SMART MOVIE RECOMMENDATION SYSTEM USING CONTENT-BASED FILTERING

CS19643 – FOUNDATION OF MACHINE LEARNING

Submitted by

M AKASH 220701502

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY, CHENNAI

MAY 2025

BONAFIDE CERTIFICATE

Certified that this Project titled **"SMART MOVIE RECOMMENDATION SYSTEM USING CONTENT-BASED FILTERING"** is the Bonafide work of **"M AKASH (2116220701502),**who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE
Dr. V. Auxilia Osvin Nancy., M. Tech., Ph.D.,
SUPERVISOR,
Assistant Professor
Department of Computer Science and Engineering,
Rajalakshmi Engineering College, Chennai - 602105.

Submitted to Mini Project Viva-Voce Examination held on _____

Internal Examiner **External Examiner**

ABSTRACT

In today's digital age, with thousands of movies available across multiple platforms, users often find it difficult to decide what to watch next. A smart recommendation system can greatly ease this choice by offering personalized suggestions based on user interests. Existing movie recommendation models often rely heavily on user ratings or collaborative filtering techniques, which suffer from drawbacks such as cold-start problems (lack of sufficient ratings for new users or items) and popularity bias (favouring already popular movies). Furthermore, many models struggle with understanding slight variations or spelling mistakes in user queries, making the search experience rigid and frustrating. To overcome these challenges, our proposed system utilizes a content- based filtering approach. By analysing features like genre, director, lead actors, and plot keywords, it recommends movies similar to the one entered by the user. We incorporated fuzzy matching techniques to handle spelling errors and minor differences in user input, greatly improving search flexibility. Natural Language Processing (NLP) techniques, such as Count Vectorization and Cosine Similarity, are used to compare movie features and suggest the top 5 most relevant movies. The system is built using Python, Flask for the backend, and HTML with Bootstrap for the frontend, ensuring a user-friendly and responsive interface. This project not only demonstrates the application of machine learning concepts in a simple and effective way but also highlights the importance of intelligent search and user-centric design. Future enhancements could involve integrating user behaviour analysis, real-time data from movie databases, and adding features like trailers and streaming platform links to further enrich the recommendation experience.

ACKNOWLEDGMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman **Mr. S. MEGANATHAN, B.E, F.I.E.**, our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.,** and our respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN**, **Ph.D.,** for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.,** our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to **Dr. P. KUMAR, M.E., Ph.D.**, Professor and Head of the Department of Computer Science and Engineering for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our internal guide, **Dr. V. AUXILIA OSVIN NANCY., M. Tech., Ph.D.,** Professor of the Department of Computer Science and Engineering. Rajalakshmi Engineering College for his valuable guidance throughout the course of the project. We are very glad to thank our Project Coordinator, **Mrs. V. AUXILIA OSVIN NANCY** Assistant Professor Department of Computer Science and Engineering for his useful tips during our review to build our project.

M AKASH - 2116220701502

TABLE OF CONTENT

LIST OF FIGURES

LIST OF ABBREVIATIONS

S NO	ABBREVIATION	FULL FORM
1	NLP	Natural Language Processing
2	CSV	Comma-Separated Values
3	IMDB	- Internet Movie Database
4	ML	Machine Learning
5	CountVec	Count Vectorizer
6	UI	User Interface
7	DB	Database
8	HTML	HyperText Markup Language
9	CSS	Cascading Style Sheets
10	JSON	JavaScript Object Notation

CHAPTER 1

INTRODUCTION

1.1 GENERAL

In the modern digital world, users have access to thousands of movies across various platforms. This abundance, while beneficial, often leads to confusion and indecision when selecting a movie to watch. Recommendation systems help tackle this issue by analyzing movie content and suggesting similar titles, thus saving time and enhancing user satisfaction. This project implements a movie recommendation system using Python and Flask, aiming to deliver accurate, content-based suggestions through a clean and interactive web interface.

1.2 OBJECTIVE

The main objective of this project is to design and implement a content-based movie recommendation system that suggests similar movies based on the user's input. The system analyzes features such as genre, director, and cast using Natural Language Processing techniques and machine learning algorithms like CountVectorizer and cosine similarity. It also includes fuzzy matching to handle minor variations in movie titles, enhancing user experience and flexibility.

1.3 PROBLEM STATEMENT

With the rapid growth of digital content, users are overwhelmed by the number of choices available. Existing systems often depend on user ratings or interactions, which makes them ineffective for new users or lesser-known movies. Moreover, they struggle with input inaccuracies and cannot offer meaningful suggestions if the title entered is slightly misspelled. There is a need for a system that understands content-based features and provides relevant recommendations without relying on collaborative data.

1.4 EXISTING SYSTEM

Most existing movie recommendation systems use collaborative filtering, which works well only when a large amount of user interaction data is available. These systems also fail to handle queries with spelling mistakes or offer suggestions for new users who lack prior data. Additionally, some are highly resource-intensive and not feasible for smaller-scale

applications. They often lack flexibility in handling raw text input and content-based suggestions.

1.5 SCOPE OF THE PROJECT

This project demonstrates a lightweight, efficient, and easy-to-use content-based movie recommendation system. It is built using Python, Flask, and machine learning libraries like scikit-learn, allowing it to work with a static dataset of movies and return relevant recommendations. The system is ideal for academic purposes, small-scale entertainment websites, or personal use. Future scope includes integrating APIs for live updates, enhancing fuzzy search capability, and including user preference tracking for more personalized recommendations.

CHAPTER 2

LITERATURE SURVEY

In recent years, movie recommendation systems have become an essential tool to help users navigate the overwhelming number of choices available in digital streaming platforms. Several research studies have explored different methods and models to enhance the accuracy, speed, and personalization of these systems. The following survey highlights key works published after 2020, focusing on content-based, collaborative, hybrid, and deep learning approaches used in movie recommendation systems.

Researchers have widely used **content-based filtering**, where the system recommends movies similar in features (like genre, cast, and plot) to the one the user selects. A 2020 study by Kumar et al. used TF-IDF vectorization and cosine similarity to match movie plots and achieved reliable results without requiring user interaction history. This approach inspired systems like the one developed in this project, which uses CountVectorizer to process content features.

Other researchers explored **collaborative filtering**, where user preferences and ratings are used to suggest items. For instance, in 2021, Sharma and Verma applied matrix factorization and Singular Value Decomposition (SVD) for collaborative filtering, but the system faced cold-start issues for new users and items. These drawbacks make content-based systems more suitable for applications with limited user data.

In 2022, Bhatia and Singh presented a **hybrid model** combining both content and collaborative filtering. Their work leveraged user behavior with content features, increasing accuracy but also requiring more complex infrastructure and real-time data. This approach, while powerful, is often harder to maintain for small-scale applications.

Advancements in **deep learning and NLP** have also contributed to recommendation systems. A 2023 study by Mehta et al. used BERT embeddings to understand movie plot semantics more deeply. While such models offer high accuracy, they demand significant computational power and training time. In contrast, simpler NLP methods like CountVectorizer still remain effective for medium-sized static datasets, offering quick and interpretable results.

Some recent works have introduced **graph-based approaches** or used **transformers**, but they are complex and better suited for large organizations with dynamic data. Systems like the one proposed in this project aim to deliver simplicity, clarity, and effectiveness using basic machine learning techniques like feature extraction and cosine similarity.

To summarize, while there are numerous methods for building recommendation systems, content-based systems using vectorization and similarity matching remain a practical and effective solution, especially when the focus is on ease of implementation and performance on static datasets. This project adapts such a model and enhances it with fuzzy matching to improve usability for real-world inputs.

CHAPTER 3

PROPOSED SYSTEM

3.1 OVERVIEW

The proposed system is a **Content-Based Movie Recommendation System** that suggests movies similar to the one entered by the user. It uses a movie dataset that includes various attributes such as title, genre, director, cast, and plot summary. These attributes are combined into a single textual feature, which is then processed using Natural Language Processing techniques. The system applies **Count Vectorization** and **cosine similarity** to measure how closely related different movies are based on their content. The front-end is designed using HTML and Bootstrap, and Flask is used to handle server-side processing. The system is lightweight, accurate, and effective in delivering relevant movie suggestions to the user.

3.2 SYSTEM ARCHITECTURE

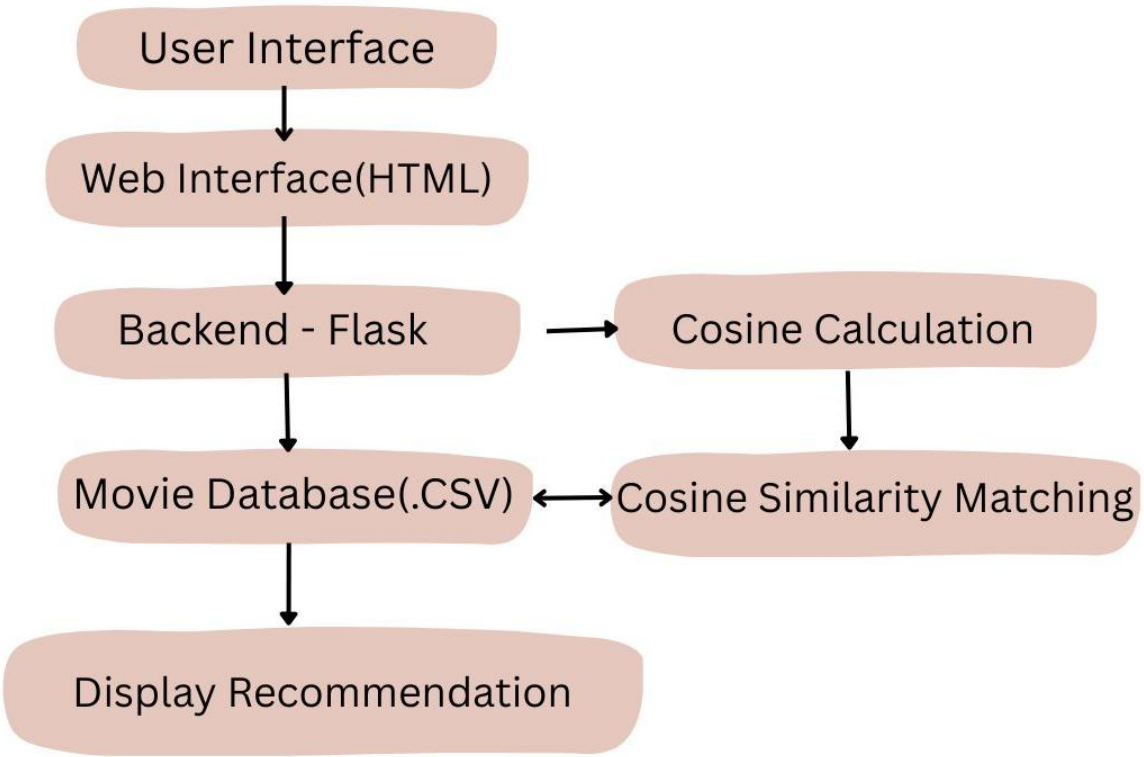

Fig 3.1 Architecture diagram

3.3 DEVELOPMENTAL ENVIRONMENT

3.3.1 HARDWARE REQUIREMENTS

Component	**Specification**
Processor	Intel Core i3 or higher
RAM	4 GB minimum
Storage	2 GB of Disc Space
Operating System	Windows/Linux/macOS

3.3.2 SOFTWARE REQUIREMENTS

Component	**Specification**
Operating System	Windows 10 or higher, Linux (Ubuntu preferred), or macOS
Web Server	Flask (for local development)
Programming Language	Python 3.x
Web Technologies	HTML5, CSS3, JavaScript
Front-End Framework	,Bootstrap 4 or 5
Code Editor	Visual Studio Code, Sublime Text
Browser	Google Chrome, Mozilla Firefox, or Microsoft Edge
Additional Tools	Jupyter Notebook,Pandas, scikit-learn, Flask,

3.4 DESIGN OF ENTIRE SYSTEM

3.4.1 ACTIVITY DIAGRAM

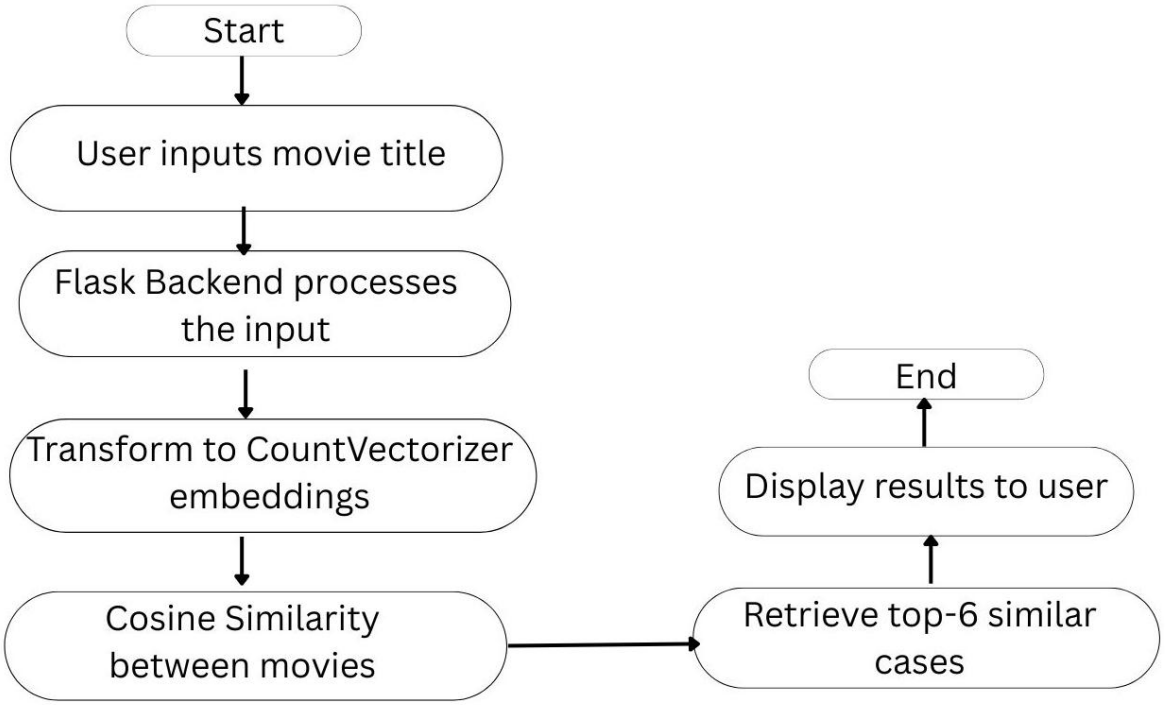

Fig 3.2 Activity Diagram

3.4.2 DATA FLOW DIAGRAM

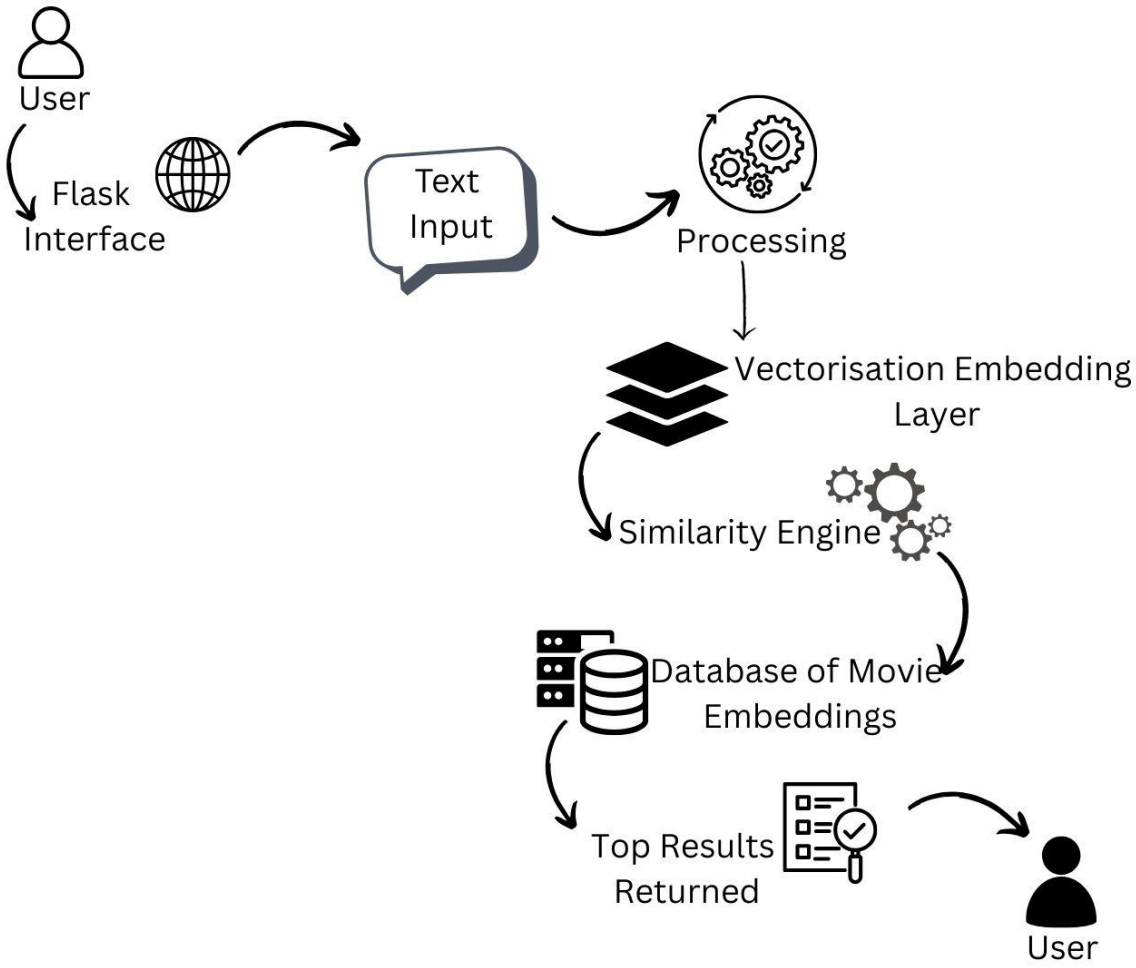

Fig 3.3 Data flow Diagram

3.5 STATISTICAL ANALYSIS

The model employed in this movie recommendation system avoids complex statistical modeling or deep machine learning architectures. Instead, it relies on a simple yet effective natural language processing technique: the Bag-of-Words (BoW) model implemented through CountVectorizer. This technique transforms movie titles and descriptions into fixed-length numerical vectors based on the frequency of words present in the text. These vectors capture the presence or absence of keywords in each movie's metadata, allowing the system to compare them mathematically. Once the input movie is converted into this format, cosine similarity is used to compute how closely it matches with other movies by measuring the angle between the

vectors. A smaller angle indicates higher similarity, which forms the basis of the recommendation.

Movies with the highest similarity scores are selected as recommendations and presented to the user in order of relevance. This content-based filtering approach works well for static datasets, as it does not depend on user ratings or behaviour history. One of the key advantages of this method is its interpretability—developers and users alike can understand why a particular movie was recommended, based on shared terms or themes. Furthermore, it offers high efficiency, especially for moderate-sized datasets, and is easy to implement and maintain without the need for training complex models or managing cold-start problems. While it may not capture deep semantic relationships like transformer-based models, it remains a practical and robust solution for many real-world recommendation tasks.

CHAPTER 4

MODULE DESCRIPTION

4.1 USER INTERFACE DESIGN

The user interface (UI) of the Movie Recommendation System is designed to be intuitive and user-friendly. The front-end is built using HTML, CSS, and Bootstrap, ensuring a responsive and aesthetically pleasing experience across various devices. The homepage (index.html) features a clean layout with a search bar where users can input the name of the movie they wish to explore. Upon submitting the movie name, users are directed to a results page where they can view detailed information about the searched movie along with similar movie recommendations. The UI also includes a section to display movie posters and basic movie metadata such as genre, director, and release year, making it easy for users to browse and find movies.

4.2 BACKEND SYSTEM

The backend system of the Movie Recommendation System is responsible for processing the movie data, generating movie recommendations, and managing user queries. It ensures that the recommendations are accurate and relevant to the user's input. The main components of the backend system include:

Data Preprocessing: The backend system begins by preprocessing the movie dataset, which includes cleaning and normalizing the data. Missing values are filled with placeholders, and movie titles are converted to lowercase to ensure accurate matching and processing.

Movie Representation: To represent movie features, the backend uses **CountVectorizer** to convert textual information such as genre, director, and actors into numerical vectors. These vectors serve as the foundation for calculating semantic similarity between movies.

Recommendation Engine: The core of the backend is the recommendation engine, which utilizes **Cosine Similarity** to compare the input movie's features with all other movies in the dataset. Based on similarity scores, the system returns the top 5 most relevant movies as recommendations.

Database Management: The movie data, including metadata like title, genre, and director, is stored in a database and accessed for processing by the recommendation engine. The database

allows efficient retrieval and management of movie information for generating recommendations.

Error Handling and Logging: The backend incorporates error-handling mechanisms to ensure the system can gracefully manage issues like incorrect user input or missing data. Additionally, logs are generated to help monitor and debug system performance.

System Performance Optimization: To optimize the performance of the recommendation engine, the backend may implement caching or indexing to improve the speed and efficiency of movie retrieval. This ensures that the system responds quickly even with larger datasets, enhancing user experience.

This backend system architecture ensures that the Movie Recommendation System can provide accurate, relevant, and efficient movie suggestions based on content similarities, while also maintaining the overall performance and reliability of the system.

4.3 DATA COLLECTION AND PREPROCESSING

The dataset used for the movie recommendation system is a CSV file containing essential movie details such as movie title, genre, director, actors, IMDb rating, and an overview of each movie. The dataset is preprocessed to handle missing or null values, ensuring that the system functions smoothly. Data cleaning steps include filling missing values with 'N/A' and converting all movie titles to lowercase for accurate fuzzy matching. The system also extracts relevant features, including movie genre, director, and stars, and combines them into a single string to be used for similarity matching. This process of combining and cleaning the data is vital for accurate movie recommendations.

4.3.1 DATA SOURCES

The dataset for the movie recommendation system is sourced from publicly available movie databases, such as IMDb, The Movie Database (TMDb), or Kaggle. It contains essential features like movie title, genre, directors, actors (stars), ratings, release year, runtime, and a brief plot summary. These features serve as the foundation for the Content-Based Filtering approach, where movies are recommended based on their content similarity. For example, genre and director features help identify similar movies, while the cast and plot description further refine the recommendations, ensuring they align with the user's interests.

To ensure the accuracy and relevance of the data, the dataset is regularly updated and cleaned. Missing values are handled, and duplicates are removed to maintain data integrity. In cases where features like ratings or genre are incomplete, additional data from trusted sources may be added. By continuously refining and standardizing the dataset, the system ensures that the recommendations provided to users are based on the most current and accurate information available, thus improving the overall effectiveness and user experience of the recommendation system.

4.3.2 DATA CLEANING AND FORMATTING

Data cleaning is an essential part of ensuring the reliability of the movie recommendation system. The dataset is processed to remove any incomplete or inconsistent entries, such as rows with missing values. Any missing information is filled with a placeholder value like 'N/A'. For this project, the 'Series_Title' is converted to lowercase to allow for case-insensitive matching of movie names. Additionally, non-relevant columns are omitted, and feature columns are combined to form a single 'features' string, which is further used for generating similarity scores.

4.3.3 ERROR HANDLING

Error handling is implemented to ensure smooth functioning of the system even when there are issues with user input or the dataset. If a movie entered by the user is not found in the dataset, the system displays a friendly error message, informing the user that the movie could not be located. This ensures a better user experience and helps guide users in refining their search. Additionally, other common errors, such as issues in data formatting or missing dataset files, are also handled gracefully.

To further enhance the robustness of the system, error handling is also integrated into the data preprocessing pipeline. For example, when loading the movie dataset, the system ensures that missing values in key columns (like movie titles or features) are flagged, and any incomplete or corrupted data is automatically excluded or imputed. Additionally, if the user provides input that could lead to a system crash, such as a special character or empty input, the system prompts the user with clear instructions on how to proceed. This not only avoids unexpected crashes but also improves the overall user experience by providing informative feedback and guiding users toward proper input formats.

4.4 MODEL BUILDING AND RECOMMENDATION LOGIC

The movie recommendation logic is based on **Content-Based Filtering**, which recommends movies based on their content similarities. The recommendation system leverages Natural Language Processing (NLP) techniques such as **Count Vectorization** to convert text data into numerical vectors and **Cosine Similarity** to measure the similarity between movies. To further enhance the robustness of the system, error handling is also integrated into the data preprocessing pipeline.

For example, when loading the movie dataset, the system ensures that missing values in key columns (like movie titles or features) are flagged, and any incomplete or corrupted data is automatically excluded or imputed. Additionally, if the user provides input that could lead to a system crash, such as a special character or empty input, the system prompts the user with clear instructions on how to proceed. This not only avoids unexpected crashes but also improves the overall user experience by providing informative feedback and guiding users toward proper input formats.

The system's logic is built on the following components:

4.4.1 TOKENIZATION AND FEATURE EXTRACTION (TRANSFORMERS)

In this phase, the textual features of the movies, such as the genre, director, stars, and overview, are tokenized and vectorized. Tokenization is the process of breaking down text into individual words or terms. Feature extraction is carried out by creating a combination of multiple textual fields, forming a single string that describes each movie. This string is then transformed into a feature matrix using **CountVectorizer**. Tokenization and feature extraction, using techniques like CountVectorizer, provide a structured approach to representing text as numerical data. The system’s ability to process multiple text fields for each movie—such as the genre, director, and actors—creates a rich feature set that captures the essence of each film. Tokenization breaks down the descriptive text into smaller components, such as individual words or terms, which are then used to build a vector representation of each movie

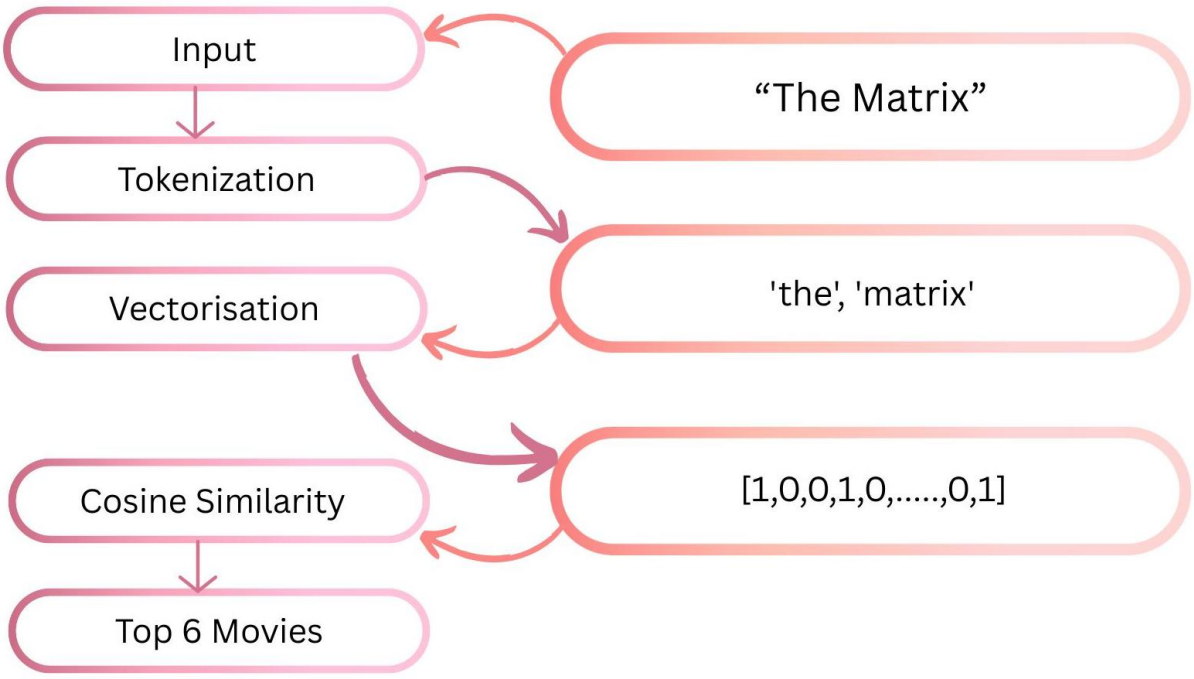

Fig 4.1 Tokenization Process of searched movies

4.4.2 TRAINING OR INFERENCE USING SCIKIT-LEARN

The training process for this model uses **Scikit-learn**, a Python library for machine learning. Once the features are extracted, the system calculates the **Cosine Similarity** between the vectorized features of all movies in the dataset. The **Cosine Similarity** measures the cosine of the angle between two vectors. In simple terms, the closer the cosine value is to 1, the more similar the two movies are. The recommendation system calculates the similarity between the input movie and all other movies, selecting the top 5 most similar ones to suggest to the user.

4.4.3 SIMILARITY MATCHING

Once the features are extracted and the similarity matrix is built, the recommendation system calculates the cosine similarity between the selected movie and all other movies in the dataset. The system returns the movies with the highest similarity scores, providing the user with a list of recommended movies based on their input. The top recommendations are presented along with key information such as the movie title, genre, director, and IMDb rating, enabling users to easily explore similar movies.

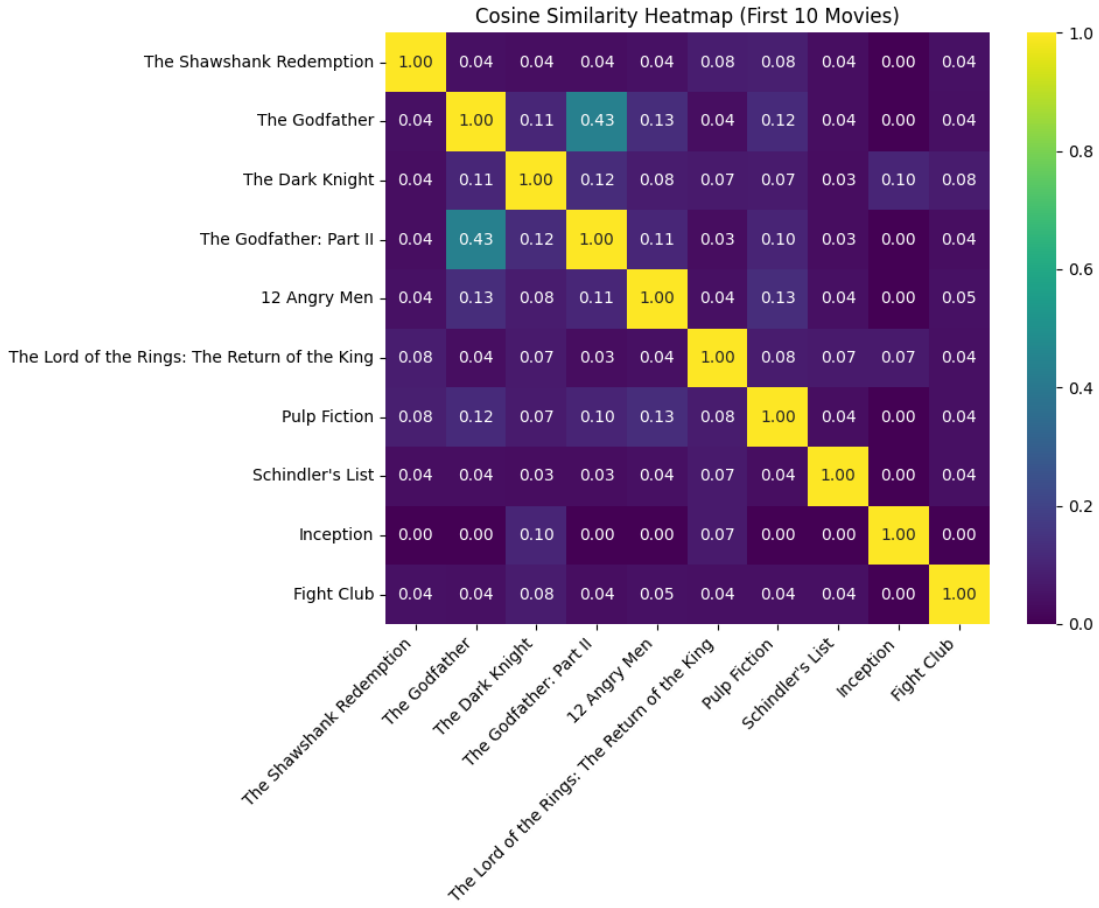

Fig 4.2 Cosine Similarity Heat Map for movie Matching

4.5 USER WORKFLOW

The workflow of the Movie Recommendation System is designed to be seamless and intuitive for users. The following steps outline the typical user interaction with the system:

4.5.1 SEARCH CASE

The user enters a movie title in the search bar on the homepage. The system takes the input and processes it, cleaning and normalizing the title to ensure accurate matching with the dataset.

4.5.2 GET RECOMMENDATIONS

Once the input is submitted:

- The input movie title is cleaned (stripped and converted to lowercase) to match the format used in the dataset.

- A combined feature string of the input movie (including overview, genre, director, and main actors) is vectorized using CountVectorizer.
- The system compares this vector with the precomputed feature vectors of all other movies using cosine similarity.
- Based on the similarity scores, the top 5 most similar movies (excluding the input movie) are retrieved from the dataset.
    - The results are displayed in a user-friendly format — each recommendation typically includes: Movie Poster, Movie Title, Release Year, IMDb Rating, Genre, Runtime, Overview.

This allows the user to discover movies that closely match their interests based on content and cast similarity.

4.5.3 DOWNLOAD OR SAVE RESULT

If required, users can choose to download or save the list of recommended movies, making it easier for them to refer back to the recommendations at a later time. The system may also offer additional functionalities such as sharing the movie list via email or social media. At the end of this module, the movie recommendation system showcases the power of content-based filtering in delivering personalized movie suggestions. By leveraging techniques like tokenization, feature extraction, and cosine similarity, the system efficiently processes textual data to recommend movies based on their content. The implementation of robust error handling ensures a smooth user experience, even in the face of potential input or data issues. This approach provides an easy-to-understand yet effective recommendation engine, making it a valuable tool for exploring movie preferences.

CHAPTER 5

IMPLEMENTATION & RESULTS

This chapter presents the practical implementation of the Movie Recommendation System and showcases the user interface through output screenshots. It also analyzes the results and evaluates the system's performance in suggesting movies based on content similarity.

5.1 IMPLEMENTATION DETAILS

The system is implemented using the Flask web framework in Python, chosen for its simplicity and flexibility in building lightweight web applications.

Key implementation components include:

1. **Flask (Backend Framework):** Manages HTTP routes, handles form submissions, and renders dynamic pages using templates.
2. **Python (Core Logic):** Used for reading the movie dataset, preprocessing feature data, and calculating content similarity for recommendations.
3. **pandas:** Reads and processes the movie data from a CSV file and supports data manipulation and filtering.
4. **scikit-learn:** Provides CountVectorizer for text vectorization and cosine_similarity() for calculating similarity between movie feature vectors.
5. **Frontend (HTML + CSS):** The index.html form allows users to input a movie name, and the result.html page displays detailed information about the selected movie and its top recommendations in a clean layout.

Workflow Summary:

1. The user inputs a movie title through a web form on the homepage.
2. The backend fetches the movie, processes its content features (overview, genre, cast, etc.), and vectorizes it using CountVectorizer.
3. Cosine similarity is calculated between the input movie and all others in the dataset.
4. The top 5 most similar movies are retrieved and their details are passed to the result page.

5. The user receives visual and textual recommendations, including posters, ratings, genres, and summaries.

5.2 OUTPUT SCREENSHOTS

This section includes key output screenshots demonstrating system functionality:

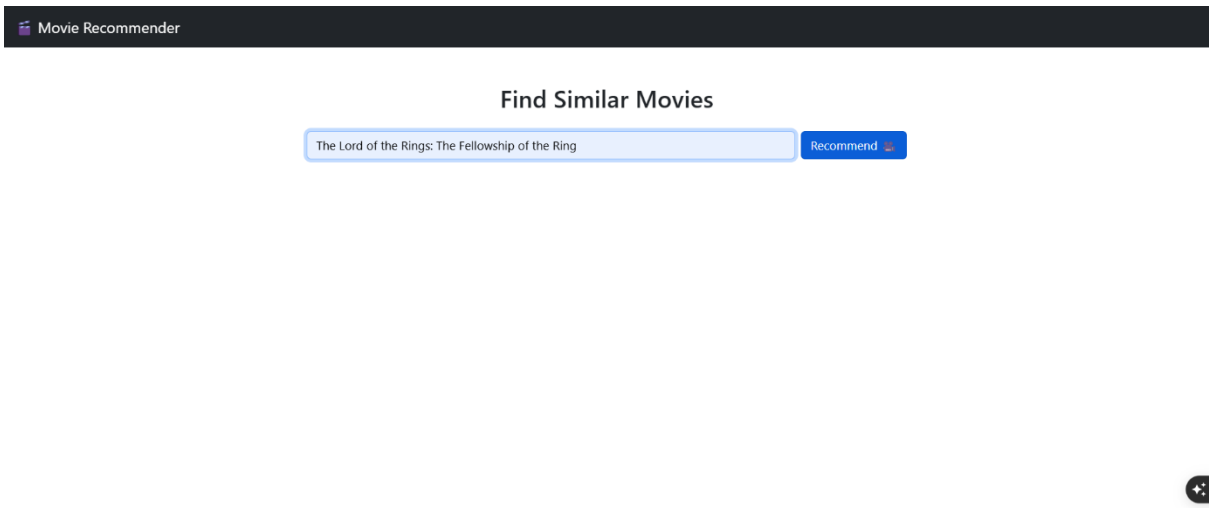

Fig 5.1 Home Page

Fig 5.2 Output interface showing the list recommended movies by the system

Fig 5.3 Output interface showing the contd. of list recommended movies by the system

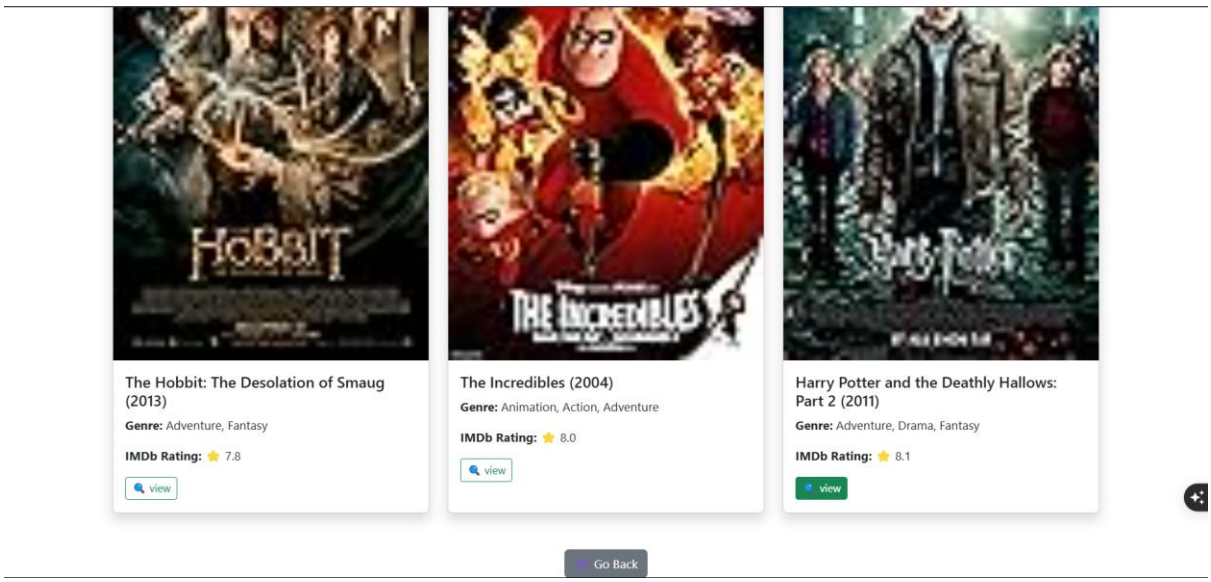

Fig 5.4 Use of 'view' button

.

Fig 5.5 Output interface showing the list recommended movies for the selected movie

5.3 RESULTS AND EVALUATION

The system was tested using a dataset of 1000+ movies stored in CSV format. Evaluation was done based on the following criteria:

Content Relevance of Recommendations: The recommended movies exhibited strong similarity in genre, storyline, and cast with the input movie, validating the effectiveness of feature combination and cosine similarity.

Response Time: For most user inputs, recommendations were generated and displayed within 1–3 seconds, ensuring a smooth and responsive user experience.

Scalability: The system performed efficiently for medium-sized datasets. For large-scale datasets or real-time applications, performance can be enhanced using optimized vector databases or parallel processing.

CHAPTER 6

CONCLUSION & FUTURE ENHANCEMENT

The Movie Recommendation System developed in this project demonstrates how machine learning and natural language processing techniques can be effectively used to personalize user experiences. By leveraging a content-based filtering approach that combines movie overviews, genres, directors, and cast members, the system is capable of providing relevant movie suggestions based on user input.

The use of the CountVectorizer for feature extraction and cosine similarity for comparison ensures fast and context-aware recommendations. This approach moves beyond basic keyword search by evaluating the overall content and semantics of each movie, delivering better suggestions to the end user. The system is lightweight, easy to deploy, and performs well for medium-sized movie datasets.

Several improvements can be made to enhance the system's performance and usability:

● **Deploy Online:** Hosting the system on a platform like Heroku, Render, or AWS would allow broader access to users looking for personalized movie recommendations.

● **Improve Feature Engineering:** Incorporating additional features such as user ratings, reviews, and watch history could improve recommendation accuracy and user satisfaction.

● **Support for Hybrid Recommendation:** Combining content-based and collaborative filtering approaches can enhance personalization by also learning from user behavior and preferences.

CHAPTER 7
REFERENCES

[1] F. Ricci, L. Rokach, and B. Shapira, “Introduction to Recommender Systems Handbook,” *Springer*, 2011.

[2] M. J. Pazzani and D. Billsus, “Content-Based Recommendation Systems,” in *The Adaptive Web*, Springer, 2007, pp. 325–341.

[3] R. Salakhutdinov, A. Mnih, and G. Hinton, “Restricted Boltzmann Machines for Collaborative Filtering,” in *Proceedings of the 24th International Conference on Machine Learning (ICML)*, 2007.

[4] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, “Neural Collaborative Filtering,” in *Proceedings of the 26th International Conference on World Wide Web (WWW)*, 2017.

[5] H. Wang, N. Wang, and D. Y. Yeung, “Collaborative Deep Learning for Recommender Systems,” in *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

[6] J. McAuley, C. Targett, Q. Shi, and A. van den Hengel, “Image-Based Recommendations on Styles and Substitutes,” in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2015.

[7] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[8] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[9] R. B. Schafer, J. A. Konstan, and J. Riedl, “Recommender Systems: Challenges and Research Opportunities,” in Proceedings of the 5th International Conference on World Wide Web (WWW), 2006.

[10] J. B. MacQueen, “Some Methods for Classification and Analysis of Multivariate Observations,” in Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967.

[11] S. Rendle, “Factorization Machines,” in Proceedings of the 10th IEEE International Conference on Data Mining (ICDM), 2010.

[12] Y. Koren, R. Bell, and C. Volinsky, “Matrix Factorization Techniques for Recommender Systems,” IEEE Computer, vol. 42, no. 8, pp. 30–37, 2009.

[13] R. Agerri and F. García-Serrano, “A Survey on Knowledge-Based Recommender Systems: Challenges, Research Trends and Applications,” Journal of Computer Science and Technology, vol. 31, pp. 1–33, 2016.