

A PROJECT REPORT

A Semantic Approach to solve Data Sparsity, Scalability and Cold Start problems in Collaborative Filtering

Submitted in partial fulfilment of the requirements for the award of degree

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

by

P NIKHIL CHANDRA (160119733037)

UDAY KIRAN REDDY NALLAGANDLA (160119733056)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)**

(Affiliated to Osmania University; Accredited by NBA and NAAC, ISO 9001:2015 Certified Institution), GANDIPET, HYDERABAD – 500 075

[2022-2023]



**CHAITANYA BHARATHI
INSTITUTE OF TECHNOLOGY (A)**
Kokapet(Village), Gandipet, Hyderabad, Telangana-500075. www.cbit.ac.in



COMMITTED TO
RESEARCH,
INNOVATION AND
EDUCATION

44
years

CERTIFICATE

This is to certify that the project work entitled “**A Semantic Approach to solve Data Sparsity, Scalability and Cold Start problems in Collaborative Filtering**” is the bonafide work carried out by

P. NIKHIL CHANDRA (160119733037)

UDAY KIRAN REDDY NALLAGANDLA (160119733056)

the students of B.E.(CSE) of Chaitanya Bharathi Institute of Technology(A), Hyderabad, affiliated to Osmania University, Hyderabad, Telangana (India) during the academic year 2022-2023, submitted in the partial fulfilment of the requirements for the award of the degree in **Bachelor of Engineering (Computer Science and Engineering)** and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Project Guide

Smt. T. Suvarna Kumari

Assistant Professor,
Department of CSE,
CBIT, Hyderabad

Head of the Department

Dr. M. Swamy Das

Professor and Head of
Department of CSE,
CBIT, Hyderabad

Place: Hyderabad

Date:

DECLARATION

We hereby declare that the project report entitled “**A Semantic Approach to solve Data Sparsity, Scalability and Cold Start problems in Collaborative Filtering**” submitted for the B.E (CSE) degree is our original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any similar titles.

P. NIKHIL CHANDRA (160119733037)

UDAY KIRAN REDDY NALLAGANDLA (160119733056)

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the project.

We would like to express our sincere gratitude and indebtedness to our Project Guide Smt. T. Suvarna Kumari, Assistant Professor, Dept. of CSE who has supported us throughout our project with patience and knowledge.

We are also thankful to Head of Department, Dr. M. Swamy Das, for providing excellent infrastructure and a conducive atmosphere for completing this project successfully.

We are also extremely thankful to our Project Coordinator, Dr. S. China Ramu, Professor, Dept. of CSE, for his valuable suggestions and interest throughout the course of this project.

We convey our heartfelt thanks to lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

P. Nikhil Chandra (160119733037)

Uday Kiran Reddy Nallagandla (160119733056)

ABSTRACT

Collaborative filtering is a popular technique used in recommender systems to provide personalized recommendations to users. However, collaborative filtering suffers from data sparsity, scalability, and cold start problems. Data sparsity occurs when the user-item matrix is sparse, and most users have rated only a small fraction of items. Scalability is a concern when the size of the user-item matrix grows, making it difficult to perform computations efficiently. The cold start problem arises when a new user or item enters the system, and there is not enough data to generate accurate recommendations.

To address these challenges, we built a single model that employs singular value decomposition to resolve all three issues. The proposed method decomposes the user-item matrix into a low-dimensional latent space, which captures the underlying semantic structure of the data. The latent space is used to represent users and items, and recommendations are generated based on the similarity between the latent representations of users and items. The proposed approach improves scalability by reducing the dimensionality of the data, making it easier to perform computations efficiently. Additionally, it mitigates data sparsity by capturing the latent factors that influence user-item interactions, even when there is limited data available. The proposed approach also addresses the cold start problem by leveraging the latent space representation. For new users, the method employs a popularity-based method to recommend high rated movies. Overall, the results of our work showed better results as compared to other SVD model recommendation systems.

LIST OF FIGURES

Figure 2.1.1	Recommendation System Hierarchy
Figure 2.2.1.1	Evaluation of memory based Collaborative Techniques
Figure 2.2.3.1	Process Model of the Study
Figure 2.2.4.1	The Process of the Hybrid Collaborative Filtering Algorithm
Figure 3.1.1	Flow Diagram of the system
Figure 3.2.1	Data Flow Diagram for data pre-processing
Figure 3.2.2	Data Flow Diagram for singular value decomposition
Figure 3.3.1	UML class diagram for the proposed system
Figure 3.4.1	Sequence Diagram for the proposed system
Figure 3.5.1	UML use case diagram for the proposed system
Figure 3.6.1	Activity Diagram for the proposed system
Figure 3.7.1	User-item matrix
Figure 3.7.2	Matrix $M = R^T R$
Figure 3.7.3	Left Singular Matrix U
Figure 3.7.4	Matrix $M = R R^T$
Figure 3.7.5	Right Singular Matrix V
Figure 3.7.6	Singular Value Matrix Σ
Figure 3.7.7	Decomposed matrices
Figure 5.1	Ratings Data Frame with 100k rows
Figure 5.2	Item matrix with top 100 latent features
Figure 5.3	Sparse user-item matrix before applying SVD.
Figure 5.4	Item matrix after applying SVD
Figure 5.5	Popularity-based Recommendations
Figure 5.6	Top movies based on a given movie
Figure 5.7	Output for given movies and their ratings.
Figure 5.8	RMSE and MAE of the system
Figure 5.9	Home Page
Figure 5.10	Search for a movie
Figure 5.11	Rate the movie
Figure 5.12	Recommended Movies

TABLE OF CONTENTS

		Page. No
	Title Page	I
	Certificate of the Guide	II
	Declaration of the Student	III
	Acknowledgement	IV
	Abstract	V
	List of Figures	VI
1.	INTRODUCTION	1
	1.1 Problem Definition and Objectives	1
	1.2 Outline of the Results	2
	1.3 Scope of the Project	3
	1.4 Motivation	3
	1.5 Organization of the Report	4
2.	LITERATURE SURVEY	4
	2.1 Introduction to Problem Domain Terminology	5
	2.2 Related Works	7
	2.2.1 Evaluation of Hybrid Collaborative Filtering approach with Context-Sensitive Recommendation System	7
	2.2.2 Accuracy Analysis of Recommendation System using Singular Value Decomposition	8
	2.2.3 User Profile Feature-Based Approach to Address the cold Start Problem in Collaborative Filtering for Personalized movie Recommendation.	8
	2.2.4 A Hybrid Collaborative Filtering Recommendation Algorithm for Solving the Data Sparsity	9
	2.2.5 Model and Popularity Based Recommendation System - A Collaborative Filtering Approach	9
	2.2.6 Movie Recommendation System Employing the User-Based CF in Cloud Computing	10
	2.3 Tools and Technologies used.	10
	2.3.1 Python 3.9	10
	2.3.2 Pandas	11
	2.3.3 Google Colab	12
	2.3.4 Surprise	12
	2.3.5 Django Web Framework	13
3.	DESIGN OF THE PROPOSED SYSTEM	14
	3.1 Flow Diagram	15
	3.2 Data Flow Diagram	15
	3.3 Class Diagram	17
	3.4 Sequence Diagram	18
	3.5 Use Case Diagram	20

	3.6 Activity Diagram	21
	3.6 Theoretical Foundation	21
4.	IMPLEMENTATION OF THE PROPOSED SYSTEM	25
	4.1 SVD Algorithm	25
	4.2 Data Pre-processing	26
	4.3 Model Preparation	27
	4.4 Dataset Analysis	28
	4.5 Testing	29
5.	RESULTS	30
6.	CONCLUSION AND FUTURE SCOPE	37
7.	REFERENCES	38
8.	APPENDIX	40

CHAPTER 1

INTRODUCTION

1.1 Problem Definition and Objectives

Considering the growing rate of advancement in information technology and social networks, the volume of digital Internet-generated data has increased dramatically in recent years, and the big data revolution is now in full swing. It is increasingly and more prominent as data volumes rise. It's challenging and tricky for people to locate the information they need from the vast volume of information. Currently, the recommender system is capable of playing the highest practical significance. In consideration of the user data and user history the recommendation system can predict efficiently based on behavioural data. Recommender systems are now used by entertainment-focused businesses to improve customer interaction and experience. These internet streaming services make content, movie, and video suggestions based on user interests, which boosts overall customer satisfaction results. It identifies the user's preferences, making things more unique content for the user. Users might be intrigued by and substantially reduce the expense of obtaining the desired information. In the study of recommender systems, people are working to improve the recommender algorithms. Now, the most common recommendation algorithms fall into the following categories: collaborative Filtering-based recommendation, content-based recommendation, and hybrid recommendation. Collaborative filtering has been used for many years to improve and increase the accuracy of recommendations. Although numerous strategies have been employed, the problems in the recommender system still exist and top ones being data sparsity, scalability, and cold start. The cold start problem occurs when the system is unable to form any relation between users and items for which it has insufficient data. Data sparsity occurs when it becomes difficult in finding sufficient reliable similar users since in general the active users only rated a small portion of items. The scalability problem occurs in a real time environment where the number of users is high, and the data being processed is more.

There is a need to build an efficient system which can solve the cold start, scalability, and data sparsity problems. We built a single model that employs singular value decomposition to resolve all three issues. By singular value decomposition (SVD), the space dimension is reduced from N to K dimensions. The factors are taken out of the factorization by breaking down the high-level (user-item-rating) matrix into three additional matrices. The three objectives of the work are as follows:

- (i) Whenever a new user signs up in the system, a cold-start issue occurs. It is a little more challenging for the model to recommend a movie that the user might like because there is no prior information about their preferences.
- (ii) Data sparsity is a result of two main factors: (a) users probably cannot view each and every movie and (b) every now and then, even after viewing a movie, users will decide not to focus on providing feedback.
- (iii) Scalability is an issue that is faced by systems having huge amounts of data to process. When the number of users grows, the ratings data grow exponentially.

Simple collaborative filtering systems use the ratings in the user-item vector to generate suggestions[8]. Since these ratings are crucial, their absence causes a significant amount of difficulty for a basic Collaborative Filtering algorithm.

1.2 Outline of the Results

This work mainly focuses on solving the three problems which are faced by traditional collaborative filtering – data sparsity, scalability and cold start. Singular Value Decomposition is a matrix factorization method that computes latent features of the user-item matrix using complex mathematical expressions. As the user-item matrix is sparse, we use the ratings given by user to predict the ratings of other movies which are not rated by the user which solves the problem of data sparsity. As Singular Value Decomposition is also a dimensionality reduction technique, we use this to solve our scalability issue. The user-item matrix is decomposed into three matrices which hold user and item profile data. We use these matrices instead of the bigger user-item matrix which is more efficient than traditional collaborative filtering algorithms in terms of scalability. The cold start problem is solved by using a popularity-based approach and by using the SVD. When there is no item data, we use popularity-based method and when user gives a single movie as input, we use SVD to find similar items using cosine similarity method.

The final results are calculated using performance metrics like Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). The cosine similarity metric is used to evaluate similarity between the movies.

1.3 Scope of the Project

Collaborative Filtering is the branch of recommendation systems where the research is intense. It is a complex area where huge amounts of data are computed to find the perfect recommendations. Nowadays, recommendation systems are used in many platforms like e-commerce to music and movie streaming platforms. There is a need for optimization in collaborative filtering algorithms. This optimization can earn millions of dollars of profits for the companies. The traditional collaborative filtering consists of computing user-item matrix and performing mathematical calculations on it. There is a need for an efficient system which is capable of solving the three problems – data sparsity, scalability and cold start, build a system which is better and compatible than the traditional approach. Singular Value Decomposition has the ability to solve all the three problems and also perform better in harsh conditions where the availability of the data is less.

1.4 Motivation

The motivation behind building a movie recommendation system using Singular Value Decomposition (SVD) is to provide users with personalized movie recommendations based on their past preferences. With the increasing popularity of movie streaming services and the abundance of movies available, users often face difficulties in finding movies that match their tastes. A recommendation system based on SVD can help to solve this problem by analyzing user ratings data and providing personalized recommendations for each user. Moreover, the use of SVD can address the issues of data sparsity, scalability, and cold start, which are common challenges in building recommendation systems. One of the main motivations behind this work is to improve the performance and usability of recommendation systems in a wide range of applications. Recommendation systems have become increasingly important in many areas, including e-commerce, social media, and entertainment. By providing personalized recommendations, these systems can help to improve user satisfaction and engagement. However, the performance of these systems is often limited by data sparsity, scalability, and cold start issues. By using SVD to build a recommendation system for movies, we can address these challenges and provide more accurate and relevant recommendations to users. This work has the potential to be applied to various other domains and help to improve the performance and usability of recommendation systems in these areas.

1.5 Organization of the report

This report started off by a brief introduction about the problem and our approach in solving it. In next chapter we get to know in detail about the hierarchy of recommendation systems and know more about collaborative filtering. We also discuss about some related works, existing solutions in solving the problem and the tools and technologies used in our work.

In the upcoming chapters, we explain in detail the design of our proposed system using various UML diagrams and flow diagrams. This helps in understanding the proposed methodology better. After the design of the system, we discuss in detail the implementation of the proposed system. We talk about various modules in the proposed system and their theoretical foundation. We also explain how SVD works by talking a user-item matrix as an example. We talk about the dataset and the testing process after that. At the end we discuss the results and conclusions of our work.

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction to Problem Domain Terminology

A recommendation system is an information filtering system that predicts user preferences and provides personalized suggestions to users for items or services that they might be interested in. The system uses various algorithms and techniques to analyse user data and generate recommendations based on that analysis[p].

There are mainly three types of recommendation systems:

1. Content-based recommendation systems: This type of recommendation system recommends items to users based on their previous preferences or ratings for similar items. The system analyses the characteristics of the items that the user has already liked or rated highly and then recommends items with similar characteristics. For example, if a user has rated several movies highly that are action movies, the system will recommend other action movies to the user.
2. Collaborative filtering recommendation systems: This type of recommendation system recommends items to users based on the preferences of other users who have similar tastes[11]. The system analyses the ratings or preferences of similar users and recommends items that those users have liked or rated highly. Collaborative filtering can be further divided into two types:
 - a. User-based collaborative filtering: In this type of filtering, the system recommends items to a user based on the preferences of other users who have similar tastes.
 - b. Item-based collaborative filtering: In this type of filtering, the system recommends items to a user based on the preferences of other users who have liked or rated similar items[14].
3. Hybrid recommendation systems: This type of recommendation system combines the above two types of recommendation systems to provide more accurate and personalized recommendations to users [12]. Hybrid recommendation systems can use various algorithms and techniques to combine the results of content-based and collaborative filtering recommendation systems.

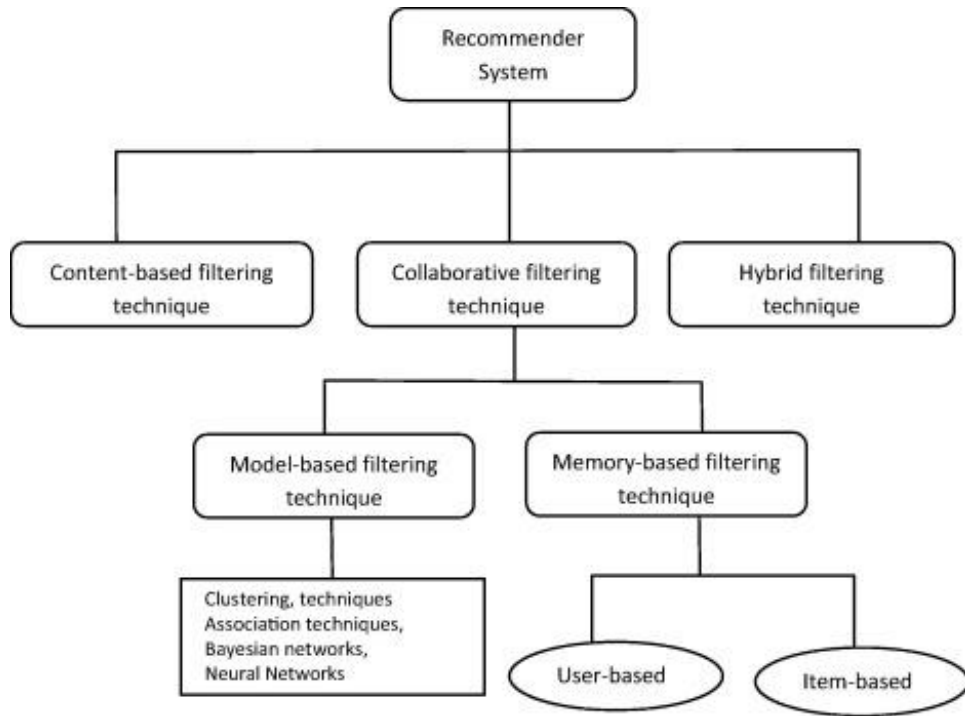


Fig 2.1.1 Recommendation System Hierarchy

SVD or Singular Value Decomposition is a matrix factorization technique used to reduce the dimensionality of data. It is widely used in various fields, including recommendation systems. It is used as a collaborative filtering technique to predict user ratings for items. The idea is to decompose the user-item rating matrix into three matrices, i.e., U , Σ , and V , where U represents the user matrix, Σ represents the singular value matrix, and V represents the item matrix.

$$A_{[m \times n]} = U_{[m \times r]} \Sigma_{[r \times r]} (V_{[n \times r]})^T$$

The matrix factorization of the user-item rating matrix assumes that each user's ratings can be represented as a combination of a small number of latent factors that represent user preferences, and each item can be represented as a combination of a small number of latent factors that represent item characteristics. The SVD technique factorizes the user-item rating matrix into U , Σ , and V matrices, where U and V are orthogonal matrices and S is a diagonal matrix with singular values. The diagonal elements in Σ represent the importance of each latent factor, and the higher the value, the more important the latent factor is in predicting the user ratings. Once the user-item rating matrix is decomposed into U , Σ , and V matrices, the system can use this factorization to predict the missing ratings for each user-item pair. The predicted rating can be calculated as the dot product of the corresponding rows in the U and V matrices, which represents the user and item latent factor vectors, respectively.

2.2 Related Works

Recommender systems are used by organizations like Netflix, Amazon, and others to encourage their customers in choosing the best item or movie for them [15]. The recommender system handles the abundance of data by filtering the most crucial information based on the information provided by a user and other aspects that consider the user's opinion and interest. It determines whether a user and an item are compatible and then assumes that they are identical to make recommendations. Many research works in collaborative filtering try to improve the ability of the system to recommend items accurately and some even proved effective. Some of them are :

2.2.1 Evaluation of Hybrid Collaborative Filtering Approach with Context-Sensitive Recommendation System

This work [1] involves the evaluation of a memory-based recommendation system, and the evaluation was carried out using three predictive methods, user collaborative filtering, item collaborative filtering, and hybrid. It uses Pearson correlation coefficient to determine which method has the best correlation coefficient between the predicted and actual values. The test was carried out with adding contextual information and without contextual information.

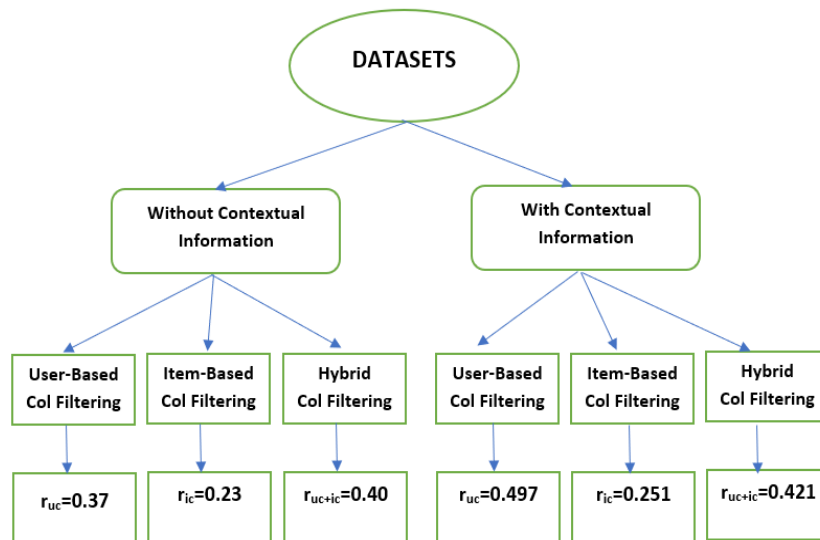


Fig 2.2.1.1 Evaluation of memory based Collaborative Techniques

2.2.2 Accuracy Analysis of Recommendation System Using Singular Value Decomposition

Singular Value Decomposition (SVD) is a most powerful algorithm to predict unknown ratings by reducing the less significant features. Before applying SVD on utility matrix, all unknown ratings should be filled with some initial values. This paper [2] focuses to generate two predictive matrixes by assigning two different initial values, where one is Zero and other is replacing unknown values with average item rating and then subtracting corresponding average user rating from the values.

The author explained SVD clearly by considering a rank 5 matrix. The second approach to feed unknown values shows better performance. But this was taken over a small matrix of rank 5 but in real world scenario the rank is much higher and the RMSE is also higher.

2.2.3 User Profile Feature-Based Approach to Address the Cold Start Problem in Collaborative Filtering for Personalized Movie Recommendation

The authors in [6] apply the relationship of user feature-scores derived from user-item interaction via ratings to optimize the prediction algorithm's input parameters used in the recommender system to improve the accuracy of predictions with less past user records. This addresses a major drawback in collaborative filtering, the cold start problem by showing an improvement of 8.4% compared to the base collaborative filtering algorithm. The feature score is calculated from user-item interaction data. For each rating the user rates, the genre's feature score is calculated. And finally, Pearson similarity is used between user profile features. The RMSE values of different systems are around 1.1.

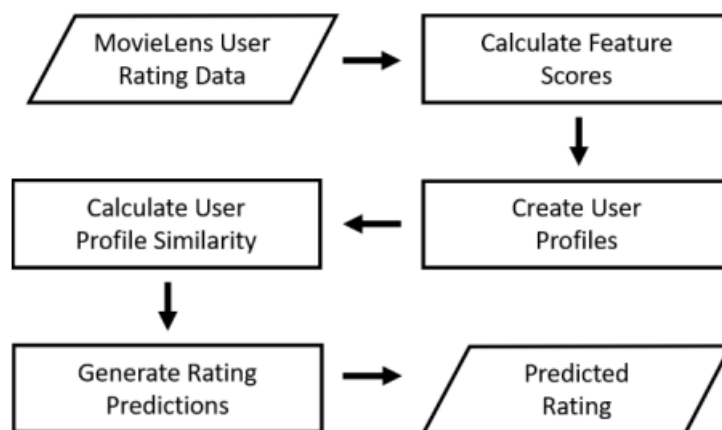


Fig 2.2.3.1 Process Model of the Study

2.2.4 A Hybrid Collaborative Filtering Recommendation Algorithm for Solving the Data Sparsity

This work [7] proposed a new recommendation algorithm that combines collaborative filtering and content-based filtering to address the data sparsity problem in recommender systems. The proposed algorithm first uses content-based filtering to identify a set of similar items to the target item, and then performs collaborative filtering on the identified set of items to generate recommendations. The algorithm also considers user preferences and item popularity to improve the accuracy of the recommendations.

The evaluation of the proposed algorithm was conducted on the Movie Lens dataset, and the performance was measured using precision, recall, and F1-score. The experimental results showed that the proposed algorithm outperformed several state-of-the-art collaborative filtering algorithms in terms of recommendation accuracy. MAE values of the system are around 0.78 to 0.84. This method also solves the cold start problem to an extent by recommending highly predicted movies.

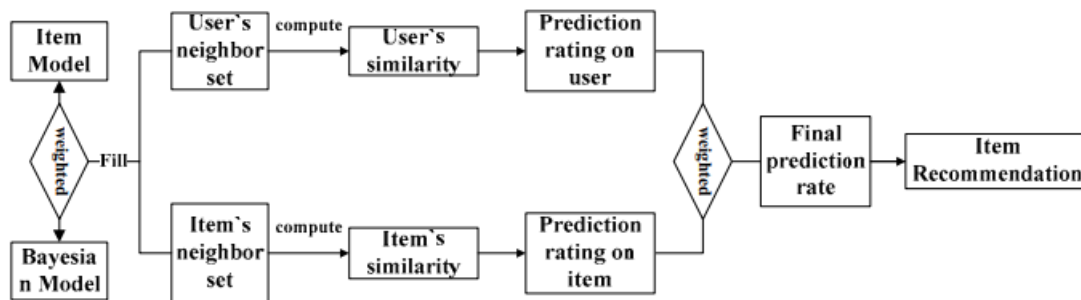


Fig 2.2.4.1 The Process of the Hybrid Collaborative Filtering Algorithm

2.2.5 Model and Popularity Based Recommendation System - A Collaborative Filtering Approach

This work [3] presents a recommendation system that combines collaborative filtering with item popularity and item models to generate accurate recommendations. The proposed system first uses collaborative filtering to identify a set of similar users, and then considers the popularity and model of items to recommend. The popularity of an item is calculated based on the number of times it has been rated, while the model of an item is determined based on the category it belongs to.

The evaluation of the proposed system was conducted on the Movie Lens dataset, and the performance was measured using precision, recall, and F1-score. The experimental results showed that the proposed system outperformed several state-of-the-art collaborative filtering algorithms in terms of recommendation accuracy. Overall, the proposed system is a promising approach that combines different factors to generate accurate recommendations in a real-world setting.

2.2.6 Movie Recommendation System Employing the User-Based CF in Cloud Computing

This work [4] proposes a movie recommendation system that utilizes user-based collaborative filtering (CF) in a cloud computing environment. The proposed system employs MapReduce and Hadoop technologies to analyse large-scale movie rating data and generate recommendations. The user-based CF algorithm is used to identify similar users and recommend movies based on their ratings. The system also incorporates a dimensionality reduction technique called Singular Value Decomposition (SVD) to improve recommendation accuracy.

The evaluation of the proposed system was conducted on the Movie Lens dataset, and the performance was measured using Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). The experimental results showed that the proposed system achieved better performance compared to a traditional user-based CF algorithm and a matrix factorization method. Overall, the proposed system provides an efficient and scalable solution for generating movie recommendations in a cloud computing environment.

2.3 Tools/Technologies used.

We have used a wide range of tools and technologies in our work. We built our model in Python programming language using Google Colab. We used many tools like Pandas, Surprise, NumPy etc. The User Interface of our work is done using Django Web framework and we used Pickle files to use our model. Let us see in detail the tools and technologies used.

2.3.1 Python 3.9

Python is a high-level, interpreted programming language that has become a popular choice among data scientists, machine learning engineers, and other professionals in the field of data science. One of the reasons for Python's popularity in machine learning is its simple syntax,

which allows developers to write concise and readable code. This, coupled with a large community of developers who have contributed to the development of powerful open-source libraries, has made Python an ideal choice for developing machine learning applications.

Python is also highly flexible and can be used for a range of machine learning tasks, including data pre-processing, feature engineering, model training, and evaluation. Python offers a range of powerful libraries such as NumPy, Pandas, Scikit-learn, TensorFlow, and Keras, which provide an extensive set of tools for building and deploying machine learning models. The availability of these libraries has made it easy for developers to implement complex machine learning algorithms and models with relative ease, enabling faster development times and improved accuracy of models.

2.3.2 Pandas

Pandas is a popular open-source library in Python that provides powerful data manipulation and analysis tools for data scientists, machine learning engineers, and other professionals working with tabular data. It is built on top of the NumPy library and provides easy-to-use data structures such as DataFrame and Series that allow for efficient data manipulation, cleaning, and preparation. Pandas also provides a wide range of functions for data exploration, aggregation, and visualization, making it an essential tool for working with data in Python. Pandas is particularly useful in machine learning due to its ability to handle large datasets and its seamless integration with other popular machine learning libraries such as Scikit-learn and TensorFlow. It can be used for data pre-processing and feature engineering, two important steps in machine learning workflows that involve cleaning, transforming, and selecting relevant features from the dataset. With pandas, users can easily manipulate, filter, and transform data to prepare it for training machine learning models.

In addition to data pre-processing, pandas can also be used for exploratory data analysis (EDA) and visualization. With built-in plotting functions, users can create a range of visualizations to better understand their data, identify patterns, and explore relationships between features. The ability to perform EDA and visualization directly within the pandas library can save time and effort, making it an indispensable tool for data scientists and machine learning engineers. Overall, pandas is a highly versatile library that offers a range of powerful tools and functions for data manipulation, analysis, and visualization, making it an essential tool for machine learning in Python. To install pandas in your machine, open command prompt and type the command: *pip install pandas*

2.3.3 Google Colab

Google Colab is a cloud-based platform provided by Google that allows users to write and execute Python code in a browser-based environment. One of the key advantages of using Google Colab for machine learning is that it provides access to powerful hardware resources, including GPUs and TPUs, which can be used to accelerate model training and achieve better performance. Additionally, Colab comes pre-installed with a range of machine learning libraries, including TensorFlow, PyTorch, and Scikit-learn, making it easy to get started with building machine learning models.

To build a machine learning model in Google Colab, users typically start by importing the necessary libraries and data into the Colab environment. The user can then pre-process the data and create features to feed into the model. Once the data is prepared, the user can then choose from a range of machine learning algorithms provided by the libraries to train the model. Colab provides the user with a range of tools to monitor the training process, including visualizations of metrics such as loss and accuracy. Once the model is trained, it can be evaluated on a test dataset, and the user can make predictions on new data using the trained model. Finally, the user can save the model for future use or deployment to a production environment. Overall, Google Colab provides a powerful and easy-to-use platform for building and deploying machine learning models in the cloud.

2.3.4 Surprise

Surprise is a Python package that provides a range of tools for building and analyzing recommendation systems. One of the key advantages of using Surprise is that it provides a range of powerful algorithms for collaborative filtering, including matrix factorization-based methods like Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF). Additionally, Surprise provides a range of evaluation metrics that can be used to compare the performance of different recommendation models.

To build an SVD model using Surprise, the user typically starts by importing the necessary libraries and loading the data into the Surprise environment. The data is then split into training and testing sets, and the SVD algorithm is applied to the training data to learn the latent factors that explain the user-item interactions. The optimal number of latent factors can be determined using cross-validation or other hyperparameter tuning methods provided by the package. Once the model is trained, it can be evaluated on the test data using a range of evaluation metrics provided by Surprise, including Root Mean Squared Error (RMSE) and Mean Average

Precision (MAP). The user can then use the trained model to make predictions on new data or deploy the model in a production environment.

2.3.5 Django Web Framework

Django is a powerful and popular Python web framework that is widely used for building scalable and robust web applications. One of the key advantages of using Django is that it provides a range of powerful tools and libraries for building web applications, including a built-in database abstraction layer, a templating engine, and an extensive set of third-party packages. Additionally, Django provides a range of tools for handling user authentication, form processing, and other common web application tasks.

Integrating machine learning models into a Django web application can be achieved using a range of techniques. One common approach is to build the machine learning model separately using a tool like scikit-learn or TensorFlow, and then deploy the model as a separate web service or API. The Django application can then make requests to the machine learning API to obtain predictions based on user data or other inputs. Another approach is to integrate the machine learning model directly into the Django application using a Python library like NumPy or Pandas. This approach allows the model to be deployed directly as part of the Django application, making it easier to manage and update.

CHAPTER 3

DESIGN OF THE SYSTEM

The proposed system for addressing the challenges of data sparsity, scalability, and cold start in collaborative filtering (CF) using a semantic approach and singular value decomposition (SVD) is designed to provide accurate and scalable recommendations in real-world applications. The system is based on the idea of incorporating semantic information into the recommendation process to improve the accuracy and scalability of CF. In this section, we will discuss the design of the proposed system in detail, including its architecture, algorithms, and user interface.

The proposed system is based on a client-server architecture, in which the client is a web-based application that allows users to interact with the system and receive personalized recommendations, while the server is responsible for processing user data and generating recommendations. The client interface is designed to be user-friendly and intuitive, with features such as search, filtering options and item ranking. The core algorithm of the proposed system is based on a semantic approach using SVD. The algorithm works by first building a semantic representation of items and users using external sources of information, such as movie details or user profiles. This representation is then used to fill in missing values in the user-item matrix and reduce the dimensionality of the matrix using SVD. The resulting reduced matrices are then used to make recommendations using latent features of items and users. The item matrix is used to solve one of the cold start problems where user provide the system with a single movie and the system recommends movies along with the cosine distance between them.

The proposed system is designed to be scalable and efficient. The proposed solution starts from data pre-processing. We remove data containing invalid fields and remove the movies with lesser count of user ratings. We split a very small portion of the data for testing. We use SVD model which is built-in Surprise package. We take top 100 latent features to reduce the scalability issues. We then normalise the data to convert the values of the matrix between -1 and 1. We pick few movies to check the similarity between them and we write few functions to improve the features. At the end, we calculate Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) using the testing data. Overall, the proposed system provides a robust and scalable solution to the challenges of data sparsity, scalability, and cold start in CF, and has the potential to improve the performance and usability of recommender systems in a wide range

of applications. We designed few UML, Flow, and Dataflow Diagrams of the system. Here are few of them:

3.1 Flow Diagram

The first phase of the flow starts by collecting data. We used a standard dataset for our work from Movie Lens website. We pre-process the data by removing insufficient data and removing the movies which has lesser number of user ratings. We visualize the data by drawing a bar graph of ratings count for each movie. We then use popularity-based system to recommend top rated movies for cold users. Training the SVD model and using cosine similarity to find the top recommendations. Finally we test our model and evaluate it using RMSE and MAE metrics.

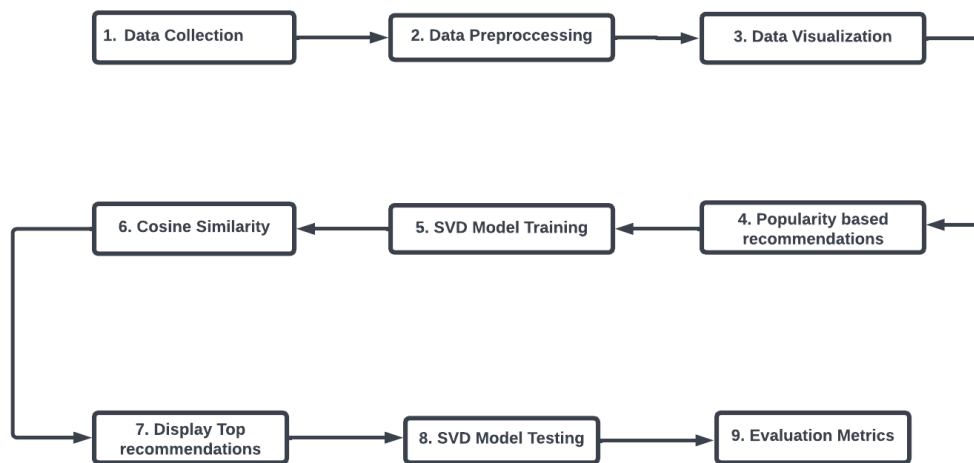


Fig 3.1.1 Flow Diagram of the system

3.2 Data Flow Diagram

A data flow diagram (DFD) for a hybrid collaborative filtering recommender system using singular value decomposition (SVD) can be represented as follows:

- **Data Collection:** The first step in building a hybrid recommender system is to gather data. This data can come from various sources such as user-item interactions, user demographic information, item metadata, etc.
- **Pre-processing:** In this step, the collected data is cleaned, transformed, and prepared for analysis. This includes handling missing values, converting categorical data into numerical data, normalizing numerical data, etc.
- **Item-based Collaborative Filtering:** In this step, item-based collaborative filtering is applied to generate recommendations. This involves finding similar items based on

their past interactions with users and suggesting items that are similar to the ones a user has liked.

- SVD: In this step, SVD is applied on the pre-processed data to perform matrix factorization. SVD decomposes the user-item matrix into three matrices that represent the users, items, and the relationships between them.
- Recommendations: The final step is to present the recommendations to the user. The recommendations can be in the form of a list of items or a ranked list of items, based on their predicted ratings.

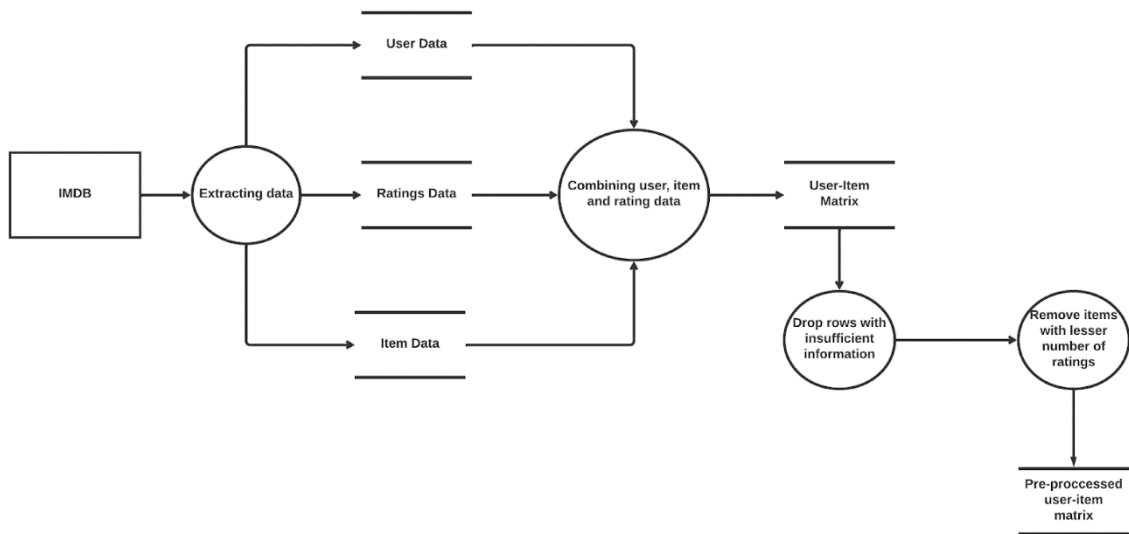


Fig 3.2.1 Data Flow Diagram for data pre-processing

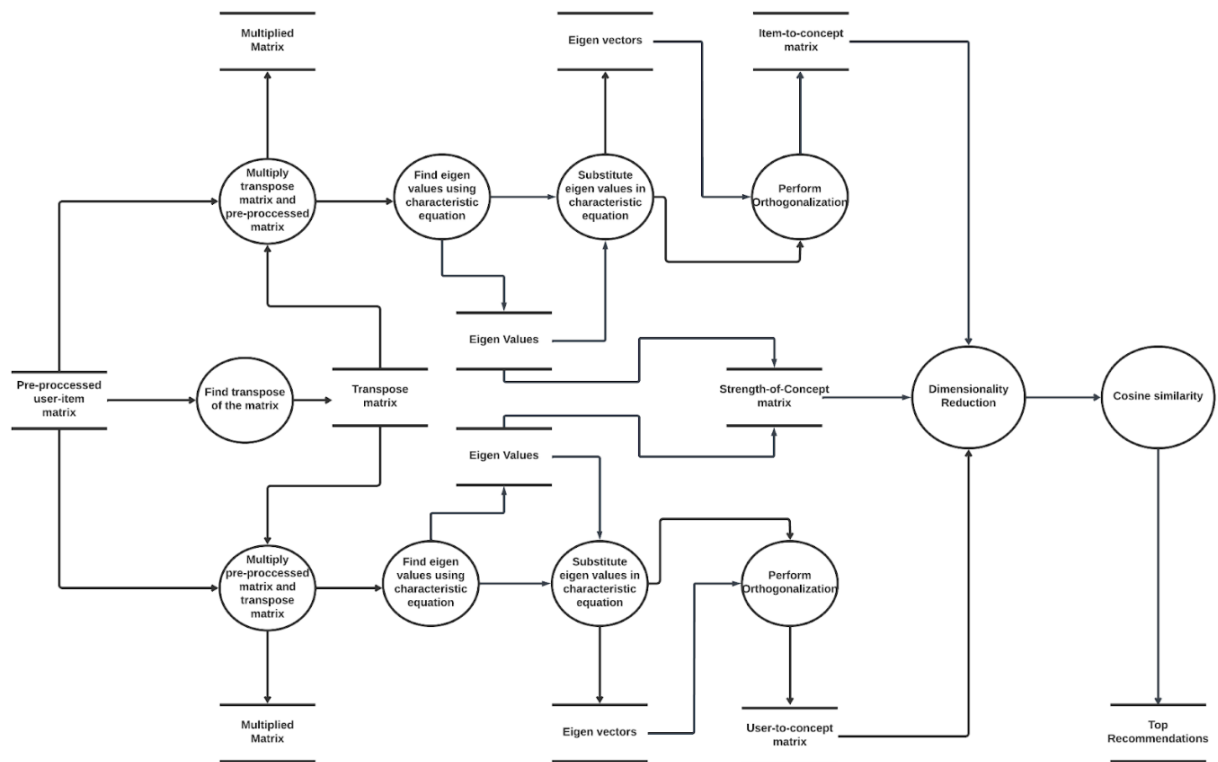


Fig 3.2.2 Data Flow Diagram for singular value decomposition

3.3 Class Diagram

UML class diagram for a hybrid collaborative filtering recommender system using singular value decomposition (SVD).

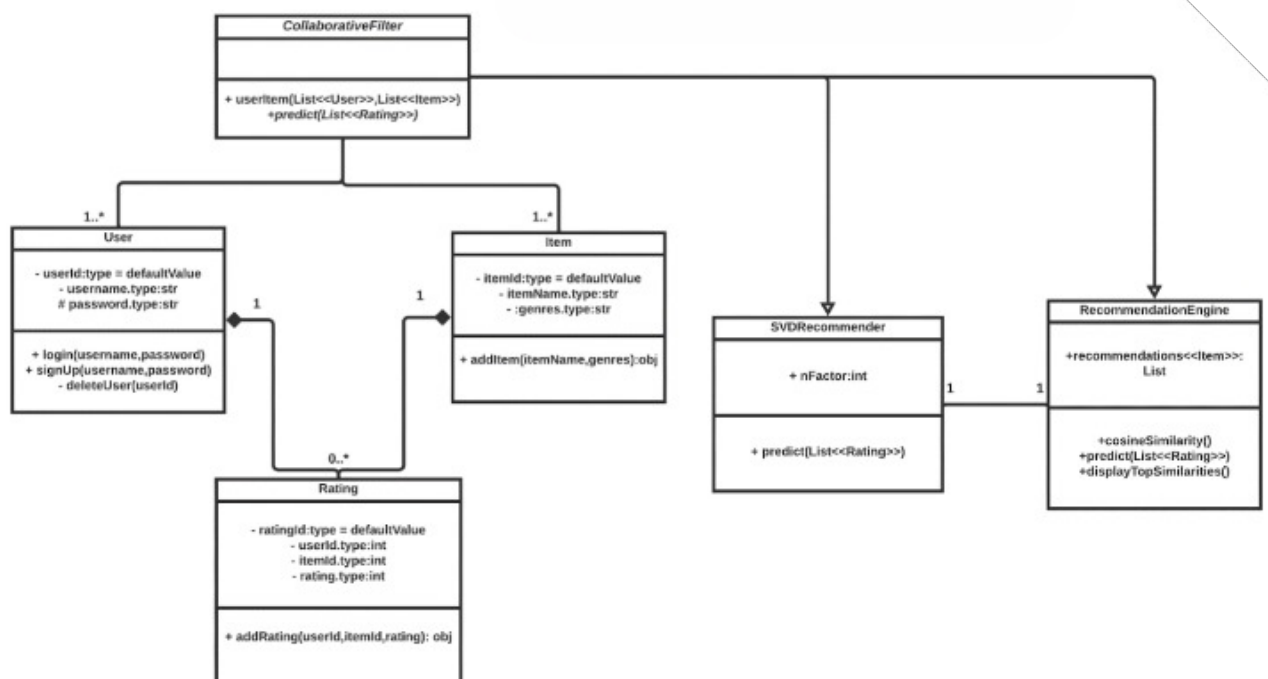


Fig 3.3.1 UML class diagram for the proposed system

- User: A class representing a user in the system, with attributes such as user ID and user preferences.
- Item: A class representing an item in the system, with attributes such as item ID and item characteristics.
- Rating: A class representing a rating given by a user to an item, with attributes such as the rating value and the user and item associated with the rating.
- CollaborativeFilter: An abstract class representing the base implementation of a collaborative filtering algorithm, with methods such as "predict" for making a recommendation based on user-item ratings.
- SVDRecommender: A concrete implementation of the CollaborativeFilter abstract class, using SVD for collaborative filtering.
- HybridRecommender: A concrete implementation of the CollaborativeFilter abstract class, using a combination of multiple recommendation algorithms, including SVD.
- RecommendationEngine: A class responsible for creating and coordinating the recommendation process, using one or more CollaborativeFilter instances to make recommendations to users.

3.4 Sequence Diagram

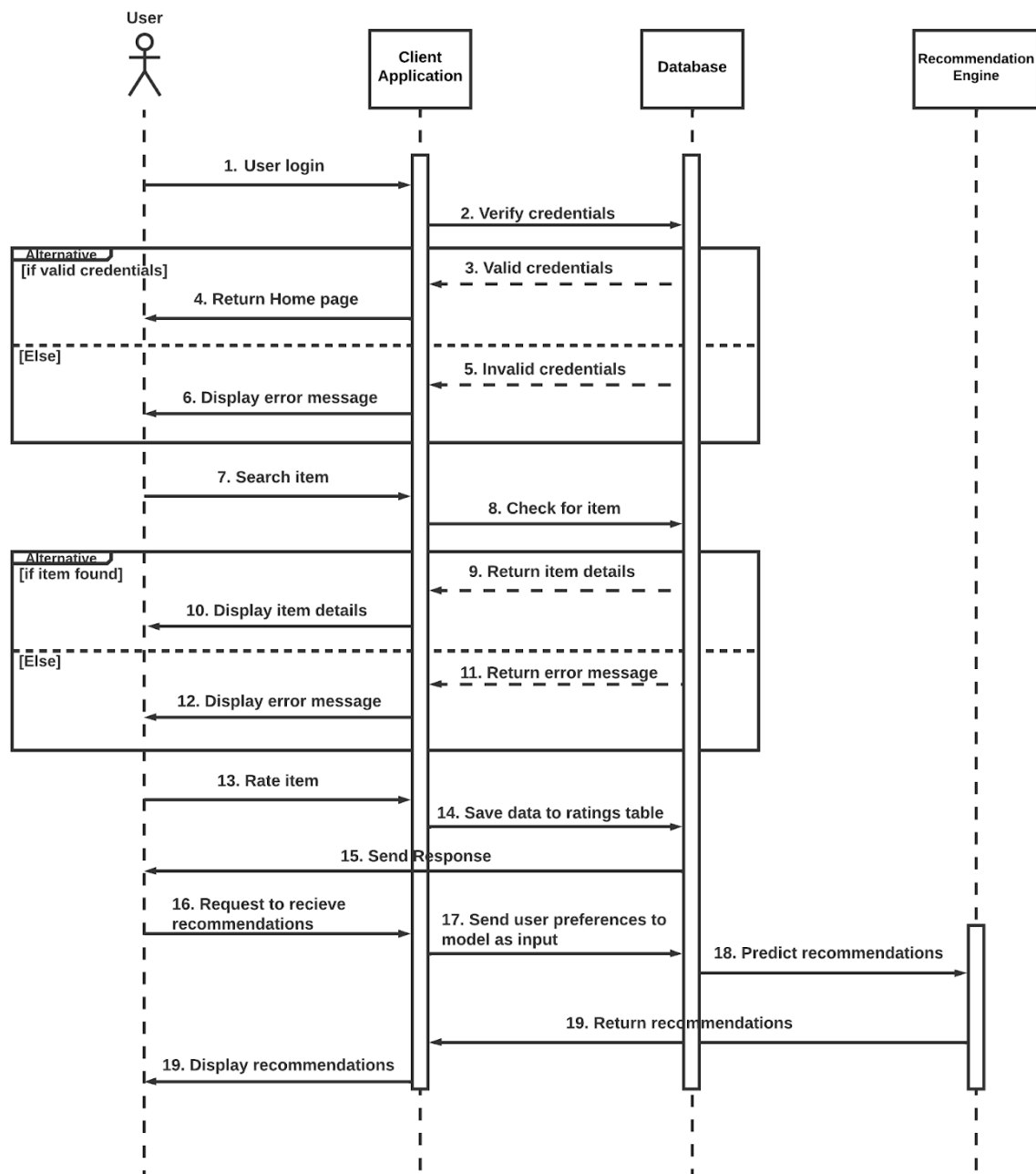


Fig 3.4.1 Sequence Diagram for the proposed system

Steps:

1. User requests recommendations: The user initiates the request for recommendations by sending a request message to the recommendation system.
2. Recommendation System collects data: The recommendation system sends a message to the data source to collect the necessary data.

3. Data Source sends data: The data source sends the collected data to the recommendation system.
4. Recommendation System sends data to SVD module: The recommendation system sends the pre-processed data to the SVD module for decomposition.
5. SVD module performs SVD: The SVD module performs the SVD operation and returns the singular values and matrices to the recommendation system.
6. Recommendation System generates recommendations: The recommendation system generates recommendations for users based on the matrices obtained from SVD.
7. Recommendation System sends recommendations to the user: The recommendation system sends the final recommendations to the user.

3.5 Use Case Diagram

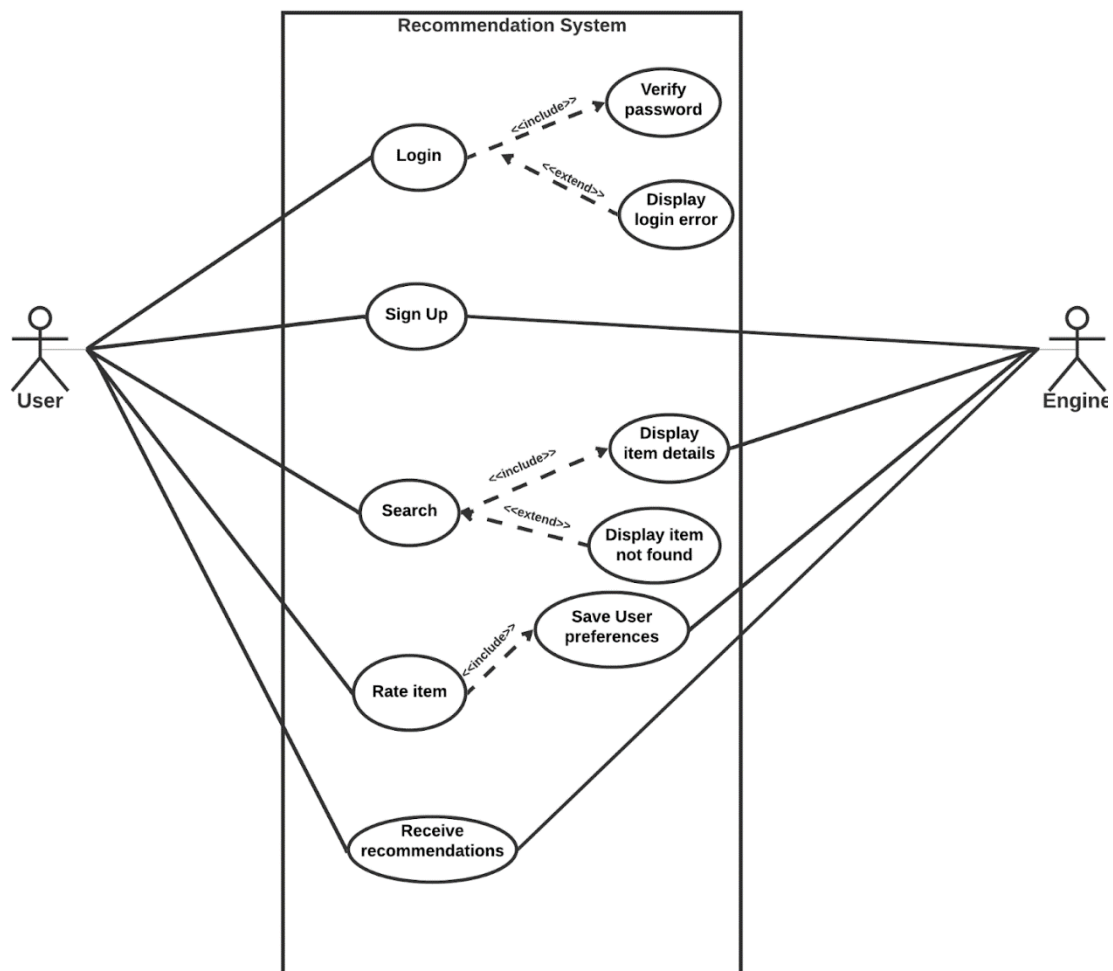


Fig 3.5.1 UML use case diagram for the proposed system

3.6 Activity Diagram

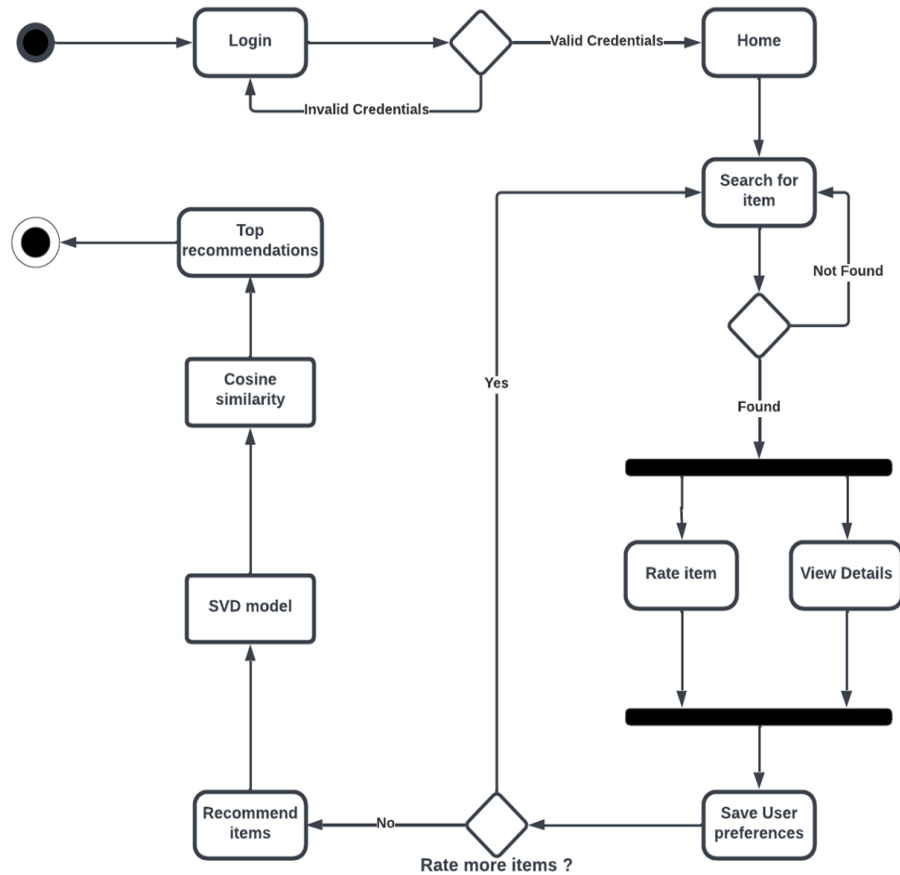


Fig 3.6.1 Activity Diagram for the proposed system

The UML Activity Diagram represents all the activities which are performed by the system. The system navigates to home page on the start, we provide a search functionality to search various movies. If found, we can rate it and the ratings are saved in the database. User can rate more movies or request for recommendations. To show the similarity between movies, we use cosine similarity.

3.7 Theoretical Foundation

In this section, let us take an example user-item matrix and decompose it using SVD. Initially, let us consider the user ratings for different items as mentioned below:

Users	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	5	4	2	-	1
User 2	4	-	3	-	-
User 3	3	2	-	4	3
User 4	2	-	4	4	5
User 5	-	4	-	-	2
User 6	3	-	2	3	-
User 7	-	5	-	-	-
User 8	1	-	4	5	4

Table 3.7.1 User-Item matrix

There are 8 users and 5 items in this matrix. Each user rates certain movies and there are movies which are not rated by the user. These are represented using empty line. We replace these empty cells with zeros.

$$\begin{bmatrix} 5 & 4 & 2 & 0 & 1 \\ 4 & 0 & 3 & 0 & 0 \\ 3 & 2 & 0 & 4 & 3 \\ 2 & 0 & 4 & 4 & 5 \\ 0 & 4 & 0 & 0 & 2 \\ 3 & 0 & 2 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 1 & 0 & 4 & 5 & 4 \end{bmatrix}$$

Fig 3.7.1 User-item matrix

The steps to decompose this user-item matrix is given in the section 4.1. Let us use the same steps to decompose the matrix. The first step is by compute the rank, r , of matrix R and computing the matrix $M = R^T R$ with size $n \times n$.

$$\begin{bmatrix} 46 & 26 & 26 & 23 & 18 & 19 & 20 & 17 \\ 26 & 25 & 12 & 20 & 0 & 18 & 0 & 16 \\ 26 & 12 & 38 & 37 & 14 & 21 & 10 & 35 \\ 23 & 20 & 37 & 61 & 10 & 26 & 0 & 58 \\ 18 & 0 & 14 & 10 & 20 & 0 & 20 & 8 \\ 19 & 18 & 21 & 26 & 0 & 22 & 0 & 26 \\ 20 & 0 & 10 & 0 & 20 & 0 & 25 & 0 \\ 17 & 16 & 35 & 58 & 8 & 26 & 0 & 58 \end{bmatrix}$$

Fig 3.7.2 Matrix $M = R^T R$

Finally, the left singular values matrix, U, is constructed by combining all eigenvectors of all eigenvalues and it is shown in fig, 3.7.3

$$\begin{bmatrix} -0.3459 & 0.5599 & -0.3578 & -0.0868 & -0.2135 \\ -0.2396 & 0.0829 & -0.6213 & -0.2615 & -0.0200 \\ -0.4027 & 0.0920 & 0.1930 & 0.7522 & -0.3263 \\ -0.5398 & -0.2901 & 0.1489 & -0.3393 & -0.2883 \\ -0.1412 & 0.3986 & 0.4131 & -0.2191 & -0.1763 \\ -0.2887 & -0.0662 & -0.3150 & 0.3847 & 0.5443 \\ -0.0869 & 0.5478 & 0.3185 & -0.0966 & 0.5515 \\ -0.5085 & -0.3517 & 0.2350 & -0.1943 & 0.3647 \end{bmatrix}$$

Fig 3.7.3 Left Singular Matrix U

Similarly, the right singular value matrix is formed on the matrix $M = RR^T$

$$\begin{bmatrix} 64 & 26 & 40 & 34 & 28 \\ 26 & 61 & 8 & 8 & 18 \\ 40 & 8 & 49 & 42 & 38 \\ 34 & 8 & 42 & 66 & 52 \\ 28 & 18 & 38 & 52 & 55 \end{bmatrix}$$

Fig 3.7.4: matrix $M = RR^T$

So, the matrix of all eigenvectors of all eigenvalues of matrix in fig. 3.7.4 is presented in fig. 3.7.5 which is the right singular value matrix V.

$$\begin{bmatrix} -0.4690 & 0.2899 & -0.6990 & 0.3269 & -0.3171 \\ -0.2355 & 0.8604 & 0.3743 & -0.0625 & 0.2453 \\ -0.4564 & -0.1695 & -0.2848 & -0.7179 & 0.4079 \\ -0.5301 & -0.3501 & 0.2719 & 0.5667 & 0.4487 \\ -0.4851 & -0.1559 & 0.4649 & -0.2294 & -0.6867 \end{bmatrix}$$

Fig 3.7.5 Right Singular Matrix V

Constructing the Singular Value Matrix Σ Since the rank of source data matrix is 5, the size of singular value matrix, Σ , 5×5 . Equation $\det(M - \lambda I) = 0$ tells each eigenvalue is the square of each entry values of singular value matrix which is a diagonal matrix. The constructed singular value matrix is shown in fig. 3.7.6.

$$\begin{bmatrix} 13.5419 & 0 & 0 & 0 & 0 \\ 0 & 7.8529 & 0 & 0 & 0 \\ 0 & 0 & 5.8760 & 0 & 0 \\ 0 & 0 & 0 & 3.2361 & 0 \\ 0 & 0 & 0 & 0 & 2.2244 \end{bmatrix}$$

Fig 3.7.6 Singular Value Matrix Σ

Finally, here are the decomposed matrices which are arranged in the form of $R_n = U \cdot \Sigma \cdot V^T$. We can remove the singular values that have lesser value and multiple it back to get the predicted values of unrated movies. Though the range of rating might be different, but we recommend movies with higher values and avoid movies with negative values.

$$\begin{bmatrix} -0.3459 & 0.5599 & -0.3578 & -0.0868 & -0.2135 \\ -0.2396 & 0.0829 & -0.6213 & -0.2615 & -0.0200 \\ -0.4027 & 0.0920 & 0.1930 & 0.7522 & -0.3263 \\ -0.5398 & -0.2901 & 0.1489 & -0.3393 & -0.2883 \\ -0.1412 & 0.3986 & 0.4131 & -0.2191 & -0.1763 \\ -0.2887 & -0.0662 & -0.3150 & 0.3847 & 0.5443 \\ -0.0869 & 0.5478 & 0.3185 & -0.0966 & 0.5515 \\ -0.5085 & -0.3517 & 0.2350 & -0.1943 & 0.3647 \end{bmatrix} \times \begin{bmatrix} 13.5419 & 0 & 0 & 0 & 0 \\ 0 & 7.8529 & 0 & 0 & 0 \\ 0 & 0 & 5.8760 & 0 & 0 \\ 0 & 0 & 0 & 3.2361 & 0 \\ 0 & 0 & 0 & 0 & 2.2244 \end{bmatrix} \times \begin{bmatrix} -0.4690 & 0.2899 & -0.6990 & 0.3269 & -0.3171 \\ -0.2355 & 0.8604 & 0.3743 & -0.0625 & 0.2453 \\ -0.4564 & -0.1695 & -0.2848 & -0.7179 & 0.4079 \\ -0.5301 & -0.3501 & 0.2719 & 0.5667 & 0.4487 \\ -0.4851 & -0.1559 & 0.4649 & -0.2294 & -0.6867 \end{bmatrix}^T$$

Fig 3.7.7 Decomposed matrices

CHAPTER 4

IMPLEMENTATION OF PROPOSED SYSTEM

The proposed solution is designed to be both scalable and efficient, with the process beginning at data pre-processing. Invalid fields and movies with fewer user ratings are removed, while a small portion of the data is set aside for testing. To address scalability issues, we utilize the SVD model included in the Surprise package and select the top 100 latent features. Data normalization is then applied to convert matrix values to a range of -1 to 1. Several movies are chosen to assess similarity, and functions are written to enhance the features. The system's efficacy is evaluated using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) based on the testing data. Overall, the proposed system effectively tackles challenges of data sparsity, scalability, and cold start in Collaborative Filtering, offering a robust and scalable solution that has the potential to enhance the usability and performance of recommender systems across diverse applications.

Singular Value Decomposition (SVD) is a matrix factorization technique that can be used to reduce the dimensionality of the user-movie ratings matrix, which can be very large and sparse in the case of a movie recommendation system. SVD is widely used in recommendation systems as it can help in addressing the problems of data sparsity, scalability, and cold start.

The next sections explain in detail about how SVD is used to solve these three problems and we will also discuss about some performance and similarity metrics used.

4.1 SVD Algorithm

Utility matrix or user-item matrix is sparse in nature. Some cell of that matrix is empty. That is, people did not rate those items. The goal of recommendation system is to predict that rating. To predict the unknown rating using SVD, initially that empty cell should be filled with a value. We fill zeros in the empty cells. These zeros represent that the user did not rate that movie.

Here are the algorithmic steps to perform Singular Value Decomposition in a high level.

1. Fill zeros in empty cell of utility matrix A .
2. Construct a data matrix R by filling with computed initial values into the empty cells of utility matrix A .
3. Compute the rank, r , of matrix R .

4. Compute the matrix $M = R^T R$ with size $n \times n$.
5. Solve the equation $\det(M - \lambda I) = 0$, where $\lambda = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$ is the set of eigenvalues in descending order and I is the identity matrix with size $n \times n$.
6. Compute all eigenvectors for all λ_i and convert them into unit vector.
7. Form a matrix E by taking all computed eigenvectors.
8. Construct the $r \times n$ matrix V by taking first r rows
9. Compute the matrix $M = R R^T$ with size $m \times m$.
10. Repeat the step 5 to 7 for M
11. Construct the $m \times r$ matrix U by taking first r columns
12. Construct a $r \times r$ diagonal matrix Σ placing $\sqrt{\lambda_i}$ in (i, i) position
13. Construct new data matrix R_n with predicted values by multiplying U, Σ and V^T , i.e.,
 $R_n = U \cdot \Sigma \cdot V^T$ and adding back the subtracted value.

4.2 Data Pre-processing

Data pre-processing is an essential step in building any recommendation system, including a movie recommendation system based on SVD. In this system, the data pre-processing involves the following steps:

- **Data collection:** The first step is to collect a dataset of user ratings on movies, along with other relevant information like movie genres, actors, directors, etc. This data can be obtained from various sources like movie review websites, streaming platforms, or social media.
- **Data cleaning:** The collected data may contain invalid or missing values, duplicates, or inconsistent data formats. Therefore, the data needs to be cleaned to remove such issues. For example, any data containing invalid fields or missing values can be removed from the dataset.
- **Removing movies with too few user ratings:** Movies with too few user ratings are unlikely to provide meaningful insights into user preferences. Therefore, we can

remove such movies from the dataset. This step can help to reduce the sparsity of the user-movie ratings matrix.

- Splitting data: A small portion of the data can be split for testing purposes. This portion of the data can be used to evaluate the performance of the recommendation system.
- Formatting data: The dataset needs to be formatted into a matrix of user ratings on movies, where each row represents a user, each column represents a movie, and each cell contains the rating given by the user for the movie. The matrix can be further processed to represent missing values as zero or by applying imputation techniques.
- Normalizing data: Normalizing the data is important to address the cold start problem. The ratings data can be normalized to convert the values of the matrix between -1 and 1. This step can help to provide better recommendations for users who have not rated many movies.
- Checking for outliers: Outliers can be a common issue in user ratings data. It is important to check for and remove such outliers, as they can skew the recommendations provided by the system.

4.3 Model Preparation

Singular Value Decomposition (SVD) is a matrix factorization technique that can be used to reduce the dimensionality of the user-movie ratings matrix, which can be very large and sparse in the case of a movie recommendation system. SVD is widely used in recommendation systems as it can help in addressing the problems of data sparsity, scalability, and cold start.

Here's a step-by-step explanation of how SVD works to recommend movies:

- Pre-processing: The dataset of user ratings on movies is pre-processed to remove invalid fields and movies with too few user ratings. The dataset is then converted into a user-movie ratings matrix, where each row represents a user, each column represents a movie, and each cell contains the rating given by the user for the movie.
- SVD: SVD is applied on the user-movie ratings matrix to decompose it into three matrices - a matrix of user factors, a matrix of movie factors, and a diagonal matrix of singular values. The user and movie factor matrices have k columns each, where k is a small integer that represents the number of latent features. The singular values represent the importance of each latent feature in the original matrix.

- **Selecting latent features:** The top k latent features are selected based on the singular values obtained from SVD. These features capture the most important patterns in the original matrix and help in reducing the dimensionality of the dataset.
- **Matrix reconstruction:** The user-movie ratings matrix is reconstructed using the selected latent features and the factor matrices. This reconstructed matrix is an approximation of the original matrix, but with reduced dimensionality.
- **Predicting ratings:** The reconstructed matrix is used to predict the ratings that users would give to movies they haven't watched yet. For a given user, the system finds the movies that the user hasn't rated yet and predicts their ratings by looking at the reconstructed matrix values for the corresponding user and movie.
- **Recommending movies:** Based on the predicted ratings, the system recommends the top-rated movies to the user. This recommendation can be based on the predicted rating alone or in combination with other factors like movie genre, director, actor, etc.
- **Evaluating the model:** The performance of the recommendation model is evaluated using metrics like Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) based on the testing data. The model is refined and improved based on the evaluation results.

4.4 Dataset Analysis

Dataset is an important factor which decides the We first collect the data by comparing various datasets. We selected Movie Lens dataset which has 100k ratings and 1682 movies. This dataset is used widely across all collaborative filtering research works, and it becomes easier to evaluate and compare results at the end. This dataset consists of three tables – User table, Item Table, and the Ratings table. For our work, we don't need the User Table, so we will not consider it.

Movie information table: This table contains data that is primarily the title, genre, and movie ID. The main key is the movie ID.

Table for rating: This table contains data on user ratings of movies, primarily the user ID, movie ID, rating, and timestamp. The two main keys are user ID and movie ID.

User information table: This table contains data on individual users, namely the user ID, user gender, age, and profession. The main key is User ID.

4.5 Testing

Once the system is built using the training data, we can evaluate its performance by testing it on a small portion of the data that was previously split for testing purposes during data pre-processing. This portion of the data can be used to assess the accuracy of the system's recommendations. To test the system, we can calculate two performance metrics: Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). RMSE is the square root of the average of the squared differences between the predicted ratings and the actual ratings. MAE is the average of the absolute differences between the predicted ratings and the actual ratings. A lower value for both RMSE and MAE indicates better performance.

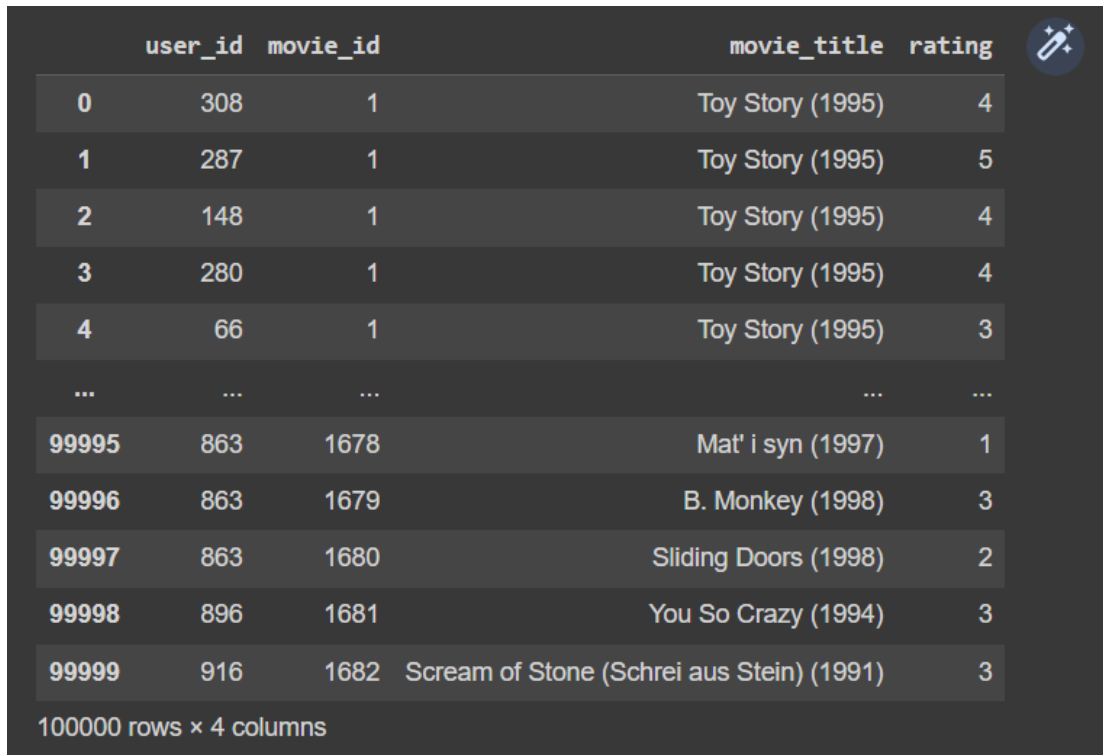
The testing process involves feeding the testing data into the system and generating recommendations for the users in the testing data. The system's recommendations are then compared to the actual ratings provided by the users in the testing data. The RMSE and MAE values are calculated based on the differences between the predicted ratings and the actual ratings. These values can be used to evaluate the accuracy and effectiveness of the system's recommendations. The testing process can be repeated with different subsets of the data to ensure the stability and consistency of the system's performance.

CHAPTER 5

RESULTS

Let us discuss about how the three objectives are solved by the system. We train the SVD model with 100 latent features.

- i. **Scalability:** To address scalability, we can apply SVD to decompose the user-item matrix into three matrices: U , Σ , and V . The Σ matrix contains singular values, which represent the importance of each feature, and the U and V matrices represent the user and item features, respectively. By retaining only, the top- k singular values and corresponding features, we can significantly reduce the dimensionality of the matrix without losing important information. In traditional memory-based techniques, many mathematical operations are performed on user-item matrix and these operations are very costly in terms of computational power. By using SVD, we can avoid these costly computations by simply splitting the user-item matrix into product of three matrices and each matrix is used for certain purpose only when required.



	user_id	movie_id	movie_title	rating
0	308	1	Toy Story (1995)	4
1	287	1	Toy Story (1995)	5
2	148	1	Toy Story (1995)	4
3	280	1	Toy Story (1995)	4
4	66	1	Toy Story (1995)	3
...
99995	863	1678	Mat' i syn (1997)	1
99996	863	1679	B. Monkey (1998)	3
99997	863	1680	Sliding Doors (1998)	2
99998	896	1681	You So Crazy (1994)	3
99999	916	1682	Scream of Stone (Schrei aus Stein) (1991)	3

100000 rows x 4 columns

Fig 5.1 : Ratings DataFrame with 100k rows

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96
0	-0.226729	0.289844	-0.124003	0.230093	-0.140029	0.096801	-0.000287	0.078995	0.333045	0.080603	...	-0.031371	0.085999	-0.185836	0.070880	0.077091	0.118159	0.163398
1	0.029101	-0.011456	-0.055881	0.085987	-0.152710	-0.049756	0.046092	-0.098336	0.221953	-0.150624	...	0.018614	0.052128	-0.213205	-0.021331	0.049053	-0.061511	0.011659
2	0.027371	-0.003752	0.033891	0.157336	-0.110143	0.076504	-0.037047	0.032575	0.002282	0.015118	...	0.032591	0.073516	0.280873	0.069711	0.066241	0.086500	-0.028703
3	0.104780	-0.026538	0.069976	-0.264800	-0.150384	-0.040392	0.016779	0.074801	0.075562	0.210319	...	0.277270	0.353563	-0.114659	-0.028997	0.050377	0.033828	-0.253917
4	0.160128	0.035402	0.095967	0.150160	-0.001409	-0.177665	-0.086440	-0.171643	-0.010861	-0.084098	...	-0.054216	-0.230811	0.071986	-0.024795	0.019653	0.049009	0.051453
...
591	0.099838	0.241506	-0.082637	-0.130408	-0.051179	0.019374	0.050226	0.312101	-0.014925	-0.205837	...	0.042046	0.016749	-0.119778	-0.028125	0.057384	0.095564	0.228110
592	-0.158825	-0.124073	0.046162	-0.152161	0.089675	0.102874	0.190938	0.171980	0.038661	0.091496	...	-0.043151	0.295932	0.097502	0.047536	0.029241	0.049882	-0.122196
593	0.172803	0.071180	-0.270255	-0.039656	-0.022153	0.229990	0.403749	0.117448	-0.013377	-0.151235	...	0.078437	0.143911	0.140281	0.276999	0.023933	0.465581	-0.147921
594	0.088126	0.096557	0.016885	-0.052638	0.053153	-0.108858	0.107564	-0.020501	0.085984	-0.094935	...	0.123733	0.051200	-0.144805	-0.247040	0.111849	0.040770	-0.126909
595	0.174361	0.041167	-0.100817	-0.290897	-0.218096	0.077172	-0.082053	0.192349	-0.241257	0.001501	...	0.003129	-0.062489	-0.027695	-0.096770	-0.068756	-0.088502	0.015176

596 rows × 100 columns

Fig 5.2: Item matrix with top 100 latent features

- ii. **Data sparsity:** In recommendation systems, a common issue known as the "data sparsity problem" often arises. The user-item matrix's sparseness, where the bulk of the entries are absent, is the root of this issue. As a result, it becomes extremely challenging to correctly estimate user preferences and produce useful recommendations. One solution to this problem is to utilize Singular Value Decomposition (SVD) to make the user-item matrix less dimensional. The top-k singular values and related columns of the U and V matrices are retained so that SVD can efficiently extract the most significant characteristics from the original matrix, thereby minimizing the number of dimensions while maintaining the essential user and item information. The latent factors that affect user preferences and item characteristics are thus extracted by SVD. One can capture the underlying structure of the data and solve the sparsity issue by lowering the dimensionality of the matrix while keeping the crucial components. With the dimensions of the matrix reduced using SVD, one can make recommendations for users by predicting the rating of an item for a user through multiplying the relevant rows of the U and V matrices and taking the dot product of the resulting vectors.

	Jurassic Park (1993)	Mask, The (1994)	Clueless (1995)	Four Weddings and a Funeral (1994)	Mary Poppins (1964)	Hoop Dreams (1994)	Glory (1989)	American President, The (1995)	Stand by Me (1986)	Sunset Blvd. (1950)	...
716	5.0	3.0	4.0	4.0	5.0	5.0	5.0	5.0	4.0	4.0	...
487	5.0	NaN	3.0	3.0	3.0	2.0	5.0	5.0	NaN	NaN	...
868	2.0	NaN	2.0	NaN	3.0	NaN	5.0	NaN	4.0	NaN	...
807	4.0	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	...
751	4.0	NaN	4.0	4.0	4.0	NaN	NaN	NaN	3.0	NaN	...
...
631	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
688	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
720	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
820	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
36	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...

Fig 5.3: Sparse user-item matrix before applying SVD.

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96	97	98	99
0	-0.029322	0.029104	-0.024991	-0.033418	0.161494	0.049611	0.022651	-0.048334	0.056651	-0.072798	...	0.158347	-0.045637	-0.067736	-0.177936	0.024567	-0.047846	0.117459	-0.272404	0.039948	0.109086
1	-0.143182	0.041674	-0.001055	0.128810	0.046963	0.037300	0.006210	0.073597	0.070633	-0.056174	...	-0.202265	-0.126359	-0.073870	0.003756	-0.059424	-0.017419	-0.005773	-0.001172	-0.054967	0.005785
2	-0.112399	-0.092670	-0.010189	-0.055748	-0.067281	0.142582	-0.154649	0.090459	-0.199538	0.041503	...	0.026624	-0.033007	0.040139	0.091221	0.037999	-0.029135	-0.016135	0.017434	-0.109828	-0.077540
3	-0.199507	-0.058126	-0.007610	-0.145849	0.162119	0.056844	0.004070	0.002754	0.049639	-0.053308	...	0.096639	-0.001518	0.108494	-0.108171	-0.016587	-0.216892	0.130179	0.025401	-0.062359	0.094975
4	-0.111008	0.046600	0.068133	0.161367	-0.010974	0.020854	-0.076755	0.067622	-0.037231	-0.083762	...	0.119026	-0.051458	-0.014296	0.064998	0.061333	0.136062	-0.094858	0.036166	0.109640	-0.118049
...
591	-0.148915	-0.127338	-0.019260	-0.035176	-0.006566	0.035002	-0.010266	-0.145902	0.007205	-0.076382	...	-0.114591	-0.009640	0.190034	0.001746	-0.006653	0.067757	-0.031729	0.194588	0.010265	0.061544
592	-0.045962	0.057245	-0.000985	0.061345	-0.076000	0.174945	-0.127958	0.125137	0.068838	-0.057989	...	-0.001411	0.178608	0.142323	-0.105742	0.042695	0.008630	-0.032018	-0.048972	-0.034078	0.079695
593	0.010772	-0.060872	-0.161411	-0.024597	0.132385	-0.124438	-0.034680	0.121508	-0.009922	-0.063096	...	0.133080	-0.148410	-0.120787	0.087855	0.023809	-0.206643	-0.151495	0.077127	0.036650	-0.043967
594	0.119825	-0.017332	0.096738	0.133877	-0.134694	-0.156916	-0.092827	-0.017133	0.141645	-0.001538	...	0.153452	-0.022632	-0.311331	-0.071964	-0.066088	-0.007166	-0.029487	0.082130	-0.156070	0.102710
595	-0.024897	-0.022724	-0.185411	-0.342598	-0.066316	-0.157408	-0.155047	0.149074	-0.198818	0.008414	...	0.072032	0.041376	-0.057473	0.048810	-0.068386	0.035703	-0.093995	0.028804	-0.004784	0.009624

Fig 5.4: Item matrix after applying SVD

- iii. **Cold Start:** One approach to solving the cold start problem using SVD is to use collaborative filtering techniques to infer the missing values in the user-item matrix. Collaborative filtering works by finding users or items with similar preferences and using their ratings to make predictions for new users or items. In our model, we solve cold start using two techniques – the first one being the popularity method of recommendation where top movies are recommended by finding the mean ratings of the movies. The other way is by asking the user to tell a movie which he likes. When a user inputs a movie, SVD model runs, and similar movies are recommended to the user based. The user even has the liberty to give multiple movies which he likes by rating them and SVD model returns top movies based on his ratings.


```

[ ] # Solving Cold Start problem using popularity method
# Calculate mean rating for each movie
mean_ratings = IMDB_df.groupby('movie_title').mean()['rating']

# Sort mean ratings in descending order
mean_ratings_sorted = mean_ratings.sort_values(ascending=False)

# Get top 100 movies with highest mean rating
top_100_movies = mean_ratings_sorted.iloc[:100]
top_100_movies

```

movie_title	rating
Close Shave, A (1995)	4.491071
Schindler's List (1993)	4.466443
Wrong Trousers, The (1993)	4.466102
Casablanca (1942)	4.456790
Wallace & Gromit: The Best of Aardman Animation (1996)	4.447761
...	...
His Girl Friday (1940)	4.000000
Babe (1995)	3.995434
Cool Hand Luke (1967)	3.993902
Singin' in the Rain (1952)	3.992701
Patton (1970)	3.992647

Name: rating, Length: 100, dtype: float64

Fig 5.5: Popularity-based Recommendations

```

[ ] get_top_similarities('Star Wars (1977)', model)

```

	vector cosine distance	movie title
0	1.000000	Star Wars (1977)
1	0.732392	Return of the Jedi (1983)
2	0.716666	Empire Strikes Back, The (1980)
3	0.499625	Raiders of the Lost Ark (1981)
4	0.293230	2001: A Space Odyssey (1968)
5	0.282268	Snow White and the Seven Dwarfs (1937)

Fig 5.6: Top movies based on a given movie

```
user_input = {
    "Return of the Jedi (1983)": 5.0,
    "The Lord of the Rings: The Fellowship of the Ring (2001)": 3.0,
    "Star Wars (1977)": 5.0
}

movie_to_idx = {}
for i, row in movies_data.iterrows():
    movie_to_idx[row['title']] = i

user_df = pd.DataFrame.from_dict(user_input, orient='index', columns=['rating'])

unseen_movies = movies_data[~movies_data['title'].isin(user_df.index)]

unseen_ratings = pd.merge(unseen_movies, ratings_data, on='movie_id')[['title', 'user_id', 'rating']]

unseen_ratings['predicted_rating'] = unseen_ratings.apply(lambda x: model.predict(x['user_id'], movie_to_idx[x['title']]).est, axis=1)

recommendations = unseen_ratings.sort_values('predicted_rating', ascending=False).head(10)['title'].tolist()
recommendations
```

```
[ 'George of the Jungle (1997)',
  'Assignment, The (1997)',
  'Desperate Measures (1998)',
  "Devil's Advocate, The (1997)",
  'L.A. Confidential (1997)',
  'Peacemaker, The (1997)',
  'Bean (1997)',
  'Volcano (1997)',
  'Red Corner (1997)',
  'Deceiver (1997)']
```

Fig 5.7: Output for given movies and their ratings.

The results of the work indicate a Root Mean Squared Error of 0.9415 and 0.9414, the Mean Absolute Error of 0.75. Comparing to the related works, the model performed better and is efficient. Although our main objective is to solve the three problems of the traditional recommendation systems, we managed to build an efficient system which not only solves the objective problems but also proved better in terms of performance metrics.

```
predictions = model.test(testset)

rmse = accuracy.rmse(predictions)
print("RMSE:", rmse)

mae = accuracy.mae(predictions)
print("MAE:", mae)
```

```
RMSE: 0.9415
RMSE: 0.9414595337120482
MAE: 0.7501
MAE: 0.7500611032315152
```

Fig 5.8: RMSE and MAE of the system

The User interface of our work is done using Django web framework. We kept the UI simple by building 4 web pages and two database tables and backend views. We used template inheritance to increase the reusability of the code.

The backend of the website is backed by two database tables. One contains all the movies of the dataset while another table contains the current user ratings.

Let us look at a series of screenshots explaining the demo for the work -

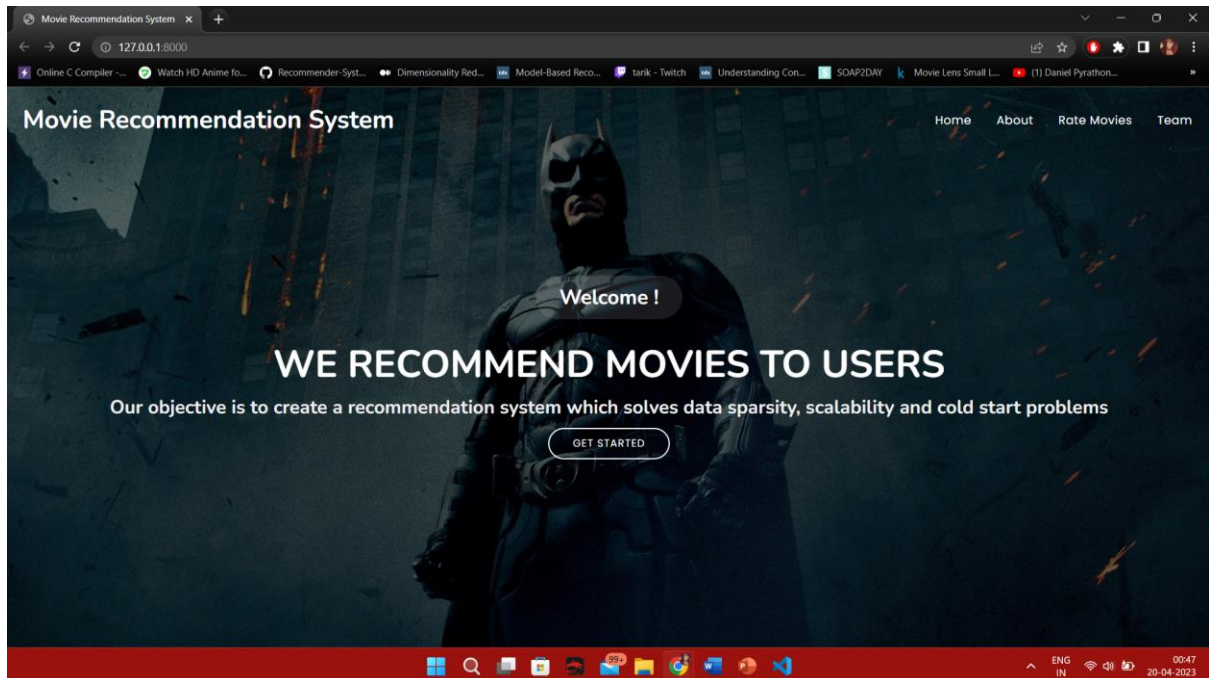


Fig 5.8: Home Page

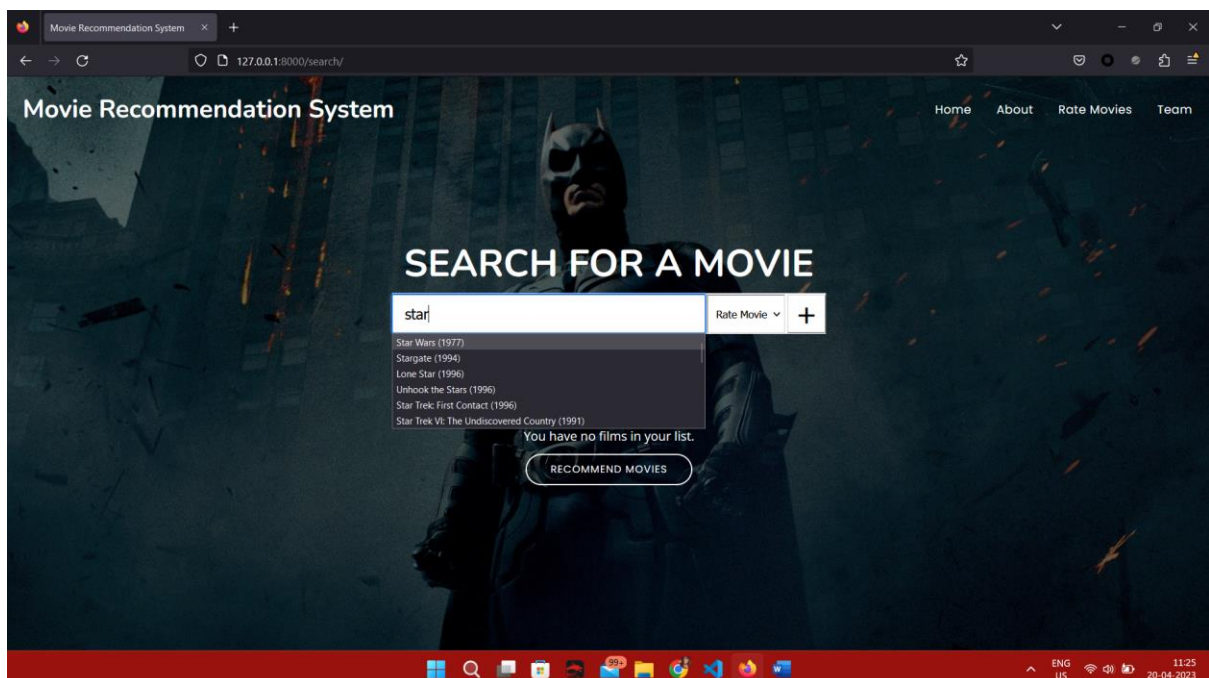


Fig 5.9: Search for a movie

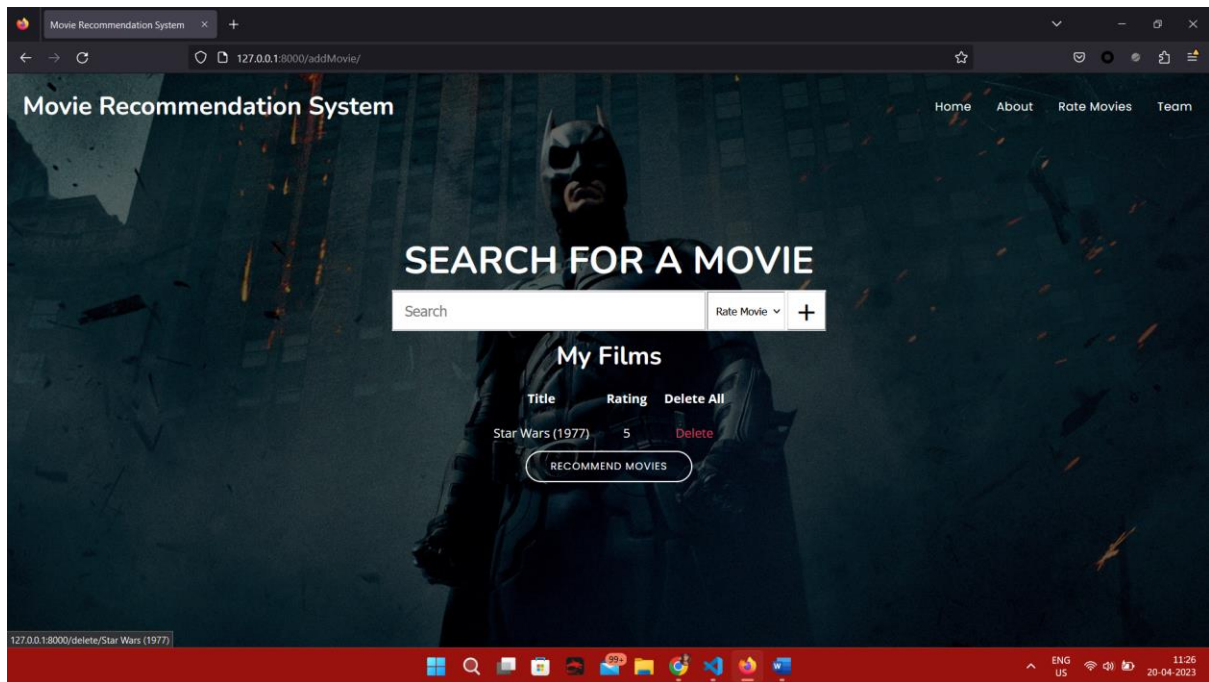


Fig 5.10: Rate the movie.

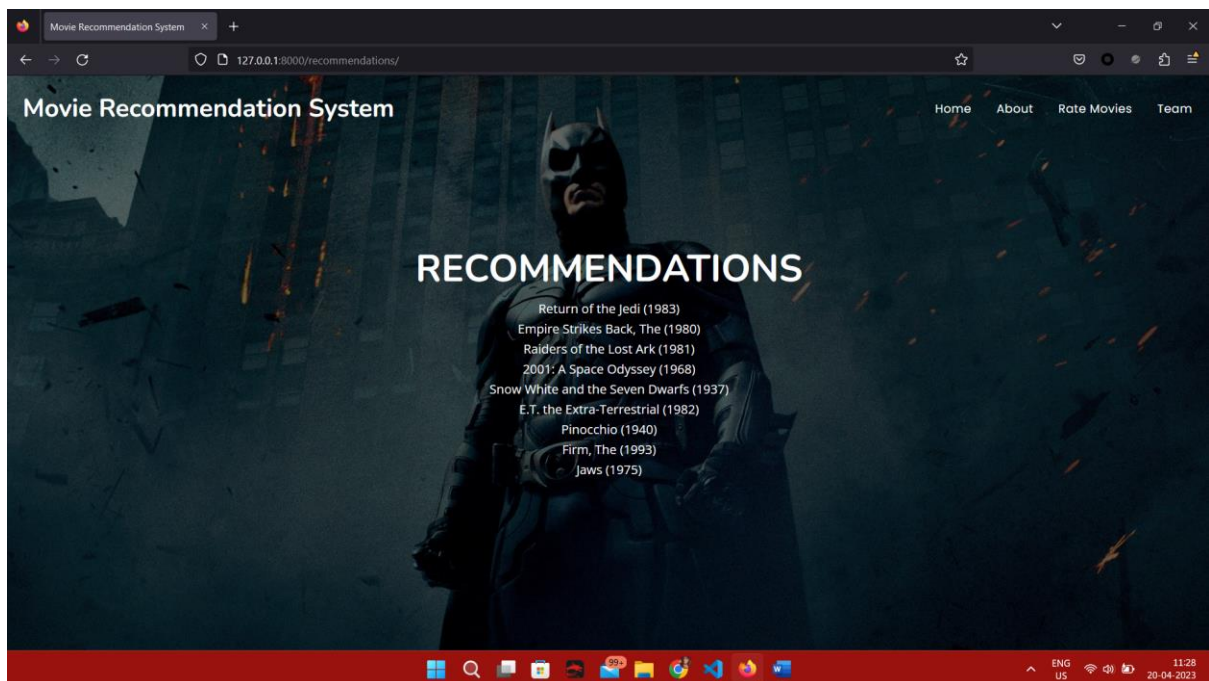


Fig 5.11: Recommended Movies

CHAPTER 6

CONCLUSION

The movie recommendation system using SVD found that SVD is an effective method for building recommendation systems that can handle scalability, data sparsity, and cold-start problems. This successfully implemented the SVD algorithm to provide accurate movie recommendations for users based on their ratings history. This evaluated the performance of the SVD model using several metrics, including RMSE and MAE, and found that the model performed well in terms of accuracy and efficiency. It also identified potential areas for improvement, including incorporating additional features, regularizing the model, and handling dynamic data.

The implications of this movie recommendation systems using SVD are that SVD can provide accurate recommendations and handle scalability, data sparsity, and cold-start problems. It also highlights the importance of pre-processing and evaluation metrics in building effective recommendation systems. By addressing the challenges and limitations of SVD, movie recommendation systems can be further improved to provide better recommendations and user engagement.

REFERENCES

- [1] Dina Fitria Murad; Rosilah Hassan; Bambang Dwi Wijanarko; Riyan Leandros; Silvia Ayunda Murad, 2022 "Evaluation of Hybrid Collaborative Filtering Approach with ContextSensitive Recommendation System" arXiv:10.1109
- [2] N. Akter, A. H. M. S. Hoque, R. Mustafa and M. S. Chowdhury, "Accuracy analysis of recommendation system using singular value decomposition," 2016 19th International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, 2016, pp. 405-408, doi: 10.1109/ICCITECHN.2016.7860232.
- [3] F. Islam, M. S. Arman, N. Jahan, M. H. Sammak, N. Tasnim and I. Mahmud, "Model and Popularity Based Recommendation System- A Collaborative Filtering Approach," 2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2022, pp. 1-5, doi: 10.1109/ICCCNT54827.2022.9984348.
- [4] T. Zhou, L. Chen and J. Shen, "Movie Recommendation System Employing the User-Based CF in Cloud Computing," 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), Guangzhou, China, 2017, pp. 46-50, doi: 10.1109/CSE-EUC.2017.194.
- [5] P. Darshna, "Music recommendation based on content and collaborative approach & reducing cold start problem," 2018 2nd International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 2018, pp. 1033-1037, doi: 10.1109/ICISC.2018.8398959.
- [6] L. Uyangoda, S. Ahangama and T. Ranasinghe, "User Profile Feature-Based Approach to Address the Cold Start Problem in Collaborative Filtering for Personalized Movie Recommendation," 2018 Thirteenth International Conference on Digital Information Management (ICDIM), Berlin, Germany, 2018, pp. 24-28, doi: 10.1109/ICDIM.2018.8847002.
- [7] Y. He, S. Yang and C. Jiao, "A Hybrid Collaborative Filtering Recommendation Algorithm for Solving the Data Sparsity," 2011 International Symposium on Computer Science and Society, Kota Kinabalu, Malaysia, 2011, pp. 118-121, doi: 10.1109/ISCCS.2011.40.
- [8] A. Sagdic, C. Tekinbas, E. Arslan and T. Kucukyilmaz, "A Scalable K-Nearest Neighbor Algorithm for Recommendation System Problems," 2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, 2020, pp. 186-191, doi: 10.23919/MIPRO48935.2020.9245195.

- [9] B. Hawashin, S. Alzubi, A. Mughaid, F. Fotouhi and A. Abusukhon, "An Efficient Cold Start Solution for Recommender Systems Based on Machine Learning and User Interests," 2020 Seventh International Conference on Software Defined Systems (SDS), Paris, France, 2020, pp. 220-225, doi: 10.1109/SDS49854.2020.9143953.
- [10] O. H. Embarak, "A method for solving the cold start problem in recommendation systems," 2011 International Conference on Innovations in Information Technology, Abu Dhabi, United Arab Emirates, 2011, pp. 238-243, doi: 10.1109/INNOVATIONS.2011.5893824.
- [11]Souptik Saha, S.Ramamoorthy, Eisha Raghav ” User Centric and Collaborative Movie Recommendation System Under Customized Platform” 3rd International Conference on Signal Processing and Communication (ICPSC) | 13 – 14 May 2021
- [12] Mostafa Khalaji1 Chitra Dadkhah1 Joobin Gharibshah “Hybrid Movie Recommender System based on Resource Allocation”, published on The CSI Journal on Computer Science and Engineering Vol. 17, No. 2, 2020
- [13] Jiang Zhang, Yufeng Wang , Zhiyuan Yuan, and Qun Jin “Personalized Real-Time Movie Recommendation System: Practical Prototype and Evaluation” Published in: TSINGHUA SCIENCE AND TECHNOLOGY ISSN11007-0214 02/12 pp180–191 DOI: 1 0. 2 6 5 9 9/T ST. 2 0 1 8. 9 0 1 0 1 1 8 V olume 2 5, Number 2, April 2020
- [14] Poonam Sharma, Lokesh Yadav “Movie Recommendation System Using Item Based Collaborative Filtering” Published in International Journal of Innovative Research in Computer Science & Technology (IJIRCST) ISSN: 2347-5552, Volume-8, Issue-4, July 2020
- [15] Nguyen Anh Khoa Dam and Thang Le Dinh b “A Literature Review of Recommender Systems for the Cultural Sector” Published in ICEIS 2020 - 22nd International Conference on Enterprise Information Systems. DOI: 10.5220/0009337807150726

APPENDIX

1. SVD_model.py

```
from surprise import SVD
from surprise import Dataset, Reader
from surprise.model_selection import cross_validate, train_test_split
import random
import numpy as np
import pandas as pd
import numpy as np
from typing import *
from scipy.spatial.distance import cosine
from surprise import accuracy
import warnings
import pickle
import pandas as pd
import matplotlib.pyplot as plt

def load_movie_dataset() -> pd.DataFrame:
    movie_data_columns = ['movie_id', 'title', 'release_date', 'video_release_date',
'url','unknown', 'Action', 'Adventure', 'Animation', "Children's",
    'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy', 'Film-Noir','Horror', 'Musical',
'Mystery', 'Romance', 'Sci-Fi', 'Thriller','War', 'Western']

    movie_data = pd.read_csv("E:\\VIII Sem\\Project-
UI\\movie_recommendation_system\\user_interface\\u.item", sep = '|', encoding = "ISO-
8859-1", header = None, names = movie_data_columns,index_col = 'movie_id')
    movie_data['release_date'] = pd.to_datetime(movie_data['release_date'])
    return movie_data

def load_rating_data() -> pd.DataFrame:
    ratings_data = pd.read_csv("E:\\VIII Sem\\Project-
UI\\movie_recommendation_system\\user_interface\\u.data",sep = '\t',encoding = "ISO-8859-
1",header = None,names=['user_id', 'movie_id', 'rating', 'timestamp'])
```



```

return ratings_data[['user_id', 'movie_id', 'rating']]

def load_user_data():
    user_data = pd.read_csv("E:\\VIII Sem\\Project-
UI\\movie_recommendation_system\\user_interface\\u.user",sep = '|', encoding = "ISO-8859-
1", header = None,names=['user_id', 'age', 'gender', 'profession','user_score'],index_col =
'user_id')
    return user_data

def coldstart():
    # Solving Cold Start problem using popularity method
    # Calculate mean rating for each movie
    mean_ratings = IMDB_df.groupby('movie_title').mean()['rating']

    # Sort mean ratings in descending order
    mean_ratings_sorted = mean_ratings.sort_values(ascending=False)

    # Get top 100 movies with highest mean rating
    top_10_movies = mean_ratings_sorted.iloc[:10]
    return top_10_movies

ratings_data = load_rating_data()
movies_data = load_movie_dataset()
user_data = load_user_data()
ratings_and_movies =
ratings_data.set_index('movie_id').join(movies_data['title']).reset_index()

IMDB_df = ratings_and_movies[['user_id','movie_id','title', 'rating']]
IMDB_df = IMDB_df.rename(columns={'title': 'movie_title'})

# Remove movies with few ratings
movie_ratings = IMDB_df.groupby('movie_title').size()
valid_movies = movie_ratings[movie_ratings > 50]

```

```
IMDB_df_filtered = IMDB_df.set_index('movie_title',
drop=False).join(valid_movies.to_frame(), how='inner').reset_index(drop=True)
del IMDB_df_filtered[0]
```

```
#shuffling the data
```

```
IMDB_df_filtered = IMDB_df_filtered.sample(frac=1)
```

```
IMDB_df = IMDB_df_filtered
```

```
df = IMDB_df
```

```
# Define the format of the data using the Reader class
```

```
reader = Reader(rating_scale=(1, 5))
```

```
# Load the dataframe into a Surprise dataset
```

```
data = Dataset.load_from_df(df[['user_id', 'movie_title', 'rating']], reader)
```

```
# Get the raw ratings from the data
```

```
raw_ratings = data.raw_ratings
```

```
# Create a dictionary to map user ids to their ratings for each item
```

```
user_item_dict = { }
```

```
for uid, iid, rating, _ in raw_ratings:
```

```
    if uid not in user_item_dict:
```

```
        user_item_dict[uid] = { }
```

```
    user_item_dict[uid][iid] = rating
```

```
# Convert the dictionary into a pandas dataframe
```

```
user_item_df = pd.DataFrame.from_dict(user_item_dict, orient='index')
```

```
# Print the resulting user-item matrix
```

```
user_item_df.fillna(0)
```

```
from numpy.core.function_base import linspace
```

```
X = IMDB_df['movie_id'].unique()
```

```
Y = IMDB_df.groupby('movie_title').count()['movie_id'].sort_values(ascending=False)[:10]
```

```
Y = Y.to_frame('count')
```

```
Y = Y.reset_index()
```

```
plt = Y.plot.bar(x = 'movie_title', y = 'count',yticks=[0,200,400,600,800])
```

```
plt = plt.figure
```

```
plt.savefig('plot.png')
```

```
rating_scale = (1, 5)
```

```
reader = Reader(rating_scale=rating_scale)
```

```
data = Dataset.load_from_df(IMDB_df[['user_id', 'movie_title', 'rating']], reader)
```

```
trainset, testset = train_test_split(data, test_size=.01)
```

```
model = SVD(n_factors=100)
```

```
model.fit(trainset)
```

```
print("Matrix pu:")
```

```
print(model.pu)
```

```
print("\nMatrix qi:")
```

```
print(model.qi)
```

```
print("\nMatrix s:")
```

```
# print(model.d)
```

```
U = model.pu # User matrix
```

```
V = model.qi # Item matrix
```

```
#normalize the data to compute cosine similarity later
```

```
q0_norm_squared = np.sum(V[0] ** 2)
#This line computes the squared L2-norm of the first row of the V matrix and stores it in the
variable q0_norm_squared.
```

```
V /= np.linalg.norm(V, ord=2, axis=1, keepdims=True)
#This line normalizes each row of the V matrix by dividing it by its L2-norm,
```

```
q0_norm_squared_normalized = np.sum(V[0] ** 2)
#This line computes the squared L2-norm of the first row of the normalized V matrix (which
is equivalent to the squared cosine similarity between the first item and all other items)
#and stores it in the variable q0_norm_squared_normalized.
```

```
def display(df: pd.DataFrame):
    item_to_row_idx_df = pd.DataFrame(list(item_to_row_idx.items()), columns=['Movie
name', 'V matrix row idx']).set_index('Movie name')
    return item_to_row_idx_df.iloc[:5]
```

```
item_to_row_idx: Dict[Any, int] = model.trainset._raw2inner_id_items
```

```
display(item_to_row_idx)
```

```
toy_story_row_idx : int = item_to_row_idx['Toy Story (1995)']
```

```
def get_vector_by_movie_title(movie_title: str, trained_model: SVD) -> np.array:
    """Returns the latent features of a movie in the form of a numpy array"""
    item_to_row_idx = trained_model.trainset._raw2inner_id_items
    movie_row_idx = item_to_row_idx[movie_title]
    return trained_model.qi[movie_row_idx]
```

```
def cosine_distance(vector_a: np.array, vector_b: np.array) -> float:
```

```

"""Returns a float indicating the similarity between two vectors"""
return 1-cosine(vector_a, vector_b)

movie_a = 'Toy Story (1995)'
movie_b = 'Jurassic Park (1993)'
vector_a = get_vector_by_movie_title(movie_a, model)
vector_b = get_vector_by_movie_title(movie_b, model)
similarity_score = 1-cosine_distance(vector_a, vector_b)
print(f"The similarity score between '{movie_a}' and '{movie_b}' is: {similarity_score:.4f}")

toy_story_vec = get_vector_by_movie_title('Toy Story (1995)', model)
wizard_of_oz_vec = get_vector_by_movie_title('Wizard of Oz, The (1939)', model)

similarity_score = 1-cosine_distance(toy_story_vec, wizard_of_oz_vec)

starwars_idx = model.trainset._raw2inner_id_items['Star Wars (1977)']
roj_idx = model.trainset._raw2inner_id_items['Return of the Jedi (1983)']
aladdin_idx = model.trainset._raw2inner_id_items['Aladdin (1992)']

starwars_vector = model.qi[starwars_idx]
return_of_jedi_vector = model.qi[roj_idx]
aladdin_vector = model.qi[aladdin_idx]

cosine_distance(starwars_vector, return_of_jedi_vector)

cosine_distance(starwars_vector, aladdin_vector)

def display(similarity_table):
    similarity_table = pd.DataFrame(
        similarity_table,
        columns=['vector cosine distance', 'movie title']
    ).sort_values('vector cosine distance', ascending=False)
    return similarity_table.iloc[:4]

```

```

def get_top_similarities(movie_title, model) -> pd.DataFrame:
    """
    Returns a DataFrame of the top 4 most similar movies to a given movie title,
    based on cosine similarity between their latent feature vectors.
    """

    movie_vector: np.array = get_vector_by_movie_title(movie_title, model)
    similarity_table = []

    for other_movie_title in model.trainset._raw2inner_id_items.keys():
        other_movie_vector = get_vector_by_movie_title(other_movie_title, model)

        similarity_score = cosine_distance(other_movie_vector, movie_vector)
        similarity_table.append((similarity_score, other_movie_title))

    return pd.DataFrame(
        sorted(similarity_table, reverse = True)[:6],
        columns=['vector cosine distance', 'movie title']
    )

print(get_top_similarities('Star Wars (1977)', model))

def multipleMovieRecommendation(user_input):
    movie_to_idx = { }
    for i, row in movies_data.iterrows():
        movie_to_idx[row['title']] = i

    user_df = pd.DataFrame.from_dict(user_input, orient='index', columns=['rating'])

    unseen_movies = movies_data[~movies_data['title'].isin(user_df.index)]

    unseen_ratings = pd.merge(unseen_movies, ratings_data, on='movie_id')[['title', 'user_id',
    'rating']]

```

```
unseen_ratings['predicted_rating'] = unseen_ratings.apply(lambda x:
model.predict(x['user_id'], movie_to_idx[x['title']]).est, axis=1)
```

```
recommendations = unseen_ratings.sort_values('predicted_rating',
ascending=False).head(10)['title'].tolist()
return recommendations
```

```
predictions = model.test(testset)
```

```
rmse = accuracy.rmse(predictions)
print("RMSE:", rmse)
```

```
mae = accuracy.mae(predictions)
print("MAE:", mae)
```

2. Views.py

```
from django.http import JsonResponse
from django.shortcuts import render
import pandas as pd
from .models import Movie, Ratings
from .SVDmodel import *
```

```
model = pickle.load(open("E:\\VIII Sem\\Project-
UI\\movie_recommendation_system\\user_interface\\SVD_model.pkl", 'rb'))
```

```
def load_movie_dataset():
    movie_data_columns = ['movie_id', 'title', 'release_date', 'video_release_date',
'url', 'unknown', 'Action', 'Adventure', 'Animation', "Children's",
'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical',
'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']
```

```

    movie_data = pd.read_csv("E:\\VIII Sem\\Project-
UI\\movie_recommendation_system\\user_interface\\u.item", sep = '|', encoding = "ISO-
8859-1", header = None, names = movie_data_columns)

    movie_data['release_date'] = pd.to_datetime(movie_data['release_date'])

    return movie_data

def dataset_to_databse():
    df = load_movie_dataset()
    df_records = df.to_dict('records')[1:]

    model_instances = []
    print(df_records[0])
    for record in df_records :

        movie_id = record['movie_id']
        title = record['title']

        if record['release_date'] != 'nan':
            release_date = record['release_date']
        else:
            release_date = ""

        if record['video_release_date'] != 'nan':
            video_release_date = record['video_release_date']
        else:
            video_release_date = ""

        if record['url'] != 'nan':
            url = record['url']
        else:
            url = ""
        genres = ""

```



```

    for genre in ['unknown', 'Action', 'Adventure', 'Animation', "Children's", 'Comedy',
'Crime', 'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical', 'Mystery',
'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']:
        if record[genre] == 1:
            genres += (genre + ', ')
        genres = genres[:-2]
        print(movie_id,title,release_date,video_release_date,url,genres)
        model_instances.append(Movie(movie_id=movie_id,title=title,release_date=release_date,video_release_date=video_release_date,url=url,genres=genres))
    Movie.objects.bulk_create(model_instances)

```

```

def home(request):
    return render(request,'home.html')

```

```

def addMovie(request):
    all_ratings = Ratings.objects.all()
    if request.method == 'POST':
        titles = [x.title for x in all_ratings]
        movie = request.POST['movie_title']
        rating = request.POST['rating']
        print(movie,rating)
        if movie not in titles:
            obj = Ratings(title=movie,rating=rating)
            obj.save()
        movies = Movie.objects.all()
        all_ratings = Ratings.objects.all()
        return render(request,'search.html',{'movies': movies,'all_ratings': all_ratings})

```

```

def search(request):
    print('Into search method')
    movies = Movie.objects.all()
    all_ratings = Ratings.objects.all()

```

```

if request.method == 'POST':
    print('Into post method')
    return render(request,'recommend.html',{ 'movies': movies,'all_ratings': all_ratings})
else:
    all_ratings = Ratings.objects.all()
    return render(request,'search.html',{ 'movies': movies,'all_ratings': all_ratings})

def deleteAll(request):
    print("inside deleteAll")
    obj = Ratings.objects.all()
    obj.delete()
    all_ratings = Ratings.objects.all()
    movies = Movie.objects.all()
    return render(request,'search.html',{ 'movies': movies,'all_ratings': all_ratings})

def delete(request,title):
    obj = Ratings.objects.filter(title=title)
    obj.delete()
    all_ratings = Ratings.objects.all()
    movies = Movie.objects.all()
    return render(request,'search.html',{ 'movies': movies,'all_ratings': all_ratings})

def rateMovie(request):
    all_ratings = Ratings.objects.all()
    movies = Movie.objects.all()
    return render(request,'rateMovie.html',{ 'movies': movies,'all_ratings': all_ratings})

def recommend(request):
    all_ratings = Ratings.objects.all()
    if request.method == 'POST':
        d={}
        if len(all_ratings) == 0:
            recommendations = coldstart()
            recommendations = pd.DataFrame(recommendations).transpose()

```

```

elif len(all_ratings) == 1:
    recommendations = get_top_similarities(all_ratings[0].title,model)
    recommendations = recommendations['movie title'].tolist()
else:
    for movie in all_ratings:
        d[movie.title] = movie.rating
    recommendations = multipleMovieRecommendation(d)
return render(request,'recommendations.html',{'recommendations': recommendations})

```

3. models.py

```

from django.db import models

# Create your models here.

from django.db import models

from django.db.models.fields import DateField

# movie_data_columns = ['movie_id', 'title', 'release_date', 'video_release_date',
'url', 'unknown', 'Action', 'Adventure', 'Animation', "Children's",
# 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical',
'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']

class Movie(models.Model):

    movie_id = models.IntegerField()

    title = models.CharField(max_length=150)

    release_date = models.CharField(default=None,max_length=150)

    video_release_date = models.CharField(default=None,max_length=150)

    url = models.CharField(max_length=500)

    genres = models.CharField(max_length=500)

```

```
language = models.CharField(max_length=150)
```

```
avg_rating = models.FloatField(default=0.0)
```

```
def __str__(self):
```

```
    return self.title
```

```
class Ratings(models.Model):
```

```
    title = models.CharField(max_length=150)
```

```
    rating = models.IntegerField(default=0)
```

```
def __str__(self):
```

```
    return self.title
```

4. urls.py

```
from django.contrib import admin
```

```
from django.urls import path, include
```

```
from user_interface import views
```

```
urlpatterns = [
```

```
    path("", views.home, name = 'home'),
```

```
    path('home/', views.home, name = 'home'),
```

```
    path('search/', views.search, name = 'search'),
```

```
    path('rateMovie/', views.rateMovie, name='rateMovie'),
```

```
    path('recommendations/', views.recommend, name='recommendations'),
```

```
    path('deleteAll/', views.deleteAll, name='deleteAll'),
```

```
    path('addMovie/', views.addMovie, name='addMovie'),
```

```
path('delete/<str:title>',views.delete,name='delete'),
```

```
]
```

5. Home.html

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="utf-8">
```

```
<meta content="width=device-width, initial-scale=1.0" name="viewport">
```

```
<title>Movie Recommendation System</title>
```

```
<meta content="" name="description">
```

```
<meta content="" name="keywords">
```

```
<!-- Google Fonts -->
```

```
<link
```

```
href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,600,600i,700,700i|Nunito:300,300i,400,400i,600,600i,700,700i|Poppins:300,300i,400,400i,500,500i,600,600i,700,700i" rel="stylesheet">
```

```
<link href="https://fonts.googleapis.com/icon?family=Material+Icons"rel="stylesheet">
```

```
<!-- Template Main CSS File -->
```

```
<link href="/static/styles.css" rel="stylesheet">
```

```
</head>
```

```
<body>
```

```
<!-- ===== Header ===== -->
```

```
<header id="header" class="fixed-top ">
```

```
<div class="navbar">
```

```
<h1 class="logo"><a href="{% url 'home'%}">Movie Recommendation  
System</a></h1>
```

```

<div class="navbarLinks">

    <ul>

        <li><a class="nav-link scrollTo" href="{% url 'home' %}">Home</a></li>

        <li><a class="nav-link scrollTo" href="#about">About</a></li>

        <li><a class="nav-link scrollTo" href="{% url 'search' %}">Rate Movies</a></li>

        <li><a class="nav-link scrollTo" href="#team">Team</a></li>

    </ul>

    <i class="bi bi-list mobile-nav-toggle"></i>

</div><!-- .navbar -->

</div>

</header><!-- End Header -->

{% block main %}

<!-- ===== Hero Section ===== -->

<section id="hero">

    <div class="hero-container">

        <h3><strong>Welcome !</strong></h3>

        <h1>We recommend movies to users</h1>

        <h2>Our objective is to create a recommendation system which solves data sparsity,
scalability and cold start problems</h2>

        <a href="{% url 'search' %}" class="btn-get-started scrollTo">Get Started</a>

    </div>

</section><!-- End Hero -->

{% endblock main %}

</body>

</html>

```