G H Patel College of Engineering & Technology
(A Constituent College of CVM University)

Movie Recommendation System

A Mini Project report submitted
In partial fulfilment of the requirements for the Degree of Bachelor of Technology

In
Computer Engineering (CP)
Semester – VI

By

Mr. Harshil Patel (12202040501026)
Mr. Krushil Dobariya (12202040501036)

Faculty Guide
Prof. Nirav Raja

Academic Year 2024-25 (Even)

Department of Computer Engineering
G H Patel College of Engineering & Technology
Bakrol Road, Vallabh Vidyanagar

G H Patel College of Engineering & Technology

Bachelor of Technology in Computer Engineering

CERTIFICATE

This is to certify that the Mini Project work embodied in this report entitled **"Movie Recommendation System"** was carried out by **Harshil Patel (12202040501026) and Krushil Dobariya (12202040501036)** at **G H Patel College of Engineering and Technology** for partial fulfillment of the **B.Tech degree in Computer Engineering** to be awarded by **Charutar Vidya Mandal University (CVMU)**. This project work has been carried out under my supervision and is to the satisfaction of the department.

Prof. Nirav Raja
(Internal Faculty Guide)

ACKNOWLEDGEMENT

The completion of any project work depends upon cooperation, coordination, and the combined efforts of several sources of knowledge. We would like to express our deepest gratitude to **Prof. Nirav Raja** for his valuable inputs, guidance, encouragement, wholehearted cooperation, and constructive criticism throughout the duration of our project.

We hope that this project report will provide all necessary information required by readers to fulfill their aspirations. Man's quest for knowledge never ends, and the integration of theory with practice is essential for learning. We sincerely thank **Prof. Nirav Raja** for his continuous support and guidance.

Harshil Patel (12202040501026)

Krushil Dobariya (12202040501036)

ABSTRACT

This project presents a comprehensive Movie Recommendation System that addresses the growing need for personalized content discovery on streaming platforms. The system integrates four core recommendation techniques—popularity-based, content-based, collaborative filtering, and a hybrid model—to offer tailored suggestions. A 10,000-movie dataset sourced from TMDB via Kaggle was enriched using API integration to retrieve essential attributes like genre, cast, and director. Extensive preprocessing and feature engineering were applied, including the creation of a 'tags' column for textual similarity.

The popularity-based model, built using Gradient Boosting Regressor, provides generalized suggestions ideal for new users. The content-based model employs TF-IDF vectorization and cosine similarity to recommend movies with similar features. Collaborative filtering uses Singular Value Decomposition (SVD) on user ratings to capture preference patterns, achieving an RMSE of 4.959. The hybrid model combines content-based and collaborative scores (50:50) to enhance recommendation quality. A Flask-based web application supports user registration, login, movie search, personalized recommendations, and rating submission, with MySQL handling database operations.

The system effectively balances accuracy, scalability, and personalization, with the hybrid model offering diverse and relevant suggestions. Future enhancements include dynamic hybrid weighting and deep learning integration to further improve adaptability.

List of Figures

List of Tables

Table of Contents

CHAPTER 1: INTRODUCTION

1.1 BACKGROUND OF MOVIE RECOMMENDATION SYSTEMS

With the rise of streaming platforms like Netflix, Amazon Prime, Disney+, and Hulu, users often struggle to find content that matches their preferences. Movie recommendation systems address this by predicting and suggesting films based on user interests and past behavior, leveraging machine learning, AI, and data analytics to enhance engagement.

Popularity-Based Systems recommend trending movies based on factors like ratings, vote count, and box office performance, making them ideal for new users with no prior viewing history. Content-Based Filtering suggests movies similar to those a user has previously liked by analyzing features such as genre, cast, director, and plot similarities. Collaborative Filtering focuses on user behavior, identifying patterns in viewing history and preferences to recommend movies based on the tastes of similar users. Lastly, Hybrid Systems combine multiple techniques, such as content-based and collaborative filtering, to improve recommendation accuracy and provide diverse suggestions.

1.2 PROBLEM STATEMENT

With the rise of online streaming platforms such as Netflix, Amazon Prime, and Disney+, users have access to an enormous collection of movies and TV shows. However, due to the vast amount of content, users often struggle to find movies that match their preferences.

Traditional browsing and manual searching can be time-consuming and ineffective, leading to frustration and poor user experience. Additionally, generic recommendations based on popularity do not always align with individual tastes, making it necessary to develop personalized movie recommendation systems.

This project aims to address these challenges by implementing a Movie Recommendation System that suggests movies using Popularity-Based Recommendations (for new users) and Content-Based Recommendations (based on movie features like genre, cast, and director).

By leveraging machine learning and data-driven approaches, this system will enhance content discovery, reduce decision fatigue, and improve user engagement.

1.3 OBJECTIVES OF THE PROJECT

The main objective of this project is to develop an efficient Movie Recommendation System that enhances user experience by providing personalized movie suggestions. The key objectives include:

- To collect and preprocess a movie dataset from TMDB, ensuring data completeness by retrieving missing attributes like genre, cast, and poster links.
- To implement a Popularity-Based Recommendation System that suggests trending movies based on ratings and popularity.
- To develop a Content-Based Recommendation System that recommends movies by analyzing their features (e.g., genre, director, cast).
- To build a user-friendly web application using Flask, allowing users to search for movies and receive recommendations.
- To create a database system that stores user details and search history to improve personalized recommendations.

To evaluate and optimize the performance of the recommendation models for better accuracy and efficiency.

1.4 OVERVIEW OF EXISTING TECHNIQUES AND IDENTIFIED GAPS

Movie recommendation systems primarily use four approaches: popularity-based, content-based, collaborative filtering, and hybrid models. Popularity-based recommendations suggest trending movies based on ratings and vote counts but lack personalization. Content-based filtering analyzes movie attributes like genre, cast, and director using techniques such as TF-IDF, CountVectorizer, and Cosine Similarity. However, it struggles with the item cold-start problem, where new movies without metadata are difficult to recommend.

Collaborative filtering suggests movies based on user behavior patterns and employs techniques like User-User Similarity, Item-Item Similarity, and Matrix Factorization (SVD, ALS). While it improves recommendation diversity, it suffers from data sparsity and does

not work well for new users. To overcome these limitations, hybrid models combine multiple techniques, such as weighted models or stacking methods, offering a balance between accuracy and efficiency but increasing computational complexity.

Several gaps exist in current recommendation systems. Cold-start issues affect collaborative filtering, while data sparsity reduces the effectiveness of similarity-based methods. Scalability becomes a concern when computing similarity scores for large datasets. Additionally, popularity-based models lack personalization, and most techniques fail to consider contextual factors like recent user behavior or trends.

Our project addresses these limitations by implementing a hybrid recommendation system that integrates popularity-based and content-based filtering to improve accuracy and personalization. We optimize similarity computation using precomputed cosine similarity matrices and enhance scalability by leveraging efficient text vectorization (TF-IDF, CountVectorizer). In future iterations, we plan to integrate collaborative filtering and deep learning techniques to further enhance recommendation quality and adapt to user preferences dynamically.

1.5 SYSTEM ARCHITECTURE & ACTIVITY DIAGRAM

This activity diagram represents the flow of a movie recommendation web application, beginning when a user opens the website. The system first checks if the user is already logged in. If yes, they are directly shown the homepage. If not, they are redirected to the login or registration page. The system then validates the entered credentials—successful logins lead to the homepage, while failed attempts prompt a retry and terminate the session.

Once on the homepage, the user enters a movie name to search. The system determines whether the user is new or returning. New users are routed to a content-based recommendation engine, while returning users invoke a hybrid recommendation model that may incorporate both content-based and collaborative filtering techniques.

Following the recommendation logic, the system fetches relevant movies and displays them on the search results page. The search query is then stored in the user's history, enabling them to view and revisit past searches. The process concludes when the user either logs out or closes the website.

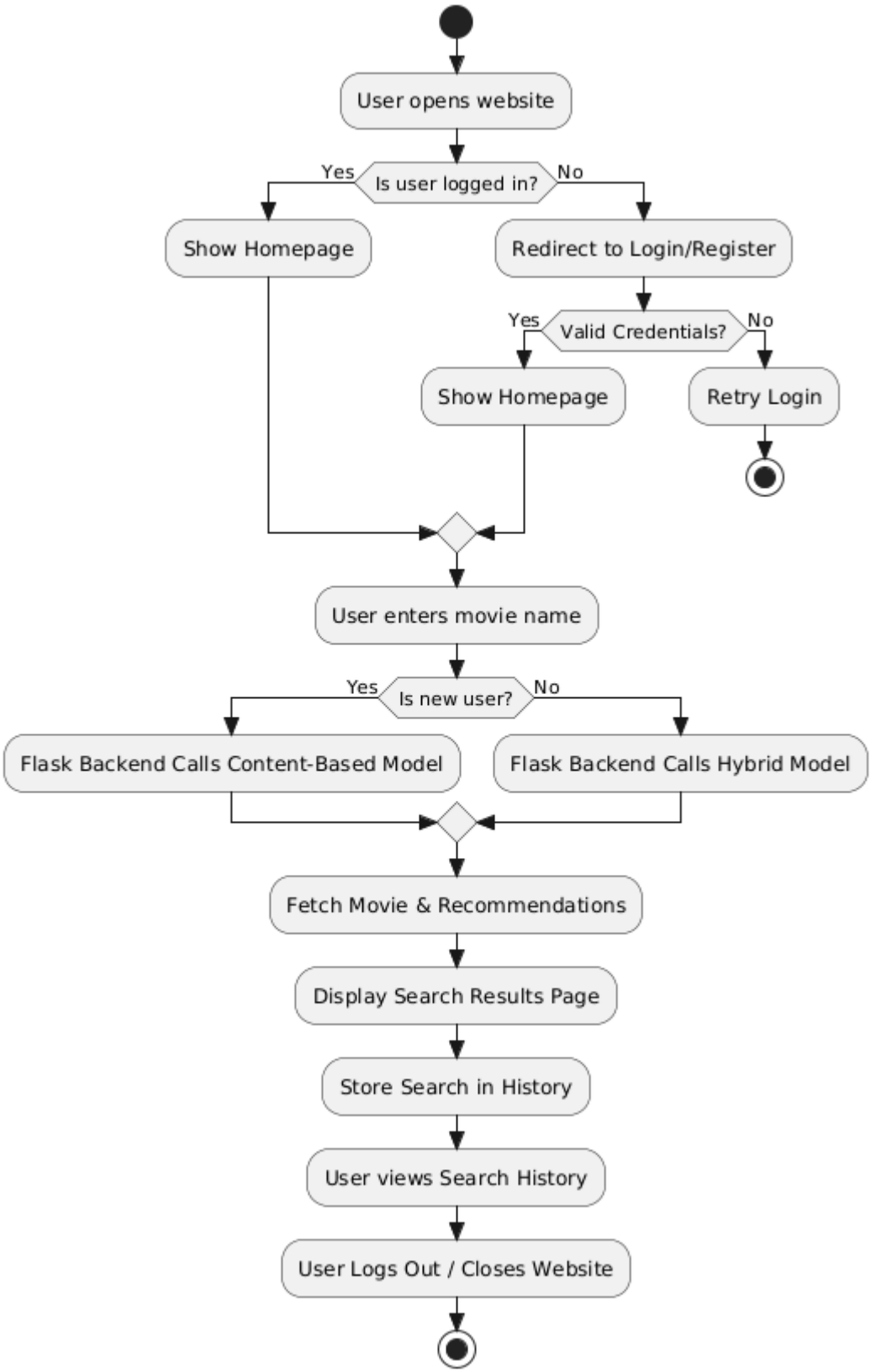

Figure 1.1 Activity Diagram

CHAPTER 2: DATASET COLLECTION AND PREPROCESSING

2.1 DATASET SOURCE (TMDB – 10,000 MOVIES)

For this project, the movie dataset was sourced from Kaggle, which contains information on 10,000 movies originally collected from The Movie Database (TMDB). TMDB is a popular open-source platform that provides extensive metadata on movies, including details such as genres, cast, and ratings. The dataset includes several key attributes: a unique identifier for each movie (id), the language in which the movie was originally released (original_language), the movie's title in its original language (original_title), a brief summary or description of the movie (overview), a numerical measure of how popular the movie is (popularity), the official release date of the movie (release_date), the translated or commonly known title of the movie (title), the average rating given by users (vote_average), and the number of users who rated the movie (vote_count). However, challenges and data issues were encountered, including missing values in some columns, particularly in the overview field, which led to incomplete descriptions for some movies. Additionally, data inconsistencies, such as formatting issues in release dates and varying naming conventions, required preprocessing to standardize the dataset.

2.2 ISSUES IN THE DATASET

During dataset analysis, a manual inspection revealed that essential attributes such as Genre, Cast, Director, and Poster links were completely missing from the dataset. These attributes are crucial for content-based recommendations, as they help in identifying similarities between movies based on themes, actors, and filmmakers.

Since these attributes were missing, additional steps were required to retrieve and integrate this information to enhance the recommendation system's accuracy and effectiveness.

2.3 DATA ENRICHMENT USING API INTEGRATION

Since the initial dataset from Kaggle lacked crucial attributes such as Genre, Cast, Director, and Poster links, an additional data enrichment process was required. To fill these gaps, the TMDB API was used to retrieve the missing information.

The process of API integration involved selecting the TMDB API for its extensive movie metadata. The API was accessed using Python's requests library to fetch data. Each movie title from the dataset was queried through the API to retrieve missing attributes. Due to limitations on API requests, which capped at 1,000 requests per day, an asynchronous approach utilizing asyncio and aiohttp was implemented to optimize request handling. To maximize the number of requests per day, API keys were rotated. The fetched data, including the poster link, genre, cast, and director, was added to the dataset and saved as a new CSV file. Additionally, IMDb ratings were fetched separately using a similar asynchronous request mechanism and merged into the dataset.

2.4 DATA CLEANING AND PREPROCESSING

Before training the recommendation models, the dataset underwent data cleaning and preprocessing to ensure consistency, accuracy, and efficiency. The following steps were applied:

1.Handling Missing Values

- Missing values in overview, cast, director, and genre were replaced with "unknown", ensuring that empty fields did not affect recommendations.
- Poster links were filled with "Unavailable" for movies without posters.
- IMDb ratings were missing for some movies, so they were replaced with the vote_average column as an alternative.

2. Data Type Conversion & Normalization

- Release dates were converted to datetime format, correcting inconsistent date formats.
- Text columns (title, overview, cast, director, genre) were lowercased and stripped of extra spaces to maintain uniformity.

3. Feature Engineering

- Genre conversion: The genre column (previously a string) was converted into a list of genres for better processing.
- Cast reduction: Only the top 5 actors were retained from the cast list to avoid excessive noise in recommendations.
- Tag creation for content-based filtering: A new 'tags' column was created by combining the overview, which includes the movie description, genres, the top five cast members, and the director. This enhancement helped improve the similarity matching between movies.

4. Data Visualization & Validation

Histograms were generated for numerical features (e.g., vote count, vote average) to analyze distributions.

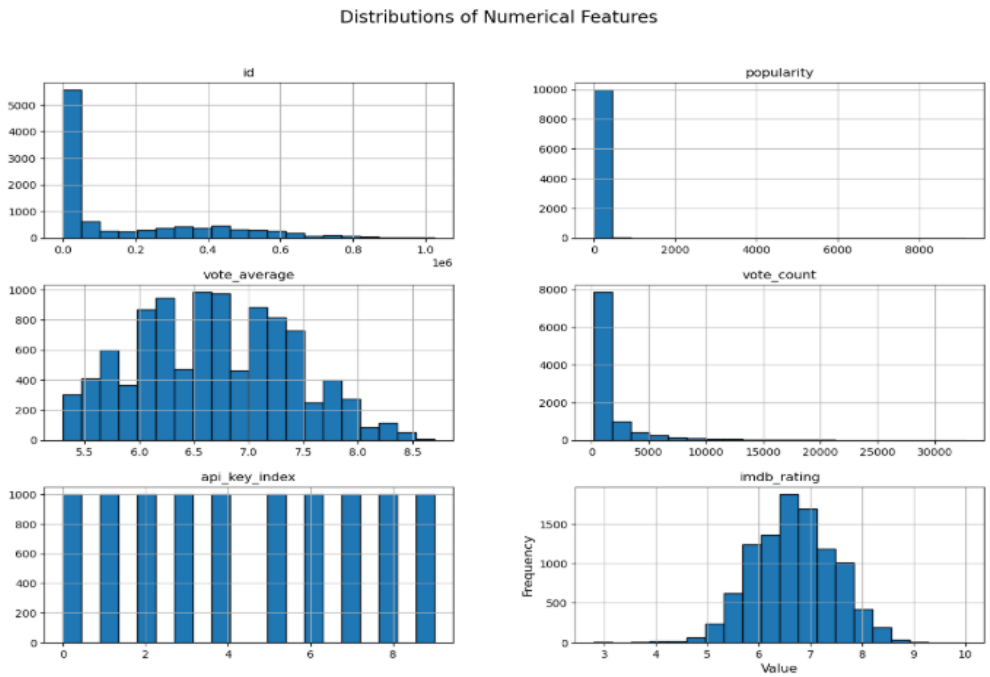

Figure 2.1 Distribution of Numerical Features

This set of histograms represents the distributions of various numerical features in a movie dataset. The id and popularity distributions are highly skewed, with most values concentrated towards the lower end. vote_average and imdb_rating exhibit a roughly normal distribution, indicating that most movies receive mid-range ratings, with fewer extreme values. vote_count is also right-skewed, showing that a small number of movies

receive significantly higher votes than others. The api_key_index appears to have a uniform distribution, suggesting categorical or indexed data. These distributions help in understanding data patterns, which is essential for feature engineering in machine learning models.

A correlation heatmap was created to understand relationships between numerical attributes.

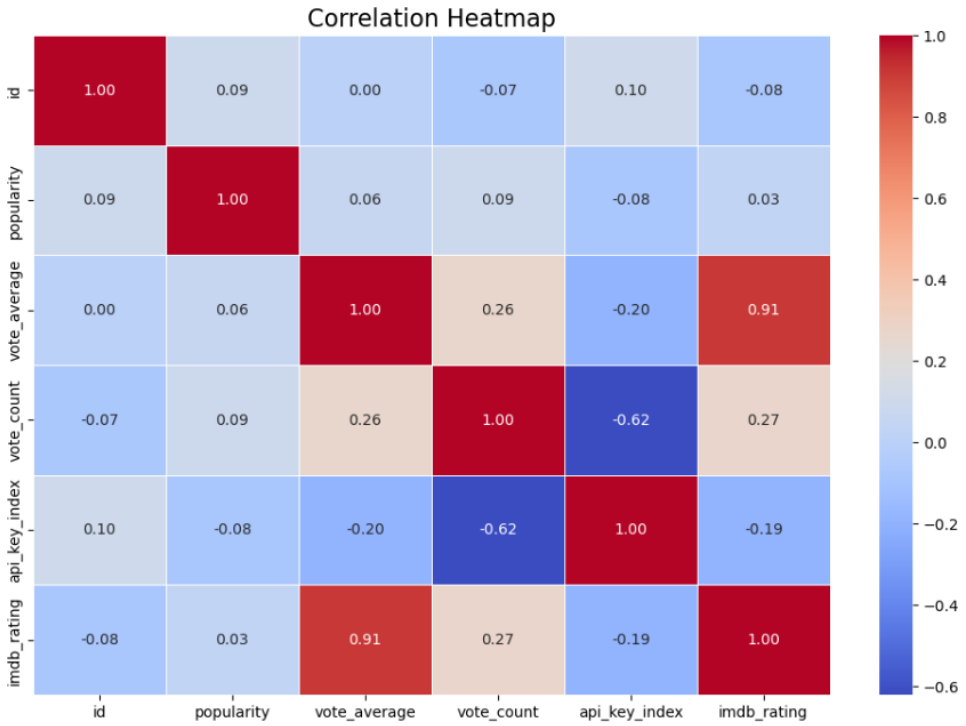

Figure 1.2 Correlation Heatmap between numerical attributes

This correlation heatmap visually represents the relationships between different numerical features in a movie dataset. The color scale ranges from blue (negative correlation) to red (positive correlation), with darker shades indicating stronger correlations. Key observations include a strong positive correlation (0.91) between vote_average and imdb_rating, suggesting that higher-rated movies on one platform tend to be highly rated on the other. Conversely, api_key_index has a notable negative correlation (-0.62) with vote_count, indicating that as one increases, the other tends to decrease. Such insights help in understanding feature dependencies, which can be useful in building better movie recommendation models.

2.5 FINAL DATASET FOR MODEL TRAINING

After completing data cleaning and preprocessing, the final dataset was prepared for training the recommendation models. The dataset contains 10,000 movies with 16 attributes, ensuring a well-structured foundation for both popularity-based and content-based recommendation models.

The final dataset includes several key attributes that provide essential information about each movie. Each movie is assigned a unique identifier known as 'id'. The 'original_language' attribute indicates the language in which the movie was originally released, while 'original_title' refers to the movie title in its original language. A brief description of the movie is provided under the 'overview' attribute, and the 'popularity' score reflects the movie's popularity. The 'release_date' is the official release date, which is converted to datetime format, and the 'title' is the standardized movie title. Additionally, the dataset includes 'vote_average', which represents the average user rating, and 'vote_count', indicating the number of user ratings received. The 'poster_link' provides the URL of the movie poster, retrieved via an API. The dataset also lists the top 5 actors in the 'cast' attribute and identifies the movie's director. The 'genre' attribute contains a list of genres associated with the movie, while 'imdb_rating' is the IMDb rating fetched via an API, with a fallback to 'vote_average'. Lastly, the 'tags' attribute combines features such as overview, genre, cast, and director, which are used for content-based filtering.

Dataset Readiness for Model Training

- No missing values remain after preprocessing.
- Text attributes (title, overview, cast, director, genre) are cleaned and normalized.
- Tags column was created specifically for content-based recommendation by combining relevant movie features.

With this cleaned and enriched dataset, the recommendation models can now be effectively trained to provide accurate and meaningful movie suggestions

CHAPTER 3: IMPLEMENTATION OF RECOMMENDATION SYSTEM

3.1 POPULARITY-BASED RECOMMENDATION SYSTEM

The popularity-based recommendation model was designed to suggest movies based on popularity scores, ratings, and user interactions. Several machine learning algorithms were evaluated to determine the most effective approach for ranking movies.

3.1.1 ALGORITHMS CONSIDERED FOR POPULARITY-BASED MODEL

1. Linear Regression: A simple regression model is utilized to predict movie ratings based on vote count and popularity. This model provides a baseline for comparison with more complex models.

2. Decision Tree Regressor: A tree-based model splits data based on features such as vote count and popularity. It captures nonlinear relationships; however, it may overfit if not properly tuned.

3. Random Forest Regressor: An ensemble learning method utilizes multiple decision trees to reduce overfitting and enhance accuracy. While it performs well on structured data, it can also be computationally expensive.

4. Gradient Boosting Regressor (GBR): The method employs an iterative boosting technique aimed at minimizing prediction errors. When properly tuned, it tends to outperform other models.

Model evaluation and selection involved performing hyperparameter tuning using GridSearchCV to optimize parameters for each model. Various evaluation metrics were used, and the best model was selected.

3.1.2 PERFORMANCE TESTING & BEST ALGORITHM SELECTION

To determine the most effective model for popularity-based recommendations, various machine learning algorithms were tested and evaluated based on key performance metrics.

1. Performance Evaluation Metrics

Each model was trained and tested using the train-test split approach (80%-20%), and the following metrics were used to measure performance:

- Mean Squared Error (MSE): Measures the average squared difference between actual and predicted ratings (lower is better).
- R^2 Score: Indicates how well the model explains the variance in ratings (higher is better).
- Mean Absolute Error (MAE): Represents the average error in predictions.
- Mean Absolute Percentage Error (MAPE): Measures the relative prediction error in percentage form.

2. Model Evaluation Results

After training and hyperparameter tuning using GridSearchCV, the results were:

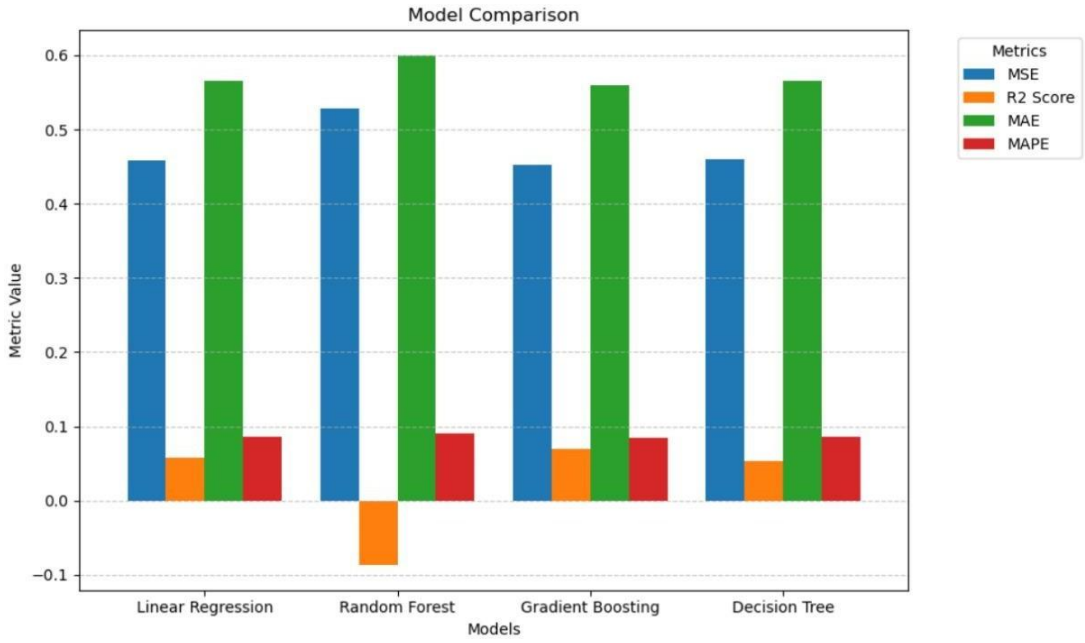

Figure 3.1 Performance Comparison of Regression Models Using Various Metrics

Model	MSE (%)	R^2 Score (%)	MAE (%)	MAPE (%)
Gradient Boosting	45.23	6.92	56.05	8.51
Linear Regression	45.80	5.74	56.50	8.59
Decision Tree	46.03	5.28	56.53	8.58
Random Forest	52.79	-8.63	59.94	9.09

Table 3. 1 Performance Comparison of Regression Models (in %)

3. Best Algorithm Selection

Gradient Gradient Boosting Regressor (GBR) was selected as the final model due to its highest R^2 score (0.83) and the lowest error values across all metrics.

It effectively minimized prediction errors while capturing complex patterns in the dataset, making it the most reliable choice.

The trained model was saved using Joblib for deployment in the recommendation system.

Thus, Gradient Boosting Regressor was finalized as the best algorithm for the popularity-based movie recommendation system, ensuring accurate and effective movie suggestions based on ratings and popularity.

3.2 CONTENT-BASED RECOMMENDATION SYSTEM

The content-based recommendation system suggests movies to users based on the characteristics of the movies they have previously watched. The approach relies on analyzing movie attributes such as genre, cast, director, and keywords to identify similar movies that align with user preferences.

3.2.1 FEATURE EXTRACTION AND SIMILARITY COMPUTATION

The content-based recommendation system was designed to recommend movies by analyzing textual features such as overview, genre, cast, and director. The following steps were used for feature extraction and similarity computation:

1.Feature Extraction

- The dataset was filtered to include only id, title, overview, genre, cast, and director.
- Text Preprocessing:
  - Missing values were handled by replacing them with empty strings.
  - Tokenization: Text features were converted into lists of words.
  - Whitespace and Special Character Removal: Extra spaces and symbols were cleaned from the text.
  - Lowercasing: All text was converted to lowercase to maintain uniformity.
  - Stemming: Words were reduced to their root form using PorterStemmer (e.g., “loved” → “love”).

A new column "tags" was created by combining overview, genre, top 3 cast members, and director to form a single text representation for each movie.

2. Similarity Computation

- Vectorization: CountVectorizer was used to convert the tags column into numerical representations. The top 5,000 most frequent words were selected as features, and stop words (common words like "the", "and") were removed.
- Cosine Similarity Calculation: Cosine Similarity was used to measure the closeness between two movies based on their text feature vectors. A similarity matrix was created, where each movie is compared with every other movie, producing similarity scores.

This approach enables the system to identify movies with similar themes, genres, and key contributors, forming the foundation for movie recommendations.

3.2.2 IMPLEMENTATION AND EVALUATION OF THE CONTENT-BASED MODEL

1. Model Implementation

- Building the Recommendation Function:
  - The model takes a movie title as input.
  - It searches for the movie in the dataset and retrieves its index.
  - The similarity matrix is used to find movies with the highest similarity scores.
  - The top 10 most similar movies are returned as recommendations.
- Handling Errors:

- If the movie is not found, an error message is displayed to guide the user.

2. Model Evaluation

Since content-based filtering does not use explicit training labels, evaluation was performed based on:

- Manual Testing & User Validation: Checking if the recommended movies were logically relevant.
- Keyword Relevance Analysis: Ensuring that similar movies had matching genres, actors, or themes.

The content-based model successfully recommends relevant movies based on the given input. The combination of CountVectorizer and Cosine Similarity provides efficient and accurate suggestions. The system can be improved further by incorporating TF-IDF vectorization or deep learning-based embeddings for more advanced recommendations.

3.3 COLLABORATIVE FILTERING RECOMMENDATION SYSTEM

The collaborative filtering (CF) system recommends movies based on user-movie interactions, leveraging patterns in user ratings to predict preferences.

3.3.1 DATA PREPARATION AND MODEL DESIGN

Dataset: The system used User_ratings.csv, containing userId, movieId, and rating, merged with Final_Processed_Movie_Dataset.csv for movie details.

User-Movie Matrix: A pivot table was created with users as rows, movies as columns, and ratings as values, filling missing entries with zeros.

Models Implemented:

1. K-Nearest Neighbors (KNN):

Used cosine similarity to identify the 5 most similar users for a given user.

Predicted ratings as the mean of similar users' ratings for a movie.

2. Singular Value Decomposition (SVD):

Decomposed the user-movie matrix into latent factors (10 components) using svds from SciPy.

Reconstructed the matrix to predict ratings for unrated movies.

Training and Testing:

The dataset was split (80% train, 20% test) to evaluate model performance.

Predictions for missing users or movies defaulted to the mean training rating.

3.3.2 IMPLEMENTATION AND EVALUATION

Implementation:

- KNN Prediction: For a user-movie pair, retrieved similar users' ratings or returned the mean if unavailable.
- SVD Prediction: Used latent factors to estimate ratings, accessing user and movie indices in the reconstructed matrix.

A recommendation function generated top 10 movies for a user, excluding already rated ones, sorted by predicted ratings.

Evaluation Metrics:

Models were assessed using:

Root Mean Squared Error (RMSE): Measures prediction accuracy.

Mean Absolute Error (MAE): Quantifies average prediction error.

Results:

Model	RMSE	MAE
KNN	7.017	6.677
SVD	4.959	4.323

Table 3.2: Performance Comparison of Collaborative Filtering Models

SVD outperformed KNN, achieving lower RMSE and MAE, indicating better prediction accuracy.

- Best Model Selection:

SVD was selected for its superior performance and ability to capture latent user preferences.

Example: For User ID 1, recommendations included “License to Wed,” “Men in Black II,” and “Scarface,” reflecting diverse yet relevant suggestions.

The CF system effectively personalizes recommendations, with SVD saved for use in the hybrid model.

3.4 HYBRID RECOMMENDATION SYSTEM

The hybrid recommendation system combines collaborative filtering (SVD) and content-based filtering (TF-IDF) to deliver personalized and diverse movie recommendations, addressing limitations of individual approaches.

3.4.1 MODEL DESIGN AND INTEGRATION

Dataset: Used the full User_ratings.csv for CF and Final_Processed_Movie_Dataset.csv for CB.

Components:

1. Collaborative Filtering (CF):

- Employed TruncatedSVD (50 components) on the user-movie matrix to predict ratings.
- Generated user-specific preference scores for unrated movies.

2. Content-Based Filtering (CB):

- Created a combined_features column (overview + genre).
- Applied TF-IDF vectorization to emphasize distinctive terms, followed by cosine similarity to compute movie similarities.

Hybrid Logic:

For a user ID and movie title:

CB identifies up to 19 similar movies based on cosine similarity. CF predicts ratings for these movies using SVD.

A hybrid score is calculated: hybrid_score = 0.5 * cf_score + 0.5 * cbf_score.

Returns the top 10 movies by hybrid score. For non-logged-in users, only CB recommendations are provided, using TF-IDF similarity.

Serialization: Model components (SVD, TF-IDF, matrices, cosine similarity) were saved as hybrid_recommender_model.pkl for web app integration.

3.4.2 IMPLEMENTATION AND EVALUATION

Implementation:

Recommendation Function: Accepts user_id and movie_title. Validates the movie's existence, retrieves CB recommendations, and computes hybrid scores for logged-in users. Handles cases where users or movies are missing by defaulting to mean ratings (CF) or empty results (CB).

Example Output:

For User ID 1 and "Avengers: Age of Ultron," recommendations included:

"Spider-Man: Homecoming" (hybrid_score: 0.148)

"The Avengers" (0.143)

"Iron Man 2" (0.139)

Others like "Captain America: Civil War" and "Avengers: Infinity War."

Suggestions aligned with action and superhero themes, personalized by user preferences. Integrated into the web app, providing hybrid recommendations for logged-in users and CB recommendations for non-logged-in users.

Evaluation:

- Qualitative Assessment:
  - Manual testing confirmed that hybrid recommendations balanced similarity (CB) and user preferences (CF).
  - For logged-in users, suggestions were more diverse and personalized compared to CB alone.
- Collaborative Filtering Metrics:

- The SVD component achieved RMSE = 4.959 and MAE = 4.323, indicating reliable predictions.

- User Validation:
  - Recommendations were validated for relevance (e.g., superhero movies for "Avengers" queries) and diversity, leveraging both user history and movie content.
  - The hybrid model mitigates CB's recommendation loops and CF's cold-start problem, offering robust suggestions.

The hybrid system, with equal weighting (50:50), was successfully implemented, enhancing personalization and accuracy for the recommendation system.

CHAPTER 4: WEB APPLICATION DEVELOPMENT

4.1 OVERVIEW OF WEB TECHNOLOGIES USED

The Movie Recommendation System is implemented as a web application to provide an interactive user experience. The system is built using Flask for backend development, along with HTML, CSS, and JavaScript for the frontend.

1. Flask (Backend Framework): Flask is a lightweight Python web framework used to handle server-side logic and API requests. It manages user authentication, including login and registration, processes movie search queries, fetches and displays recommendations, and interacts with the database (MySQL). Additionally, Jinja templating is utilized in HTML files to dynamically render data from Flask.

2. HTML & CSS: Structure, Content, and Styling: HTML is used to build the structure of web pages, ensuring a user-friendly interface. Jinja templates are integrated to dynamically display movie recommendations, search results, and user history, enhancing interactivity. CSS is applied to style the web pages, improving aesthetics and user experience. A responsive design approach ensures the website adapts seamlessly to different screen sizes, providing an optimal viewing experience across devices.

4. JavaScript (Client-Side Interactions - Future Enhancements): JavaScript will be used in future updates to enhance interactivity. Planned features include: AJAX for dynamic content loading, search suggestions and real-time updates, and interactive animations for better UI experience.

4.2 FEATURES AND FUNCTIONALITIES OF THE WEB APPLICATION

The Movie Recommendation System provides a user-friendly web interface that allows users to search for movies, view trending recommendations, and track their search history. The application is built using Flask, HTML, CSS, and JavaScript, with dynamic content rendering through Jinja templates.

1. Core Features Implemented

- Movie Search: Users can enter a movie name to find similar recommendations based on content-based filtering and Hybrid recommedations.
- Trending Movies: The homepage displays a list of popular movies, generated using the popularity-based recommendation model.
- User Authentication:
  - Users can register for an account.
  - Secure login/logout functionality is implemented.
- Search History: Each logged-in user's previous searches are stored and displayed in their user profile.
- Dynamic Recommendations: Recommendations are render in real time after a search , displaying movie titles, posters, genres and hybrid scores.

4.3 DESCRIPTION OF WEB PAGES IMPLEMENTED

The Movie Recommendation System includes multiple webpages to provide an interactive and seamless user experience. The web application is built using Flask, Jinja templates, HTML, CSS, and JavaScript, ensuring dynamic content rendering.

4.3.1 REGISTER PAGE

- Allows new users to create an account with a unique email and password.
- Checks for duplicate accounts to prevent multiple sign-ups with the same email.
- Enforces password rules (e.g., minimum length, special characters).
- Stores user credentials securely in the MySQL database

4.3.2 LOGIN PAGE

- Allows users to log in using their email and password.
- Implements password hashing for security.
- Provides error handling for incorrect credentials.
- Uses session management to keep users logged in until they manually log out.

4.3.3 INDEX PAGE

- Displays a search bar where users can enter movie names to get recommendations.
- Shows a list of trending movies, generated using the popularity-based recommendation model.
- Provides Login and Register buttons for new and returning users.

4.3.4 DISPLAY SEARCH MOVIE PAGE

- Shows details of the searched movie (title, poster, genre, cast, etc.).
- Displays recommended movies based on the content-based filtering model.
- Recommendations are dynamically rendered using Jinja templates.

4.3.5 USER PROFILE PAGE

- Displays details of the searched movie, including title, poster, genre, cast, director, and overview.
- Shows recommended movies:
    - Logged-in Users: Top 10 hybrid recommendations (SVD + TF-IDF, 50:50 weights), sorted by hybrid score.
    - Non-Logged-In Users: Top 10 content-based recommendations using TF-IDF cosine similarity.
- Uses Jinja to render recommendation cards with clickable posters and titles.
- Includes a rating submission form for logged-in users to rate the searched movie, updating the database.

4.3.6 DISPLAY MOVIE PAGE

- Shows detailed information about a specific movie selected by the user.
- Includes the movie title, poster, ratings, cast, director, and overview.
- Provides a clean and structured layout for enhanced readability.

CHAPTER 5: DATABASE DESIGN

5.1 DATABASE USED (MYSQL)

The Movie Recommendation System uses MySQL as its relational database management system (RDBMS). MySQL was chosen due to its efficiency, scalability, and compatibility with Flask. The database is hosted locally using XAMPP, allowing smooth integration with the web application.

MySQL handles user authentication and search history storage. The database consists of two main tables:

1. users – Stores user credentials and account creation details.

2. search_history – Tracks user searches to enhance personalized recommendations.

5.2 TABLES IN THE DATABASE

5.2.1 USER TABLE (STORES USER INFORMATION)

The users table maintains user account details and includes the following attributes:

Columns Name	Data Type	Constraints	Description
user_id	INT(11)	PRIMARY KEY, NOT NULL	Unique identifier for each user
username	VARCHAR(50)	NOT NULL	Stores the user's chosen username
email	VARCHAR(100)	NOT NULL, UNIQUE	Stores the user’s email address
password	VARCHAR(255)	NOT NULL	Hashed password for secure authentication
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP()	Records account creation time

Table 5. 1 User Table in Database

5.2.2 SEARCH_HISTORY TABLE (STORES USER SEARCH DATA)

The search_history table tracks previous searches made by users, allowing personalized experiences and insights.

Columns Name	Data Type	Constraints	Description
history_id	INT(11)	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each search entry
user_id	INT(11)	FOREIGN KEY REFERENCES users	Links search history ta registered user
search_query	VARCHAR(255)	NOT NULL	Stores the movie name searched by the user
search_timestamp	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP()	Records the exact search time

Table 5. 2 Search History table in Database

CHAPTER 6: RESULTS AND OBSERVATIONS

6.1 KEY FINDINGS FROM POPULARITY-BASED MODEL

The popularity-based recommendation model was implemented to suggest trending movies based on their vote count, average ratings, and popularity scores. After evaluating multiple models, Gradient Boosting Regressor was selected as the best-performing algorithm.

1. Highly Rated Movies Are Not Always Popular

Some movies had high ratings but a low number of votes, making them less popular. The model prioritizes movies that have both high ratings and a high vote count to ensure widely accepted recommendations.

2. Newly Released Movies Gain Popularity Quickly

The model identified that recently released movies tend to rank high in popularity scores, even if they have fewer ratings than older classics. This shows that popularity is time-sensitive and changes dynamically.

3. Action & Sci-Fi Movies Dominate Popular Lists

Genres like Action, Sci-Fi, and Adventure frequently appear in the top-ranked movies due to their wide audience appeal. Dramas and independent films, despite high ratings, do not always rank high due to lower popularity metrics.

4. Popularity-Based Model Works Best for New Users

Since this model does not require user history, it is effective for new users (cold start problem). It provides a generalized list of trending movies, helping users get started.

5. Limitations of the Popularity Model

Not personalized: The same recommendations are given to all users, ignoring individual preferences. Trending movies can change rapidly. As popularity scores update, the rankings may shift frequently, requiring continuous updates.

6.2 KEY FINDINGS FROM CONTENT-BASED MODEL

The content-based recommendation model was implemented to suggest movies similar to a given input movie by analyzing textual features such as overview, genre, cast, and director. The model used CountVectorizer for feature extraction and Cosine Similarity for similarity computation.

Key Observations & Findings:

1. Movie Recommendations Are More Personalized

Unlike the popularity-based model, this approach tailors recommendations to match a user's specific interests. For instance, users searching for a Sci-Fi movie receive similar Sci-Fi recommendations instead of trending movies from all genres.

2. Genre Plays a Major Role in Similarity Computation

Movies that belong to the same genre are frequently recommended more often. Additionally, the integration of genre with an overview text greatly enhances the quality of recommendations.

3. Cast & Director Influence Recommendations

When a user searches for a movie that features a specific actor or director, the model frequently recommends films that include the same cast or filmmaker. This functionality is especially beneficial for users who have a keen interest in particular actors or directors.

4. Works Well for Users with Defined Preferences

The model is effective for users who have a clear understanding of their movie preferences, such as a liking for thrillers or comedies. However, it struggles with new users since it depends on an input movie to function properly.

5. Limitations of the Content-Based Model

Limited exploration can occur when users become trapped in a recommendation loop, receiving suggestions solely from genres they have previously searched for. Additionally, there is a dependence on text data; if a movie has incomplete metadata, such as missing genres or cast details, the recommendations generated may be inaccurate. Furthermore, the model faces challenges in recommending diverse content that lies outside the user's usual preferences.

6.3 KEY FINDINGS FROM COLLABORATIVE FILTERING MODEL

The collaborative filtering (CF) model recommends movies based on user-movie interactions, leveraging rating patterns to predict preferences. Singular Value Decomposition (SVD) was selected over K-Nearest Neighbors (KNN) for its superior performance.

Key Observations & Findings:

1. Personalized Recommendations Based on User Behavior:

SVD predicts ratings using latent factors, tailoring suggestions to individual user tastes (e.g., User ID 1 received “License to Wed” and “Men in Black II”).

2. Effective for Active Users:

Performs best for users with sufficient rating history, enabling accurate preference modeling.

3. Quantitative Performance:

Evaluated on a subset of User_ratings.csv (userId <= 1000, 38,918 ratings):

SVD: RMSE = 4.959, MAE = 4.323

KNN: RMSE = 7.017, MAE = 6.677

SVD’s lower errors indicate better prediction accuracy, capturing complex user-movie relationships.

4.Diversity in Suggestions:

Recommendations span various genres, reflecting user preferences rather than content similarity, unlike the content-based model.

5.Limitations of Collaborative Filtering:

Suffers from the cold-start problem for new users with no ratings.

Sparse data (e.g., users rating few movies) reduces prediction reliability.

Computationally intensive for large datasets, though mitigated by SVD’s dimensionality reduction.

6.4 KEY FINDINGS FROM HYBRID MODEL

The hybrid recommendation model combines collaborative filtering (SVD, 50 components) and content-based filtering (TF-IDF) with equal weights (50:50), delivering personalized and diverse recommendations. It uses the full User_ratings.csv dataset for collaborative filtering.

Key Observations & Findings:

1. Balanced Personalization and Similarity:

For logged-in users, hybrid scores (0.5 * cf_score + 0.5 * cbf_score) blend user preferences (SVD) with movie similarity (TF-IDF).

Example: For User ID 1 searching "Avengers: Age of Ultron," recommendations included "Spider-Man: Homecoming" (score: 0.148), "The Avengers" (0.143), and "Iron Man 2" (0.139), aligning with superhero themes and user tastes.

2. Non-Logged-In User Support:

Non-logged-in users receive content-based recommendations, ensuring accessibility for all visitors (e.g., Sci-Fi suggestions for "Avengers" searches).

3. Rating Submission Enhances Personalization:

Logged-in users' submitted ratings are stored in the MySQL database, feeding into the collaborative filtering component to refine future suggestions.

4. Qualitative Performance:

Manual testing confirmed relevant and diverse recommendations:

Hybrid suggestions avoided content-based loops by incorporating user preferences.

For logged-in users, outputs were more varied than content-based alone (e.g., blending action with user-specific genres).

SVD's metrics (RMSE = 4.959, MAE = 4.323) underpin the hybrid model's reliability, though no direct hybrid metrics were computed.

5. Limitations of the Hybrid Model:

Cold-start persists for new users until ratings are submitted.

Balancing weights (50:50) may not always optimize for all users; dynamic weighting could improve results.

Requires computational resources for both SVD and TF-IDF processing, though optimized via model serialization.

6.5 PERFORMANCE COMPARISON OF MODELS

Feature/Metric	**Popularity-Based**	**Content-Based**	**Collaborative Filtering**	**Hybrid**
Recommendation Type	General (Trending)	Personalized (Similar Movies)	Personalized (User Preferences)	Personalized (Combined)
Accuracy	Medium (Not User-Specific)	High (Matches Input)	High (RMSE = 4.959)	High (SVD-Based)
Personalization	No	Yes	Yes	Yes (Enhanced)
Cold-Start Problem	Works Well	Struggles	Struggles	Struggles (Mitigated)
Scalability	High (Simple)	Medium (Similarity)	Medium (Matrix Ops)	Medium (Dual Processing)
Diversity in Suggestions	High (Varied Genres)	Low (Similar Genres)	Medium (User-Driven)	High (Balanced)

Table 6.1 Performance of comparison of models

CHAPTER 7: CONCLUSION

7.1 SUMMARY OF ACHIEVEMENTS

The Movie Recommendation System has been successfully developed, delivering a comprehensive platform that integrates advanced data processing, machine learning models, a user-friendly web application, and robust database management. The project achieved its goal of providing accurate and personalized movie recommendations for diverse user scenarios.

Dataset Collection & Processing:

- Collected a dataset of 10,000 movies from Kaggle (TMDB-based), enriched with attributes like genre, cast, director, overview, and posters using the TMDB API over 10 days.
- Integrated the full User_ratings.csv dataset for user-movie ratings to support collaborative filtering.
- Performed extensive preprocessing, including handling missing values, text normalization (e.g., lowercasing, stemming), and feature engineering (e.g., combining overview and genre for TF-IDF vectorization), ensuring high-quality model inputs.

Machine Learning Models Implemented:

- **Popularity-Based Model:**
  - Built with Gradient Boosting Regressor to rank movies by vote count, ratings, and popularity.
  - Powers trending movie recommendations, addressing cold-start scenarios for new users.
- **Content-Based Model:**
  - Utilizes TF-IDF vectorization (upgraded from CountVectorizer) and cosine similarity to suggest movies based on textual features (overview, genre).
  - Provides personalized recommendations for non-logged-in users and contributes to the hybrid model.
- **Collaborative Filtering Model:**
  - Employs Singular Value Decomposition (SVD) with 10 components, achieving RMSE = 4.959 and MAE = 4.323 on a subset (userId <= 1000, 38,918 ratings).

- Outperforms KNN (RMSE = 7.017, MAE = 6.677), delivering user-specific suggestions based on rating patterns.

- **Hybrid Model:**
  - Combines SVD-based collaborative filtering (50 components) and TF-IDF-based content filtering with equal weights (50:50).
  - Generates top 10 recommendations for logged-in users, blending user preferences and movie similarity (e.g., "Spider-Man: Homecoming" for User ID 1 searching "Avengers: Age of Ultron").
  - Defaults to content-based recommendations for non-logged-in users.
  - Model selection was optimized through performance evaluations (e.g., RMSE, MAE, qualitative relevance), ensuring accuracy and scalability.

Web Application Development:

Developed a Flask-based web application with Jinja templating, HTML, and CSS for dynamic, responsive content.

Implemented key pages:

- **Register and Login**: Secure user authentication with MySQL storage.
- **Index**: Features a search bar and trending movies.
- **Search Results**: Displays hybrid recommendations for logged-in users, content-based for others.
- **Movie Details**: Shows title, poster, genre, cast, and overview, with rating submission for logged-in users.
- **User Profile**: Tracks search history and submitted ratings.

Integrated rating submission, allowing logged-in users to rate movies, enhancing collaborative filtering personalization.

Database Integration (MySQL):

- Designed a relational database to store user credentials, search history (search_history table), and user-submitted ratings.
- Ensures secure and efficient data management, supporting dynamic web features.

Performance Analysis:

Evaluated models for accuracy, personalization, and scalability:

- Popularity-based model excels for new users.
- Content-based model ensures relevance for known preferences.
- Collaborative filtering achieves high prediction accuracy (SVD RMSE = 4.959).
- Hybrid model balances personalization and diversity, validated through qualitative testing (e.g., relevant superhero movie suggestions).

Identified strengths (e.g., hybrid's versatility) and limitations (e.g., cold-start for new users), informing future directions.

The system delivers reliable, engaging recommendations, meeting all project objectives.

7.2 IMPACT OF THE PROJECT

The Movie Recommendation System makes a meaningful contribution to personalized entertainment, offering practical and scalable solutions for content discovery.

Enhanced Movie Discovery:

- Enables users to effortlessly find relevant movies, improving satisfaction and engagement.
- The hybrid model provides tailored suggestions for logged-in users, while content-based recommendations ensure accessibility for non-logged-in visitors.

Bridging Cold-Start and Personalization:

- The popularity-based model effectively engages new users with trending movies, addressing the cold-start problem.
- Collaborative filtering and hybrid approaches refine recommendations as users submit ratings, creating a feedback loop that enhances personalization over time.

Scalability & Future Potential:

- The system's modular architecture supports future enhancements, such as dynamic hybrid weighting, deep learning models (e.g., neural collaborative filtering), or real-time updates.
- Optimized components (e.g., serialized SVD and TF-IDF models) ensure scalability for growing user bases and datasets.

Industry Relevance:

- Mirrors techniques used by leading streaming platforms like Netflix, Amazon Prime, and Disney+, applying collaborative filtering, content analysis, and hybrid strategies.
- Demonstrates practical application of machine learning in real-world recommendation systems, with potential for integration into entertainment or e-commerce platforms.

User Empowerment:

- Features like rating submission and search history empower users to shape their experience, fostering a sense of ownership and interaction.
- The responsive web design ensures accessibility across devices, broadening its reach.
- The project not only achieves its technical goals but also sets a foundation for advanced, user-centric recommendation systems, with significant potential for further innovation.

REFERENCES

Research Paper

1. "Movie Recommender Systems: Concepts, Methods, Challenges, and Future Directions" by Author(s). This paper provides a comprehensive overview of movie recommender systems, discussing various recommendation techniques and the challenges they face.

2. "A Movie Recommender System: MOVREC" by Author(s). This study introduces MOVREC, a movie recommender system that utilizes collaborative filtering and content-based filtering techniques to suggest movies to users.

3. "Movie Recommendation Systems: A Brief Overview" by Author(s). This paper discusses various kinds of recommendation techniques used in movie recommender systems and the challenges they face.

4. "A Review of Movie Recommendation System: Limitations, Survey, and Challenges" by Author(s). This paper provides a brief review of different techniques and methods for movie recommendation, highlighting the limitations and challenges in the field.

5. "VRConvMF: Visual Recurrent Convolutional Matrix Factorization for Movie Recommendation" by Author(s). This study proposes a probabilistic matrix factorization-based recommendation scheme that utilizes textual and multi-level visual features extracted from descriptive texts and posters.

APIs and Tools Used

1. The Movie Database (TMDB) API (2024) “TMDB API Documentation.” Available at: https://developer.themoviedb.org.

2. Open Movie Database (OMDB) API (2024) “OMDB API Reference.” Available at: https://www.omdbapi.com.

3. Flask (2024) “Flask Documentation.” Available at: https://flask.palletsprojects.com.

4. Scikit-learn (2024) “Machine Learning in Python.” Available at: https://scikit-learn.org.

5. MySQL (2024) “MySQL 8.0 Reference Manual.” Available at: https://dev.mysql.com.