TU DUBLIN, TALLAGHT CAMPUS

BSc in Computing with AI/ML

Movie Recommendation System

Braulio Fenollosa López

Supervised by:
John Cardiff

January 15, 2025

Contents

1 Introduction

In an era where entertainment options are abundant, finding the right movie to watch can be a challenging task for viewers. To address this, recommendation systems have emerged as a pivotal tool, leveraging data and algorithms to provide personalized suggestions. This project focuses on developing a movie recommendation system that combines content-based techniques and an interactive user interface.

The foundation of this system lies in a meticulously curated dataset, enriched with additional metadata from the OMDb API, to ensure detailed and high-quality information about each movie. The recommendation engine uses vectorization of TF-IDF and cosine similarity to identify and rank movies that align with user preferences. Complementing this robust backend is a user-friendly web application built using Streamlit, allowing users to interact dynamically with the system by selecting movies, applying filters, and receiving tailored recommendations.

By addressing the growing demand for personalized content and showcasing a scalable, adaptable design, this project not only highlights the potential of recommendation systems but also paves the way for further innovations in this domain.

2 Data Collection

The data collection process for this project focused on compiling, cleaning, and enriching a dataset of movies to build a robust recommendation system. The methodology combined leveraging a pre-existing dataset, integrating external API data, and applying normalization techniques to ensure high-quality and comprehensive information. Below, the steps involved in the data collection are detailed.

2.1 Initial Dataset

The foundation of this project was an initial dataset extracted from the MovieLens database, a widely recognized resource for movie recommendation research. This dataset included essential attributes such as:

- **Movie ID**: A unique identifier for each movie.
- **Title**: The name of the movie.
- **Release Year**: The year the movie was released.
- **Genres**: A set of genres associated with the movie (e.g., Action, Comedy, Drama).
- **IMDB URL**: A direct link to the movie's page on IMDB.

The data was loaded into a Pandas DataFrame for further processing. The dataset provided a structured starting point but lacked rich descriptive attributes like detailed plots, ratings, and box office performance, which are crucial for recommendation systems.

2.2 Data Cleaning and Normalization

To prepare the data for further use, cleaning and normalization steps were implemented. One significant transformation involved correcting movie titles to ensure consistency. Some titles contained prepositional phrases or articles (e.g., "The", "A", "An") at the end, which were moved to the beginning for better readability and uniformity, especially in order to make clear requests to the API. This was achieved using a Python function that:

- Splits the title into components based on commas.
- Identifies common prepositions or articles.
- Reconstructs the title in a standardized format.

Additionally, missing or incorrect entries in key columns were identified and corrected where possible.

2.3 Integration of OMDb API Data

While the MovieLens dataset provided a foundational structure, its descriptive depth was insufficient for the desired recommendation functionality. To address this, the Open Movie Database API (OMDB) was used to enrich the data set. The OMDb API offers detailed metadata about movies, including:

- IMDB Ratings: Viewer ratings as recorded on IMDB.
- Genres: Additional or more specific genre information.
- Plot Summary: A brief description of the story of the movie.
- Runtime: Duration of the movie.
- Box Office Performance: Revenue data for the movie.
- Metascore and IMDB Votes: Indicators of critical reception and audience.
- Poster URL: Links to promotional images or posters.

2.3.1 Implementation of API Integration

1. **API Access**: An API key was used to authenticate requests. The title of each movie was passed as a query parameter in the HTTP GET requests to the OMDb endpoint.
2. **Data Enrichment**: A custom Python function iterated over the movie titles in the DataFrame. For each movie:
   - The API response was parsed to extract relevant fields.
   - New columns were added to the DataFrame to store the enriched attributes.
   - Error handling mechanisms ensured that API failures or missing data did not disrupt the process.
3. **Rate Limiting**: To adhere to OMDb's rate limits (one request per second for free-tier users), a delay was introduced between consecutive API calls. This ensured compliance and prevented service disruption.
4. **Handling Missing Data**: Not all movies in the data set had corresponding entries in OMDb. For such cases, the missing attributes were left as null, allowing the recommendation system to gracefully handle incomplete data during processing.

2.4 Challenges and Solutions

The data collection process faced several challenges, including:

- **Data Inconsistency**: The titles in the MovieLens dataset occasionally did not match the OMDb naming conventions. This was mitigated by preprocessing titles to remove discrepancies, such as extra spaces or inconsistent capitalization.
- **Rate Limitations**: The free-tier limitation of one API call per second significantly extended the time required to process the dataset. Implementing a delay and optimizing the sequence of API calls minimized disruptions.
- **API Failures**: Intermittent connectivity issues or unavailability of certain movies in OMDb required robust error handling. The implemented solution skipped failed requests and logged them for further investigation.

2.5 Final Dataset

After enriching the MovieLens dataset with OMDb data, the final dataset included a wide array of attributes, making it highly suitable for a recommendation system. The structure included:

- Core attributes of the MovieLens dataset.
- Enriched metadata from OMDb, including ratings, posters URLs, and plot summaries.

This comprehensive data set forms the basis for the recommendation system, enabling it to provide personalized and nuanced movie suggestions based on user preferences and movie characteristics.

3 Data Preprocessing

Data preprocessing is a critical step in preparing the dataset for analysis and model development. For this project, the preprocessing phase focused on exploring the enriched dataset, identifying and addressing data quality issues, and ensuring that the data werea were in a suitable format for building a recommendation system. The following sections detail the steps performed during preprocessing.

3.1 Loading the Dataset

The enriched dataset, which was augmented with additional metadata from the OMDb API, was loaded from a CSV file. The Pandas library was utilized for this purpose due to its efficient data handling capabilities. The initial load provided a structured view of the dataset, allowing for immediate inspection and manipulation. The dataset included a wide range of attributes, such as movie titles, genres, release years, and additional descriptive and numerical features obtained during the enrichment process.

3.2 Exploratory Data Analysis (EDA)

A thorough exploration of the dataset was performed to understand its structure and content. Key exploratory steps included:

- **Inspecting Data Samples:** Using the `head()` method, the first 10 rows of the dataset were reviewed to gain insights into its format and verify the presence of expected columns. This step also ensured that the enrichment process had been successfully applied.
- **Checking Dimensions:** The `shape` method provided the number of rows and columns in the dataset, confirming its scale and completeness. This information was essential for planning subsequent steps.
- **Descriptive Statistics:** The `describe()` method was used to calculate summary statistics for numerical columns, including mean, standard deviation, minimum, and maximum values. These statistics highlighted the distribution of numeric attributes, such as IMDB ratings and box office revenue.

3.3 Handling Missing Data

Missing data can significantly impact the performance of machine learning models. To address this, the dataset was examined for null or missing values using the `isna().sum()` method. This analysis revealed:

- **Columns with Missing Values:** Certain attributes, particularly those derived from the OMDb API, contained null values due to unavailable data for specific movies.
- **Strategies for Handling Missing Data:**
  - For critical columns like movie titles and release years, rows with missing values were marked for review or exclusion.

– For optional attributes like box office revenue or IMDB scores, where the number of null values was high, they were removed completely from the database.

3.4 Data Cleaning and Formatting

Although the correction of movie titles (e.g., reordering articles like "The" and "A") was performed earlier during data enrichment, additional cleaning steps were applied here to ensure consistency:

- **Ensuring Data Types:** Columns were reviewed to confirm that their data types matched their intended use. For instance, numeric columns like IMDB ratings were checked for non-numeric entries or formatting issues.
- **Removing Duplicates:** Any duplicate entries in the dataset were identified and removed to avoid redundancy and potential bias in recommendations. The same was done with redundant columns.
- **Standardizing Genres:** Genre columns, often represented as comma-separated strings, were inspected to ensure consistent naming conventions and formatting, converting them into a binary column format for each genre.

3.5 Summary of Preprocessed Data

After completing the preprocessing steps, the dataset was validated to ensure it met the requirements for subsequent modeling. Key attributes of the preprocessed dataset included:

- **Cleaned Titles:** Movie titles were normalized to a consistent format.
- **Complete Key Attributes:** Essential columns, such as movie ID, title, and genres, were verified for completeness.
- **Managed Missing Data:** Columns with missing values were either removed or left as nulls, depending on their significance.
- **Standardized Structure:** All columns were formatted to ensure compatibility with recommendation algorithms.

3.6 Challenges and Insights

During the preprocessing phase, several challenges were encountered, including:

- **Irregularities in Metadata:** Data sourced from the OMDb API occasionally included inconsistencies or errors, such as unusual formats in runtime or genre fields. These were manually inspected and corrected where necessary.
- **Managing Missing Data:** Balancing the need to retain as much data as possible while ensuring its quality required careful decision-making. For instance, rows with missing critical attributes were excluded, while non-critical missing values were handled more leniently.

The preprocessing phase provided valuable insights into the dataset's structure and quality. This understanding informed decisions about feature engineering and model design in subsequent phases of the project.

4 Models and Evaluation

In this project, we developed a content-based movie recommendation system. Below, we describe the strategy employed, including the models, techniques, and evaluation processes used.

4.1 Data Preprocessing

The dataset was sourced from an enriched CSV file named `items_enriched_final.csv`. This file contains detailed information about movies, including their titles, plots, release years, and genre information. To create a feature-rich representation of each movie, we followed these steps:

- A new column, `combined_features`, was generated by concatenating the movie title, plot description, release year, and genres.
- Genres were dynamically extracted for each movie by filtering columns where the genre indicator was set to 1. These genres were then concatenated into a single string.
- Missing values in the plot description were handled by replacing them with empty strings.

This approach ensured that relevant textual and categorical information about each movie was captured in a single representation.

4.2 Feature Representation

To enable comparison between movies, we transformed the `combined_features` column into a numerical representation using the following technique:

1. **TF-IDF Vectorization:**
    - We used the `TfidfVectorizer` from the `scikit-learn` library to convert textual data into a matrix of TF-IDF features. This method evaluates the importance of terms in a document relative to their occurrence in the dataset.
    - Stop words were removed to avoid common but irrelevant terms from influencing the similarity calculation.

The resulting TF-IDF matrix is sparse and high-dimensional, with each row representing a movie and each column corresponding to a term in the vocabulary.

4.3 Similarity Calculation

To identify movies similar to user-selected ones, we calculated the cosine similarity between movies:

- For a given set of selected movies, we extracted their corresponding rows from the TF-IDF matrix.
- The mean similarity scores for the selected movies were computed by averaging their pairwise cosine similarities with all other movies.

- This approach allowed us to prioritize movies that were collectively similar to the user's choices.

4.4 Recommendation Strategy

The recommendation process involved ranking movies based on similarity scores and applying optional filters to refine the results:

1. **Ranking:**
    - Movies were sorted in descending order of their similarity scores.
    - The top N recommendations were selected, where N was defined by the user (default: 10).
2. **Filters:**
    - Users could apply additional filters to customize the recommendations. These filters could target attributes such as genre, year, or other metadata.
    - Filters were implemented as flexible conditions, supporting callable functions, specific values, or ranges.
3. **Exclusion of Selected Movies:**
    - To avoid redundancy, movies already selected by the user were excluded from the final recommendation list.

4.5 Evaluation

The performance of the recommendation system was evaluated using both qualitative and quantitative approaches:

1. **Relevance of Recommendations:**
    - To assess the system's effectiveness, we conducted a manual evaluation using a controlled test. Five movies were selected as input, and a test set of 30 movies deemed relevant to these five was prepared. The system was expected to recommend movies from this test set.
2. **Precision Metrics:**
    - **Precision@10:** The fraction of relevant movies in the top 10 recommendations. This metric was measured at 0.7, indicating that 7 out of the top 10 recommendations were relevant to the input movies.
    - **Overall Precision:** The ratio of relevant movies recommended to the total number of relevant movies in the test set. This metric was computed as 0.234, reflecting that 23.4% of all relevant movies were successfully recommended.
3. **Interpretation of Results:**

- The high Precision@10 suggests that the system performs well at identifying the most relevant recommendations, particularly for a small set of top-ranked results. This aligns with user expectations in real-world scenarios, where the top few recommendations are critical.
- The lower overall precision highlights the challenge of covering all relevant movies, especially in a larger dataset. This indicates potential for improvement in diversifying and expanding the recommendation set.

4. **Scalability Testing:**

   - The system's computational efficiency was tested by measuring the time required to process the dataset and generate recommendations. Results showed that the TF-IDF vectorization and cosine similarity calculations scaled well with the dataset size, making the system suitable for large-scale applications.

5. **Diversity:**

   - The diversity of recommendations was examined to ensure that the system did not overly concentrate on a narrow subset of movies. By including multiple selected movies in the similarity computation, the system generated a broader range of suggestions.

4.6 Evaluation Results

- **Processing Time:** The preprocessing and vectorization steps were completed in under a few seconds for a dataset of approximately 1,600 movies, demonstrating good scalability.
- **Precision Metrics:** The Precision@10 score of 0.7 indicates strong performance in prioritizing highly relevant recommendations. The overall precision of 0.234 suggests room for improvement in covering a broader range of relevant movies.
- **Qualitative Feedback:** When tested with various input selections, the recommendations consistently included relevant and diverse movies, satisfying different user preferences.

5 Interactive Application

As part of this project, we developed an interactive movie recommendation application using Streamlit, a Python framework for building web applications. This section details the design, functionality, and implementation process of the app.

5.1 Objective

The purpose of the application was to provide an intuitive interface for users to:

1. Browse and select movies based on their preferences.
2. Apply filters to refine recommendations.
3. Receive personalized movie recommendations dynamically.

5.2 Application Workflow

The app guides users through a series of steps:

1. **Welcome Page:**
   - Users are greeted with an introductory page, including a description of the system's capabilities.
   - A call-to-action button allows users to start exploring movies.
2. **Movie Browsing and Selection:**
   - Movies are displayed in pages, each containing a balanced selection of films across genres. This ensures diversity and prevents overrepresentation of specific genres.
   - Users can select movies they like by checking boxes associated with each title. Selected movies are highlighted in the sidebar for easy reference.
3. **Filter Application:**
   - Filters are available in the sidebar, allowing users to refine recommendations based on:
     - Genre: Multi-select dropdown to specify desired genres.
     - Year: Slider to include movies released after a specific year.
     - IMDb Score: Slider to set a minimum rating for recommended movies.
4. **Movie Recommendations:**
   - Once movies are selected, users can generate recommendations. The system displays up to 10 personalized recommendations, dynamically updated based on filters and selections.
5. **Detailed Movie Information:**
   - Each recommended movie is presented with its poster, title, release year, plot summary, runtime, and IMDb score.

5.3 Technical Implementation

1. **Data Handling:**
    - The app utilizes a preprocessed dataset (`items_final_poster.csv`), containing detailed metadata for movies, including titles, genres, plots, posters, and IMDb scores.
    - Preprocessing steps (e.g., combining features for recommendations) are implemented directly in the app script to ensure seamless integration.
2. **Streamlit Features:**
    - **Session State:**
        - Variables such as the current page, selected movies, and filter preferences are stored using Streamlit's session state. This enables smooth navigation and user experience across pages.
    - **Sidebar:**
        - The sidebar houses filter options and a summary of selected movies, enhancing usability and accessibility.
    - **Columns:**
        - Movies and recommendations are displayed using Streamlit's column layout, ensuring a visually appealing and organized interface.
3. **Recommendation Algorithm:**
    - The recommendation engine relies on TF-IDF vectorization and cosine similarity, implemented in the backend. Selected movies serve as the basis for calculating similarity scores with other movies.
    - Filters are applied programmatically to refine the results based on user-defined criteria.
4. **Pagination:**
    - To enhance browsing, the app divides movies into pages. Each page contains 9 movies, and users can navigate between pages using "Previous" and "Next" buttons.

5.4 Key Features

1. **Balanced Initial Selection:**
    - Movies displayed in the browsing phase are sampled randomly across genres, ensuring a balanced selection.
2. **Dynamic Recommendations:**
    - The recommendations update dynamically as users select movies and adjust filters, providing an engaging and responsive experience.
3. **Interactive Sidebar:**

- The sidebar includes filters, selected movies, and user options, centralizing control and improving accessibility.

4. **Visual Elements:**
    - Posters and detailed descriptions make the app visually appealing and informative.

5.5 Challenges and Solutions

1. **Diversity in Recommendations:**
    - **Challenge:** Ensuring diverse recommendations that are not overly similar to the selected movies.
    - **Solution:** Calculating average similarity across multiple selected movies and limiting the number of recommendations per genre.
2. **User Engagement:**
    - **Challenge:** Maintaining user interest during the browsing phase.
    - **Solution:** Implementing pagination and displaying visually appealing posters to create a more interactive experience.
3. **Performance:**
    - **Challenge:** Efficiently handling large datasets and real-time updates.
    - **Solution:** Optimizing TF-IDF vectorization and precomputing genre-based selections to reduce computation time.

5.6 Evaluation of the App

The app was tested for functionality, usability, and performance:

1. **Functionality:**
    - All features, including browsing, filtering, and recommendations, worked as intended.
2. **Usability:**
    - User feedback highlighted the intuitive layout and ease of use.
    - Visual elements, such as movie posters and detailed descriptions, enhanced user engagement.
3. **Performance:**
    - The app handled datasets with thousands of movies efficiently, with recommendations generated in real-time without noticeable delays.

6 Future Work

While the movie recommendation system and its accompanying interactive app provide a strong foundation, there are several areas for improvement and extension. This section outlines potential future work and enhancements that can improve the system's functionality, scalability, and user experience.

6.1 Expanding the Database

To improve the scope and quality of recommendations:

- **Adding Movies:** Regularly updating the database with new movie releases and older classics ensures the system remains relevant and comprehensive.
- **Enriching Metadata:** Incorporating additional information such as actors, directors, awards, and user reviews can enhance the feature set, leading to more accurate recommendations.

6.2 Improving the Similarity Model

The current model is based on TF-IDF vectorization and cosine similarity. Future improvements could include:

- **Advanced NLP Models:** Using pre-trained embeddings like BERT or Word2Vec to capture contextual and semantic relationships between movies more effectively.
- **Feature Weighting:** Assigning different weights to features like plot, genre, and year to better reflect their importance in recommendations.
- **Hybrid Systems:** Combining content-based filtering with collaborative filtering to leverage both user preferences and item similarity.

6.3 Enhancing the Interactive Application

The interactive app could be improved in several ways:

- **User Authentication:** Implementing user accounts to allow users to save their preferences, history, and recommendations for future sessions.
- **Enhanced Filters:** Adding filters for more specific attributes, such as runtime, language, director, or availability on streaming platforms.
- **UI Improvements:** Fixing issues with navigation buttons and optimizing the layout for better usability on different devices.
- **Open Access:** Publishing the app on GitHub or other platforms to make it freely accessible to a wider audience.

6.4 Addressing Current Limitations

1. **Cold Start Problem:**
    - Incorporating external data sources, such as user ratings or social media mentions, could help address the issue of limited metadata for certain movies.
2. **Collaborative Filtering:**
    - Including user behavior data, such as viewing history and ratings, can enhance the recommendation quality by identifying shared preferences among users.
3. **Evaluation Metrics:**
    - Incorporating rigorous evaluation metrics like recall, mean reciprocal rank (MRR), and normalized discounted cumulative gain (NDCG) will provide a more quantitative assessment of the system's performance.

6.5 Publishing and Sharing

To promote collaboration and transparency:

- **Open-Source Repository:** Publishing the app and codebase on GitHub will allow others to contribute to its development and adapt it for their own use.
- **Documentation:** Providing detailed documentation, including setup instructions and a user guide, to facilitate adoption by developers and researchers.

6.6 Advanced Features for Recommendations

To make the recommendation system more robust and versatile:

- **Real-Time Recommendations:** Implementing streaming data pipelines to update recommendations based on real-time user interactions.
- **Social Recommendations:** Incorporating social media data and user networks to provide recommendations based on friends' preferences.
- **Recommendation Diversity:** Ensuring that recommendations include a balance of highly relevant and diverse suggestions to avoid overspecialization.

6.7 Scalability and Performance

As the system grows:

- **Optimized Algorithms:** Refactoring computationally intensive processes to improve efficiency.
- **Distributed Systems:** Implementing distributed computing solutions for handling larger datasets and concurrent user requests.

7 Conclusion

The movie recommendation system developed in this project successfully combines a sophisticated content-based approach with an interactive and visually appealing application. Through the use of enriched metadata and TF-IDF vectorization, the system delivers accurate and relevant suggestions, while the Streamlit-powered interface provides an engaging and intuitive user experience.

Key achievements include:

- **Data Enrichment**: The integration of OMDb API data significantly enhanced the dataset.
- **Recommendation Accuracy**: Metrics like Precision@10 demonstrate the system's effectiveness.
- **Scalability and Usability**: The app handles a dataset of thousands of movies efficiently.

Despite its success, the system faces challenges such as the cold start problem and the absence of collaborative filtering. Addressing these limitations and implementing advanced features form the basis for future work.

8 References

- MovieLens Dataset. GroupLens Research. Available at: `https://grouplens.org/datasets/movielens/`
- OMDb API Documentation. Available at: `https://www.omdbapi.com/`
- Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12, pp. 2825-2830.
- Streamlit Documentation. Available at: `https://docs.streamlit.io/`
- Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval.* Cambridge University Press.
- Ricci, F., Rokach, L., & Shapira, B. (2011). *Introduction to Recommender Systems Handbook.* Springer.
- Aggarwal, C. C. (2016). *Recommender Systems: The Textbook.* Springer.
- Hu, Y., Koren, Y., & Volinsky, C. (2008). "Collaborative Filtering for Implicit Feedback Datasets." *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining,* pp. 263-272.
- Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). "Deep Learning Based Recommender System: A Survey and New Perspectives." *ACM Computing Surveys (CSUR),* 52(1), pp. 1-38.
- Burke, R. (2002). "Hybrid Recommender Systems: Survey and Experiments." *User Modeling and User-Adapted Interaction,* 12(4), pp. 331-370.