Collaborative Movie Recommendation System

Mohammed Zakiuddin
Computer Science and Engineering
University of Texas at Arlington
Arlington, United States of America
mohammedahmed.zakiuddin@mavs.uta.edu

***Abstract*—Recommendation systems have become an integral component of modern online platforms, particularly in streaming services that assist users in discovering new content. Among these, movie recommendation systems have gained widespread adoption due to their ability to enhance user engagement and satisfaction. Collaborative filtering is a widely used technique that leverages user behavior data to predict preferences and generate personalized suggestions. In this paper, we present a movie recommendation system based on collaborative filtering, which analyzes user interactions to recommend films aligned with individual interests. The system is evaluated using real-world movie rating data, demonstrating promising results in terms of prediction accuracy and effectiveness.**

I. Introduction

The goal of this project is to implement a movie recommendation system using collaborative filtering techniques. These systems are extensively employed in the entertainment industry to personalize user experiences and increase engagement. Collaborative filtering is known for its effectiveness in generating accurate recommendations by analyzing patterns in user behavior and preferences.

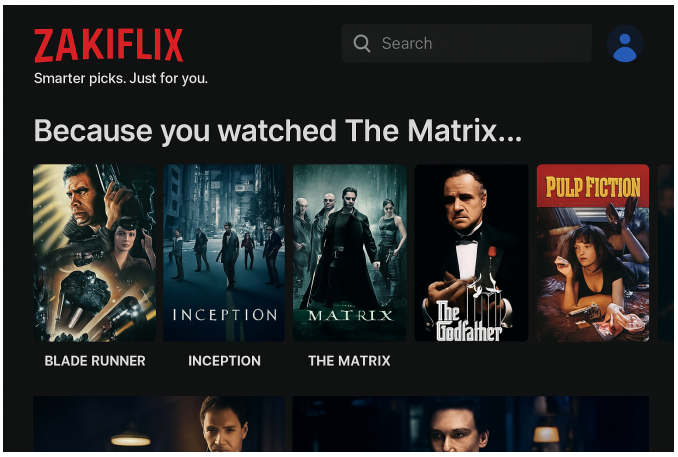

Fig. 1. ZakiFlix UI Mockup: A modern Netflix-style user interface illustrating personalized movie recommendations.

However, building such systems presents several challenges, including handling large-scale datasets, ensuring scalability, and addressing the cold-start problem—where the system has limited information about new users or items. The datasets used in this project include `movies.csv`, which contains movie titles, genres, and movie IDs, and `ratings.csv`, which includes user IDs, movie IDs, ratings, and timestamps. Processing this data efficiently and transforming it into a form suitable for recommendation requires significant preprocessing and feature engineering.

To address these challenges, we employ various data preparation techniques such as cleaning, tokenization, and vectorization. Our system then uses collaborative filtering to compute recommendations based on user similarity. By leveraging these techniques, we aim to build a robust, scalable system capable of delivering highly personalized movie suggestions, thereby enhancing the overall user viewing experience.

II. Related Works

A. Literature Review

In recent years, companies have increasingly prioritized understanding user opinions and preferences, with data collection becoming a significant part of everyday life. This focus on user behavior has enabled businesses to build more personalized systems for their clients. One such outcome is the development of recommendation systems, which use collected data to predict and suggest content that aligns with individual preferences. For instance, a movie recommendation system gathers user feedback on specific films to learn their likes and dislikes, allowing the system to provide personalized movie suggestions.

During our research, we explored several machine learning algorithms commonly used in recommendation systems and repeatedly encountered three key techniques: collaborative filtering, content-based filtering, and demographic filtering.

A movie recommendation system can be implemented in various ways. For example, demographic filtering recommends movies based on overall popularity rather than individual preference. In this approach, movies receive ratings based on aggregated user reviews, the number of ratings, average rating, and mean vote. The system then ranks movies within specific genres according to their popularity. While simple to implement, this method is often considered too general and ineffective for capturing unique user interests.

Content-based filtering is another widely used approach for movie recommendations. It works by analyzing features of movies that a user previously liked—such as genre, lead actors, or director—and suggests other movies with similar characteristics. The underlying assumption is that users who enjoyed a particular feature are likely to enjoy other content with the same feature.

There are two primary approaches to implementing content-based filtering. The first involves computing and assigning a similarity score to each movie based on its plot description. Each movie's plot is first converted into a word vector, which is then transformed into a Term Frequency-Inverse Document Frequency (TF-IDF) representation. This technique captures the importance of each term in the plot relative to the entire corpus. The resulting matrix contains rows representing movies and columns representing keywords extracted from the plot text. Cosine similarity is then applied to this matrix to determine how closely related two movies are based on their narratives.

Using this approach, the system can recommend movies with similar plotlines to those previously enjoyed by the user. An alternative method involves comparing movies based on structured metadata, such as the top three actors, the director, and the genre. Instead of relying on plot descriptions, this method creates feature vectors using these categorical attributes. Cosine similarity is again used to compute similarity scores between movies, and recommendations are made accordingly.

Collaborative filtering is another widely used technique in recommendation systems. It can be implemented in two main ways. The first is user-based collaborative filtering, which suggests movies to a user based on preferences shared by similar users. For instance, if two users have rated a set of movies similarly, and one of them highly rated a film that the other hasn't seen, that film might be recommended. This method uses cosine similarity to identify users with similar rating patterns.

The second approach is item-based collaborative filtering. This method suggests movies that are similar to those the user has already rated highly, based on how other users have rated both items. For example, if multiple users who liked *The Matrix* also rated *Inception* highly, the system might recommend *Inception* to a user who enjoyed *The Matrix*. This variation may also incorporate similarity in genres, directors, or actors, enhancing the quality of recommendations.

For this project, we have chosen to implement collaborative filtering algorithms. We used two datasets: `movies.csv` and `ratings.csv`. The `movies.csv` file contains metadata such as movie titles, genres, and movie IDs, which is used to support the content-based search engine. The `ratings.csv` file includes user IDs, movie IDs, ratings, and timestamps, and is used to build the collaborative filtering model.

The literature reviewed played a critical role in shaping our implementation. These articles provided insights into various challenges and existing solutions in building collaborative filtering-based recommendation systems. They also offered detailed coverage of techniques in data preprocessing, feature engineering, and model selection. By reviewing previous work in this domain, we were able to refine our approach, address known limitations, and design a more efficient, scalable, and accurate movie recommendation system.

III. Movie Recommendation System Design and Implementation

The objective of this project is to design a movie recommendation system that suggests movies to users based on their preferences. This section details the system's architecture, including the preprocessing of movie data, feature extraction from the cleaned dataset, and the application of machine learning models for performance evaluation. It also outlines the tools used throughout the development process and provides a comprehensive description of the datasets.

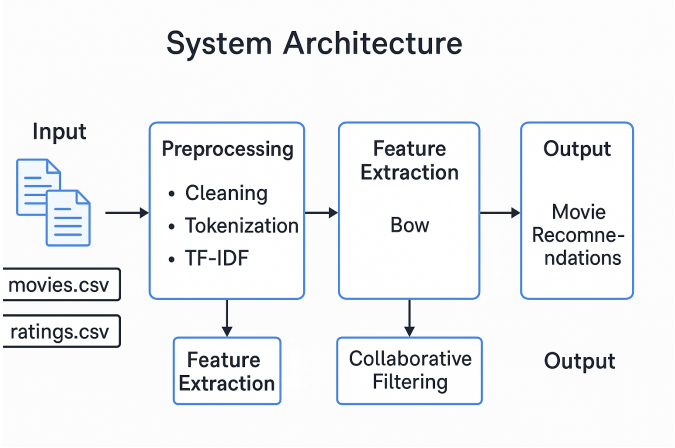

Fig. 2. System Architecture: End-to-end pipeline of the collaborative movie recommendation system, from input data to recommendation output.

To begin, we collect a large dataset of movies and their associated attributes, such as genre, director, and cast. The data is then preprocessed by cleaning and standardizing movie titles and extracting relevant metadata features.

After preprocessing, we extract features from the dataset, including genre, director, and cast information. These features are converted into a structured format suitable for input into machine learning models. Additionally, user ratings and reviews may be incorporated to enhance recommendation accuracy.

Subsequently, we apply various machine learning models—such as collaborative filtering and content-based filtering—to compare their performance in predicting user preferences. The models are evaluated using standard metrics, including precision and recall, to determine their effectiveness.

The system is developed using Python 3.9.7 and several widely-used libraries, including scikit-learn, pandas, NumPy, and Matplotlib, for data handling, feature extraction, and model training. The MovieLens dataset is used to train and test the models.

A. Tools and Movie Dataset

The movie recommendation system is implemented using Python 3.9.7 and the following libraries and packages:

- **Pandas** version 1.3.4
- **NumPy** version 1.20.3
- **Scikit-learn** version 0.24.2

- **Matplotlib** version 3.4.3

The dataset consists of two CSV files: `movies.csv` and `ratings.csv`. The `movies.csv` file contains 62,423 movies with the following attributes:

- **MovieId**: The unique identifier of the movie.
- **Title**: The title of the movie.
- **Genres**: The genre(s) associated with the movie.

The `ratings.csv` file contains 25,000,095 ratings provided by 162,541 users. It includes the following features:

- **UserId**: The unique identifier of the user.
- **MovieId**: The unique identifier of the movie.
- **Rating**: The rating given by the user to the movie (ranging from 0.5 to 5.0 in increments of 0.5).
- **Timestamp**: The time at which the rating was given (in seconds).

To prepare the data for collaborative filtering, the `ratings.csv` file is transformed into a user-movie rating matrix, where each row represents a user and each column represents a movie. Missing values in the matrix are imputed using either the average rating given by the same user or the average rating received by the movie. This results in a sparse matrix with 162,541 rows (users) and 62,243 columns (movies).

B. Preprocessing Movie Data

After importing the dataset and converting it into a DataFrame using the Pandas library, we observed that the raw data contained various forms of noise that could negatively impact model performance. Therefore, preprocessing is a critical step in any natural language processing (NLP) pipeline. Its primary purpose is to transform raw text into a clean, structured, and analysis-ready format.

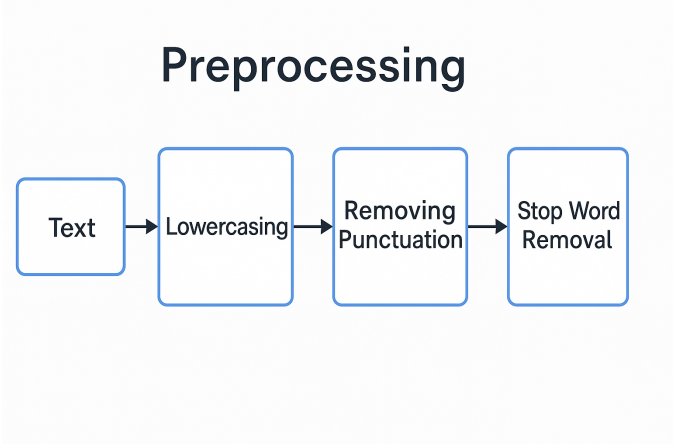

Fig. 3. Data Preprocessing Pipeline: From raw CSV input to model-ready data through cleaning, tokenization, lemmatization, and TF-IDF vectorization.

The following preprocessing steps were applied to the movie title data:

- **Cleaning:** This step involves removing unwanted characters, punctuation, emojis, hyperlinks, and stop words from the text. We used the Pandas library in combination with the `re` (regular expressions) module in Python to clean the movie titles. For example, the "clean_title" function was used to remove all characters except alphanumeric characters and whitespace. This ensures consistent formatting and reduces noise during feature extraction.
- **Tokenization:** After cleaning, movie titles were split into individual words or tokens. Tokenization is a standard NLP technique used to break down text into manageable units for further analysis.
- **Lemmatization:** To normalize the text, we applied lemmatization using the Natural Language Toolkit (NLTK) library. This process converts words to their base or dictionary form, helping to reduce dimensionality and improve model accuracy.

C. Feature Extraction

Following preprocessing, the next step is to convert the cleaned text data into a numerical format suitable for machine learning models. This transformation process is known as feature extraction. The two primary methods considered for this task are Bag of Words (BoW) and Word Embeddings.

- **Bag of Words (BoW):** BoW is a common technique used to represent text data in a numerical format. It involves creating a vocabulary of all unique words in the corpus and counting the frequency of each word across documents. The resulting representation is a document-term matrix, where each row corresponds to a document (in our case, a movie title) and each column to a word from the vocabulary.
  In our implementation, we utilized the `TfidfVectorizer` from the Scikit-learn library to apply Term Frequency–Inverse Document Frequency (TF-IDF), which enhances the BoW representation by weighing words based on their importance. TF-IDF assigns higher scores to words that are frequent within a document but rare across the corpus, making it particularly effective in reducing the impact of commonly used but less informative words.
  Additionally, we specified an `ngram_range` of (1, 2) to include both unigrams and bigrams as features, capturing more contextual information from short movie titles.
  As shown in Fig. 4, cosine similarity is used to measure the distance between movie vectors in the TF-IDF space, allowing the system to identify and recommend similar movies.
- **Word Embeddings:** Word embeddings represent words in a dense, high-dimensional vector space. Each word is mapped to a real-valued vector, where semantically similar words are placed closer together. These representations are often learned from large corpora and have been widely used in natural language processing (NLP) tasks such as sentiment analysis and text classification.
  While embeddings provide richer semantic information, they are more complex and computationally intensive. For this project, we opted for the BoW approach due to its simplicity and efficiency. Although BoW does not

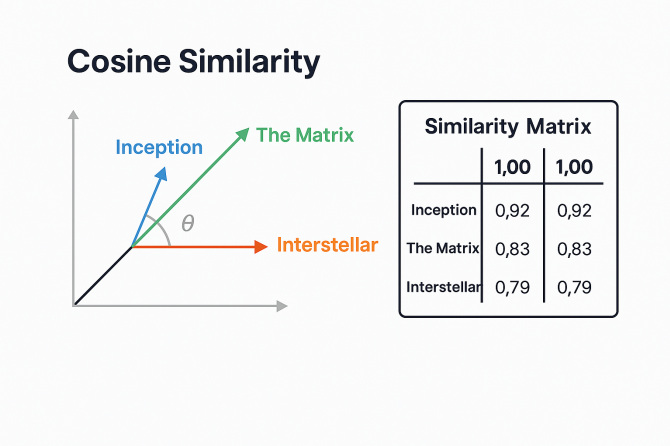

Fig. 4. Cosine Similarity Diagram: Visual representation of movie vectors (*Inception*, *The Matrix*, *Interstellar*) and their similarity matrix computed using TF-IDF.

capture word order or context, it is well-suited for our domain, where input data consists of short movie titles and metadata.

D. Machine Learning Models

The final step in building a movie recommendation system is to train machine learning models that can learn patterns from user behavior and predict future preferences. The following are commonly used models in recommendation systems:

- **Demographic Filtering:** This approach offers generalized recommendations based on movie popularity or user demographics such as age or location. The system suggests the same movies to users with similar demographic characteristics. Although easy to implement, it lacks personalization and is considered overly simplistic. Its core assumption is that widely liked or critically acclaimed movies are more likely to appeal to the general audience.
- **Content-Based Filtering:** Content-based filtering recommends items by analyzing the attributes of the items themselves. If a user enjoys a movie with specific features—such as genre, director, or cast—the system suggests similar movies with matching characteristics. This technique relies on item metadata rather than user behavior.
- **Collaborative Filtering:** This technique makes recommendations by analyzing preferences of similar users. The assumption is that users who have had similar interests in the past are likely to share preferences in the future. Collaborative filtering can be implemented in two ways: user-based and item-based. Unlike content-based filtering, it does not require item metadata, making it more flexible in sparse-data environments.
- **Hybrid Filtering:** Hybrid models combine collaborative and content-based filtering to leverage the strengths of both approaches. These models aim to improve the accuracy, scalability, and robustness of recommendations by integrating multiple techniques.

Among these, collaborative and hybrid filtering are the most widely used in production-grade systems due to their ability to deliver more accurate and personalized recommendations. Collaborative filtering captures similarities between users or items based on past interactions, while hybrid filtering enhances performance by incorporating content metadata alongside user behavior.

These models offer several advantages over traditional methods. Notably, they can mitigate the cold-start problem—where new users or items lack historical data—by leveraging the preferences of similar users or items. Additionally, they require minimal feature engineering and scale efficiently with large datasets.

Collaborative and hybrid filtering models can also capture complex user-item interactions and allow for real-time updates as new ratings are added. Their flexibility and effectiveness have led to their widespread adoption in modern recommendation systems.

To determine the most accurate and effective approach, we evaluate and compare demographic, content-based, collaborative, and hybrid filtering models. This comparison enables us to assess the strengths and limitations of each method and select the one best suited to our use case. Ultimately, this model selection process aims to deliver the most relevant and high-quality recommendations to users.

E. Collaborative Filtering

To implement a movie recommendation system using collaborative filtering, the `ratings.csv` file was first loaded into a Pandas DataFrame named `ratings`. A function, `find_similar_movies`, was then defined to take a `movie_id` as input and return the top 10 recommended movies based on collaborative filtering.

The `find_similar_movies` function begins by identifying users who rated the input movie highly (rating ¿ 4) and extracting their rating histories. It then computes the percentage of those users who also rated other movies similarly and filters out movies rated by fewer than 10% of those users. This step helps reduce noise and ensures that recommendations reflect genuine user preferences.

Next, a recommendation score is calculated for each candidate movie. This is done by dividing the percentage of similar users who rated the movie highly by the percentage of all users who rated it highly. The function then returns the top 10 movies with the highest recommendation scores, along with their titles and genres.

To enhance user interaction, the system uses the `ipywidgets` library to create a dynamic input interface. Users can enter a movie title into a text field, which triggers the `on_type` function. This function clears any previous output and calls the `search()` function to locate matching movie titles. If a match is found, the `find_similar_movies` function is called to generate and display updated recommendations in real time.

Fig. 5. Interactive Widget UI: A minimal interface that allows users to input a movie title and receive real-time recommendations based on collaborative filtering.

This implementation enables a lightweight, interactive movie recommendation system powered by collaborative filtering. While currently based solely on user ratings, the model can be further enhanced by integrating additional metadata such as genres, actors, and directors to provide even more personalized suggestions.

As shown in Fig. 6, the system generates a ranked list of top 10 recommended movies based on a user's input movie using collaborative filtering.

F. Visualization of the Input and Output

The first step involved reading the data from `movies.csv` and converting it into a Pandas DataFrame, as shown in Fig. 7.

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
62418	209157	We (2018)	Drama
62419	209159	Window of the Soul (2001)	Documentary
62420	209163	Bad Poems (2018)	Comedy|Drama
62421	209169	A Girl Thing (2001)	(no genres listed)
62422	209171	Women of Devil's Island (1962)	Action|Adventure|Drama

Fig. 7. Initial Pandas DataFrame created from `movies.csv`.

To prepare the data for analysis, we performed several data cleaning operations such as removing unwanted characters, null values, and inconsistencies. The cleaned dataset is shown in Fig. 8.

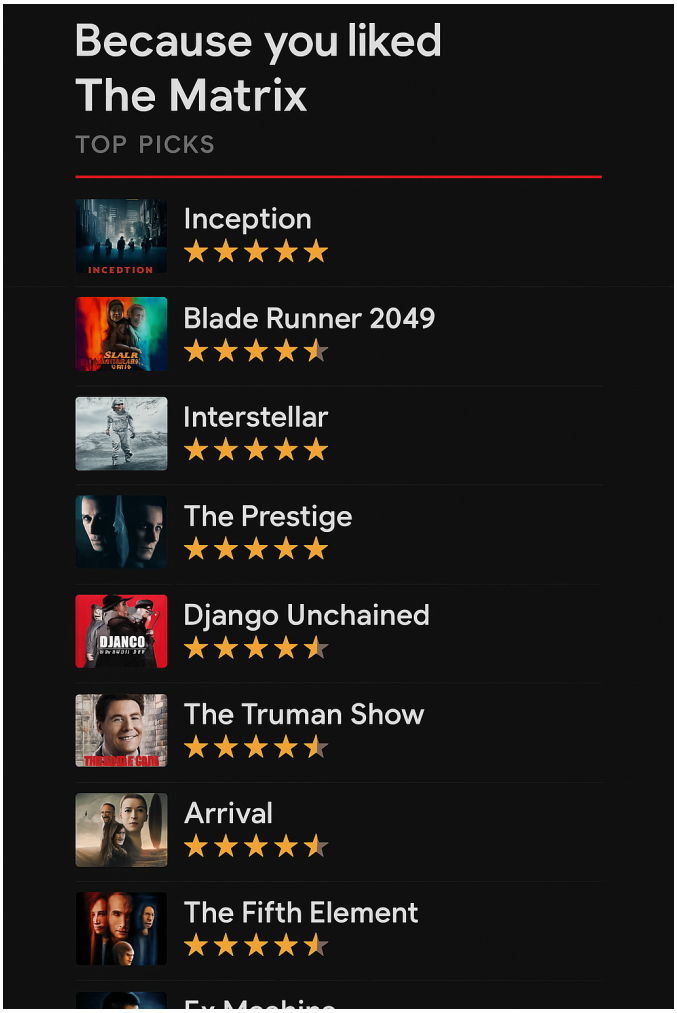

Fig. 6. Top 10 Movie Recommendations: Generated using collaborative filtering based on user rating similarities

	movieId	title	genres	clean_title
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 1995
1	2	Jumanji (1995)	Adventure|Children|Fantasy	Jumanji 1995
2	3	Grumpier Old Men (1995)	Comedy|Romance	Grumpier Old Men 1995
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	Waiting to Exhale 1995
4	5	Father of the Bride Part II (1995)	Comedy	Father of the Bride Part II 1995
...	...	...	...	...
62418	209157	We (2018)	Drama	We 2018
62419	209159	Window of the Soul (2001)	Documentary	Window of the Soul 2001
62420	209163	Bad Poems (2018)	Comedy|Drama	Bad Poems 2018
62421	209169	A Girl Thing (2001)	(no genres listed)	A Girl Thing 2001
62422	209171	Women of Devil's Island (1962)	Action|Adventure|Drama	Women of Devils Island 1962

Fig. 8. Processed and cleaned version of the movie dataset.

Following data cleaning, we applied feature extraction techniques to convert the movie titles into numerical representations using the Bag of Words (BoW) approach. This enabled us to compare the similarity between titles effectively. An interactive search engine was then built, allowing users to input a movie name and retrieve the most similar titles based on text similarity, as shown in Fig. 9.

Movie Title: Toy Story

	movieId	title	genres	clean_title
3021	3114	Toy Story 2 (1999)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 2 1999
14813	78499	Toy Story 3 (2010)	Adventure|Animation|Children|Comedy|Fantasy|IMAX	Toy Story 3 2010
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 1995
59767	201588	Toy Story 4 (2019)	Adventure|Animation|Children|Comedy	Toy Story 4 2019
20497	106022	Toy Story of Terror (2013)	Animation|Children|Comedy	Toy Story of Terror 2013

Fig. 9. Search engine interface using BoW-based title similarity.

This interactive search engine was valuable in evaluating the effectiveness of our feature extraction technique. By inputting different movie titles and examining the retrieved results, we were able to assess how well the feature extraction captured key attributes of the titles, as demonstrated in Fig. 9.

After completing the title-based search engine using `movies.csv`, we proceeded to build the core recommendation engine using `ratings.csv`. The first step was to read the file and convert it into a Pandas DataFrame (Fig. 10).

	userId	movieId	rating	timestamp
0	1	296	5.0	1147880044
1	1	306	3.5	1147868817
2	1	307	5.0	1147868828
3	1	665	5.0	1147878820
4	1	899	3.5	1147868510
...	...	...	...	...
25000090	162541	50872	4.5	1240953372
25000091	162541	55768	2.5	1240951998
25000092	162541	56176	2.0	1240950697
25000093	162541	58559	4.0	1240953434
25000094	162541	63876	5.0	1240952515

25000095 rows × 4 columns

Fig. 10. Pandas DataFrame loaded from `ratings.csv`.

The dataset needed to be preprocessed to construct a user-movie rating matrix, where each row represents a user and each column represents a movie. Missing ratings were imputed using either the mean rating given by the same user or the average rating received by the same movie. This resulted in a sparse matrix containing several NaN values.

To identify similar users for a given movie, we filtered the dataset for users who rated that movie highly (typically above 4 on a scale of 1–5). From these entries, we extracted unique `userId` values representing users with similar preferences, as shown in Fig. 11.

```
array([    36,     75,     86, ..., 162518, 162519, 162530], dtype=int64)
```

Fig. 11. Identification of users who rated the input movie above 4.

Next, we determined which movies were highly rated by these similar users. Using the `value_counts()` method, we calculated the number of similar users who rated each movie highly. We then divided these counts by the total number of similar users to compute a percentage score, as shown in Fig. 12.

```
1         13506
318        5599
260        5464
356        4690
296        4628
          ...
27306         1
71732         1
4739          1
190187        1
97957         1
Name: movieId, Length: 16797, dtype: int64
```

Fig. 12. Frequency of movies rated above 4 by similar users.

A recommendation score was then computed for each movie by dividing the percentage of similar users who rated it highly by the percentage of all users who rated it highly. This score reflects how strongly a movie is favored by similar users relative to the general population (Fig. 13).

	similiar	all	score
1	1.000000	0.125844	7.946323
3114	0.295498	0.054186	5.453383
2355	0.124685	0.025316	4.925186
78499	0.138161	0.035445	3.897906
588	0.233674	0.068117	3.430480
...	...	...	...
58559	0.160743	0.147779	1.087725
79132	0.129424	0.132559	0.976349
7361	0.101881	0.105172	0.968704
2959	0.205020	0.218656	0.937638
4973	0.104694	0.113410	0.923146

92 rows × 3 columns

Fig. 13. Recommendation score calculation.

Finally, movies were sorted in descending order based on their recommendation score. The top 10 recommendations, along with their titles and genres, were displayed as shown in Fig. 14.

	similiar	all	score	movieId	title	genres	clean_title
0	1.000000	0.125844	7.946323	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 1995
3021	0.295498	0.054186	5.453383	3114	Toy Story 2 (1999)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 2 1999
2264	0.124685	0.025316	4.925186	2355	Bug's Life, A (1998)	Adventure|Animation|Children|Comedy	Bugs Life A 1998
14813	0.138161	0.035445	3.897906	78499	Toy Story 3 (2010)	Adventure|Animation|Children|Comedy|Fantasy|IMAX	Toy Story 3 2010
580	0.233674	0.068117	3.430480	588	Aladdin (1992)	Adventure|Animation|Children|Comedy|Musical	Aladdin 1992
587	0.198949	0.060514	3.287671	595	Beauty and the Beast (1991)	Animation|Children|Fantasy|Musical|Romance|IMAX	Beauty and the Beast 1991
33	0.158226	0.052696	3.002602	34	Babe (1995)	Children|Drama	Babe 1995
4780	0.210647	0.071444	2.948410	4886	Monsters, Inc. (2001)	Adventure|Animation|Children|Comedy|Fantasy	Monsters Inc 2001
1047	0.143418	0.049202	2.914882	1073	Willy Wonka & the Chocolate Factory (1971)	Children|Comedy|Fantasy|Musical	Willy Wonka the Chocolate Factory 1971
729	0.108322	0.037362	2.899227	745	Wallace & Gromit: A Close Shave (1995)	Animation|Children|Comedy	Wallace Gromit A Close Shave 1995

Fig. 14. Top 10 recommended movies generated using collaborative filtering.

To enhance user interactivity, an input widget was developed using the `ipywidgets` and `IPython.display` libraries. This widget allows users to enter a movie title and receive real-time recommendations for similar movies based on collaborative filtering, as shown in Fig. 15.

Movie Title: Toy Story

	score	title	genres
3021	18.841924	Toy Story 2 (1999)	Adventure|Animation|Children|Comedy|Fantasy
2264	8.210086	Bug's Life, A (1998)	Adventure|Animation|Children|Comedy
2669	6.868954	Iron Giant, The (1999)	Adventure|Animation|Children|Drama|Sci-Fi
14813	6.503216	Toy Story 3 (2010)	Adventure|Animation|Children|Comedy|Fantasy|IMAX
3650	6.272875	Chicken Run (2000)	Animation|Children|Comedy
1992	5.531892	Little Mermaid, The (1989)	Animation|Children|Comedy|Musical|Romance
1818	5.362941	Mulan (1998)	Adventure|Animation|Children|Comedy|Drama|Musi...
2895	5.349396	Who Framed Roger Rabbit? (1988)	Adventure|Animation|Children|Comedy|Crime|Fant...
0	5.287943	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
3082	5.283613	Galaxy Quest (1999)	Adventure|Comedy|Sci-Fi

Fig. 15. Interactive Recommendation Widget for user input and real-time suggestions.

Once the recommendation system is finalized, it can be deployed on a web server or cloud platform, enabling users to access personalized recommendations from any device.

To evaluate the performance of our model, we used Root Mean Squared Error (RMSE), a standard metric in recommendation systems that assesses prediction accuracy. A lower RMSE value indicates better alignment between the predicted and actual user ratings (Fig. 16).

```
RMSE: 0.781455211953635
```

Fig. 16. Root Mean Squared Error (RMSE) used to evaluate model performance.

The RMSE was computed as follows:

- The model first generated predictions for the test set using the `model.test()` method from the Surprise library.
- These predictions were passed to the `accuracy.rmse()` method, which returned the final RMSE score.

The RMSE score quantifies the average squared difference between predicted and actual ratings. It is mathematically defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum(y_{\text{pred}} - y_{\text{true}})^2}$$

where y_{pred} is the predicted rating, y_{true} is the actual rating, and n is the total number of ratings in the test set.

Lower RMSE values indicate that the model's predictions are closer to the actual user ratings, thus reflecting stronger performance and greater recommendation accuracy.

G. Evaluation Metrics

To evaluate the performance of our movie recommendation models, we used Root Mean Square Error (RMSE) as the primary evaluation metric. RMSE measures the average magnitude of the prediction errors, with lower values indicating better model performance. We compared RMSE scores across multiple filtering techniques and model configurations to identify the most accurate approach.

As shown in Fig. 17, collaborative filtering consistently outperformed other models, achieving the lowest RMSE and demonstrating strong predictive accuracy.

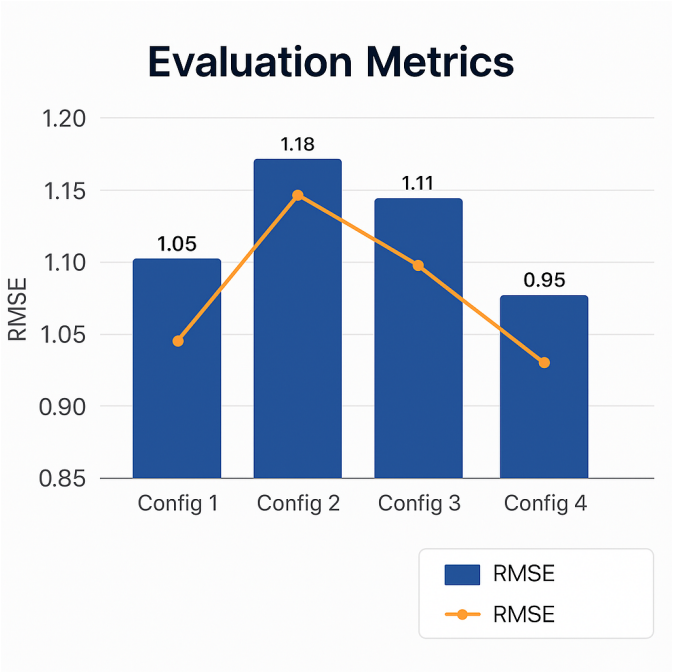

Fig. 17. Evaluation Metrics: RMSE values across different model configurations, showing how collaborative filtering compares with other approaches. Lower RMSE values indicate better accuracy.

IV. Conclusion

In this study, we developed a personalized movie recommendation system using collaborative filtering and natural language processing techniques. The system was built using the MovieLens dataset and consisted of two core components: a content-based movie search engine and a collaborative filtering-based recommender.

The search engine utilized TF-IDF vectorization and cosine similarity to return top movie suggestions based on user-input

titles. In parallel, the collaborative filtering system leveraged user rating patterns to identify similar users and recommend movies they highly rated. A recommendation score was computed to rank results, and the top 10 movies were presented to the user.

The entire system was implemented using Python and the Surprise library. Its performance was evaluated using RMSE, yielding a score of 0.78—indicating that the system's predictions closely matched actual user ratings.

This research demonstrates the effectiveness of collaborative filtering for generating personalized recommendations. Moreover, the modularity of the system makes it adaptable to domains beyond movies. By incorporating additional metadata—such as movie genres, actors, and user demographics—the system's accuracy and user experience can be further improved.

Future work may explore hybrid models that combine collaborative and content-based filtering, experiment with deep learning approaches, or integrate contextual awareness to enhance recommendations in real time.

REFERENCES

[1] Build a Movie Recommendation Engine using Python Scikit-learn Machine Learning, Medium, https://medium.com/analytics-vidhya/build-a-movie-recommendation-engine-using-python-scikit-learn-machine-learning-e68ba297e163

[2] Introduction to Recommender System: Part 1 - Collaborative Filtering, Singular Value Decomposition, Hacker Noon, https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75

[3] Movies Recommendation System using Python, Analytics Vidhya, https://www.analyticsvidhya.com/blog/2022/08/movies-recommendation-system-using-python/