not perfect but pretty close.

	id	name	minute	rating	year	Action	Adventure	Animation	Comedy	Crime	...
309	1027102	Mukkabaaz	0.0588	0.717949	2017	0	0	0	0	0	...

1 rows × 166 columns

Figure 1: User Profile with only one film watched

	id	name	minute	rating	year	Action	Adventure	Animation	Comedy	Crime	...
57	1024948	Games of Love and Chance	0.050225	0.711538	2003	0	0	0	0	0	...
36	1003574	Paris, Je T'Aime	0.048591	0.666667	2006	0	0	0	0	0	...
4535	1457802	La ladra	0.037566	0.656997	1955	0	0	0	0	0	...
1439	1116005	The Moment of Truth	0.043691	0.656997	1952	0	0	0	0	0	...
1307	1120680	Jardins du Palais Royal	0.001633	0.656997	2011	0	0	0	0	0	...
950	1080037	Nest of Vipers	0.042466	0.604359	1978	0	0	0	0	0	...
4524	1456359	Malafemmena	0.036341	0.604359	1957	0	0	0	0	0	...
6918	1709825	Le due madonne	0.038383	0.604359	1949	0	0	0	0	0	...
674	1063040	Happy Times Will Come Soon	0.040425	0.612179	2016	0	0	0	0	0	...
1106	1140419	Our Happy Lives	0.059616	0.604359	1999	0	0	0	0	0	...

Figure 2: Recommendation for the user profile with only one film watched in Fig. 1

In Fig. 3 we have another example of a user profile with more than 1 title watched, and in Fig. 4 we have the suggested recommendation. Again, the results are not bad.

7 Extras

Here are two nice plots about the data. One regarding the utility matrix (Fig. 5) and the other one regarding the number of watcher per film and average film

	id	name	minute	rating	year	Action	Adventure	Animation	Comedy	Crime	...
143	1030618	Carmen Comes Home	0.034708	0.644231	1951	0	0	0	1	0	...
338	1018748	Lightning	0.035116	0.839744	1952	0	0	0	0	0	...
3176	1318593	The Ghost of Iwojima	0.035525	0.585737	1959	0	0	0	0	0	...
3184	1312916	Forbidden Path	0.045733	0.656997	1952	0	0	0	0	0	...

Figure 3: User Profile with more than one films watched

watchers(Fig. 6).

8 Declaration

"I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study."

	id	name	minute	rating	year	Action	Adventure	Animation	Comedy	Crime	...	Uk
39	1036099	Knox Goes Away	0.046141	0.583333	2023	0	0	0	0	0	...	
1687	1172747	Look Who's Stalking	0.025356	0.545940	2023	0	0	0	0	0	...	
6983	1748586	Power Slap 2: Wolverine vs. The Bell	0.085341	0.583333	2023	1	0	0	0	0	...	
2509	1234562	A Future Together	0.000817	0.580769	2021	0	0	0	0	0	...	
3931	1372937	The Heart	0.001225	0.638889	2022	0	0	0	0	1	...	
4530	1471892	Tristan and Isaac	0.004900	0.580769	2020	0	0	0	0	0	...	
1341	1111835	Terror Eyes	0.032258	0.545940	2021	0	0	0	0	0	...	

Figure 4: Recommendation result for Fig. 3

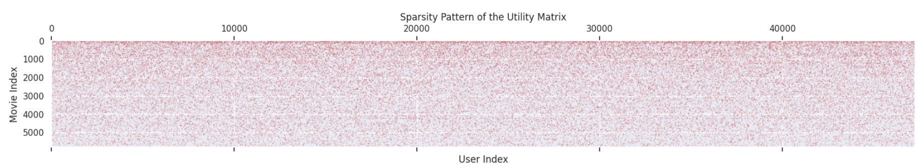

Figure 5: Sparsity Pattern

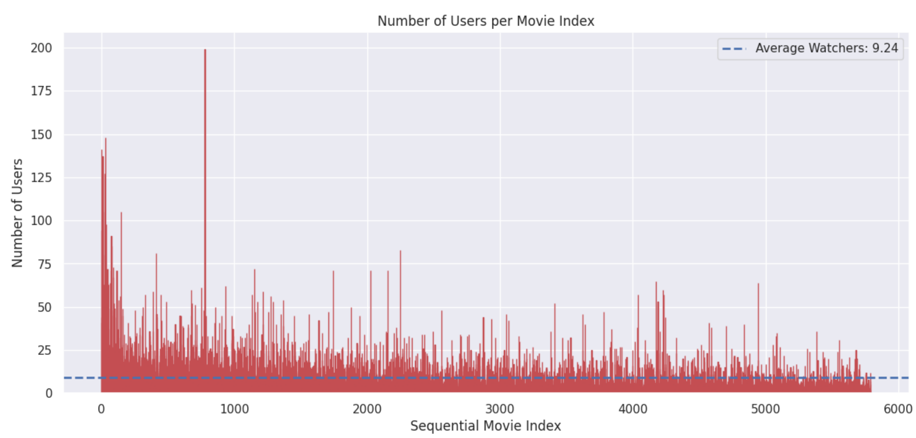

Figure 6: Number of Users per Movie