POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Numerical Analysis For Machine Learning Project

Movie Recommendation System using Sentiment Analysis from Microblogging Data

Author(s): **Irfan Cela - 10694934**

Fabio Lusha - 10882532

Bianca C. Savoiu Marinas - 10684465

Group ID: **70**

Academic Year: 2023-2024

Contents

1 | Assigned Paper and Recommender System task

1.1. Problem description

In this project, we developed some movie recommender systems leveraging sentiment analysis data from the **MovieTweetings dataset**, trying to follow the ideas outlined in the given paper `"Movie Recommendation System Using Sentiment Analysis from Microblogging Data"`. The paper combines sentiment analysis with traditional recommendation methods such as Collaborative Filtering (CF) and Content-Based Filtering (CB) to enhance recommendation accuracy and precision. Instead of performing sentiment analysis, we used the pre-existing sentiment scores, found in the dataset associated with the paper, and enriched it by integrating additional movie metadata from the **TMDB API**, trying to create a more robust content-based filtering system and to build a hybrid recommender system that integrates similarity among users, movie content, and public sentiment.

1.2. Outline of the project

In this report, we detail the development of a movie recommender system leveraging microblogging data from the **MovieTweetings dataset** and enhancing it with metadata from the **TMDB API**. The following sections outline the major steps taken during the project:

1.2.1. Data Preprocessing and Data Analysis

- **Initial Data:** Preprocess and analyze the initial dataset containing user ratings from pre-computed scores of microblogging data, and basic movie data.
- **Data Augmentation:** Integration of additional movie metadata (e.g., director, cast, runtime) from the **TMDB API** to enrich the dataset.

- **Final Dataset:** The final dataset used for building recommendation models, then adapted for the CF and CB recommenders.

1.2.2. Collaborative Filtering Recommendation System

- **Data Preparation:** Organizing data for collaborative filtering (CF) recommender systems.
- **Most Popular Recommender:** A simple recommender system based on movie popularity.
- **CF recommender with Funk-SVD:** A CF recommender system based on user-movie interactions.

1.2.3. Content-Based Recommendation System

- **Data Preparation:** Structuring the enriched dataset for content-based filtering (CB), maintaining mainly the most relevant attributes.
- **Simple CB Recommender:** A basic CB model, recommending movies based on movie attributes.
- **Feature Augmentation:** Expanding the feature set using the metadata from the **TMDB API**, specifically the **overview** data, to improve recommendations.

1.2.4. Hybrid Recommendation System

- **Overview:** Introduction to hybrid recommendation techniques combining CF and CBF.
- **Mixed Hybrid System:** Combining CF and CBF directly to improve recommendations, using the intersection and the union, depending on the common recommended movies.
- **Meta-Level Hybrid System:** Using the collaborative filtering recommender's output as input for the content based filtering system.

1.2.5. Model Evaluation

- **With Ground Truth:** Evaluating the models against known recommendations, downloaded from **TMDB API**, used as a ground truth.

- **Without Ground Truth:** Approximating performance without explicit ground truth recommendations.

This report covers each of these components more in detail, explaining some of the main steps we followed to develop, implement, and evaluate our movie recommender system.

2 | Data preprocessing and Data Analysis

2.1. Initial data from preprocessed microblogging data

In this section, we begin by focusing on the raw datasets sourced from microblogging platforms, specifically from Twitter, where user interactions and reviews about movies are captured in the form of ratings and textual comments. The primary datasets consist of three main components: movies, ratings, and users. Our goal is to preprocess the data effectively before augmenting it with external sources and performing analysis.

2.1.1. Movies Dataset

The original movies dataset from the `MovieTweetings` DB forms a critical part of the recommendation system. This dataset includes a collection of movies that users have rated in their tweets, along with metadata describing each movie.

The `movies.dat` dataset contains metadata for `37.342` movies. The dataset is structured in the following format:

- `movie_id`: A unique identifier for each movie.
- `movie_title (movie_year)`: The title of the movie, followed by its release year, in parentheses.
- `genre|genre|genre`: A pipe-separated list of genres associated with the movie.

The genres provide essential information for content-based recommendation systems, which compare users' past ratings with movies in similar categories. This metadata is used to form basic feature representations for the recommendation algorithm.

Example Record: `0110912::Pulp Fiction (1994)::Crime|Thriller`

- `movie_id`: `0110912` (unique identifier for **Pulp Fiction**).

- `movie_title (movie_year)`: `Pulp Fiction (1994)` (the title with release year).
- `genre|genre`: `Crime|Thriller` (the genres associated with the movie).

This data is fundamental for understanding the types of movies users rate, as well as the genres they may prefer. It serves as the foundation for initial content-based analysis and later models that rely on user preferences for similar movie genres.

2.1.2. Ratings Dataset

The `ratings.dat` file contains user rating data, which forms the backbone of the recommendation system. It stores the ratings that users have provided for various movies. Each entry in the dataset follows the format:

```
user_id::movie_id::rating::rating_timestamp
```

Where:

- `user_id`: A unique identifier for the user who rated the movie.
- `movie_id`: A unique identifier for the movie being rated (corresponding to the `movie_id` in the `movies.dat` dataset).
- `rating`: The user's rating for the movie, on a scale from 1 to 10.
- `rating_timestamp`: A timestamp representing when the rating was given, stored as an epoch time.

The ratings' dataset is essential for building collaborative filtering models, which rely on user preferences to make recommendations. It captures both user behavior and movie evaluations, allowing the system to learn which movies are similar based on the ratings they receive from different users.

Example Record: `14927::0110912::9::1375657563`

- `user_id`: `14927`, representing a specific user.
- `movie_id`: `0110912`, corresponding to the movie *Pulp Fiction* (as identified in the `movies.dat` dataset).
- `rating`: `9`, meaning the user rated the movie 9 out of 10.
- `rating_rimestamp`: `1375657563`, which represents the time the rating was submitted.

2.1.3. Data Preprocessing

Movies: Drop Duplicates We start by removing exact duplicate entries from the movies' dataset. Duplicate entries can arise from data collection issues or errors during data integration, and can negatively impact the recommendation model by inflating the representation of certain movies.

Movies: Conversion of Genre into List The '`Genre`' column in the movies dataset consists of a string where multiple genres are separated by a '`|`' symbol. To transform this data into a more structure format, we first split each genre string into a list of individual genres. This allows for more efficient data manipulation and analysis, especially when creating features for the recommendation system.

Movies: Filter Movies by Genre To analyze the distribution of movie genres in the dataset, we first extract and count the frequency of each genre from the '`Genre`' column. To maintain data integrity, we first filter out movies that have undefined or incorrectly formatted genres in the '`Genre`' column. To focus the recommendation system on movies with genres that have sufficient representation, we filter the dataset to retain only movies whose genres appear at least 20 times.

Movies: Extracting Title and Year To enhance the usability of the movie dataset, we process the `MovieTitle(Year)` column to extract and separate the movie's title and release year. We use a regular expression to split the combined title and year into two distinct features: `Title` and `Release_year`.

Movies: Filter Movies by Release Year To understand the distribution of movie releases across different years, we create a histogram that shows the number of movies released each year. This visualization helps identify trends.

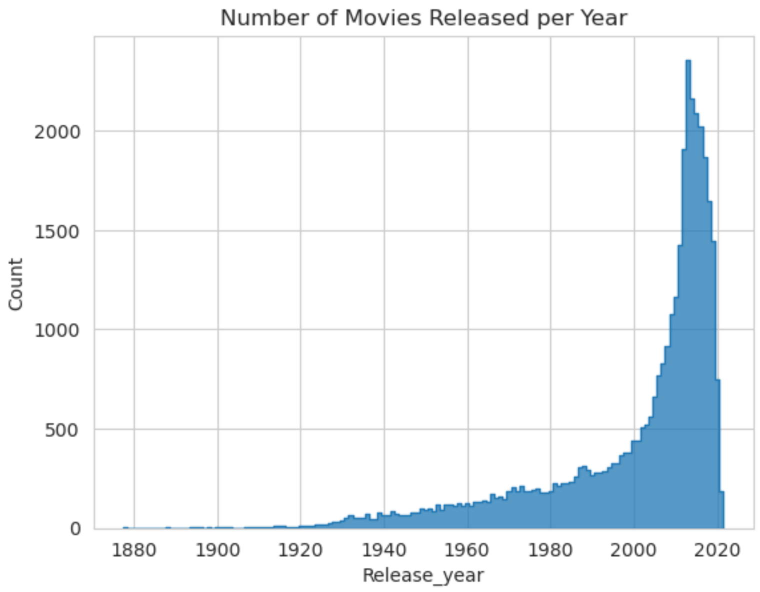


Figure 2.1: **A Century of Cinematic Growth**. *The graph reveals a dramatic increase in movie production over time, with a relatively low and steady output until around 1980, followed by a sharp exponential rise through the early 2000s. The peak is around 2013, with over* `2.000` *movies released annually. There's a noticeable drop-off in the most recent years, possibly due to incomplete data for the most recent period.*

To focus the analysis on recent movies, we filter the dataset to include only those movies released between 2014 and 2017, inclusive. This timeframe was selected based on insights from the original paper considered for the project, which emphasized the importance of analyzing recent trends to ensure the recommendation system aligns with current movie preferences.

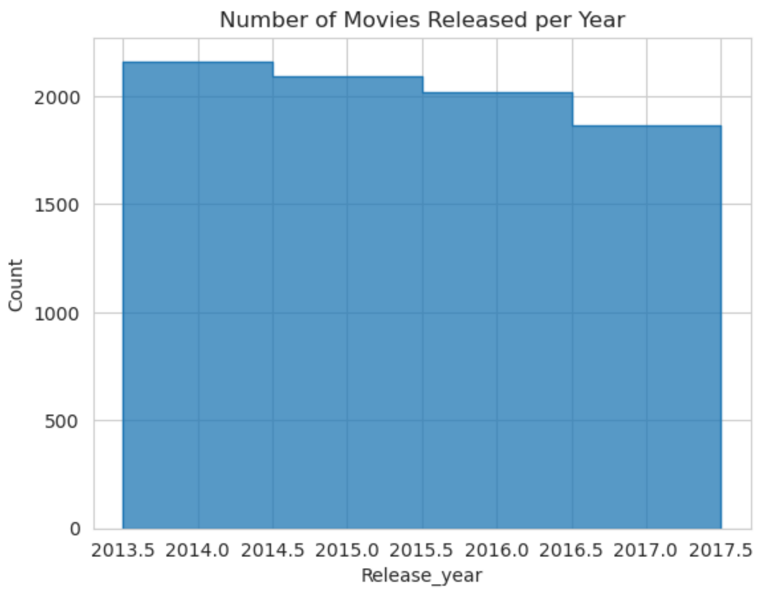


Figure 2.2: **A Declining Trend from 2014 to 2017**. *This graph illustrates the annual movie release count from 2014 to 2017.*

Ratings: Filter Ratings by Movie ID To ensure that our ratings data is relevant and corresponds to movies in our movies' dataset, we filter the ratings DataFrame to retain only those ratings for movies that are also present in the movies' dataset. This step is crucial for maintaining data consistency and ensuring that our recommendation system operated on valid data.

Ratings: Filter Ratings by User ID Counts We filter the ratings DataFrame to retain only those users who have provided at least 20 ratings. This step helps in focusing on users who are sufficiently engaged, which is important for generating reliable recommendations.

Movies: Filter Movies by Movie ID We filter the movies DataFrame to retain only those moves present in the ratings' dataset. This step is crucial for maintaining consistency between the movies and ratings datasets.

2.2. Augmentation of the movies data using TMDB

The original movies dataset from the `MovieTweetings` DB provides basic metadata for each movie, such as the title, release year, and genres in the following format:

```
movie_id::movie_title (movie_year)::genre|genre|genre
```

For example:

```
0110912::Pulp Fiction (1994)::Crime|Thriller
```

While this structure offers valuable information, it is somewhat limited. For instance, it lacks features that may enhance the performance of the recommendation system.

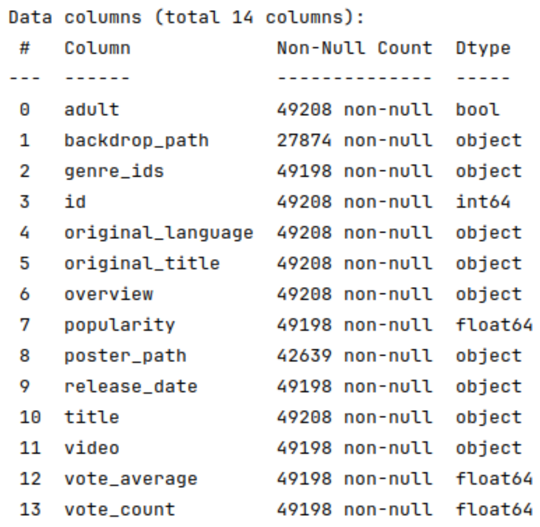
As suggested by the paper, to address these limitations, we augmented the original dataset by using the information provided by `TMDB`, which allows access to some of its movie information through a free API.

This augmented dataset allows for more sophisticated recommendation models, enabling the system to not only recommend movies based on genre but also based on factors such as plot similarity, cast members, their popularity, and other features that may be important for the users.

This part was also essential since `MovieTweetings` dataset uses the IMDb ID as an identifier for the movie. We first had to retrieve information about the movies by making API calls using the title as an identifier. Then we were able to merge the information with the data in `MovieTweetings` consistently by retrieving more movie details with the TMDB ID, which contained also the IMDb ID, guaranteeing a successful and consistent merge.

We gather detailed movie information through several data retrieval functions.

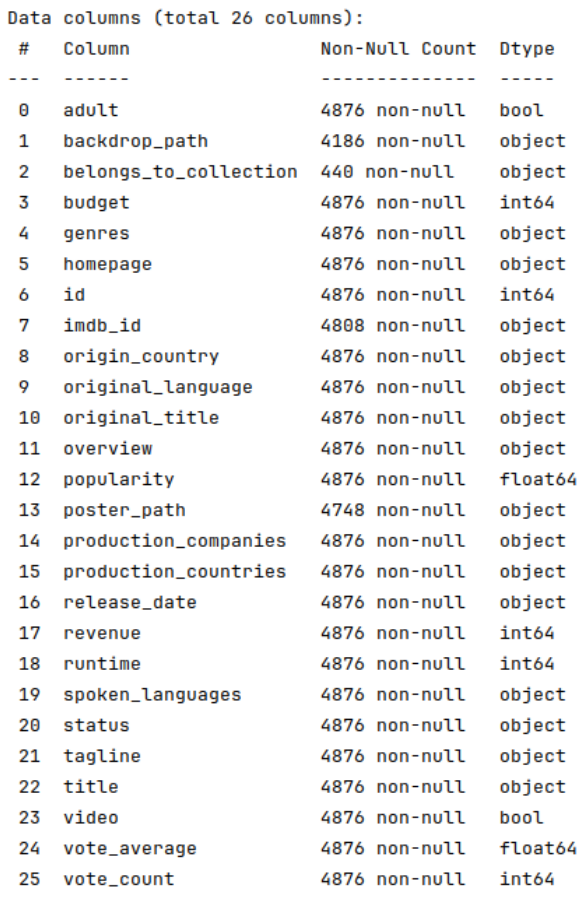
1. **Movie Details by Title**: Retrieves comprehensive details about movies based on their titles. This includes essential information such as movie IDs, overviews, release dates, and genres.



```
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   adult               49208 non-null  bool
 1   backdrop_path       27874 non-null  object
 2   genre_ids           49198 non-null  object
 3   id                  49208 non-null  int64
 4   original_language   49208 non-null  object
 5   original_title      49208 non-null  object
 6   overview            49208 non-null  object
 7   popularity          49198 non-null  float64
 8   poster_path         42639 non-null  object
 9   release_date        49198 non-null  object
 10  title               49208 non-null  object
 11  video               49198 non-null  object
 12  vote_average        49198 non-null  float64
 13  vote_count          49198 non-null  float64
```

Figure 2.3: **TMDB's Metadata by Title**.

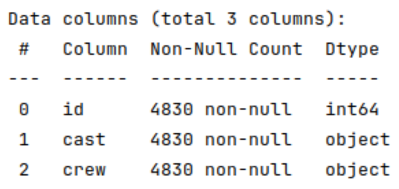
2. **Movie Details by ID**: Fetches additional movie details using unique movie IDs. This function provides more granular information, including metadata that may not be available through title-based searches. We will keep only a part of this data, the ones specified in the paper.



```
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   adult                  4876 non-null   bool
 1   backdrop_path          4186 non-null   object
 2   belongs_to_collection  440 non-null    object
 3   budget                 4876 non-null   int64
 4   genres                 4876 non-null   object
 5   homepage               4876 non-null   object
 6   id                     4876 non-null   int64
 7   imdb_id                4808 non-null   object
 8   origin_country         4876 non-null   object
 9   original_language      4876 non-null   object
 10  original_title         4876 non-null   object
 11  overview               4876 non-null   object
 12  popularity             4876 non-null   float64
 13  poster_path            4748 non-null   object
 14  production_companies   4876 non-null   object
 15  production_countries   4876 non-null   object
 16  release_date           4876 non-null   object
 17  revenue                4876 non-null   int64
 18  runtime                4876 non-null   int64
 19  spoken_languages       4876 non-null   object
 20  status                 4876 non-null   object
 21  tagline                4876 non-null   object
 22  title                  4876 non-null   object
 23  video                  4876 non-null   bool
 24  vote_average           4876 non-null   float64
 25  vote_count             4876 non-null   int64
```

Figure 2.4: **TMDB's Metadata by Movie ID**.

3. **Movie Credits**: Acquires information on the cast and crew involved in each movie. This includes data about actors, directors, producers, and other key personnel.



```
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      4830 non-null   int64
 1   cast    4830 non-null   object
 2   crew    4830 non-null   object
```

Figure 2.5: **TMDB Movie Credits**.

2.3. Final obtained datasets

The effects of the preprocessing steps are reflected in the final datasets, since the distribution of values across columns changed.

Distribution of Genres before and after Preprocessing We conducted an analysis of the genre distribution within the movies dataset by extracting and counting the frequency of each genre listed in the `Genre` column. The analysis revealed 28 distinct genres with the following frequency distribution:

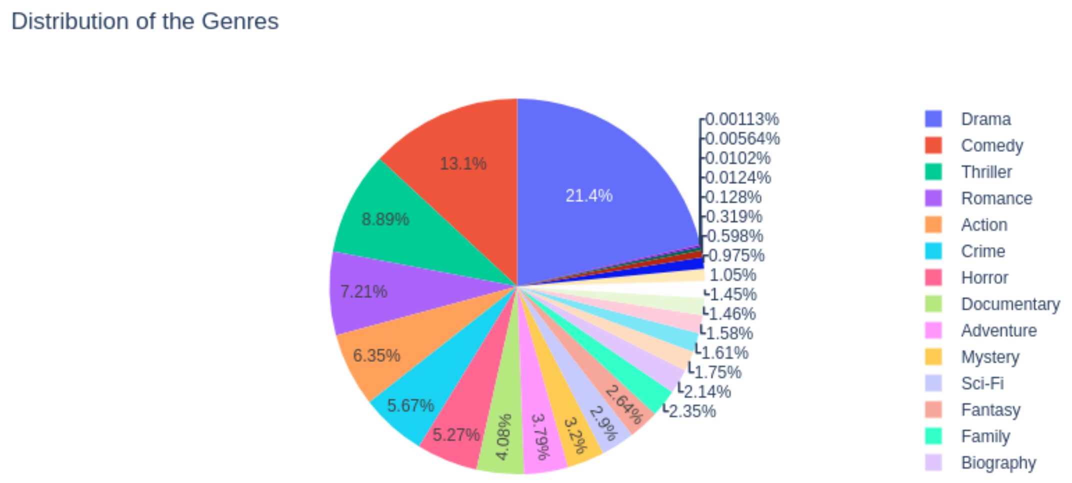


Figure 2.6: **Original Genres Distribution**.

Upon analyzing the genre distribution, we observed that some genres were represented by a very small number of movies, so we decided to discard movies associated with genres that had fewer than 20 instances.

After completing the preprocessing steps for both the movies and ratings datasets, we observed a shift in the genre distribution.

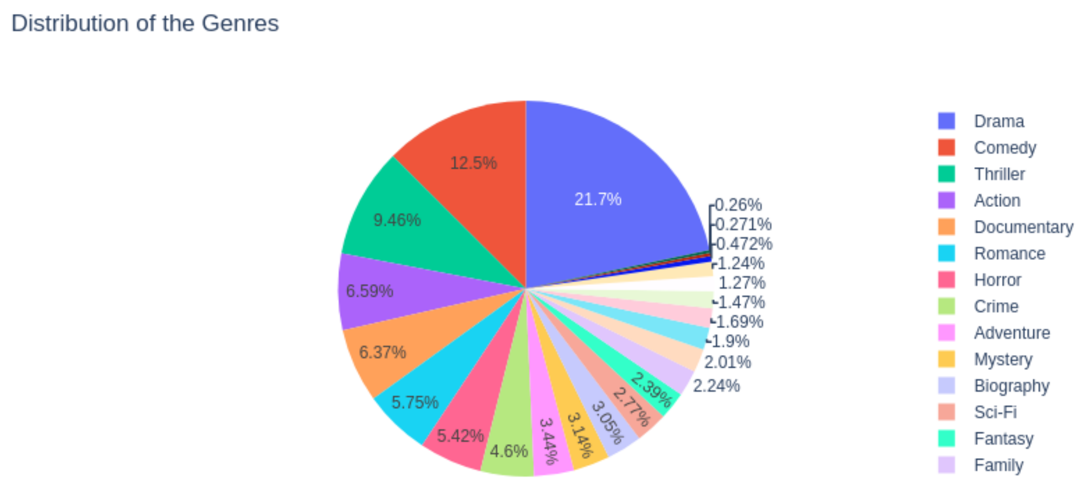


Figure 2.7: **Filtered Genres Distribution**.

Genres Correlogram One key insight from the final dataset is the interdependence of genres, which challenges the initial intuition that genres are entirely independent. The correlogram - presented below - illustrates how genres are often associated with each other, forming distinct clusters within the dataset. The dendrogram at the top suggests genre clustering based on similarity.

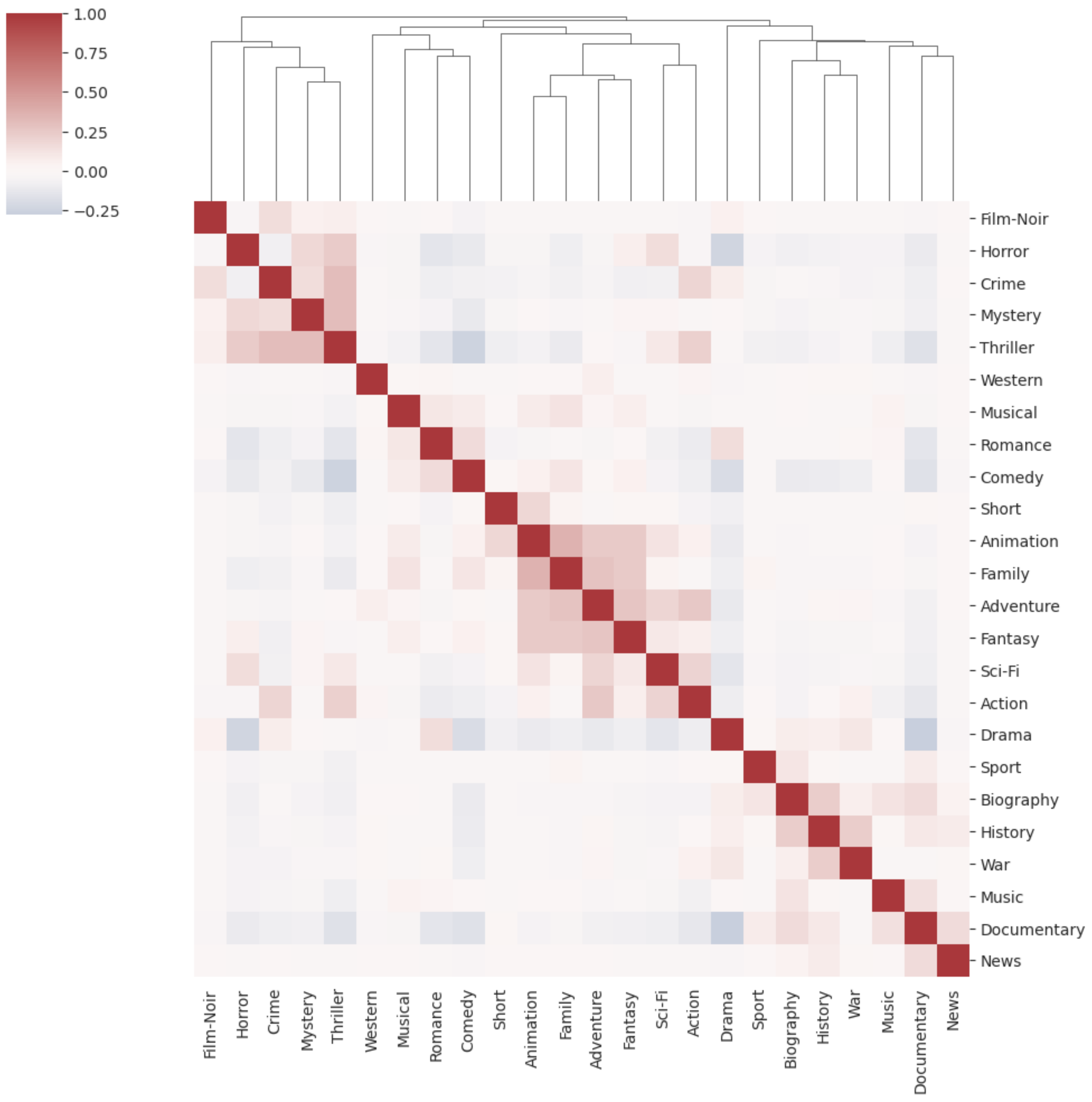


Figure 2.8: **Revealing Relationships**. This heatmap illustrates the correlations between various film genres. The diagonal red squares represent perfect self-correlation, while shades of red and blue indicate positive and negative correlations between different genres.

Distribution of Ratings The distribution of ratings in the dataset reveals a noticeable skew towards higher scores. This observation is illustrated in the graph below, which shows that a significant proportion of ratings are concentrated at the higher end of the rating scale. The skewness towards higher ratings can introduce a bias in the recommendation system, potentially leading to an overemphasis on highly-rated movies. This might affect the diversity of recommendations, as the system could prioritize movies with higher ratings

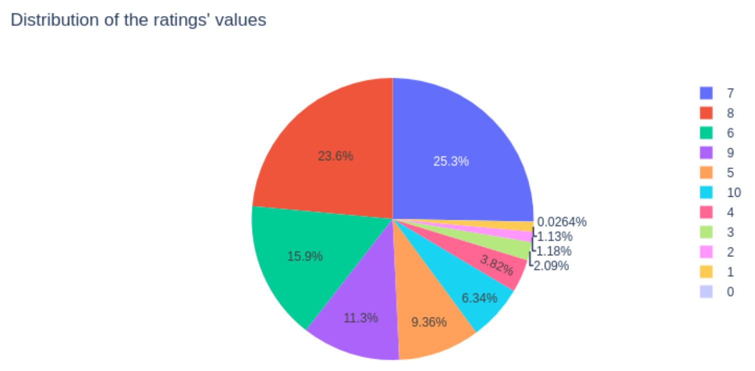


Figure 2.9: **A Skew Towards Higher Scores**. The visualization reveals that the majority of ratings cluster around the higher scores, with ratings of 7 and 8 being the most common.

more frequently.

Distribution of Number of Users per Number of Reviews The analysis of user rating activity in the dataset reveals a distinct trend: as the number of ratings per user increases, the number of users providing those ratings decreases. Specifically, the majority of users have contributed around 25 ratings.

The concentration of ratings around 25 per user indicates that many users have relatively sparse profiles. This sparsity can pose challenges for recommendation algorithms, which rely on sufficient data to generate accurate predictions and recommendations.

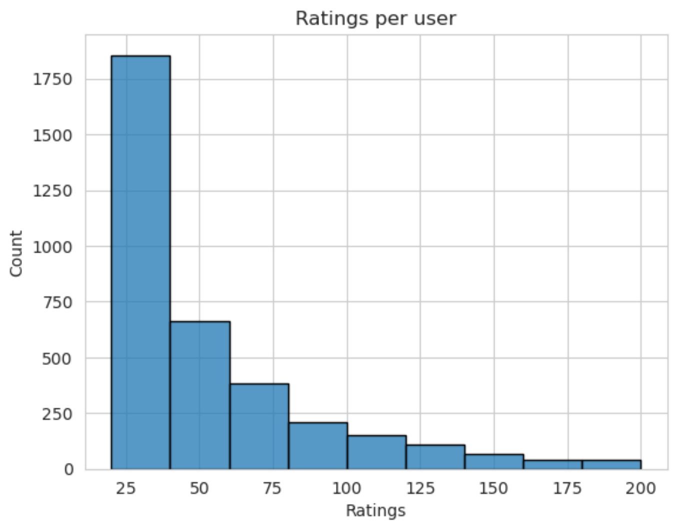


Figure 2.10: **Distribution of Ratings per User**. This bar chart illustrates the frequency of users providing different numbers of ratings. The x-axis shows the number of ratings given, while the y-axis represents the count of users. The data reveals a clear trend of decreasing user counts as the number of ratings increases, with the majority of users providing around 25 ratings.

Temporal Distribution of Movie reviews (2014-2017) The graph reveals distinct peaks in review activity, which are likely associated with major film releases or significant events in the movie industry. A noticeable drop in reviews is evident during the last month of each year, which could be attributed to the holiday season or end-of-year fatigue. Conversely, there is a consistent increase in reviews at the beginning of each new year. This trend suggest that users are more active in reviewing movies shortly after the year ends, possibly due to year-end reflections or resolutions to engage more with new content.

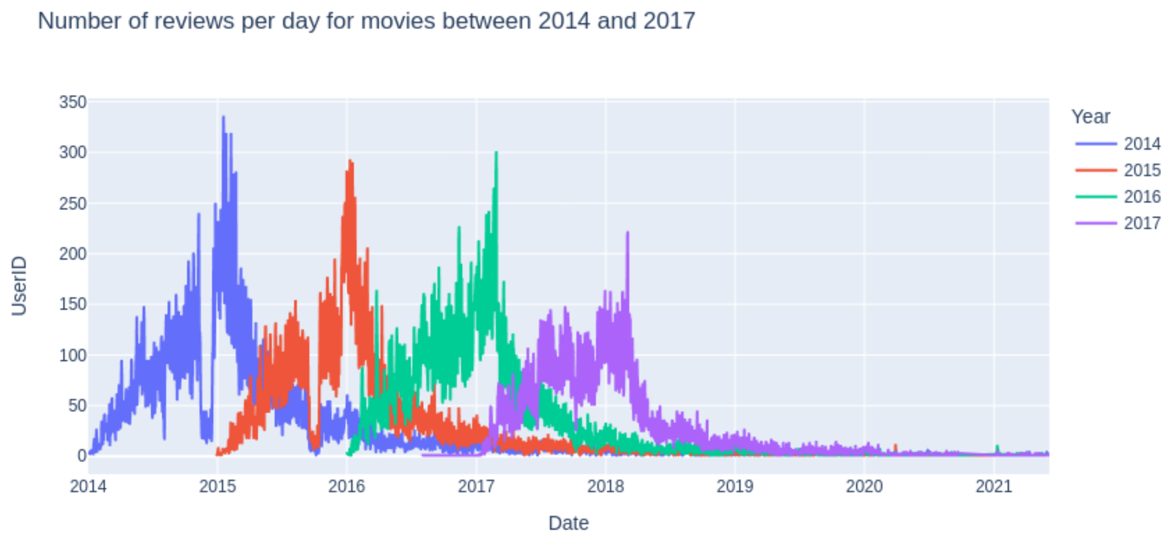


Figure 2.11: **Temporal Distribution of Movie Reviews (2014-2017)**. This line graph depicts the daily number of movie reviews from 2014 to 2017, with data extending into 2021. Each year is represented by a different color, showing distinct peaks in review activity. The graph illustrates how review frequency evolves over time, with noticeable spikes likely corresponding to major film releases or events in the movie industry.

3 | Collaborative filtering Recommendation System

The collaborative filtering approach to recommender systems does not require any item attributes, instead it relies on the opinions of a community of users to make suggestions. More specifically, in user-based collaborative filtering, the recommender system makes recommendations to a user based on the preferences and behaviors of other users who are similar to the target user. The most common input source for a recommender system is the User-Item Interaction Matrix, often referred to as the User Rating Matrix (URM). This matrix stores the interactions between users and items, with users typically represented in the rows and items in the columns. The interactions recorded in the URM can take various forms, such as implicit feedback (e.g., purchases, clicks, views), or explicit ratings as it is our case.

In our work, we have built two collaborative-filtering based systems. The first one is a user-based approach system which we refer to as the "Most Popular" recommender, which suggests movies with the highest mean rating among users. The second system is a Matrix-Factorization approach based on the popularized Funk SVD technique.

3.1. Data preparation for a Collaborative Filtering Recommender

This section didn't require much data preparation, as the preprocessing done earlier was sufficient. We only needed the user rating data, which contains information about the interactions between users and the explicit ratings they have provided, which functions as a COO (Coordinate) sparse representation of the URM (in the notebook is referred to as `flattened_URM`).

3.2. Most Popular recommender

The "Most Popular" recommender is one of the simplest user-based Collaborative filtering techniques. It consists of a recommender system which identifies and suggests the movies that have the highest average rating across all users who have rated that movie.

The underlying assumption is that movies with the highest average rating are likely to be the most popular and well-liked by the user community, and it can be assumed that they will also be liked by the users who haven't watched them yet. The advantage of this approach is that the system will always suggest items that are likely to be liked by users. However, this comes at the cost of making static recommendations that are not personalized to the individual user's preferences.

However, due to characteristics of the URM, we may have some movies that have been rated by hundreds or thousands of users, while others have only a single rating. Blindly computing the mean rating of movies across all users who have rated them would therefore not be appropriate, as movies with just a few good ratings could end up being recommended as the most popular, which would defeat the purpose of a recommender system that aims to base its suggestions on the collective opinions of the user community. To avoid this negative side effect we introduce in the denominator of the mean computation a correcting factor also known as **"Shrinkage factor"**, which helps to lower the importance of movies that have only a few ratings, ensuring that the most popular recommendations are based on a more robust set of user opinions.

Most popular
1. Interstellar
2. Whiplash
3. Hacksaw Ridge
4. La La Land
5. Inside Out

Most popular
1. New Life
2. Wizard Mode
3. Negar
4. Arvydas Sabonis 11
5. Shock Room

Most popular
1. Interstellar
2. Dangal
3. Coco
4. Whiplash
5. Hacksaw Ridge

Figure 3.1: *Comparison of ranking obtained with three different shrinkage factors. From left to right, the shrinkage factor was set to 20, 0 and 3 (the median of number of ratings).*

In Figure 3.1, we can observe how different shrinkage factor values influence the movie rankings. On the left, we have set a shrinkage factor of 20, above which the rankings tend to stabilize. With a shrinkage factor of 0, we can clearly observe the complication we discussed earlier, where directly computing the mean rating leads to movies with just a few good ratings being ranked as the most popular, which defeats the purpose of the recommender system. Finally, with a shrinkage factor equal to 3, which is the median of the number of ratings for each movie, we begin to see fairly popular movies being recommended, but we still have a few outliers.

3.3. CF Recommender System based on Funk SVD matrix factorization

Next, we implemented a method based on a matrix factorization technique that is quite appreciated in the field of Recommender Systems - the Funk SVD. Despite the name, the Funk SVD method is not directly based on the traditional Singular Value Decomposition (SVD). The one thing they have in common is that they are both matrix decomposition processes. Funk SVD was conceived to deal with sparse data, such as the URM, by factorizing the matrix into two smaller, low-rank matrices, without the need to compute the full SVD.

$$R \approx P^T Q \ \text{ where } \ P \in \mathbb{R}^{F\times m}, Q \in \mathbb{R}^{F\times n}$$

The two matrices, P and Q, are obtained through a loss minimization approach. We compute the common squared error loss with Ridge regularization for each non-missing rating, and then use a gradient-based optimization technique, such as Stochastic Gradient Descent (SGD).

$$\mathcal{L}(p,q) = \sum_{(u,i)\in \text{ Train}} \left(r_{ui} - \sum_{f=1}^{F} p_{uf} q_{if} \right)^2 + \lambda \left(\|p_u\|^2 + \|q_i\|^2 \right)$$

To compute the Funk-SVD, we resorted to an already efficient implementation provided by the **surprise** library.

4 | Content based Recommendation System

After implementing the CF system, we developed a content-based (CB) recommender system to offer a different approach to recommendations. While CF relies on user behavior and interactions (e.g., ratings) to suggest items, CB focuses on the content of the movies themselves.

In the CB system, recommendations are generated by analyzing the characteristics of movies, such as genre, actors, directors, and other key attributes. The implemented CB system identifies movies that share similar features to the one inserted by the user as favorite movie, providing personalized recommendations based on the content. This makes CB effective when user data is limited or new items are introduced.

4.1. Data preparation for a Content Based Recommender

For this content-based (CB) recommendation system, we selected key movie attributes: **genre, production companies, actors, directors, writers, production countries, and original language**. These features were chosen because in our opinion they best represent the characteristics that can help users find similar movies. We will also try the same system with a variation of the feature vector, in which we will integrate also the attribute of the **overview**, trying to exploit also the similarity among the descriptions of the movies.

In the preprocessing step of the CB system, we processed the data by handling missing values and combining the selected features into a single textual representation for each movie. Then, using TF-IDF (Term Frequency-Inverse Document Frequency), we transformed the combined text into numerical vectors to represent each movie based on its content.

Finally, we generated a cosine similarity matrix, which computes the similarity between movies. This matrix serves as the foundation for the recommendation system, allowing us to suggest movies based on their similarity to the movies rated by the user. This enriched dataset provides a good basis for making some personalized recommendations using content-based filtering.

4.2. A simple CB Recommender System

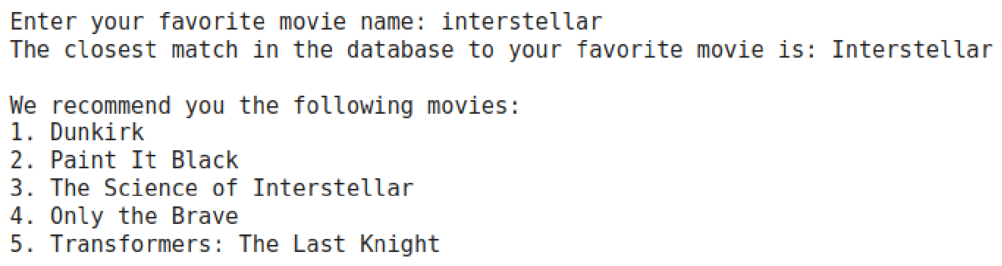
For our simple CB recommender system, we implemented a function that suggests movies similar to a user's favorite movie based on the content features of the movies. The system relies on the cosine similarity matrix derived from the movie attributes in the preprocessing phase.

The function `cb_similar_movies_recommendation` prompts the user to input their favorite movie, finds the closest match in the dataset using the Longest Common Subsequence (LCS) algorithm, and retrieves movies with the highest similarity scores. The top recommendations are then displayed based on the similarity rankings.

To enhance the user experience, we decided to use an auxiliary function, `most_similar_word`, which matches the input movie title to the closest one in the dataset, using the LCS algorithm, ensuring that minor spelling variations are handled effectively. The system returns a list of the most similar movies, allowing users to explore content based on the characteristics of the movie they like.

By combining similarity scores with intuitive title matching, our CB system recommends movies that seem to be somewhat aligned with the user's preferences.

Study case of some recommendations, given by the system:



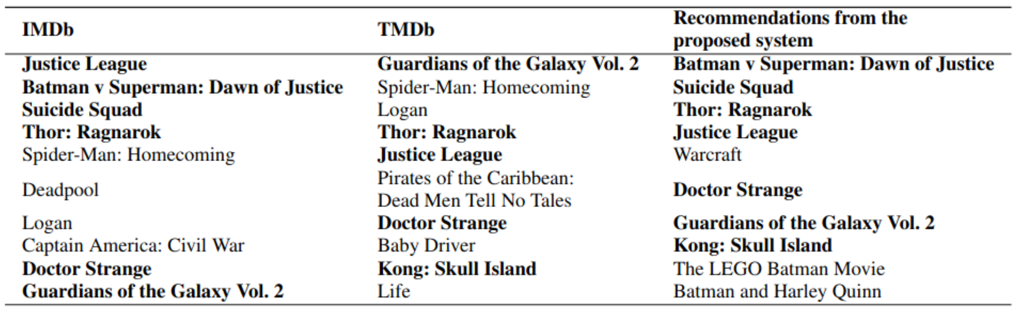
```
Enter your favorite movie name: interstellar
The closest match in the database to your favorite movie is: Interstellar

We recommend you the following movies:
1. Dunkirk
2. Paint It Black
3. The Science of Interstellar
4. Only the Brave
5. Transformers: The Last Knight
```

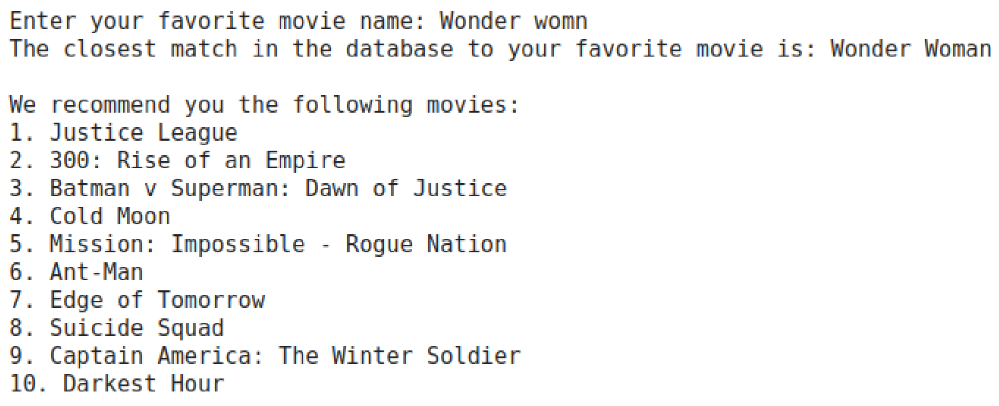
From the results shown in the image, we can see that for the movie **Interstellar**, we obtain 5 different recommendations. After a thorough research and analysis of the results, from a qualitative perspective, we can conclude that the system appears to be working reasonably well by recommending movies that share certain content attributes such as directors, actors, or production companies, as evidenced by Dunkirk and The Science of Interstellar. However, it also generates some less intuitive recommendations (Paint It Black, Transformers: The Last Knight) due to its reliance on attribute similarity rather than focusing deeply on genre and themes. This could be expected given that the genres are just a small part of the features. In fact, with some data analysis previously specified, we identified the presence of around 25 genres, but after enriching the feature vector for the CB, we have these genres contained in the vector made of 53904 words, so they have only a limited impact on the cosine similarity score and consequently on the recommendations.

Compare system recommendations with the paper results:

Table 6: Qualitative analysis of *Wonder Woman* movie: Language (English). Movies in **bold** are intersecting with either IMDb or TMDb



IMDb	TMDb	Recommendations from the proposed system
Justice League	**Guardians of the Galaxy Vol. 2**	**Batman v Superman: Dawn of Justice**
Batman v Superman: Dawn of Justice	Spider-Man: Homecoming	**Suicide Squad**
Suicide Squad	Logan	**Thor: Ragnarok**
Thor: Ragnarok	**Thor: Ragnarok**	**Justice League**
Spider-Man: Homecoming	**Justice League**	Warcraft
Deadpool	Pirates of the Caribbean: Dead Men Tell No Tales	**Doctor Strange**
Logan	**Doctor Strange**	**Guardians of the Galaxy Vol. 2**
Captain America: Civil War	Baby Driver	**Kong: Skull Island**
Doctor Strange	**Kong: Skull Island**	The LEGO Batman Movie
Guardians of the Galaxy Vol. 2	Life	Batman and Harley Quinn



```
Enter your favorite movie name: Wonder womn
The closest match in the database to your favorite movie is: Wonder Woman

We recommend you the following movies:
1. Justice League
2. 300: Rise of an Empire
3. Batman v Superman: Dawn of Justice
4. Cold Moon
5. Mission: Impossible - Rogue Nation
6. Ant-Man
7. Edge of Tomorrow
8. Suicide Squad
9. Captain America: The Winter Soldier
10. Darkest Hour
```

The results show both systems, our CB recommender and the one explained in the paper, recommend superhero and action films related to `Wonder Woman`. From the image, we can also notice that in our system, even if the user misspells the movie, we match the closest one and find the movie without re-asking the user to insert the correct one, enhancing the user experience when interacting with the system. While the system proposed in the paper focuses more narrowly on recent DC Comics movies, our CB system offers a broader range of action/adventure films, including some Marvel titles and historical action movies.

There's some overlap in recommendations like `Justice League` and `Suicide Squad`, but our system includes more diverse choices beyond just the superhero genre. Both systems capture the action theme of `Wonder Woman`, but our CB system recommendations also consider strong female leads and war themes, focusing also on other movie's aspects present in the feature vector. This depends on the attributes chosen to create the feature vector, and different choices can be explored to improve the results. The system might also benefit from improved genre or thematic weighting to make recommendations more aligned with the user expectations for similar films, but this could not always be desirable, reducing serendipity.

4.3. Augmentation of the feature vector for the same Recommender System

The goal of adding the **overview** attribute to the content-based (CB) recommender system was to enhance the recommendations by including a broader textual description of each movie. The **overview** field typically contains a summary of the movie's plot, which we expected to provide additional context about each film's themes, storyline, and general content. By including this attribute, the feature vector representing each movie becomes more detailed, and the overall feature space increases from `53.904` words to `66.974` words. The expectation was that this added information would help the system make more refined distinctions between movies and produce better, more relevant recommendations.

Despite significantly increasing the feature vector when including the movie overview, the system produces more or less the same recommendations as before. For example, for `Interstellar`, the recommendations like `Dunkirk, The Science of Interstellar` and `Transformers: The Last Knight` remain largely unchanged.

Also, for `Wonder woman`, we can see that the recommendations are more or less the same.

```
Enter your favorite movie name: Wonder wmn
The closest match in the database to your favorite movie is: Wonder Woman

We recommend you the following movies:
1. Justice League
2. Batman v Superman: Dawn of Justice
3. 300: Rise of an Empire
4. Ant-Man
5. Darkest Hour
6. Suicide Squad
7. Edge of Tomorrow
8. Mission: Impossible - Rogue Nation
9. Big Eyes
10. Captain America: The Winter Soldier
```

Some of the recommendations, with respect to the CB system with the feature vector without overview, are ordered differently, because the cosine similarity matrix has different values when also using the `overview` attribute, but we can notice that, even if it changes some similarities, in general the same movies remain the ones considered as neighbors of the given one.

While the goal of including the overview was to enrich the feature vectors with narrative details, in practice, the system continues to produce similar recommendations. This suggests that simply adding more text-based information does not always guarantee improved results, especially if the new information doesn't strongly differentiate movies in ways that impact user preferences. The system may require further tuning, such as applying feature weighting or other advanced NLP techniques, to leverage the overview more effectively.

5 | Hybrid Recommendation System

After implementing both the CF and CB systems, we developed a hybrid recommender system to combine the strengths of these two approaches. While CF leverages user behavior and interactions to suggest items, and CB focuses on the content of the movies, hybrid systems aim to integrate both methods to enhance recommendation accuracy and overcome the limitations of each individual approach.

In the hybrid system, we explored two key methods. The first is a mixed approach, which generates recommendations by combining the results of CF and CB through operations like intersection and union, offering suggestions that balance user preferences with content similarity. The second is a meta-level approach, where CF recommendations serve as a starting point, and CB is used to refine or expand those recommendations based on content analysis.

While we expected this combination to yield more robust and accurate recommendations, the evaluation using our precision function actually resulted in lower performance compared to the individual systems. This could be due to an overlap in the limitations of CF and CB, or possibly an insufficient balance in weighting their contributions, which caused the hybrid system to recommend less relevant items in some cases. Additionally, data-related issues, such as insufficient or noisy user interaction data, incomplete content attributes, or too sparse data for certain movies (like movies with one or very few ratings), may have impacted the hybrid system's ability to provide accurate recommendations.

5.1. Hybrid techniques overview

In our research on hybrid recommender systems, we learned about several common techniques for combining CF and CB approaches, each with its own method of leveraging the strengths of both systems:

1. **Weighted Hybrid**: This method combines the outputs of both CF and CB models

by assigning weighted averages to the recommendations from each, balancing the contributions of both systems.

2. **Switching Hybrid**: Here, the system switches between CF and CB depending on specific criteria, such as user behavior or characteristics of the items being recommended, ensuring flexibility in the recommendation process.
3. **Mixed Hybrid**: This technique presents results from both CF and CB models, either showing them separately or combining them, with the intersection or the union of the systems' results. This can help users see a wider variety of recommendations from both systems.
4. **Meta-Level Hybrid**: The meta-level hybrid, works as a cascade, using the output of one recommender system (typically CF) as input for another system (CB), refining or expanding the recommendations based on the content features of the items.

For our project, we implemented two of these hybrid techniques: **Mixed Hybrid and Meta-Level Hybrid**.

We chose to implement the Mixed Hybrid and Meta-Level Hybrid techniques primarily because of their simplicity and clarity. The Mixed Hybrid method, which combines the results of CF and CB systems using union or intersection, is straightforward and easy to interpret, allowing us to use both approaches without introducing unnecessary complexity.

The Meta-Level Hybrid was attractive for its ability to refine CF recommendations using CB. This sequential process combines the user-based strength of CF with the content-based focus of CB, enabling more personalized and accurate suggestions.

The Mixed Hybrid strikes a balance between user-based and content-based recommendations, while the Meta-Level Hybrid helps fine-tune CF results, especially for new or less common items.

5.2. Mixed Hybrid Recommender System

5.2.1. CF and CB extensions for the hybrid recommender

To implement the hybrid system, we first extended the CF and CB recommender systems. For the CF recommender, we only changed some technical aspects of the function in order to reuse it for the hybrid system, so, in particular, we extended the CB recommender algorithm in two ways, allowing it to generate recommendations based on all movies rated by the user rather than just one input movie.

The first algorithm, `cb_recommendation_user_based`, considers all movies a user has rated, and for each one creates the recommendations using the basic CB recommender we explained in the previous chapter.

More specifically, the algorithm works as follows:

1. It fetches the k most similar neighbors for each movie the user has rated.
2. Then calculates a rating prediction for each neighbor by weighting the rating of the watched movie by the similarity score between the neighbor and the watched movie.
3. If a candidate movie is a neighbor to multiple watched movies, its rating prediction is updated to consider the similarity to all those watched movies.
4. Finally, a normalizing factor (the sum of the similarity scores) is applied to bring the weighted rating within the range of 1 to 10. Additionally, a shrinkage factor of 0.1 is used to penalize candidate movies that are neighbors to only a few watched movies.
5. This results in a final weighted score for each recommendation, which is used to rank and provide the top recommendations. The system suggests the top n candidates with the highest predicted ratings.

The second algorithm, `cb_recommendation_user_based_rating_filtering`, also considers all movies rated by the user, but only includes movies where the user's rating is greater than 7. It aggregates recommendations for these higher-rated movies and calculates the average score for each recommended movie based on how frequently it appears. This method filters out, from the start, the movies that may not strongly reflect the user's preferences. In this technique, though, it may also be necessary to fine tune the rating decided to use as higher-rated movies. In our case, we chose a really restrictive threshold.

While both methods extend the original CB approach, we found that the first algorithm consistently provided better accuracy in the scoring, and this result could also be influenced by the threshold chosen in the second algorithm. The chosen algorithm, that will be used in the following hybrid recommender systems, considers all ratings rather than filtering by higher ratings, which may result in more diverse, but still relevant recommendations.

5.2.2. Mixed hybrid recommender implementation

The `ensemble_recommendation_intersection_based` function integrates CF and CB recommendation methods to generate movie recommendations. The implemented algorithm works as follows:

1. **Collaborative Filtering Recommendations**: The function first uses a collaborative filtering model to predict which movies the user might like, based on similar users' preferences. It retrieves a list of movie IDs recommended by this method.
2. **Content-Based Recommendations**: Next, it performs content-based filtering. This involves recommending movies similar to those the user has already rated, using a similarity matrix that measures how alike different movies are. It extracts a list of movie IDs recommended by this approach.
3. **Combining Recommendations**: The function finds the intersection of the CF and CB recommendations. If there are common movies, it returns these as the final recommendations. If not, it combines the recommendations from both methods and sorts them by their similarity scores to provide a cohesive list.
4. **Handling Shortages**: If there aren't enough common recommendations, it supplements the list with additional movies from the combined recommendations, ensuring the final list has the desired number of suggestions.

Overall, this hybrid approach aims to leverage the strengths of both CF and CB methods to provide well-rounded and relevant movie recommendations.

5.3. Meta-level Hybrid Recommender System

The `ensemble_recommendation_meta_level` function implements a meta-level hybrid recommender system that combines CF and CB recommendations to generate movie suggestions. The meta-level hybrid implementation works as follows:

1. **Generating CF Recommendations** : The function begins by using a collaborative filtering model to generate a list of movie recommendations for the user, exactly like the previous mixed hybrid recommender system. This model predicts movies that the user might like based on the preferences of similar users.
2. **Refining Recommendations with CB Data**: For each movie recommended by the CF model, the function then retrieves content-based recommendations. It uses a similarity matrix to find other movies similar to the CF-suggested movies. This helps to enrich the recommendations with additional relevant content.
3. **Aggregating and Scoring**: As it gathers content-based recommendations for each CF-suggested movie, it checks for duplicate entries and collects unique movie IDs

along with their content-based scores. The function aggregates these scores to provide a comprehensive list of recommendations.

4. **Sorting and Returning Recommendations**: The collected recommendations are then sorted based on their scores in descending order. The function returns the top recommendations, ensuring that they are relevant and highly rated according to the content-based approach.

Overall, this meta-level hybrid approach aims to leverage the strengths of both CF and CB methods. By combining CF recommendations with additional CB insights, it seeks to provide more diverse movie suggestions, but it seems to work even worse than the previous model, probably because getting the recommendations of the recommendations we introduce some bias in the system and reduce the precision of the result.

6 | Model evaluation

We evaluated the models in two ways: qualitatively, by analyzing the recommendations given by the systems, some of which we also showed in this report, and quantitatively, by comparing them with a ground truth or by computing some precision metrics on a test set created from the available data.

6.1. Model evaluation using the ground truth

To evaluate our recommendation models, we tried to use data obtained from TMDB. Specifically, we used the API to download what we refer to as the "ground-truth" data, which consists of the movies that TMDB recommends when given a specific movie ID. This ground-truth data could have been important for us because it was intended to serve as a benchmark to assess the accuracy and performance of our model.

After downloading the recommendations for the movies in our system, we went through a preprocessing step to clean and organize the data, making it suitable for evaluation. However, as we started analyzing the ground-truth, we noticed a significant problem. The recommendations provided by TMDB for the movie IDs we were working with did not adhere to the time frame that our model was trained on. While our model was designed to recommend movies released between 2014 and 2017, the TMDB recommendations spanned a much wider range, with movies dating all the way back to the 1980s and continuing up to the present.

This mismatch in time frames caused a fundamental issue in our evaluation process. Since our model only has knowledge of movies released between 2014 and 2017, it struggled to find common ground with the broader set of movies recommended by TMDB. As a result, there was very little overlap between the movies suggested by TMDB and the ones our model was capable of recommending, making a fair comparison impossible. This lack of intersection meant that our evaluation was not meaningful, as we could not accurately measure how well our model performed against a ground-truth set that included movies far outside the time range our system was designed to handle.

Furthermore, it's important to note that the "ground-truth" we obtained from TMDB is not necessarily an objective or definitive set of correct recommendations. The movies provided by TMDB are simply its own algorithm's suggestions based on the given movie ID, and there's no guarantee that these recommendations are actually the most appropriate or relevant ones for users in all cases. So, to properly evaluate our model, we'd need actual user feedback. Relying on TMDB's recommendations alone isn't enough, as we can't confirm they're the best fit for users. User input would help us better assess whether our model is providing useful recommendations.

6.2. Model evaluation without the ground truth, computing the precision

Given that the obtained ground-truth recommends a lot of movies that are not in the desired range of movies from 2014 to 2017, it is really hard to evaluate our model with the obtained data from `TMDB` recommendations, as explained before. It could be used only by expanding our range of interest, so this can be a future development of our project.

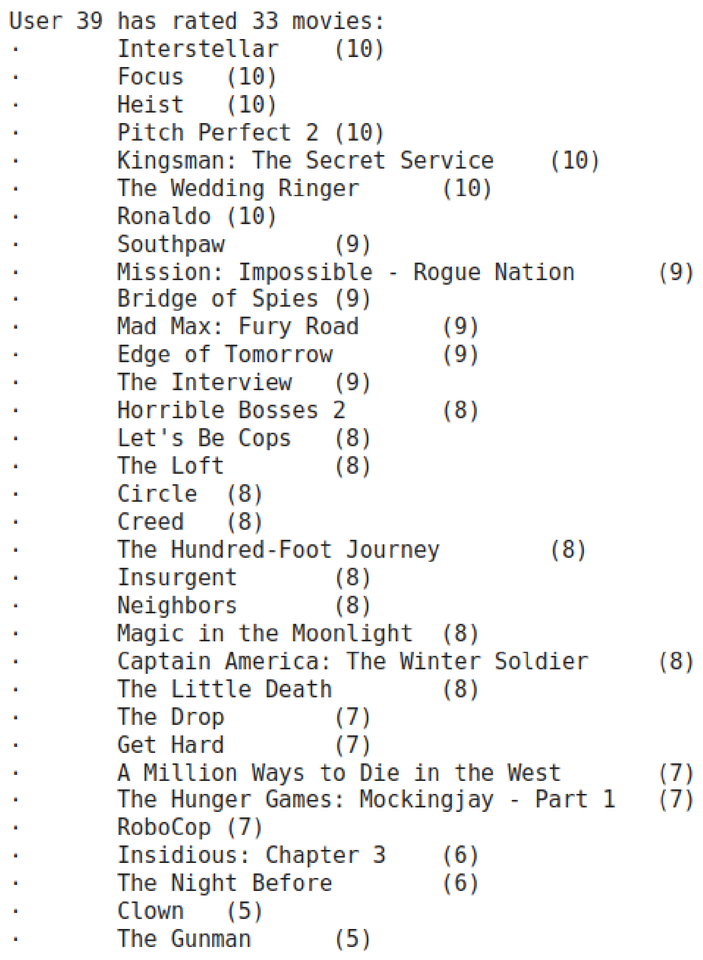
Right now, instead, we will try to evaluate the model without using any ground-truth, by leveraging the already performed train and test split in order to compute some metrics on the models, in particular the `precision@5` metric. From these values and from some recommendations analyzed qualitatively, we try to extract some more insights on our models and our results.

6.2.1. Comparison of the complex models

Let's start from comparing the more complex implemented models, the mixed hybrid recommender system and the meta-level one.

The meta-level hybrid recommender system shows a lower precision compared to the mixed hybrid recommender system, as indicated by its `precision@5` of `0.0078` versus `0.0172` for the mixed system. This difference in performance can be explained by examining the nature of the recommendations each system produces and how they relate to user preferences. We will try, next, to analyze a case study from the qualitative point of view and examine the systems' metrics, to give an explanation of the obtained results.

Qualitative case study on a specific user



```
User 39 has rated 33 movies:
·       Interstellar    (10)
·       Focus   (10)
·       Heist   (10)
·       Pitch Perfect 2 (10)
·       Kingsman: The Secret Service    (10)
·       The Wedding Ringer      (10)
·       Ronaldo (10)
·       Southpaw        (9)
·       Mission: Impossible - Rogue Nation      (9)
·       Bridge of Spies (9)
·       Mad Max: Fury Road      (9)
·       Edge of Tomorrow        (9)
·       The Interview   (9)
·       Horrible Bosses 2       (8)
·       Let's Be Cops   (8)
·       The Loft        (8)
·       Circle  (8)
·       Creed   (8)
·       The Hundred-Foot Journey        (8)
·       Insurgent       (8)
·       Neighbors       (8)
·       Magic in the Moonlight  (8)
·       Captain America: The Winter Soldier     (8)
·       The Little Death        (8)
·       The Drop        (7)
·       Get Hard        (7)
·       A Million Ways to Die in the West       (7)
·       The Hunger Games: Mockingjay - Part 1   (7)
·       RoboCop (7)
·       Insidious: Chapter 3    (6)
·       The Night Before        (6)
·       Clown   (5)
·       The Gunman      (5)
```

We compared the mixed hybrid and meta-level recommender systems for user 39, who has rated 33 movies. The mixed hybrid system recommended films like `Pitch Perfect 3` and `Neighbors 2`, which are sequels to movies the user rated highly. It also suggested films in line with the user's taste for action and drama, such as `Dunkirk and The Hateful Eight`. This shows the system effectively considers user preferences, including genre and sequels, resulting in better-targeted recommendations.

On the other hand, the meta-level recommender suggested less relevant content, including short films like `Feast`, `Lava` and `Interstellar`, which the user had already rated, given that they are not excluded from the new recommendations of the CF-recommendations results. This system seems to rely more on metadata, leading to less useful recommendations for this user. The precision values highlight this difference, indicating that the mixed hybrid system performs better, giving more relevant suggestions by aligning better with the users' viewing history.

Possible reasons of lower precision in the meta-level hybrid system:

The meta-level system aggregates recommendations from content-based approaches based on CF-suggested movies. If CF recommendations do not align closely with user preferences, the content-based recommendations derived from these suggestions might not be so relevant. So, the meta-level approach can sometimes lead to irrelevant recommendations if the initial CF suggestions are not well-targeted. Since content-based recommendations are based on these suggestions, they might also miss the mark, resulting in less relevant final recommendations. The system, also, relies heavily on content similarity rather than user-specific interactions. As a result, it might recommend movies that are similar in content but not necessarily aligned with the user's personal tastes or high ratings. On the contrary, the mixed hybrid system might better balance CF and CB methods to target user preferences more accurately.

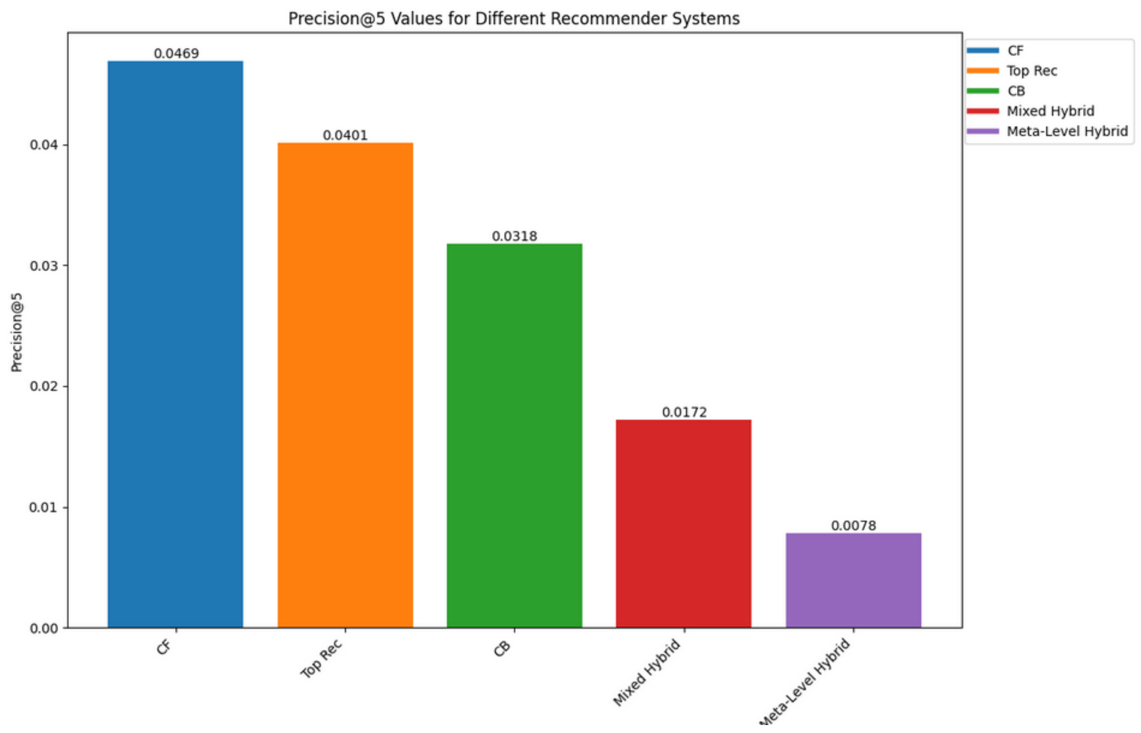
6.2.2. Comparison of hybrid models with the basic ones

We performed a train test split on the user-movie-rating list (`flattened_URM`) and the split consisted of simply hiding some user-movie rating entries to then evaluate the capability of the system to recommend these hidden items. As can be imagined, this technique does not offer the best evaluation metric. This is because a suggested movie may be just as well-liked by a user, but the mere fact that this movie is not present in the dataset constitutes an incorrect prediction, which penalizes the overall accuracy metric of the model. The precision function was computed as:

$$\text{Precision@k} = \frac{1}{N}\sum_{u=1}^{N}\frac{|\ \text{relevant}(u)\ \cap\ \text{recommended}(u,k)\ |}{|k|} \tag{6.1}$$

Where for relevant items, we intend movies in the test set with a rating above a predefined threshold, which we set to 5.

The observed `precision@5` scores indicate that both the mixed hybrid and meta-level hybrid recommender systems underperform compared to the individual CF and CB recommenders.



The hybrid systems, as observed in terms of precision, are less effective than the individual CF and CB recommenders due to the integration challenges they present. The mixed hybrid approach might not effectively combine recommendations, leading to less relevant suggestions, while the meta-level system's reliance on CF-based suggestions to drive content-based recommendations can result in less precise outcomes if the initial CF suggestions are not well-aligned with user preferences. In contrast, individual CF and CB methods are more specialized and optimized for capturing specific aspects of user preferences, leading to higher precision in their recommendations.

Thus, the hybrid approaches struggle with the integration and reliance on potentially flawed initial recommendations, resulting in lower precision compared to the more focused and effective individual CF and CB methods. Also, the sparsity of data in the dataset, particularly the lack of reviews for many movies, can pose a significant challenge when combining the methods. Both CF and CB may encounter difficulties when working with sparse user data, and their combination may amplify these issues rather than resolve them, leading to poorer results for the hybrid system than for the individual systems. This also explains why the precision values of the Most Popular recommender and the more sophisticated Funk SVD-based approach are not vastly different. The sparsity of the data, combined with its skewness toward a few highly popular items frequently rated by users, does not allow us to appreciate the effectiveness of the Funk SVD, as in this scenario even the simpler Most Popular approach is able to achieve comparable performance.

Bibliography

[1] **GitHub repository that contains the notebook of all the experiments described in this report** https://github.com/BiancaSavoiu/NAMLproject