Collaborative Filtering Recommendation System Report

Lorenzo Leoncini

Matricola: 47403A

1 Introduction

This report presents the development of a movie recommendation system based on collaborative filtering. The system is designed to recommend movies to actors based on their participation in films.

2 Dataset Description

2.1 Dataset Version

The dataset used for this project is letterboxd from Kaggle, accessed on 12/09/2024. The dataset includes the following files:

- actors.csv - Contains actor IDs and other relevant details.
- genres.csv - Contains genre information for movies.
- movies.csv - Contains movie IDs, titles, and ratings.
- others...

2.2 Considered Parts

In this project, we focused on the ‘movies‘ and ‘actors‘ datasets. The ‘genres‘ dataset was considered but used only for potential future improvements.

3 Data Organization

The data was organized as follows:

- Actors DataFrame (`df_actors`): Contains the following columns:
  - actor_id - Unique identifier for each actor.
  - movie_index - Index mapping each movie in the dataset.
  - movie_id - Unique identifier for each movie, used for linking with the Movies DataFrame.

- Movies DataFrame (`df_movies`): Includes the following columns:
  - id - Unique identifier for each movie.
  - title - Title of the movie.
  - rating -Rating of the movie.

4 Pre-Processing Techniques

4.1 Pre-processing of the Movies Dataset

After examining the movies dataset, I identified missing data in several key columns: `tagline` (85.20%), `rating` (90.34%), `date` (9.76%), `description` (17.08%), and `minute` (19.28%).

Handling Missing Data:

- Tagline: I removed this column due to the high percentage of missing values.
- Rating: I filled missing values with the average rating to ensure the dataset remains useful for recommendations.
- Date and minute: I handled missing values by removing rows where data was incomplete.
- Description: I dropped this column since it wasn't used in the analysis.

After pre-processing, I removed unnecessary columns, handled missing values appropriately, and checked for duplicates. Additionally, I identified films with duplicate names but different IDs.

4.2 Pre-processing of the Actors Dataset

I began by examining the structure of the `df_actors` DataFrame, checking for missing values and analyzing the data distribution. The initial inspection revealed some inconsistencies and duplicate rows.

Handling Missing Data and Columns:

- I found no significant missing data except in the `role` column, which was irrelevant for the analysis, so I removed it.

Duplicate Management:

- I grouped the data by `id` and `name` to identify duplicate actor entries. A few duplicate associations between actors and their IDs were identified and removed.
- I dropped rows with duplicate actor names and standardized the `name` column by converting all names to lowercase.

Optimizations:

- The `id` column was optimized by converting it to `int32` format, improving memory usage.
- After these steps, I verified the dataset for unique actor names and their corresponding participations in films, creating a cleaner and more consistent dataset.

After processing, I ensured the dataset was free of missing data, duplicates, and inconsistencies. I also generated a summary of actor participations, visualizing the distribution of how often each actor appeared across films.

5 Algorithm and Implementation

In the implementation phase, I began by standardizing actor names in the dataset, converting them to lowercase and create extra spaces for consistency. I then created **two mappings**: one to associate each **actor's name** with a unique **ID**, and another to **reverse** this mapping.

After that I introduced an actor_id column to the dataset based on these mappings. I handled potential duplicates by grouping actors by their names to check for cases where the same actor name had multiple IDs. After identifying and removing such duplicates, I optimized the dataset by dropping the name column and renaming the id column to movie_id. I then mapped the movie_id values to a continuous index range, which was shuffled to ensure randomness in the recommendation process.

To handle memory efficiency and sparsity, I constructed a sparse utility matrix, where each cell represents the participation of an actor in a movie. This matrix was built using a **custom function to create a compressed sparse row (CSR) matrix format**. I calculated the density of the utility matrix to gauge the sparsity of the data.

For the seek of visualization I plotted the sparse matrix to show the distribution of actor-movie associations, folloed by calculating the **cosine similarity matrix** to assess similarities between actors based on their shared movie participations.

At the end to generate **recommendation** I built a system that checks whether an actor has sufficient data to receive detailed recommendations. If so, it retrieves movie recommendations based on the actor's similarity to others. If not, it provides random movies with a rating above 3.5 as an alternative.

To ensure that a different actor is selected each time for movie recommendations, I developed a function that **randomly selects** an actor and provides recommendations. Depending on the actor's available data, the system either returns detailed suggestions or random films from a highly rated pool.

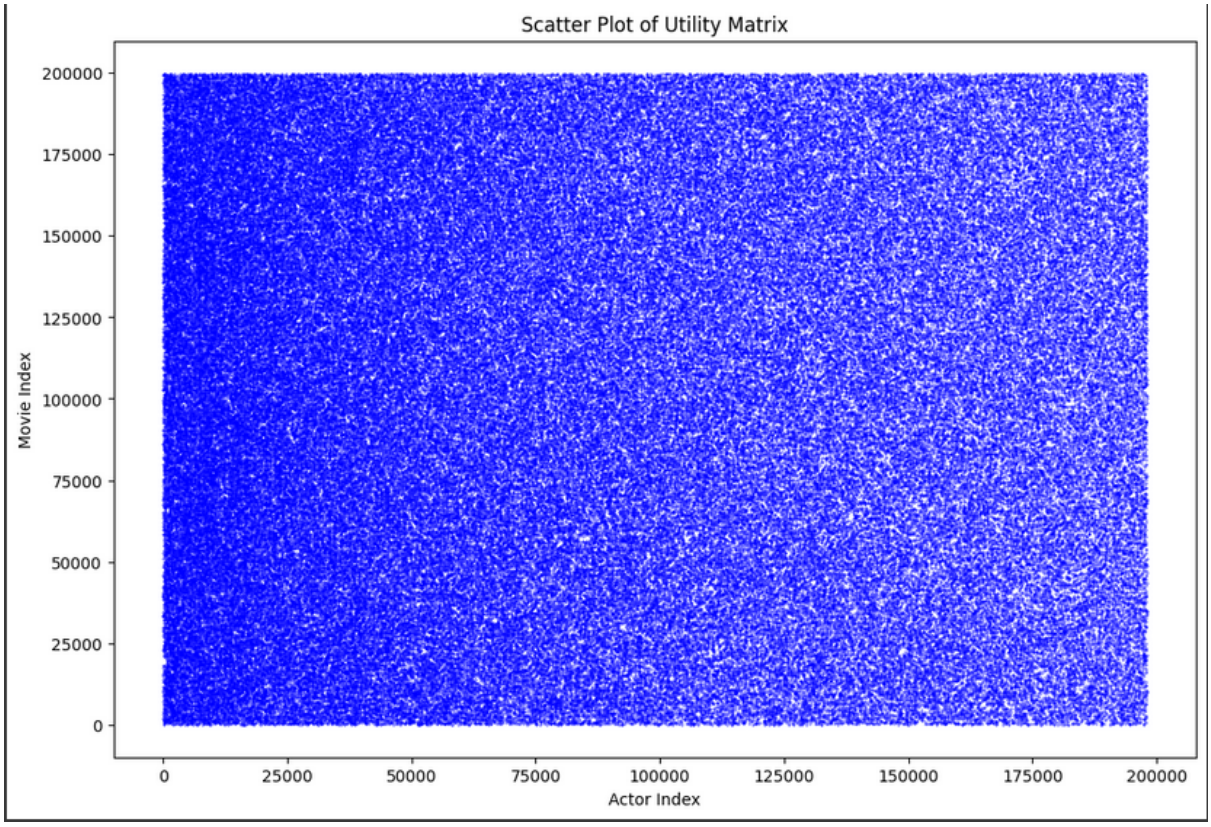

Figure 1: Scatter Plot of the Utility Matrix

6 Scalability

- To address scalability and memory usage, I utilized data chunking. This approach divides the dataset into chunks, which are processed sequentially. This technique makes the recommendation system more efficient and scalable.
- Efficient Indexing: Mapping movie and actor IDs to continuous indices optimizes data retrieval and processing.
- Sparse Matrices: Using sparse matrices reduces memory consumption and speeds up matrix operations by storing only non-zero entries. Efficient Indexing: Mapping movie and actor IDs to continuous indices optimizes data retrieval and processing.

7 Experiments

The following experiments were conducted:

- I tested the system with a randomly chosen actor to evaluate the quality of recommendations.

- I compared recommendations when using a bigger dataset versus subsampled dataset.

7.1 Results and Discussion

- I analized that this recommendation system excels with extensive interaction data between users or actors. With smaller datasets, it still works but is less effective due to limited interaction data.
- I experimented with both CPU and TPU for processing the dataset. While the CPU provided satisfactory performance for smaller samples, the TPU enabled me to handle significantly larger dataset samples, resulting in improved scalability and faster computations.
- The recommendation system performed well for actors with multiple movie participations.
- Future work will focus on incorporating genre information to improve recommendation quality.

	id	name	date	minute	rating
25632	1025633	It's Okay, That's Love	2014	0.278629	3.99
104901	1104902	Anglian Lives: Alan Partridge	2003	0.008416	3.62
31955	1031956	The Sound of the Shaking Earth	1990	0.027653	3.77
51149	1051150	Motor Mania	1950	0.001803	3.65
39825	1039826	Misfits	2015	0.021942	3.76

Figure 2: Example of a final recommendation

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.