Universitat Politècnica de Catalunya

Facultad de Informática de Barcelona

Recommender Systems

Búsqueda y Análisis de Información Masiva

Autores

Nicolas Llorens

Ricardo Lopez

02/11/2025

Contents

1 Understanding the input data

We will be using MovieLens dataset, specifically the ml-latest-small dataset. The goal is to gain a comprehensive understanding of the data before proceeding with building a recommender system.

The dataset contains information about:

- **Movies:** Movie titles, genres, and release years.
- **Ratings:** User ratings for movies on a scale of 1 to 5.
- **Tags:** User-assigned tags to movies.
- **Links:** Links to external movie databases (IMDb and TMDb).

We will use the *utils.py* script provided to load the dataset so we will only be using this function:

- $load_dataset_from_source$: $Loads the dataset from the specified directory.$

1.1 Movies Dataset

The movies dataset contains the following columns:

- **movieId**: A unique identifier for each movie.
- **title**: The title of the movie.
- **genres**: A list of genres associated with the movie.
- **year**: The year the movie was released.

1.1.1 Genres

We will start examining genres, to visualize the distribution of genres in the dataset, which are the most common genres, how many genres are there, etc.

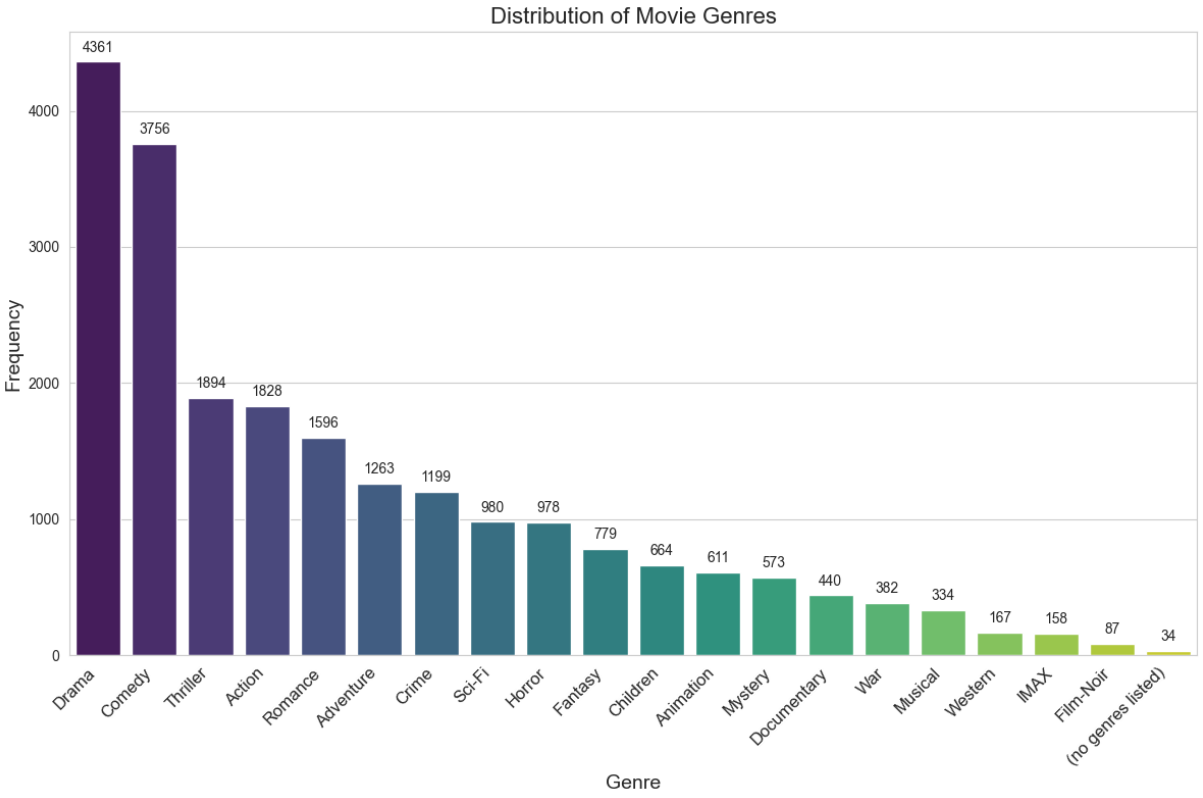

Figure 1: Distribution of Movie Genres

As we can see in plot 1, Drama and Comedy are the most common genres in the dataset. We also have a total of 20 unique genres. It's also interesting to note that some movies have multiple genres associated with them.

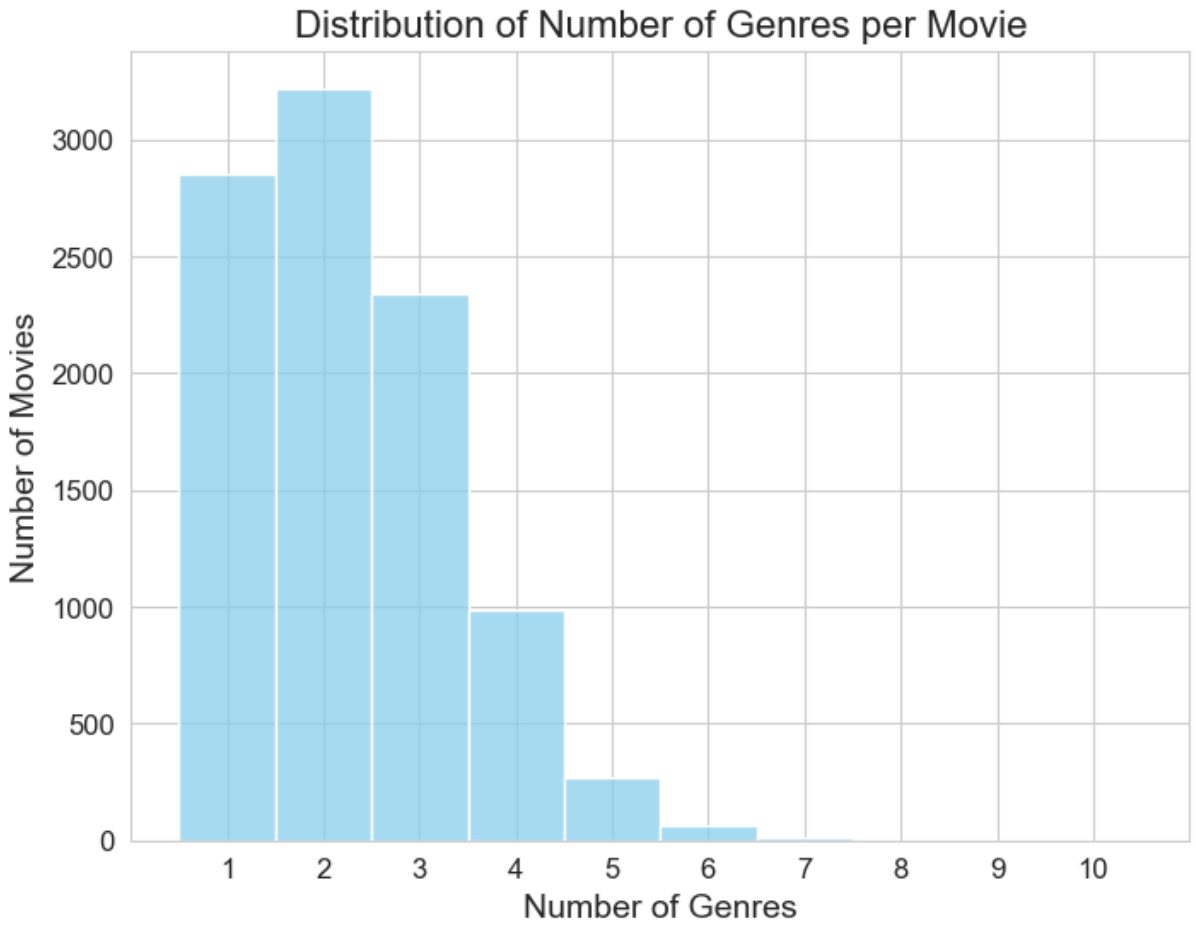

Figure 2: Distribution of Genres per Movie

In plot 2 we observe that most movies have 1 or 2 genres, but there are some movies with up to 6 genres. The distribution of the number of genres per movie is right-skewed.

1.1.2 Years

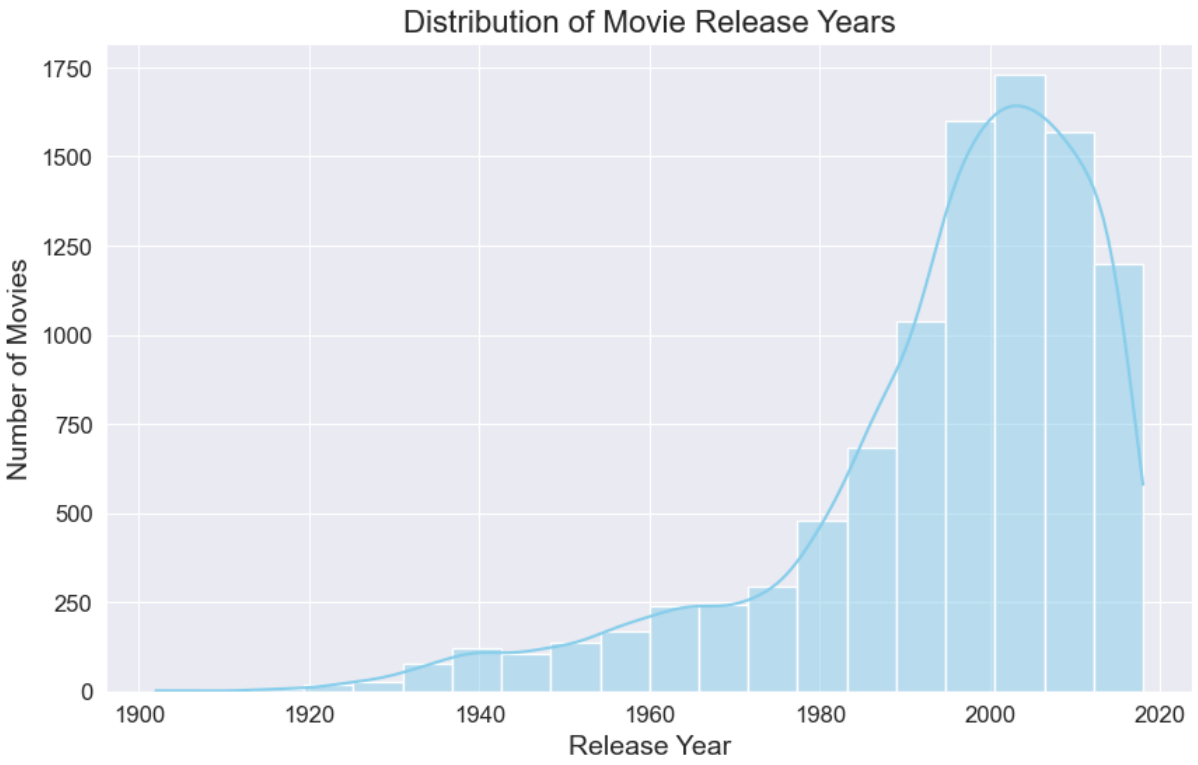

Figure 3: Distribution of Release Years

The number of movies released per year has been increasing over time. The dataset contains movies released from 1902 to 2018, with the majority of movies released after 1990. It's also interesting to note that the number of movies released in 2018 is significantly lower than in previous years.

1.2 Ratings Dataset

The ratings dataset contains the following columns:

- **userId**: A unique identifier for each user.
- **movieId**: A unique identifier for each movie.
- **rating**: The rating given by the user (on a scale of 0.5 to 5).
- **timestamp**: The timestamp when the rating was created.

For this dataset we will be analyzing this three things:

- **Rating Distribution**: Histogram of the ratings to see how users rate movies in general (e.g., are ratings skewed towards higher values?).
- **Number of Unique Users**: Calculate and display the number of unique users in the dataset.

- **User Activity**: Analyze how many ratings each user has given (e.g., histogram, statistics like mean, median, etc.). This could provide an idea of user engagement.

1.2.1 Ratings Distribution

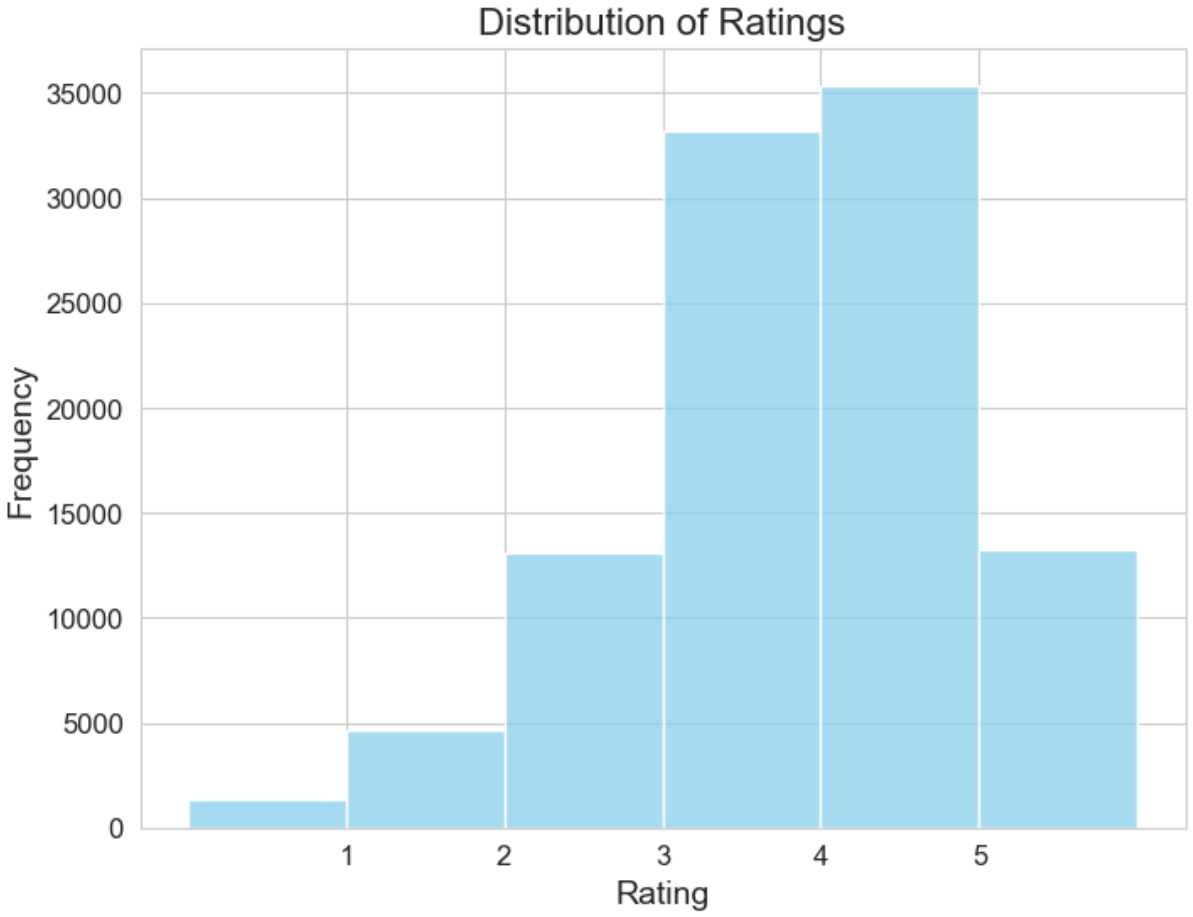

Figure 4: Distribution of Movie Ratings

The ratings are skewed towards higher values, with the majority of ratings falling between 3 and 4. The distribution is right-skewed, with a peak around 4.

1.2.2 Number of Unique Users

The are 610 unique users in the ratings dataset.

1.2.3 User Activity

It's important to measure the user activity of our dataset, if it's frequent to rate movies or not.

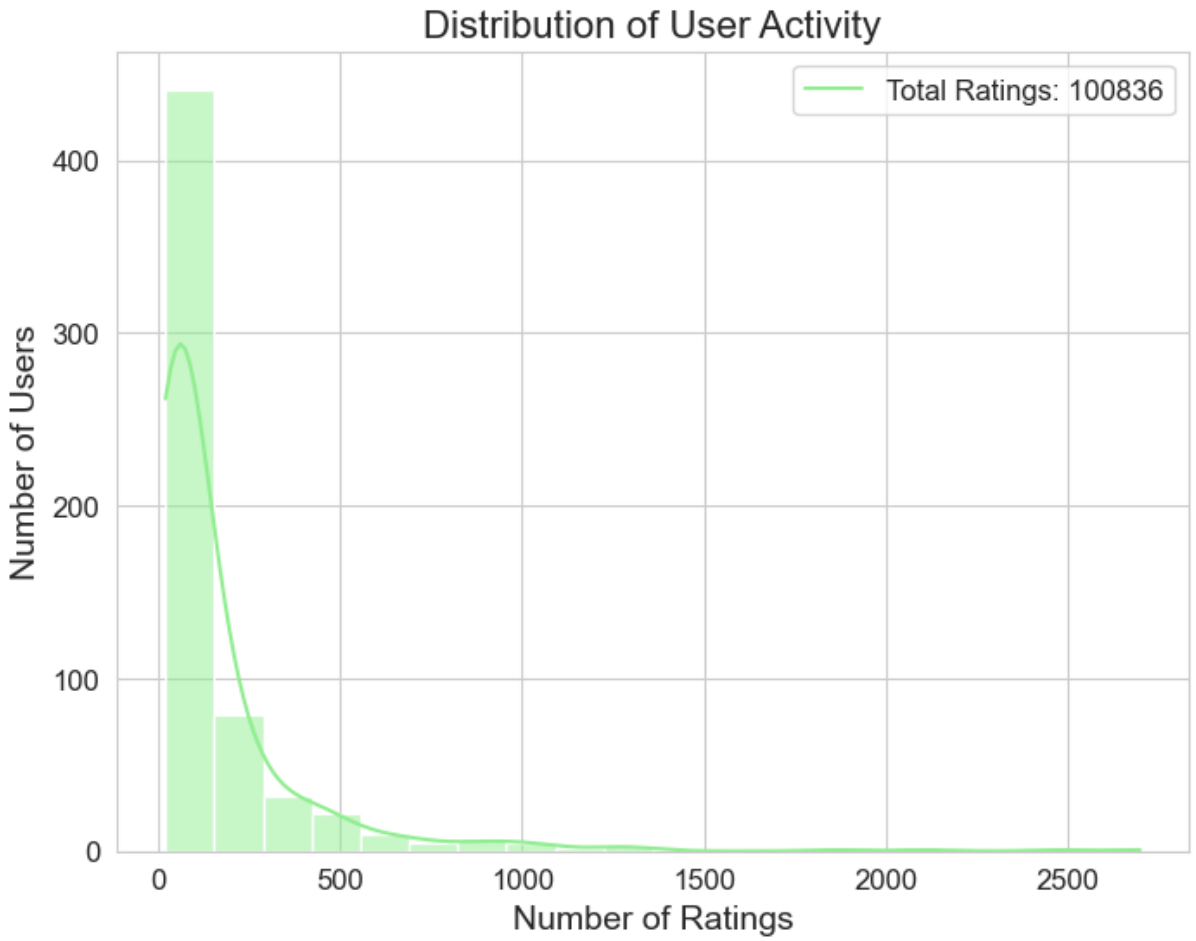

Figure 5: Distribution of User Activity

For better understanding the user activity, we will also need some statistics.

Table 1: User Activity Statistics

Statistic	Value
Mean	165.30
Median	70.50
Minimum	20
Maximum	2698

Plot 5 and table 1 show that users are very active in the dataset, with the majority of users having rated more than 100 movies on average.

1.3 Tags Dataset

The tags dataset contains the following columns:**userId**: A unique identifier for each user. **movieId**: A unique identifier for each movie. **tag**: The tag assigned by the user. **timestamp**: The timestamp when the tag was created.

For this dataset we will be analyzing this two things:**Most Common Tags**: Display the most common tags in the dataset. **Distribution of Tags per Movie**: Analyze how many tags each movie has on average.

1.3.1 Most Common Tags

As we have a large amount of total unique tags (1589), we can't display all of them. So we will display the 30 most common tags.

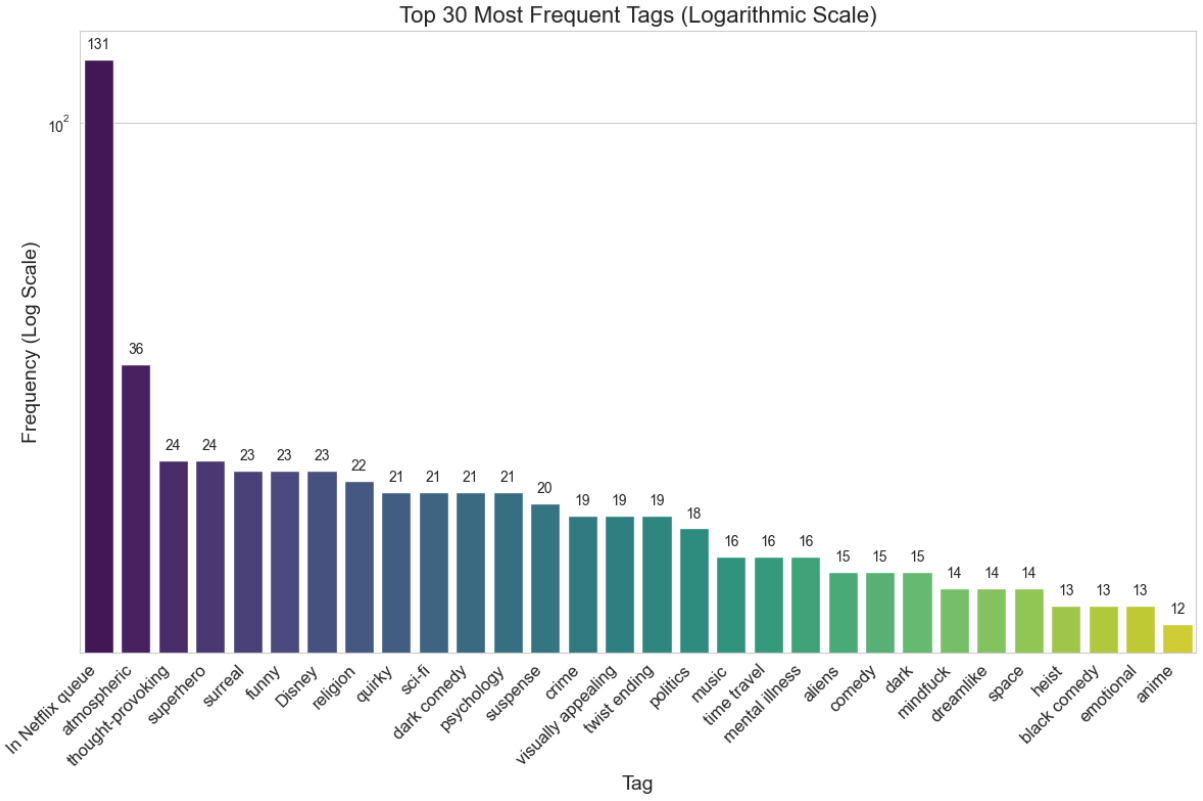

Figure 6: Distribution of Movie Tags

Plot 6 shows that tags are in fact different from genres, and they are more specific and subjective. This can be also interesting for the recommendation system, as it can provide more detailed information about the user preferences and make the recommendations more accurate.

1.3.2 Distribution of Tags per Movie

We want to see how many tags each movie has on average, so we can understand how many tags are usually assigned to a movie and maybe relay on Tags for the recommendation system.

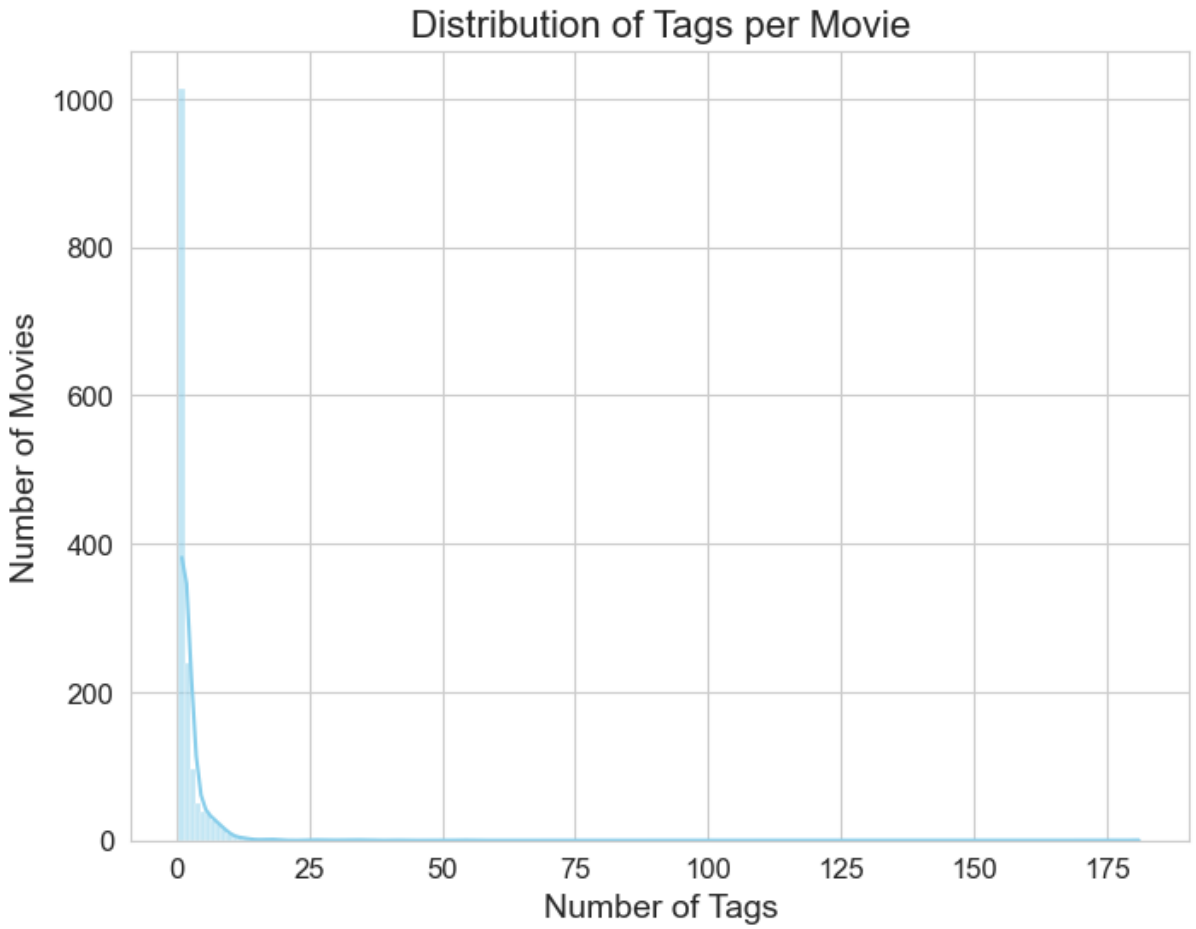

Figure 7: Distribution of Tags per Movie

Table 2: Tags on movie metrics

Statistic	Value
Mean	2.34
Median	1.00
Minimum	1
Maximum	181

We can see that most movies have 1 or 2 tags assigned to them. As there are not many tags per movie, we can consider using them as additional features for the recommendation system.

1.4 Links dataset

The links dataset contains the following columns:

- **movieId**: A unique identifier for each movie.
- **imdbId**: The IMDb identifier for the movie.

- **tmdbId**: The TMDb identifier for the movie.

This datset doesn't have much information, but it can be useful for linking the MovieLens dataset with other external databases.

2 Building a Naive Recommender

A naive recommender system was built as a baseline. This system recommends films with the highest average ratings, regardless of individual user preferences.

2.1 Implementation

The `naive_recommender` function calculates the average rating for each movie and sorts them in descending order. The top k movies with the highest average ratings are then recommended.

2.2 Time Complexity

The time complexity of this recommender is dominated by the sorting operation, which takes $\mathcal{O}(n \log n)$ time, where n is the number of movies.

2.3 Limitations

This naive recommender suffers from several limitations:

- **Popularity Bias:** It recommends popular movies with high average ratings, ignoring individual user preferences.
- **No User History:** It does not consider the user's past ratings or viewing history.
- **Cold Start Problem:** It struggles to recommend new or less popular movies with few ratings.

2.4 Potential Improvements

To overcome these limitations, more sophisticated recommender systems can be employed.

1. **Collaborative Filtering:**
   - **User-based:** Recommends items based on the preferences of similar users.
   - **Item-based:** Recommends items similar to those the user has liked in the past.
2. **Content-based Filtering:** Recommends items based on their features (genre, tags, etc.).
3. **Hybrid Approaches:** Combines multiple techniques for improved performance.
4. **Addressing Cold Start:** Uses content-based filtering or popularity-based recommendations for new items.

3 User-based Recommender

This section describes the implementation and evaluation of a user-based collaborative filtering recommender system.

3.1 Methodology

The recommender estimates the interest of a target user a in an item s based on the ratings of similar users:

$$\text{interest}(a, s) = \bar{r}_a + \sum_{b \in U} w(a, b)(r_{b,s} - \bar{r}_b),$$

where:

- $\bar{r}_a$ is the average rating of user a.
- U is the set of users most similar to user a.
- $w(a, b)$ is the normalized cosine similarity between user a and user b.
- $r_{b,s}$ is the rating of movie s by user b.

3.2 Implementation Details

We had to solve different problems during the implementation, from choosing an efficient way to store the user-movie rating matrix with adequate data structures to defining the set of similar users (U).

- **Data Structure:** A user-movie rating matrix M was created using the `generate_m` function, where M[user][movie] stores the rating. This matrix was efficiently generated using the `pivot` operation in pandas.
- **Similarity Metric:** Cosine similarity was used to measure the similarity between the rating vectors of the users. The `compute_similarity` function in `similarity.py` calculates this similarity. It handles missing values ('NaN') by replacing them with 0 and addresses cases where one or both vectors have zero norm.
- **Selection of Similar Users (Defining U):** The set U of similar users was dynamically determined in the `user_based_recommender` function. The code calculates the cosine similarity between the target user and all other users. Then, it sorts the users by similarity in descending order. The top N most similar users (where N can be a parameter) are considered for the prediction.
  - **Justification for Dynamic Selection:** This dynamic selection of similar users is preferable to a fixed set because it adapts to each target user. Different users may have different "neighborhoods" of similar users, and this approach captures that variability.
- **Handling Missing Ratings:** Missing ratings were handled in two ways:

  - During similarity calculation: 'NaN' values were replaced with 0 in the `compute_similarity` function.
  - During prediction: Only users who have rated a specific movie are considered when predicting its rating for the target user.

- **Prediction:** The `user_based_recommender` function predicts the rating for unseen movies by calculating a weighted average of ratings from similar users who have rated the movie. This weighted average is adjusted by subtracting the mean rating of each similar user to account for differences in rating scales.
- **Optional Genre Filtering:** The `user_based_recommender` function includes an optional feature to filter recommendations based on the target user's genre preferences. This is done using the `filter_by_genre_preferences` function, which selects movies that match the user's most-watched genres.

4 Validation Based on Genres

This section compares the naive recommender and the user-based collaborative filtering recommender based on their ability to recommend movies with genres similar to those in the users' validation sets.

4.1 Evaluation Methodology

For each user in a test set, the top k recommended movies from each recommender were compared against the movies in the user's validation set. The genre sets of the recommended movies and the validation set movies were compared using Jaccard similarity.

To mitigate the impact of data sparsity, we increased the number of movies held out for validation ('val_movies') to 100. This ensures that the user-based collaborative filtering recommender has sufficient data to identify similar users and make accurate predictions.

4.2 Experimentation

When we were first testing our user based recommender, we got very poor results, as we can see on plot 8.

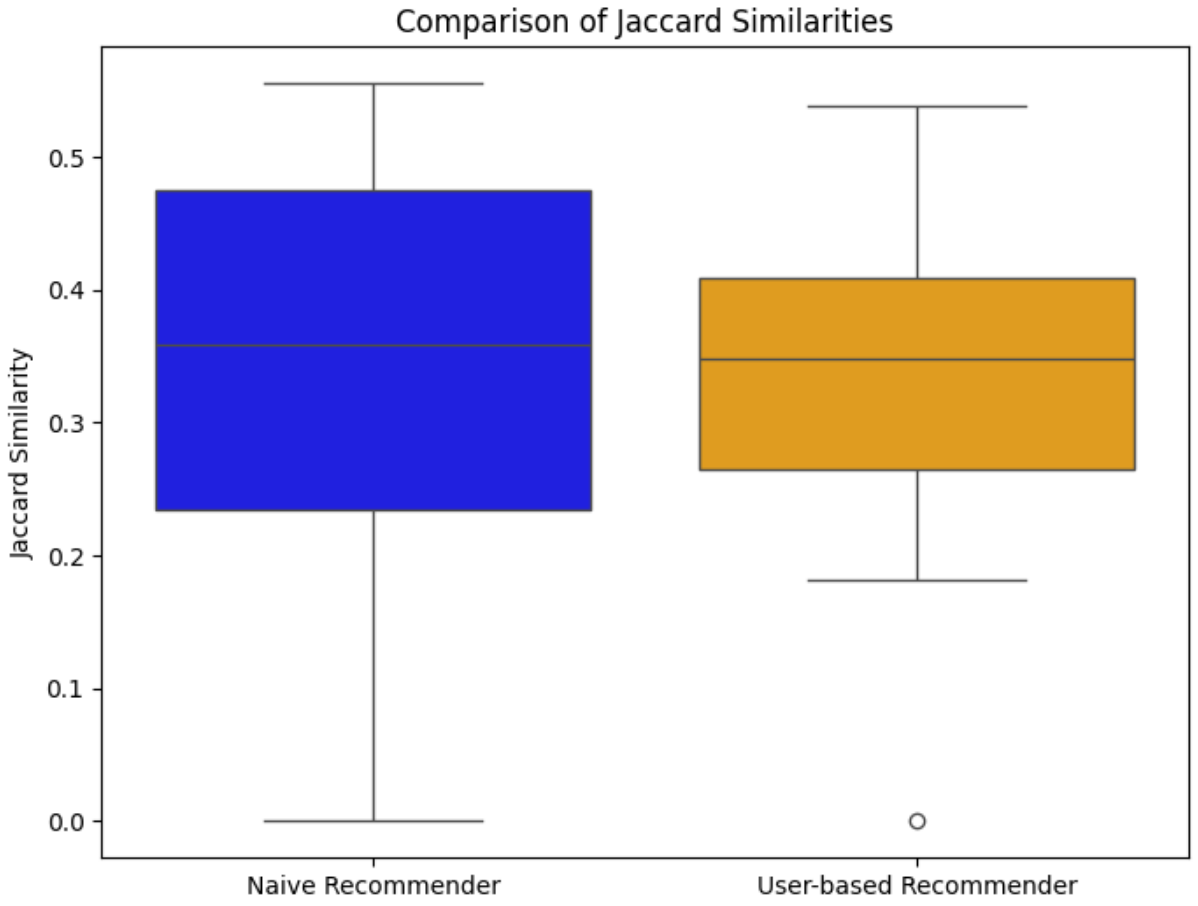

Figure 8: Naive vs UserBased Jaccard Comparison

4.2.1 Impact of `val_movies` on User-Based Recommender Performance

We then thought that we where limiting the amount of movies that a user had rated to improve performance, we intially set this to 5, as we thought it would be enough. But this led to poor performance for this reasons:

- **Increased Data Sparsity:** Holding out only 5 movies for validation severely limits the training data, exacerbating the data sparsity problem inherent in collaborative filtering. This makes it difficult to find similar users with sufficient overlapping ratings for reliable predictions.
- **Unreliable Similarity Calculation:** With limited training data, the calculated similarities between users become less reliable. Small variations in the few available ratings can significantly distort similarity scores, leading to inaccurate identification of similar users.
- **Poor Predictions:** The recommender relies on the ratings of similar users to predict ratings for unseen movies. With unreliable similarity scores and sparse data, the predicted ratings are likely to be inaccurate, resulting in poor recommendations.
- **Insufficient Genre Information:** A small validation set (5 movies) might not accurately represent the user's true genre preferences, making it difficult to assess

the recommender's ability to recommend relevant genres.

4.3 Results

Having adjusted the maximum amount of movies rated per user to an optimal amount (100), we can benchmark more accurately the recommender.

The average Jaccard similarity scores for both recommenders are shown in Table 3.

Table 3: Comparison of recommenders based on genre similarity.

Recommender	Average Jaccard Similarity
Naive	0.1772
User-based	0.4704

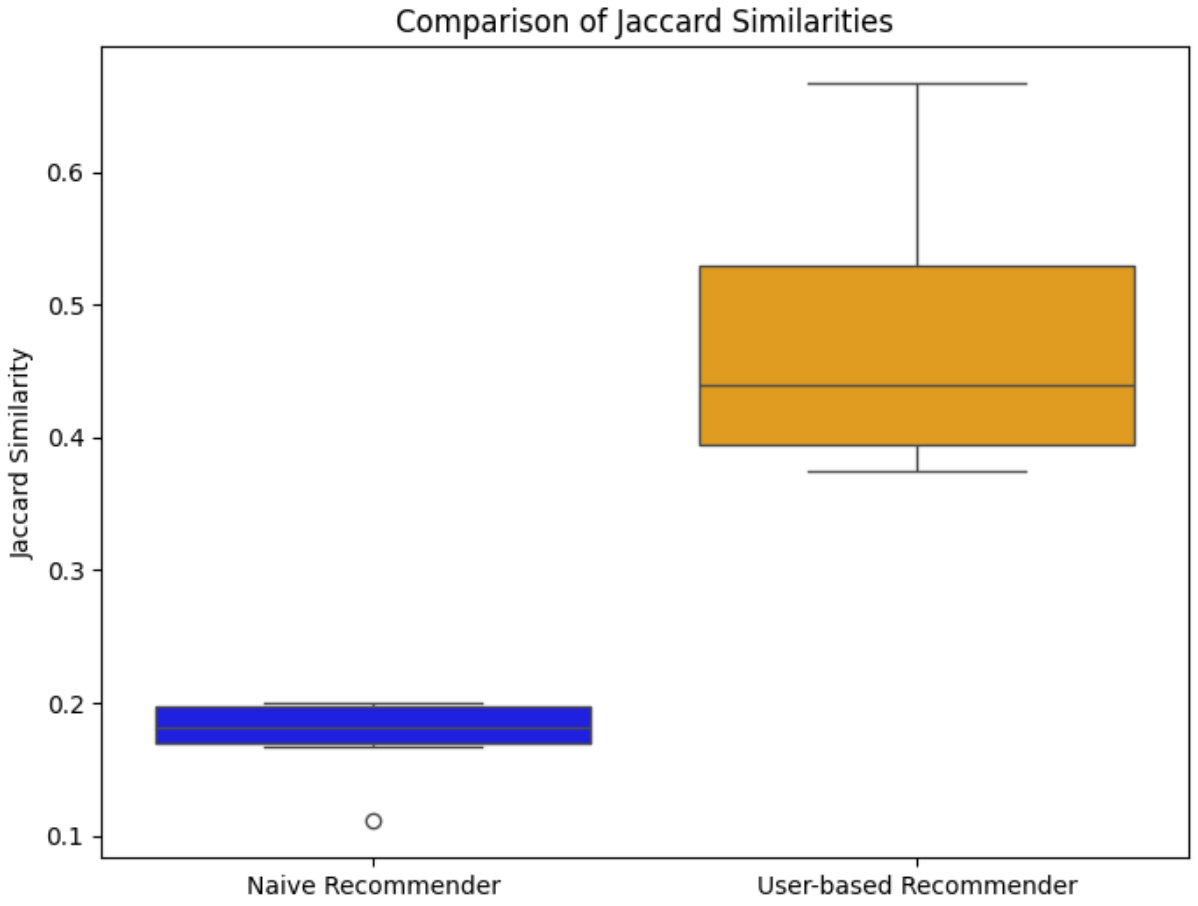

Figure 9: Naive vs UserBased Jaccard Comparison

The plot 9 suggests that the user-based recommender generally outperforms the naive recommender in terms of genre similarity. However, the user-based recommender also exhibits more variability in its performance. This could be due to factors like data sparsity, the varying effectiveness of similarity calculations between users, or the influence of the cold start problem for new users.

A t-test was performed to compare the means of the Jaccard similarity scores for the two recommenders. The results indicate a statistically significant difference in performance (t-statistic = -9.4322, p-value = 0.0000).

As we have seen, user based recommenders work better with more data, but this leads to a different problem, time performance. As we can see in table 4 user based recommender is much slower.

Table 4: Comparison of recommenders based on execution time.

Recommender	Average Time (seconds)
Naive	0.0040
User-based	8.2639

4.4 Discussion

The user-based collaborative filtering recommender significantly outperforms the naive recommender in terms of genre similarity. This suggests that the user-based approach is more effective at capturing user preferences and recommending movies with similar genres to those in the validation set.

The improvement in performance after increasing the maximum amount of movies rated per user highlights the importance of addressing data sparsity in collaborative filtering. By providing more data for training, the recommender is able to identify similar users more accurately and make better predictions.

But this also highlights the importance of performance if user base recommenders rely this much on big quantities of data.

For future implementations, this dataset gives the possibility of using movie tags, which can be a good possibility to explore that could maybe personalize even more user recommendations.

4.5 Answer to the Question

Based on the evaluation results, the **user-based collaborative filtering recommender** is more suitable for this task. Despite its higher complexity, it provides significantly better accuracy in recommending movies with genres similar to those in the users' validation sets. The naive recommender, while simpler, suffers from limitations such as popularity bias and is not effective at all capturing user preferences.

5 Conclusions

This report explored the development and evaluation of two recommender systems: a naive recommender and a user-based collaborative filtering recommender. The naive recommender, while simple to implement, suffers from limitations such as popularity bias and fails to capture individual user preferences. The user-based collaborative filtering recommender, on the other hand, demonstrated significantly better accuracy in recommending movies with genres similar to those in the users' validation sets.

The analysis highlighted the crucial role of addressing data sparsity in collaborative filtering. By increasing the amount of training data, the user-based recommender was able to identify similar users more accurately and make better predictions. However, this improvement in accuracy came at the cost of increased computational complexity and execution time.

Future work could focus on exploring more sophisticated techniques to address data sparsity and scalability challenges in collaborative filtering. Additionally, incorporating content-based filtering or hybrid approaches could further enhance the recommender's performance and address the cold start problem for new users.

Overall, this study provides valuable insights into the development and evaluation of recommender systems, emphasizing the trade-offs between accuracy, complexity, and efficiency. The user-based collaborative filtering recommender, despite its higher complexity, emerges as the more suitable choice for this task due to its superior ability to capture user preferences and provide personalized recommendations.