### PROFESSIONAL TRAINING REPORT

**at**

### Sathyabama Institute of Science and Technology (Deemed to be University)

Submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering

By

## LOMADA VENKATA GNANA CHAITANYA REDDY(Reg.No- 42110715)



### DEPARTMENT OF COMPUTER SCIENCE ENGINEERING SCHOOL OF COMPUTING

**SATHYABAMA**

**INSTITUTE OF SCIENCE AND TECHNOLOGY (DEEMED TO BE UNIVERSITY)**

**CATEGORY- 1 UNIVERSITY BY UGC**

**Accredited “A++” by NAAC I Approved by AICTE JEPPIAAR NAGAR, RAJIV GANDHI SALAI CHENNAI - 600119**

**OCTOBER - 2024**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**BONAFIDE CERTIFICATE**

This is to certify that this Professional Training-1 Report is the bonafide work of

**LOMADA VENKATA GNANA CHAITANYA REDDY(42110715)** who carried out the Project entitled **“MOVIE RECOMMENDATION SYSTEM USING COLLABORATIVE FILTERING”** under my supervision from June 2024 to October 2024.

**Internal Guide**

**MS.V.DHARANI,M.E.,**

**Head of the Department**

**Dr. L. LAKSHMANAN, M.E., Ph.D.,**

**Submitted for Interdisciplinary Viva Voce Examination held on**

**Internal Examiner External Examiner**

**DECLARATION**

I, **LOMADA VENKATA GNANA CHAITANYA REDDY(Reg. No- 42110715),**hereby declare that the Professional Training-1 Report entitled **“MOVIE RECOMMENDATION SYSTEM USING COLLABORATIVE FILTERING”** done byme under the guidance of **MS.V.DHARANI,M.E.,** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

**DATE:**

**PLACE: Chennai SIGNATURE OF THE CANDIDATE**

**ACKNOWLEDGEMENT**

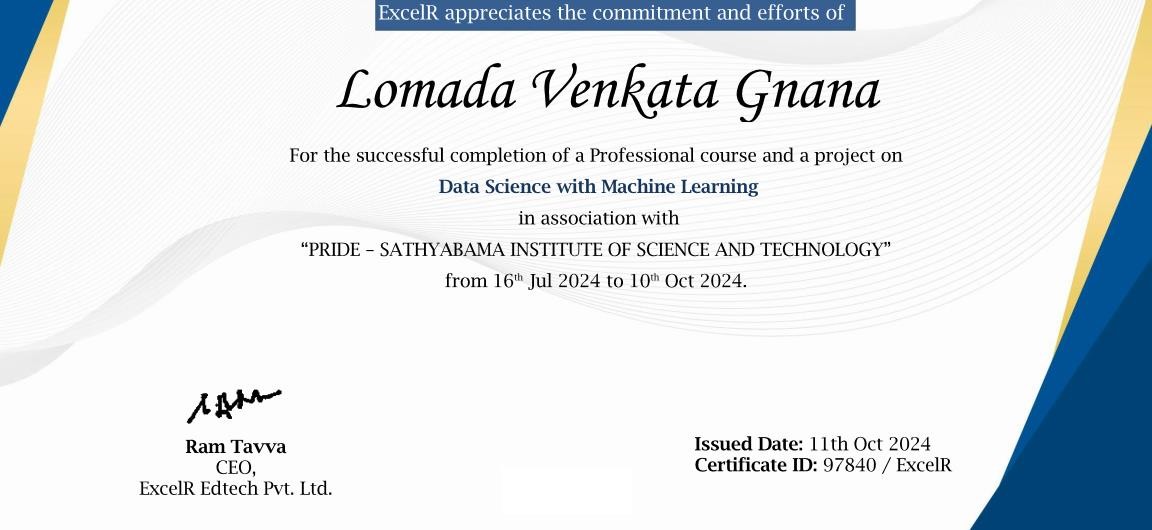
I am pleased to acknowledge my sincere thanks to **BOARD OF MANAGEMENT** of **Sathyabama Institute of Science and Technology** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph. D.**, **Dean**, School of Computing, and **Dr. L. Lakshmanan, M.E., Ph.D., Head of the Department** of Computer Science and Engineering for providing me with necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **MS.V.DHARANI,M.E.,** for her valuable guidance, suggestions, and constant encouragement paved the way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

**TRAINING CERTIFICATE**



## ABSTRACT

This project focuses on the development of a movie recommendation system using collaborative filtering, a widely employed method in recommendation engines. The system analyses historical movie ratings data to predict user preferences for movies they have not yet seen. Two datasets are used: one containing movie information and the other containing user-generated ratings. After loading and preprocessing the data, the ratings are structured into a matrix where rows represent movies, columns represent users, and the values correspond to ratings. Collaborative filtering is implemented in two ways: user-based filtering, which recommends movies based on the preferences of similar users, and item-based filtering, which recommends movies similar to those a user has rated highly. To address the challenge of sparse data, matrix factorization techniques like Singular Value Decomposition (SVD) may be used to reduce dimensionality and reveal latent patterns between users and movies. The project covers essential steps such as handling missing values, normalizing the ratings data, and applying similarity measures like cosine similarity or Pearson correlation. Evaluation metrics like Root Mean Square Error (RMSE) are likely explored to assess the performance of the recommendation system. Ultimately, the project demonstrates a practical application of machine learning in the entertainment industry, showing how

data-driven techniques can personalize user experiences. The approach could be expanded to other domains like books or music, using similar collaborative filtering techniques.

*.*

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER**  **NO** | **TITLE** | **PAGE**  **N0** |
| **ABSTRACT** | | **vi** |
| **LIST OF FIGURES** | | **ix** |
| **1 INTRODUCTION** | | **1** |
| 1.1 OVERVIEW | | 1 |
| 1.2 OBJECTIVES | | 2 |
| 1.3 PROBLEM STATEMENT | | 3 |
| 1.4 SCOPE OF THE PROJECT | | 4 |
| **2 LITERATURE REVIEW** | | **5** |
| 2.1 RELATED WORK IN RECOMMENDATION SYSTEM | | 5 |
| **3 SYSTEM ANALYSIS** | | **8** |
| 3.1 AIM OF THE PROJECT | | 8 |
| 3.2 EXISTING SYSTEM | | 8 |
| 3.3 PROPOSED SYSTEM | | 8 |
| 3.4 SYSTEM REQUIREMENTS | | 9 |
| 3.4.1 SOFTWARE REQUIREMENTS | | 9 |
| 3.4.2 HARDWARE REQUIREMENTS | | 10 |
| 3.4.3 DATA REQUIREMENTS | | 10 |
| 3.5 OVERVIEW OF THE PLATFORM | | 11 |
| 3.5.1 PYTHON | | 11 |
| 3.5.2 JUPYTER NOTEBOOK | | 12 |
| 3.5.3 COLLABORATIVE FILTERING | | 13 |
| 3.5.4 KKN ALGORITHM OVERVIEW | | 14 |

1. SYSTEM DESIGN 16
   1. SYSTEM STUDY 16
      1. [BENEFITS 16](#_TOC_250008)
      2. [DIFFERENT TYPES 17](#_TOC_250007)
      3. CHALLENGES 18
   2. [DATA PRE-PROCESSING 19](#_TOC_250006)
   3. [DATA SET USED 20](#_TOC_250005)
   4. [MODULE DESCRIPTION 21](#_TOC_250004)
   5. [RECOMMENDATION FUNCTION 23](#_TOC_250003)
2. SYSTEM IMPLEMENTATION 26
   1. [CODING 26](#_TOC_250002)
   2. [SAMPLE SCREENS 30](#_TOC_250001)
3. RESULTS AND DISCUSSION 33
   1. RESULTS AND DISCUSSIONS 33
4. CONCLUSION 34
   1. [CONCLUSION 34](#_TOC_250000)
   2. FUTURE ENHANCEMENT 35
5. REFERENCES 36
   1. REFERENCES 36

|  |  |  |
| --- | --- | --- |
|  | **LIST OF FIGURES** |  |
| **FIG NO** | **FIGURES** | **PAGE NO.** |
| 3.5.1 | PYTHON | 11 |
| 3.5.2 | JUPYTER NOTEBOOK | 12 |
| 3.5.3 | COLLABORATIVE FILTERING | 13 |
| 4.3 | DATA SET USED | 20 |
| 5.2 | SAMPLE SCREENS | 31 |
|  |

**CHAPTER-1**

**INTRODUCTION**

### Overview

A Movie Recommendation System is a tool that suggests movies to users based on their preferences, aiming to enhance user experience by providing personalized recommendations. Collaborative filtering, one of the most popular techniques used in recommendation systems, plays a pivotal role in learning from user behavior and improving the quality of suggestions over time.

#### Why Movie Recommendation Systems Matter

Movie recommendation systems have become increasingly important in today's digital age, where users are overwhelmed with a vast array of movie options. With the rise of online streaming services such as Netflix, Hulu, and Amazon Prime, users have access to thousands of movies at their fingertips. However, this abundance of choice can lead to decision paralysis, making it difficult for users to find movies that align with their tastes.

#### Key Features of Collaborative Filtering:

* + 1. Data-Driven Approach: Collaborative filtering uses historical data (e.g., movie ratings, user interactions) to find patterns and similarities among users or movies.
    2. No Need for Content Analysis: Unlike content-based filtering, which requires detailed information about movie attributes, collaborative filtering relies solely on user behavior (i.e., what movies users like or dislike).
    3. Two Main Approaches:
       - User-based Collaborative Filtering: Identifies similar users based on their movie preferences and recommends movies that similar users liked.
       - Item-based Collaborative Filtering: Looks for similarities between movies based on user ratings and recommends movies that are similar to ones the user liked.

### OBJECTIVES

#### Load and Understand Data:

* + - * Load movie and rating datasets from CSV files.
      * Understand the structure of the datasets (columns like movieId, Movie, Year, Ratings, etc.).

#### Data Preprocessing:

* + - * Clean and preprocess the data (handling missing values, converting data types if needed) Merge the datasets if necessary to create a unified structure for the recommendation system.
      * Merge the datasets if necessary to create a unified structure for the recommendation system.

#### Exploratory Data Analysis (EDA):

* + - * Analyze the distribution of movies and user ratings.
      * Identify popular genres, top-rated movies, and user behaviors.

#### Collaborative Filtering Model:

* + - * Implement a collaborative filtering algorithm (likely matrix factorization or nearest- neighbor based methods).
      * Train the model using the user-item rating matrix.
      * Predict user preferences for unseen movies.

#### Model Tuning and Optimization:

* + - * Tune model parameters to optimize recommendation accuracy.
      * Implement techniques like SVD (Singular Value Decomposition) or ALS (Alternating Least Squares) to improve predictions.

#### Exploratory Data Analysis (EDA):

* + - * Analyze the distribution of movies and user ratings.
      * Identify popular genres, top-rated movies, and user behaviors.

#### Collaborative Filtering Model:

* + - * Implement a collaborative filtering algorithm (likely matrix factorization or nearest- neighbor based methods).
      * Train the model using the user-item rating matrix.
      * Predict user preferences for unseen movies.

#### Model Tuning and Optimization:

* + - * Tune model parameters to optimize recommendation accuracy.
      * Implement techniques like SVD (Singular Value Decomposition) or ALS (Alternating Least Squares) to improve predictions.

#### Evaluation of the Model:

* + - * Evaluate the recommendation system's performance using metrics such as Root Mean Square Error (RMSE) or Mean Absolute Error (MAE).
      * Use train-test splitting or cross-validation for robust evaluation.

#### Recommendation System Deployment:

* + - * Generate movie recommendations for specific users based on the collaborative filtering model.
      * Create a user interface or API (optional) for real-time recommendations.

### Problem Statement:

With the rise of digital streaming platforms and a vast library of available movies, users often struggle to discover movies they would enjoy. The abundance of options can overwhelm users, leading to dissatisfaction and indecision. To enhance user experience and engagement, a recommendation system is needed to suggest movies that align with individual user preferences based on their historical interactions and the behavior of similar users.

The goal is to build a movie recommendation system using collaborative filtering that predicts a user's rating for movies they haven't watched yet. By leveraging user ratings and interactions, the system will provide personalized recommendations to help users find movies they are likely to enjoy.

#### Key Challenges:

* + 1. Data Sparsity: Many users rate only a few movies, leading to a sparse user-movie interaction matrix.
    2. Scalability: Handling large datasets with thousands of movies and users while maintaining efficient recommendation performance.
    3. Cold Start Problem: Providing recommendations for new users or movies with limited or no historical data.
    4. Accuracy: Ensuring the recommendations reflect users' true preferences by minimizing prediction errors.

### Scope of the Project:

The project involves building a movie recommendation system using a collaborative filtering approach. The scope can be understood from the following key steps in the notebook:

* + 1. **Data Import:** It imports movie and rating data from CSV files (movie.csv and rating.csv), loading these into pandas DataFrames.
    2. **Data Processing:** The movie dataset contains information such as movie IDs, titles, genres, years, and posters. The ratings data is pivoted into a matrix where movie IDs form the rows, user IDs form the columns, and the cells contain the corresponding movie ratings.
    3. **Collaborative Filtering**: The final dataset appears to be prepared for collaborative filtering, likely using user-item interactions to recommend movies based on similar user preferences.

**CHAPTER-2**

**LITERATURE SURVEY**

### Related Work in Recommendation System:

Recommendation systems are an essential area of research in machine learning and data mining, focusing on predicting user preferences and suggesting items accordingly. Here’s an overview of some key areas and methodologies within recommendation systems

#### Collaborative Filtering (CF)

Collaborative filtering is a technique used in recommendation systems that relies on the behavior and preferences of multiple users to provide personalized recommendations. The underlying idea is that if two users have similar tastes, they are likely to appreciate the same items.

#### Types of Collaborative Filtering:

**User-based Collaborative Filtering:**

This approach finds users that are similar to the target user and recommends items that those similar users have liked.

#### Process:

* + - * Compute the similarity between users (often using metrics like cosine similarity or Pearson correlation).
      * Select a group of similar users (k-nearest neighbors).
      * Recommend items that these neighbors have rated highly but the target user has not yet interacted with.

#### Item-based Collaborative Filtering:

Instead of finding similar users, this approach identifies items that are similar based on user ratings. It recommends items similar to those the user has already liked.

#### Process:

* + - * Calculate the similarity between items based on user ratings.
      * Recommend items similar to those the user has rated highly.

#### Matrix Factorization

Definition: Matrix factorization is a method used to decompose the user-item interaction matrix into lower-dimensional matrices, revealing latent features that can explain observed ratings.

#### Key Concepts:

**Latent Factors**: Hidden characteristics of users and items that explain observed interactions. For example, in a movie recommendation system, latent factors could include genres, themes, and user preferences.

#### Singular Value Decomposition (SVD):

A common matrix factorization technique that reduces the dimensionality of the user-item interaction matrix, capturing the most important features.

#### Alternating Least Squares (ALS):

Another approach that alternates between optimizing user and item matrices iteratively.

* + 1. Evaluation Metrics

To assess the performance of recommendation systems, various metrics are employed:

**Precision:** Measures the proportion of recommended items that are relevant to the user.

**Precision = True Positive**

**True Positives + False Positives**

**Recall:** Measures the proportion of relevant items that are recommended.

#### Recall = True Positives

**True Positives + False Negatives**

**F1 Score:** The harmonic mean of precision and recall, providing a balance between the two.

**Root Mean Square Error (RMSE):** Measures the average deviation between predicted ratings and actual ratings.

**RMSE=**

#### Cold Start Problem

The cold start problem arises when a recommendation system lacks sufficient data to make accurate recommendations. This can occur in three scenarios:

**New User:** The system has no prior interaction data for a new user, making it challenging to predict preferences.

**New Item:** Similar to new users, when a new item is introduced, there may be no ratings or interactions to base recommendations on.

**New System:** When the system is newly implemented, it lacks historical data to provide personalized recommendations.

#### Hybrid Recommendation Systems

Hybrid recommendation systems combine multiple recommendation techniques to improve the overall quality of recommendations. This can help mitigate the limitations of individual approaches.

#### Examples of Hybrid Approaches:

**Combining Collaborative and Content-based Filtering:** Use collaborative filtering to recommend items similar to those a user has liked, while also considering the content attributes of those items.

**Weighted Hybrid:** Assign weights to different recommendation techniques and combine their outputs based on performance metrics.

**CHAPTER-3**

**SYSTEM ANALYSIS**

### Aim of the Project:

The primary aim of the project is to design and implement a movie recommendation system that utilizes collaborative filtering techniques to provide personalized movie recommendations to users based on their preferences and the preferences of other users.

### Existing System

#### Collaborative Filtering:

* + - Uses user-based or item-based collaborative filtering for recommendations.
    - Limited by data sparsity and scalability issues.

#### Cold Start Problem:

* + - * Struggles with new users and movies due to lack of interaction data.

#### Matrix Factorization:

* + - * Basic methods like Singular Value Decomposition (SVD) are used but are sensitive to sparse data.

#### Limited Accuracy:

* + - * Recommendations are less accurate due to sparsity and lack of advanced algorithms.

#### Minimal UI:

* + - * User interface is typically basic with no user feedback integration.

### Proposed System:

#### Hybrid Approach:

* + - Combines collaborative filtering, content-based filtering, and matrix factorization for more personalized recommendations.

#### Cold Start Solution:

* + - Leverages user demographics and movie metadata to handle new users and movies effectively.

#### Improved Algorithms:

* + - Use advanced matrix factorization techniques (ALS, NCF) and possibly deep learning

for better recommendations.

#### Scalability:

* + - Optimized for large datasets using cloud infrastructure and caching mechanisms.

#### User-Friendly Interface:

* + - Develops an intuitive UI with Gradio, offering interactive feedback and real-time recommendations.

### System Requirements

* + 1. **Software Requirements**

#### Programming Language:

**Python**: Widely used for machine learning, data science, and recommendation systems.

#### Libraries:

**NumPy:** For numerical computations. **Pandas**: For data manipulation and handling. **SciPy:** Useful for scientific computations.

**Scikit-learn:** For building machine learning models, including collaborative filtering algorithms.

**Surprise:** A specialized library for building recommendation systems, including matrix factorization methods.

**TensorFlow/PyTorch:** For advanced deep learning models if you want to explore neural networks for recommendation systems.

**Matplotlib/Seaborn:** For data visualization and plotting results.

**Gradio**: For building a user-friendly interface where users can input movie preferences and receive recommendations.

#### IDE and Tools:

**Jupyter Notebook:** For prototyping, exploratory data analysis, and building recommendation models.

**PyCharm/VS Code:** Full-fledged IDEs for coding and project management.

#### Machine Learning Platforms:

Google Colab or Kaggle Notebooks: For running experiments, training models in the cloud with free access to GPUs.

### Hardware Requirements

**CPU:** Minimum Intel i5 or equivalent, preferably multi-core.

**RAM:** At least 8 GB for small datasets, but for larger datasets and complex models, 16 GB or

more is recommended.

**Storage:** Minimum 256 GB SSD (to store movie datasets, user interaction logs, etc.).

**GPU:** Not necessary for basic collaborative filtering models. However, if you plan to integrate deep learning models, a GPU like NVIDIA GTX 1060 or better (e.g., NVIDIA RTX 3000 series) will significantly speed up training times.

### 3.4.3Data Requirements

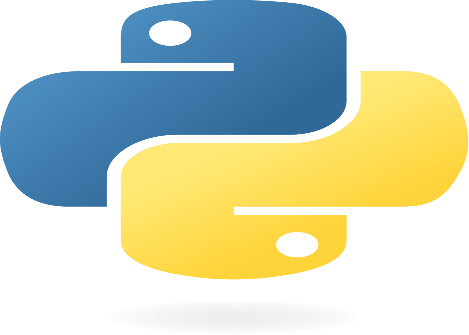
#### Datasets:

**MovieLens Dataset**: Commonly used dataset for movie recommendation projects, providing user ratings, movie metadata, and interaction data.

**IMDB API:** For additional movie metadata, such as genres, actors, directors, etc.

**Custom Data**: You may collect additional user interaction data, reviews, or ratings to train and test the model.

### Overview of the Platform

* + 1. **Python**

The programming language you'll be using for your movie recommendation system is Python. Python is widely chosen for such projects due to its simplicity, large community, and vast library support for machine learning, data analysis, and web development. Here's why Python is an excellent choice for this project:

#### Why Python?

**Extensive Libraries for Machine Learning:**

* + - * **Scikit-learn:** A widely used library for implementing collaborative filtering, matrix factorization (SVD, ALS), and other recommendation algorithms.
      * **Surprise:** Specialized for building and evaluating collaborative filtering models.
      * **TensorFlow/PyTorch:** If you want to explore deep learning techniques like Neural Collaborative Filtering (NCF).
      * **Pandas and NumPy:** For data handling, manipulation, and numerical computations.

#### Data Handling:

* **Pandas**: Efficiently handles large datasets such as user-item interaction matrices or movie metadata.
* **NumPy:** For fast numerical operations on arrays and matrices, which are essential for matrix factorization techniques.

#### Web Development:

* **Gradio:** Simplifies building an interactive, user-friendly web interface for your recommendation system.

#### Community Support and Documentation:

* Python has an extensive ecosystem of tools, libraries, and frameworks, with active support from the developer community, making it easier to find solutions and tutorials for specific issues.

#### Cross-Platform:

Python is cross-platform, meaning the code can run on various operating systems (Windows, macOS, Linux) without significant modifications.

### Jupyter NotebooksJupyter Notebook

Jupyter Notebook is an open-source, web-based development environment that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. It is widely used in data science, machine learning, and scientific computing because it supports interactive computing, where users can write and execute code in small, manageable chunks. This makes it ideal for iterative development, especially when experimenting with different algorithms, testing data preprocessing techniques, or visualizing data. Its support for multiple programming languages, including Python, R, and Julia, further extends its versatility.

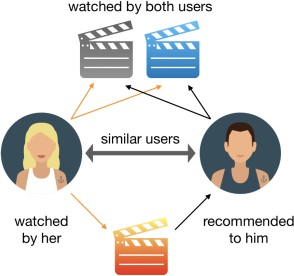
In the context of building a movie recommendation system, Jupyter Notebook is extremely useful for prototyping and data exploration. You can import datasets, visualize movie ratings, and experiment with different recommendation algorithms (like collaborative filtering or matrix factorization) in real time. The ability to add markdown cells alongside code cells allows thorough documentation, making it easy to explain and track the development process. Furthermore, Jupyter's support for visualizations (e.g., via libraries like Matplotlib and Seaborn) makes it convenient to display graphs, charts, and statistical summaries directly within the notebook, aiding in understanding how the data behaves and how well different models perform.

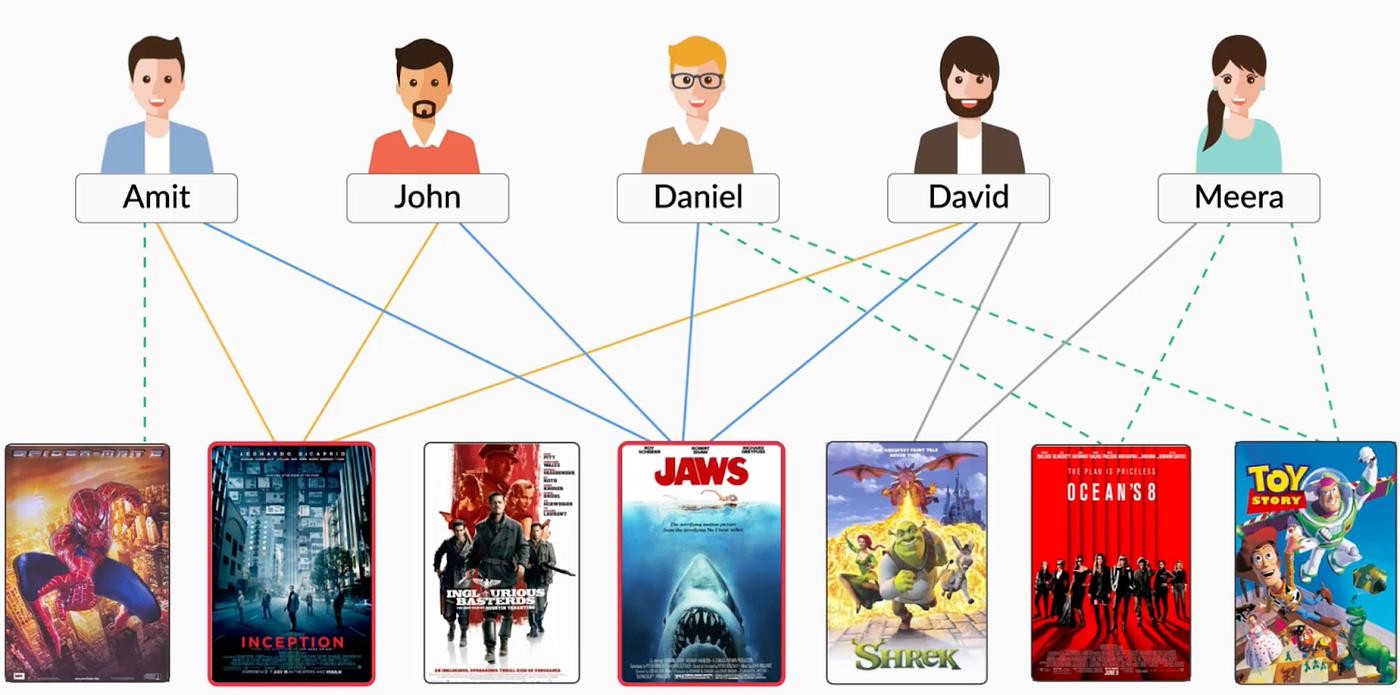
### 3.5.3Collaborative Filtering

Collaborative filtering is a widely used technique in recommendation systems that makes predictions about a user's interests by collecting preferences from many users. The core idea is to analyze the behavior and preferences of users to provide personalized recommendations. Instead of relying solely on the attributes of items, collaborative filtering utilizes the patterns of interactions among users and items. By leveraging the vast amount of user data, collaborative filtering can generate recommendations that reflect the tastes and preferences of similar users, making it a powerful approach for systems like movie or music recommendations.

#### Types of Collaborative Filtering

There are two primary types of collaborative filtering: user-based and item-based collaborative filtering.

* + - * **User-Based Collaborative Filtering:** This method identifies users with similar preferences or behaviors to a target user and recommends items that these similar users have liked or rated highly. For instance, if User A and User B have similar movie ratings, and User B has rated a specific movie highly, that movie would be recommended to User A. The effectiveness of this approach depends on the availability of sufficient user data and the ability to identify meaningful similarities among users. However, it can face challenges such as scalability issues and the "cold start" problem for new users who lack sufficient interaction history
      * **Item-Based Collaborative Filtering:** In contrast, item-based collaborative filtering focuses on finding similarities between items based on user ratings. It calculates the similarity between items (e.g., movies, products) and recommends items that are similar to those the user has already liked. For example, if a user enjoys a particular movie, the system will recommend other movies that have been liked by users who also liked that movie. Item- based filtering tends to perform better than user-based filtering in many scenarios, as item similarity is more stable over time compared to user preferences.



### 3.5.4K-Nearest Neighbors (KNN) Algorithm Overview

The K-Nearest Neighbors (KNN) algorithm is a popular and straightforward method used in recommendation systems, particularly in collaborative filtering. KNN operates on the principle of similarity, where it identifies a predefined number of nearest neighbors (denoted as K) to a target user or item based on their characteristics or ratings. The algorithm calculates the similarity between users or items using distance metrics, such as Euclidean distance or cosine similarity.

Once the nearest neighbors are identified, KNN can recommend items that these neighbors have

rated highly, effectively leveraging the collective preferences of similar users.

In recommendation system, the KNN algorithm can be employed to recommend movies to users based on the ratings provided by other users with similar tastes. To implement KNN, first, you will need to construct a user-item interaction matrix where rows represent users and columns represent movies. Each cell in the matrix contains the rating given by a user to a movie. To find similar users or movies, you can compute the distance between the rows (users) or columns (movies) in the matrix using the following formula for Euclidean distance:

**𝑑(𝑢,𝑣)=**

#### Where:

* 𝑑(𝑢,𝑣) is the distance between user 𝑢 and user v,
* 𝑢𝑖 and 𝑣𝑖 are the ratings of the two users for the ith movie,
* 𝑛 is the total number of movies rated by both users.

**Chapter-4**

**SYSTEM DESIGN**

### System Study

A movie recommendation system utilizing collaborative filtering is designed to suggest films to users based on the preferences and behaviors of similar users. This approach leverages the collective data of user interactions, such as ratings or viewing histories, to identify patterns and correlations among users and items. Collaborative filtering can be implemented through user- based or item-based methods. User-based collaborative filtering recommends movies by finding users with similar tastes and suggesting films that these similar users enjoyed. Conversely, item- based collaborative filtering focuses on the relationships between movies, recommending films that are similar to those a user has previously rated highly.

The effectiveness of collaborative filtering hinges on the availability of sufficient user data, which can pose challenges, particularly with new users or movies due to the cold start problem. To enhance recommendation accuracy, hybrid approaches that integrate content-based filtering techniques can be employed, combining the strengths of both methods. Advanced techniques, including matrix factorization and deep learning, further improve the system's ability to deliver personalized recommendations by uncovering latent factors in the user-item interaction data.

Overall, collaborative filtering remains a cornerstone of many contemporary recommendation systems, enhancing user experiences by providing tailored suggestions in the vast landscape of movie content.

### Benefits

Collaborative filtering in movie recommendation systems offers significant benefits that enhance user experience and engagement. One of the primary advantages is its ability to provide personalized recommendations tailored to individual tastes. By analyzing patterns and preferences among similar users, the system can suggest films that align closely with a user's interests, leading to increased satisfaction. Additionally, collaborative filtering adapts dynamically to user interactions, allowing the recommendations to evolve based on real-time behavior. This ensures that users receive relevant suggestions that reflect their current preferences, ultimately encouraging them to explore more content.

Another key benefit of collaborative filtering is its capacity to facilitate the discovery of new movies that users may not have encountered otherwise. By leveraging the collective insights of a diverse user base, the system can recommend films that are popular among users with similar tastes, broadening the viewing experience and introducing users to a wider array of content.

Furthermore, collaborative filtering does not rely heavily on detailed metadata about movies, making it easier to implement, especially in cases where such information is limited. This combination of personalized and diverse recommendations not only enhances user engagement but also fosters loyalty, as users are more likely to return to platform that consistent understands and meets their preferences.

### 4.1.2Different Types:

Recommendation systems can be classified into several types, each leveraging different methods to provide tailored suggestions to users. **Content-based filtering** is one of the foundational approaches, focusing on the characteristics of items and user preferences. In this method, recommendations are made based on the attributes of the items themselves, such as genre, director, or cast in the case of movies. For instance, if a user enjoys action films featuring a specific actor, the system will recommend other action films with similar characteristics. The advantage of content-based filtering is its ability to provide recommendations even for new items that lack extensive user interaction data, making it particularly useful in scenarios where item metadata is rich.

**Collaborative filtering** is another popular approach that relies on the behavior and preferences of users to generate recommendations. This method operates under the assumption that users with similar tastes will rate or engage with items similarly. Collaborative filtering can be divided into two subtypes: user-based and item-based. **User-based collaborative** filtering suggests items based on the preferences of similar users; for example, if User A and User B have rated several movies similarly, the system may recommend highly rated films from User B to User A. In contrast, **item- based collaborative filtering** focuses on the relationships between items themselves, recommending films that are popular among users who also liked the films that a target user has previously enjoyed. This approach excels in identifying trends and patterns across a large user base, leading to diverse and relevant recommendations.

Hybrid systems combine multiple recommendation techniques to enhance accuracy and user satisfaction. By integrating content-based and collaborative filtering approaches, hybrid systems can mitigate common challenges, such as the cold start problem, which occurs when new users or items lack sufficient interaction data. For example, a movie recommendation system might leverage user ratings, as well as item metadata such as genre, cast, and synopsis, to provide a more comprehensive set of recommendations. Other methods, like knowledge-based and demographic- based systems, focus on explicit user preferences or demographic information, tailoring suggestions based on specific criteria. Context-aware systems take this a step further by considering additional contextual factors, such as time and location, to ensure that recommendations are not only relevant but also timely. Together, these diverse types of recommendation systems work to create a rich, engaging user experience in various applications, from streaming services to e-commerce platforms.

### 4.1.3Challenges a Recommendation System Face

#### Cold Start Problem:

Recommendation systems struggle to make accurate suggestions for new users or items due to a lack of sufficient data. New users may not have rated any items yet, and new items may not have received any ratings or interactions,making it difficult for system to generate recommendations.

#### Data Sparsity:

In many cases, user interaction data is sparse, meaning that most users have only rated a small fraction of available items. This lack of data can make it challenging for collaborative filtering methods to find meaningful connections and provide relevant recommendations.

#### Scalability:

As the number of users and items increases, maintaining performance becomes a challenge. Recommendation algorithms need to be able to scale efficiently to handle large datasets without sacrificing response times or recommendation quality.

#### Bias and Fairness:

Recommendation systems can inadvertently reinforce existing biases present in the data or algorithm. If certain groups of users or items are overrepresented, the system may provide skewed recommendations, leading to unfair or biased outcomes.

#### Explainability:

As recommendation algorithms become more complex, particularly with the use of deep learning, explaining why a particular item was recommended becomes difficult. Users often want to understand the reasoning behind recommendations for trust and transparency.

#### Dynamic User Preferences:

Users’ preferences can change over time, making it challenging for systems to keep recommendations relevant. Maintaining up-to-date user profiles that accurately reflect current tastes is crucial but can be difficult.

### 4.2Data Pre-Processing

#### Remove Redundant, Missing, or Null Values

* + Clean data is essential for accurate analysis. Redundant records can skew results, while missing or null values may lead to incomplete recommendations. This step ensures a more reliable dataset by eliminating inconsistencies.

#### Filter the Data

* + Filtering helps focus on relevant data. For instance, you might exclude users with too few interactions or movies with very few ratings, which helps improve the quality and accuracy of the recommendations.

#### Identify Ambiguous Data

* + Ambiguities can arise from unclear or overlapping data, such as similar movie titles or multiple entries for the same movie. Resolving these ambiguities is crucial to avoid confusion in the recommendation output.

#### Include Relevant Information

* + Enriching the dataset with features like movie genres, release years, actor names, and director names provides contextual information that can enhance the recommendation algorithms. This additional context helps in better capturing user preferences and item similarities.

#### Handle Cold Start Problems

* + **For New Users**: Utilize demographic information (e.g., location, age, gender) and browsing behavior (e.g., device type) to generate initial recommendations. This approach can help tailor suggestions even before enough user interaction data is available.
  + **For New Movies**: Leverage movie attributes such as genre, cast, and crew to recommend new releases to relevant users. This allows the system to make informed suggestions despite a lack of historical interaction data.

#### Standardize Data Formats

* + Ensure consistency in data formats across the dataset. For example, dates should follow a single format, and categorical variables should be encoded uniformly. Standardization facilitates seamless integration and processing of data.

#### Create Interaction Matrices

* + Construct user-item interaction matrices to represent user preferences. This matrix serves as the foundation for collaborative filtering algorithms, allowing the model to identify patterns and relationships between users and items.

### 4.3Data Set Used

#### Movie Data

**Contents:** Basic information about the movies.

#### Columns:

movieId: Unique identifier for each movie.

title: Title of the movie.

genres: Genres associated with the movie (can be multiple). year: Year of release.



#### Rating Data

**Contents:** User ratings for movies.

#### Columns:

userId: Unique identifier for each user.

movieId: Unique identifier for each movie (links to Movie Data).

rating: The rating given by the user to the movie (typically on a scale of 1 to 5).

#### Movie Poster Data

**Contents:** URLs for the movie posters.

#### Columns:

movieId: Unique identifier for each movie. poster\_url: URL of the movie poster image.



### 4.4Module Description

1. **Library Imports:** The notebook starts with importing essential Python libraries like *pandas*

and *numpy.*

##### Code:

import pandas as pd import numpy as np

1. **Data Loading:** Two datasets are loaded:
   * *movies.csv*: Contains movie details such as title, year, genre, overview, runtime, and poster links.
   * *ratings.csv:* Contains user ratings of movies.

##### Code:

movies = pd.read\_csv('movies.csv') ratings = pd.read\_csv('ratings.csv')

1. **Data Preparation**: The ratings are reshaped into a pivot table format, where:
   * The index is *movieId.*
   * Columns represent *userId.*
   * Values are the movie *ratings.*

***Code:*** final\_dataset=ratings.pivot(index="movieId",columns="userId",values="Rating") print(final\_dataset)

final\_dataset.head() final\_dataset.fillna(0,inplace=True) final\_dataset.head()

### Output:

*userId 0 1 2 3 4 5 6 7 8 9 ... 1398 1399*

*movieId*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *0* | *8.1* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0 ... 0.0* | *0.0* |
| *1* | *0.0* | *8.2* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0 ... 0.0* | *0.0* |
| *2* | *0.0* | *0.0* | *8.1* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0 ... 0.0* | *0.0* |
| *3* | *0.0* | *0.0* | *0.0* | *5.4* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0 ... 0.0* | *0.0* |
| *4* | *0.0* | *0.0* | *0.0* | *0.0* | *4.6* | *0.0* | *0.0* | *0.0* | *0.0* | *0.0 ... 0.0* | *0.0* |

1. *rows × 1400 columns*

### 4.5Recommendation Function

This get\_recommendation function is designed to provide movie recommendations based on a movie's name, using the k-Nearest Neighbors (kNN) algorithm.

#### Function Components:

1. **Input:**
   * The function takes a movie name (movie\_name) as input, which is the movie you want recommendations based on.

#### Search for Movie:

* + It searches the movies DataFrame for movies whose titles contain the input string using:

#### Code

movie\_list = movies[movies['Movie'].str.contains(movie\_name)]

* + This finds all possible movies with similar names to the input and stores them in

movie\_list.

* + It checks if the list has any entries. If not, it returns "Movie not Found..".

#### Identify Movie Index:

* + If a movie is found, it retrieves the movieId from the first match using:

#### Code

movie\_idx = movie\_list.iloc[0]['movieId']

* + This movieId is used to locate the corresponding row in the final\_dataset, which is a matrix (likely the user-movie rating matrix created earlier in the notebook).

#### kNN for Recommendation:

* + The function uses k-Nearest Neighbors (kNN) to find similar movies. The kneighbors method returns two items:
  + distance: The distance between the input movie and its nearest neighbors.
  + indices: The indices of the nearest neighbors.

#### Code

distance, indices = knn.kneighbors(csr\_data[movie\_idx], n\_neighbors=11)

* + Here, csr\_data is a compressed sparse matrix of the rating data, used to store user-movie interactions efficiently.

#### Process Recommendations:

* + It zips the movie indices and distances, sorts them based on distance, and excludes the first one (which is the movie itself):

#### Code

rec\_movies\_indices = sorted(list(zip(indices.squeeze().tolist(), distance.squeeze().tolist())), key=lambda x: x[1])[:0:-1]

* + For each recommended movie, it retrieves the movieId from the final\_dataset, finds the corresponding movie title in the movies DataFrame, and appends it to the recommended\_movies list along with the distance from the original movie.

#### Return Recommendations:

* + The function creates a DataFrame (df) with the top 10 recommended movies and their distances from the original movie:

#### Code

df = pd.DataFrame(recommended\_movies, index=range(1, 11))

* + If no movie is found, it returns the message "Movie not Found..".

#### Example of how it works:

**Code:**

get\_recommendation(“Gabbar singh”)

#### Output:

**Movie Distance**

1. Victory 1.0
2. Screenplay of an Indian Love Story 1.0
3. Chal Mohana Ranga 1.0
4. Andha Oru Nimidam 1.0
5. Maavichiguru 1.0
6. Sri Krishnarjuna Vijayam 1.0
7. Gudachari No.1 1.0
8. Billa Ranga 1.0
9. Vicky Dada 1.0
10. Kondura (The Sage from the Sea) 1.

**CHAPTER-5**

**SYSTEM IMPLEMENTATION**

### Coding

#### #Importing Libraries:

import pandas as pd import numpy as np **#Loading Datasets:**

movies = pd.read\_csv("movie.csv")

ratings = pd.read\_csv("rating.csv")

#### #Displaying the Movies Data:

movies.head()

#### #Displaying the Ratings Data:

ratings.head()

#### #Creating a Pivot Table for the Dataset:

final\_dataset = ratings.pivot(index="movieId", columns="userId", values="Rating") print(final\_dataset)

#### #Viewing the Pivot Table:

final\_dataset.head()

**#Filling NaN Values in the Dataset:** final\_dataset.fillna(0, inplace=True) **#Visualizing the Distribution of Ratings:** import matplotlib.pyplot as plt

import seaborn as sns plt.figure(figsize=(10, 6))

sns.histplot(ratings['Rating'], bins=10, kde=True) plt.title('Distribution of Ratings') plt.xlabel('Rating')

plt.ylabel('No of movies')

plt.show()

#### #Converting the Dataset to a Sparse Matrix:

from scipy.sparse import csr\_matrix csr\_data = csr\_matrix(final\_dataset.values) final\_dataset.reset\_index(inplace=True) print(csr\_data)

#### #Using KNN for Collaborative Filtering:

from sklearn.neighbors import NearestNeighbors

knn = NearestNeighbors(metric='cosine', algorithm= 'brute', n\_neighbors = 20, n\_jobs=-1) knn.fit(csr\_data)

#### #Function to Get Movie Recommendations:

def get\_recommendation(movie\_name):

movie\_list = movies[movies['Movie'].str.contains(movie\_name)] print(movie\_list)

#### #Example Call for Recommendations:

get\_recommendation("Arya")

#### #Modified Movie Recommendation Function with Validation:

def get\_recommendation(movie\_name):

movie\_list = movies[movies['Movie'].str.contains(movie\_name)] if len(movie\_list):

print("Movies Found") else:

print("Movie not Found..")

**#Example Calls for Different Movies:**

get\_recommendation("Devara")

get\_recommendation("Rebel")

**#Fetching the Movie Index:**

def get\_recommendation(movie\_name):

movie\_list = movies[movies['Movie'].str.contains(movie\_name)] print(movie\_list)

if len(movie\_list):

movie\_idx = movie\_list.iloc[0]['movieId']

movie\_idx = final\_dataset[final\_dataset['movieId'] == movie\_idx].index[0] print(movie\_idx)

else:

print("Movie not Found..")

#### #Recommendation with KNN Neighbors:

def get\_recommendation(movie\_name):

movie\_list = movies[movies['Movie'].str.contains(movie\_name)] print(movie\_list)

if len(movie\_list):

movie\_idx = movie\_list.iloc[0]['movieId']

movie\_idx = final\_dataset[final\_dataset['movieId'] == movie\_idx].index[0] distance, indices = knn.kneighbors(csr\_data[movie\_idx], n\_neighbors=11)

rec\_movies\_indices = sorted(list(zip(indices.squeeze().tolist(), distance.squeeze().tolist())),

key=lambda x: x[1])[:0:-1]

recommended\_movies = [] for val in rec\_movies\_indices:

movie\_idx = final\_dataset.iloc[val[0]]['movieId']

idx = movies[movies['movieId'] == movie\_idx].index recommended\_movies.append({'Movie': movies.iloc[idx]['Movie'].values[0], 'Distance':val[1]})

df = pd.DataFrame(recommended\_movies, index=range(1, 11)) return df

else:

return "Movie not Found.

**Example Call for Recommendation:** get\_recommendation("Gabbar Singh")

#### Integrating with Gradio for Web App Interface:

import gradio as gr

def recommend\_movies(movie\_name):

df = get\_recommendation(movie\_name) if isinstance(df, pd.DataFrame):

return df.to\_string(index=False) else:

return "No recommendations found."

#### # Create Gradio Interface

app = gr.Interface( fn=recommend\_movies, inputs="text", outputs="text",

title="Movie Recommendation System", description="Enter a movie name to get recommendations"

)

#### # Launch the app

app.launch()

### Sample Screens:

#### Sample of the First 5 rows of movie.csv file being printed

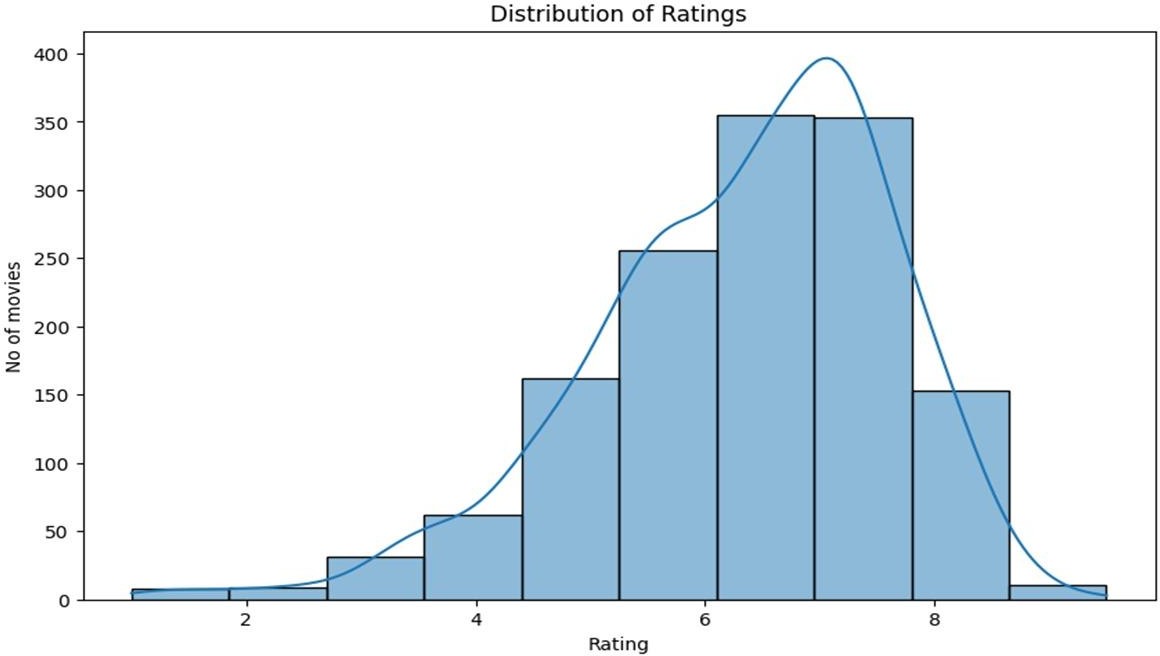
**movieId Movie Year Certificate Genre**

|  |  |  |  |
| --- | --- | --- | --- |
| 0 Bahubali: The Beginning | 2015 | UA | Action, Drama |
| 1 Baahubali 2: The Conclusion | 2017 | UA | Action, Drama |
| 2 1 - Nenokkadine | 2014 | UA | Action, Thriller |
| 3 Dhoom:3 | 2013 | UA | Action, Thriller |
| 4 Ra.One | 2011 | U | Action, Adventure, Sci-Fi |

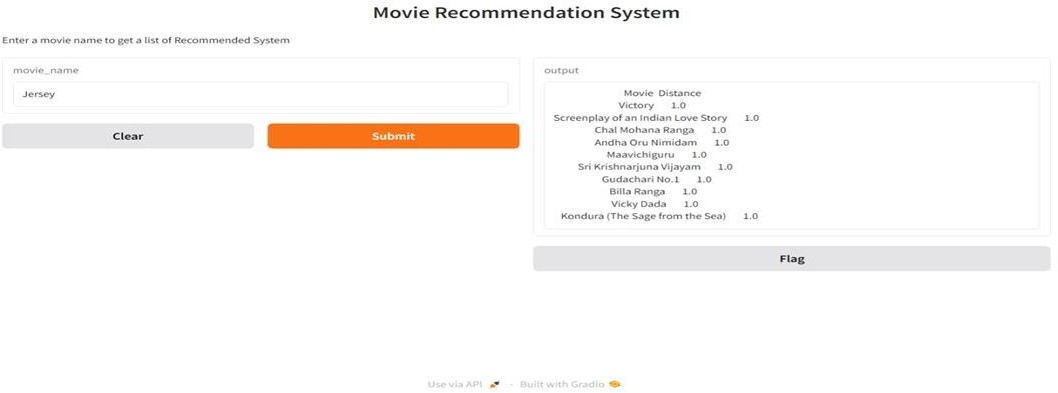
#### Sample of the First 5 rows of rating.csv file being printed

|  |  |  |  |
| --- | --- | --- | --- |
| **userId** | **movieId** | **Rating** | **No.of.Ratings** |
| 0 | 0 | 8.1 | 99114 |
| 1 | 1 | 8.2 | 71458 |
| 2 | 2 | 8.1 | 42372 |
| 3 | 3 | 5.4 | 42112 |
| 4 | 4 | 4.6 | 37211 |

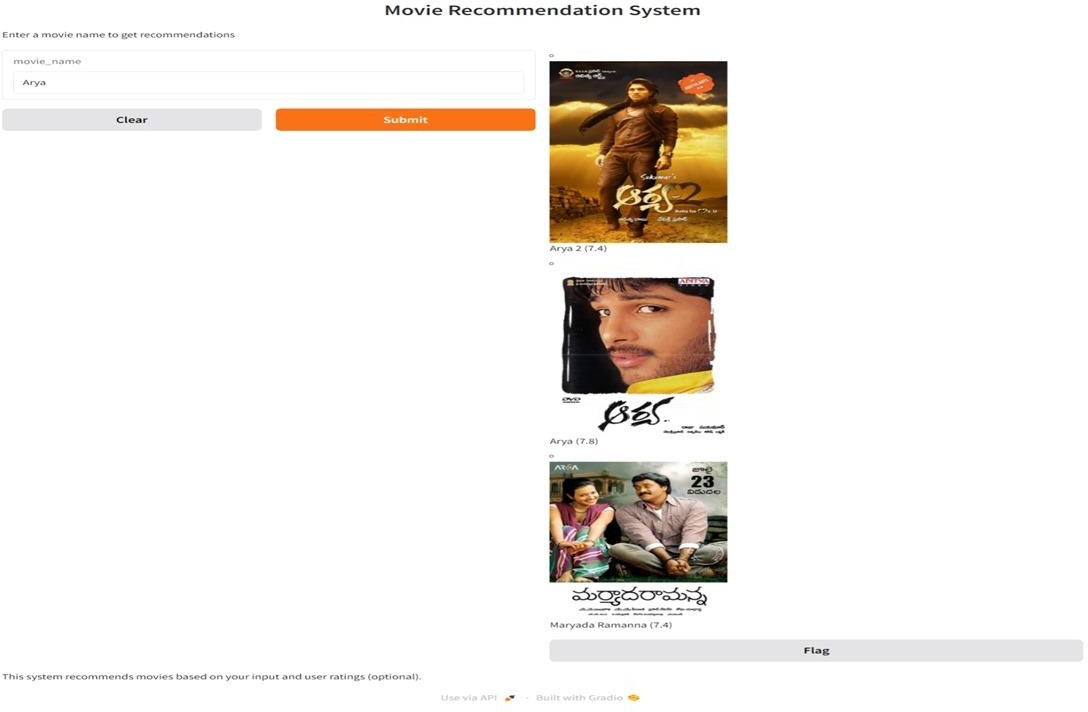
**Visualization of Dataset(rating.csv):**



**Recommendation Interface using Gradio Interface:**



**Final Recommendation Interface:**



**Chapter-6**

**RESULTS AND DISCUSSIONS**

### Results and Discussions:

The core result of this project centers around generating personalized movie recommendations based on users' historical ratings. The primary goal is to provide users with a list of films they are most likely to enjoy, derived from their past behavior and the preferences of similar users. By comparing a user's preferences with those of others who share similar tastes, the system can predict which movies a user has not yet rated but would likely appreciate. For instance, if a user frequently rates action films highly, the system will recommend other top-rated action movies that similar users enjoyed, culminating in a curated list of Top-N recommendations.

In addition to generating lists of recommended films, the system also predicts the ratings users might assign to movies they have not yet watched. This capability enhances the personalization of recommendations, allowing users to prioritize films based on predicted ratingsThis predictive feature is critical in refining the recommendation process and improving user satisfaction.

The system's underlying architecture includes a user-item matrix, or pivot table, which illustrates the ratings given by users to various movies. This matrix is essential for collaborative filtering, as it highlights the relationships between users and films, allowing the algorithm to detect patterns in ratings. Furthermore, analyzing vote counts—specifically, the number of users who rated each movie and the number of films each user has rated—enables the system to prioritize recommendations. Movies with higher user engagement can be weighted more significantly in the recommendation process, ensuring that the suggestions are based on robust data.

Finally, evaluating the system’s performance through accuracy metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Precision, and Recall is crucial for assessing its effectiveness. These metrics provide insights into how closely predicted ratings align with actual user ratings and how relevant the Top-N recommendations are. By implementing user feedback mechanisms, the system can also gauge user satisfaction and engagement, allowing for continuous improvements. Overall, this project successfully demonstrates the potential of data- driven collaborative filtering to enhance user experiences in the realm of movie recommendations.

**Chapter-7**

**CONCLUSION**

### CONCLUSION:

In conclusion, the movie recommendation system developed in this project effectively demonstrates the power of collaborative filtering to provide personalized movie suggestions tailored to individual user preferences. By analyzing historical ratings and leveraging patterns in user behavior, the system is capable of generating meaningful recommendations that enhance the user experience. The ability to predict ratings for unwatched movies further refines the recommendations, ensuring that users are presented with content that aligns with their tastes.

Throughout the project, we explored various metrics to evaluate the system's performance, including RMSE, MAE, Precision, and Recall. These metrics reveal the accuracy and relevance of the recommendations, highlighting the system's effectiveness in delivering valuable insights to users. The user-item matrix and vote counts provide a solid foundation for collaborative filtering, demonstrating the system's readiness to handle diverse datasets and detect intricate patterns in user preferences.

Moreover, the implementation of user feedback mechanisms is crucial for continuously improving the recommendation engine. By allowing users to rate their recommendations, we can gather data that informs future iterations of the model, enhancing its predictive capabilities and overall effectiveness. This feedback loop not only boosts user satisfaction but also fosters engagement, encouraging users to interact more with the system.

Ultimately, this project showcases the significant impact that a well-designed recommendation system can have on user engagement in the entertainment sector. As we move forward, further enhancements—such as integrating content-based filtering or exploring hybrid models—could lead to even more accurate and diverse recommendations, ensuring that users enjoy a tailored cinematic experience. The insights gained from this project lay a strong foundation for future advancements in personalized recommendation systems.

### Future Enhancement

Future enhancements to the movie recommendation system can significantly improve its effectiveness and user engagement by integrating content-based filtering with collaborative filtering to consider both user ratings and movie attributes like genre and director. Implementing user clustering techniques and real-time recommendations can adapt to user behavior dynamically, while a robust feedback loop that allows users to provide qualitative insights will refine the model further. Exploring advanced algorithms, such as deep learning, and incorporating sentiment analysis of user reviews can enhance predictive accuracy. Additionally, personalized user interfaces, cross-platform recommendations, and the integration of social media data will create a more comprehensive experience, ensuring diversity and novelty in suggestions to keep recommendations fresh and engaging.

**Chapter-8**

**REFERENCES**

### References:

1. Ricci, F., Rokach, L., & Shapira, B. (2015). Recommender Systems Handbook.Springer. This comprehensive handbook covers various techniques and algorithms used in recommendation systems, including collaborative filtering, content-based filtering, and hybrid methods.
2. Bennett, J., & Lanning, S. (2007). The Netflix Prize. In Proceedings of the 2007 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.This paper discusses the Netflix Prize competition and the collaborative filtering techniques used to improve movie recommendations.
3. Koren, Y. (2009). Matrix Factorization Techniques for Recommender Systems.In Proceedings of the 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. This paper presents matrix factorization methods, which are widely used in collaborative filtering approaches for recommendation systems.
4. Schafer, J. Ben, Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative Filtering Recommender Systems. In The Adaptive Web: Methods and Strategies of Web Personalization.Springer. This paper provides an overview of collaborative filtering techniques and their application in recommender systems.
5. “Recommendation Systems: An Overview” by IBM Cloud Education. A comprehensive introduction to recommendation systems, covering different types of recommendations and the algorithms behind them. Available at [IBM Cloud Education](https://[www.ibm.com/cloud/learn/recommendation-systems).](http://www.ibm.com/cloud/learn/recommendation-systems))
6. “Building a Movie Recommendation System using Collaborative Filtering” on Towards Data Science. A practical guide that walks through the process of creating a movie recommendation system using collaborative filtering techniques. Available at [Towards Data Science](https://towardsdatascience.com/building-a-movie-recommendation-system-using- collaborative-filtering-bb9f1d3a5832).
7. Apache Spark MLlib is a scalable machine learning library that includes collaborative filtering algorithms. Documentation can be found on the [Apache Spark website](https://spark.apache.org/docs/latest/ml-collaborative-filtering.html).
8. Surprise is a Python library for building and analyzing recommender systems that deal with explicit rating data. Documentation is available on the [Surprise GitHub page](https://surpriselib.com/).
9. Adomavicius, G., & Tuzhilin, A. (2005). Toward the Next Generation of Recommender

Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering, 17(6), 734-749. This paper reviews the current state of recommendation systems and discusses potential advancements in the field.

1. Zhang, Y., & Chen, L. (2018). A Survey on Collaborative Filtering-Based Recommender Systems for Movie Recommendations. IEEE Transactions on Emerging Topics in Computing. This paper provides a comprehensive survey of collaborative filtering systems.