Movie Recommendation System

Documentation

Final Year Project Report

Date: April 12, 2025

Department of Computer Science
Academic Year: 2024–2025

Contents

1 Introduction

Recommender systems are powerful tools used to suggest relevant items to users based on their preferences and behavior. Our Movie Recommendation System employs a Machine Learning approach using the K-Nearest Neighbors (KNN) algorithm from the Scikit-Learn library.

1.1 Background

Recommender systems are crucial in today's data-driven world, enabling personalized experiences for users by predicting their preferences and interests. They are extensively used in e-commerce, streaming platforms, and online education to suggest products, movies, or courses based on users' behaviors and preferences.

The purpose of this project is to develop a Machine Learning powered movie recommendation system that helps users discover movies based on their preferences. This system will utilize machine learning techniques to recommend movies by analyzing genres, keywords, cast, crew, and movie overviews.

- There are two primary types of movie recommendation approaches:

1. **Content-Based Filtering:** Suggests movies based on the attributes of items (e.g., genres, directors) that a user previously liked.
2. **Collaborative Filtering:** Relies on user-item interactions, predicting preferences based on similar users or items.

- For this project, we will focus on **Content-Based Filtering** due to its ability to generate personalized recommendations without requiring extensive user data.

1.2 Scope

The system is designed for movie enthusiasts, streaming platforms, and small businesses in the entertainment sector. It aims to provide personalized recommendations by processing movie metadata and utilizing content-based filtering. It will enhance user engagement by suggesting movies tailored to individual tastes.

Limitations:

- The system will not include advanced collaborative filtering or hybrid recommendation models.
- Data sources will be limited to pre-processed movie datasets (e.g., Kaggle datasets).

1.3 Objectives

- Create a comprehensive dataset by merging and preprocessing movie metadata.
- Build a recommendation model using machine learning techniques.
- Deliver an intuitive and user-friendly interface for seamless interaction.
- Ensure system scalability, reliability, and performance.

1.4 How KNN Works

KNN is a simple yet effective algorithm that identifies the k nearest data points to a given input. In our system:

- We use the TF-IDF (Term Frequency-Inverse Document Frequency) matrix to represent movie data numerically based on textual features like tags and genres.
- Cosine similarity is calculated to measure the proximity between movie vectors in the TF-IDF space.
- The system identifies movies closest to the input movie and recommends them.

1.5 Why TF-IDF?

TF-IDF helps convert textual data into numerical vectors by emphasizing unique and relevant terms while reducing the weight of common terms. This ensures accurate similarity calculations.

1.6 Recommendation Process

The system processes the input movie as follows:

1. The user inputs a movie title.
2. The TF-IDF matrix is queried for the vector corresponding to the movie.
3. Using the KNN model, the system identifies the nearest neighbors.
4. The details of these recommended movies are fetched from the TMDB API and displayed to the user.

1.7 KNN Similarity Visualization Using PCA Components

To better understand how the K-Nearest Neighbors (KNN) algorithm selects similar movies, we use a 2D visualization of the dataset where movie vectors are reduced to two principal components using Principal Component Analysis (PCA). The selected movie is represented by a red dot, and its nearest neighbors are shown as blue dots, while the rest of the movie dataset is plotted in gray.

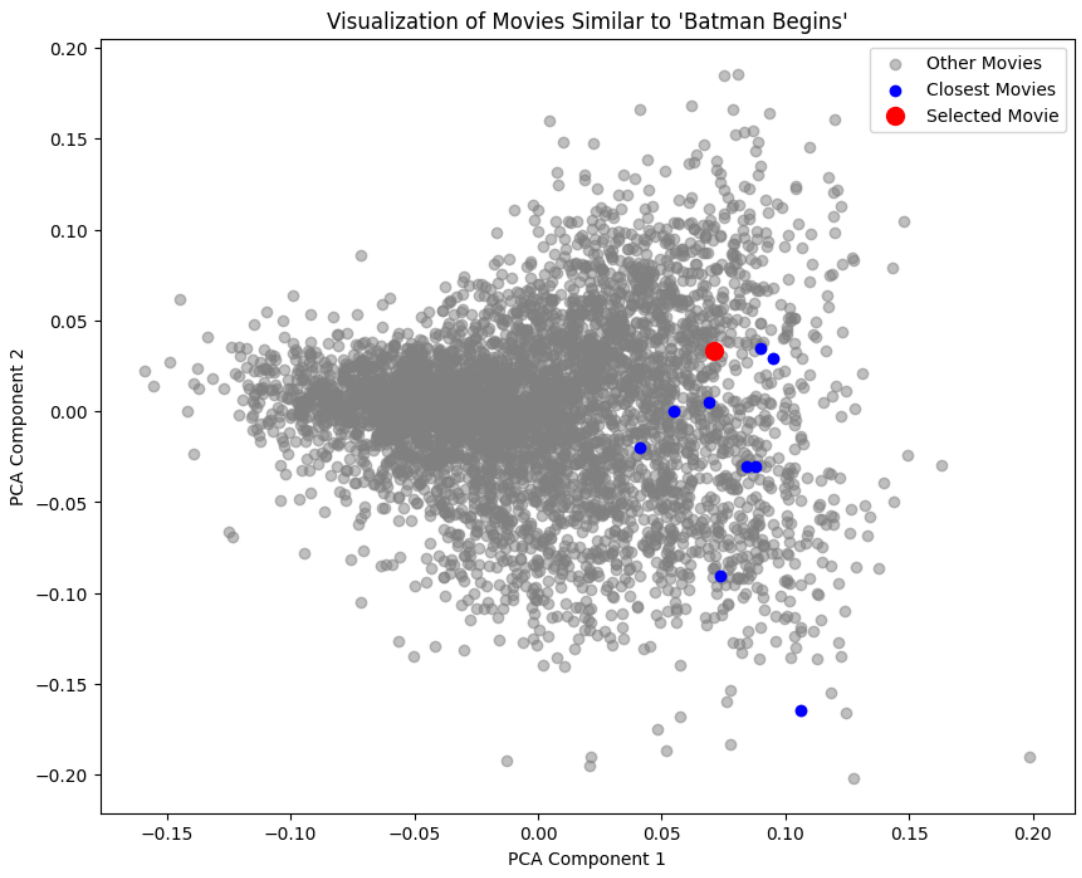

Figure 1: 2D PCA Scatter Plot of Movies with KNN Highlighting

2 Dataset

2.1 Source

The dataset used is the *TMDB Top 5000 Movies Dataset*, which contains detailed movie metadata.

2.2 Data Types

The dataset includes two files:

- **Movies.csv:** Contains columns like `budget`, `genres`, `title`, `keywords`, `popularity`, `runtime`, and `release_date`.
- **Credits.csv:** Contains `movie_id`, `title`, `cast`, and `crew`.

3 Preprocessing Steps

1. Download `Movies.csv` and `Credits.csv` from the Kaggle dataset.

2. Merge `Movies.csv` and `Credits.csv` on the `movie_id` column.
3. Retain only necessary columns: `movie_id`, `title`, `genres`, `keywords`, `overview`, `cast`, and `crew`.
4. Remove rows with null or duplicate values in the selected columns.
5. Convert dictionary data in columns (e.g., `genres`, `keywords`) into a list of words.
6. Retain only the top 3 cast members and the director from the `crew` column.
7. Concatenate `overview`, `genres`, `keywords`, `cast`, and `crew` into a single `tags` column.
8. Convert all `tags` to lowercase and remove spaces for uniformity.

4 Functional Requirements

1. **Data Integration:** Merge and preprocess datasets containing movie metadata and remove unnecessary or duplicate data.
2. **Recommendation Functionality:** Generate personalized movie recommendations based on user input using content-based filtering.
3. **Search and Retrieval:** Allow users to search movies by title or tags and retrieve detailed movie information.
4. **Model Training and Updates:** Implement a machine learning model to analyze movie data and periodically update the model to improve accuracy.

5 Non-Functional Requirements

- **Performance:** The system should respond to user queries in minimum time and handle more users without performance degradation.
- **Scalability:** Support seamless integration of new movie data and scale up as the number of users increases.
- **Usability:** Provide a clean and intuitive interface for all user types, ensuring features are easy to navigate.

6 Front-End Interface Design

6.1 Key Screens

- **Search Screen:** Allows users to input a movie name and request recommendations.
  - **Inputs:** A search bar for movie names and a *Recommend* button.
  - **Outputs:** List of recommended movies with titles, posters, and ratings.
- **Recommendations Screen:** Displays movie recommendations based on user input.
  - **Outputs:** Movie details (title, poster, and rating) in a visually appealing grid layout.
- **Movie Details Screen:** Displays detailed information for a selected movie, including the top three cast members and the director's name.

6.2 Visual Mockup

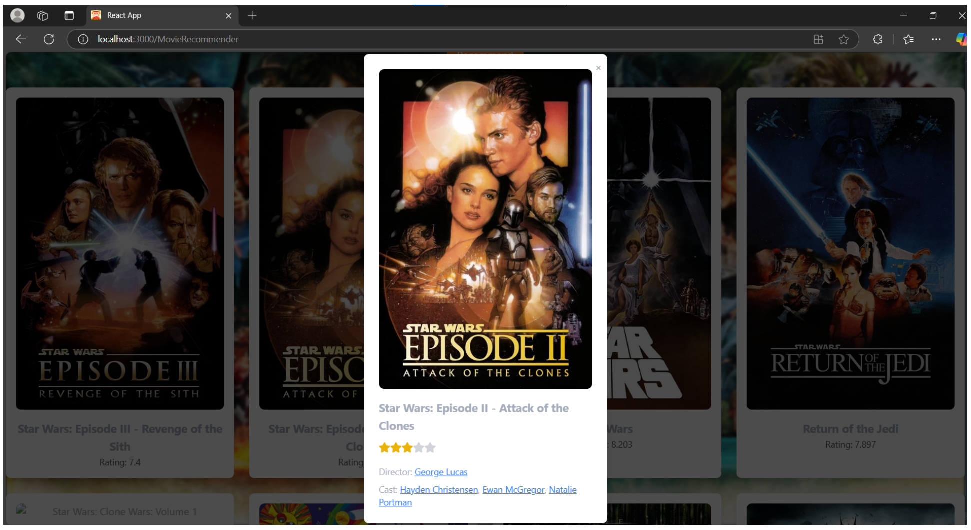
React App
localhost:3000/MovieRecommender
Star Wars: Episode III - Revenge of the Sith
Rating: 7.4
Star Wars: Episode II - Attack of the Clones
Director: George Lucas
Cast: Hayden Christensen, Ewan McGregor, Natalie Portman
Rating: 8.203
Return of the Jedi
Rating: 7.897
Star Wars: Clone Wars: Volume 1

Figure 2: UI Mockup

7 Sequence Diagram

7.1 Description

The sequence diagram demonstrates the flow of interactions between the user, frontend, backend, and external APIs. It outlines the following steps:

1. The user inputs a movie name on the React-based interface and clicks *Recommend*.
2. The frontend sends a POST request to the Flask backend with the movie title.
3. The backend verifies the input, retrieves the movie index, and computes recommendations using the KNN algorithm.

4. For each recommended movie, the backend fetches details (poster, rating) from the TMDB API.
5. The backend sends the recommendations to the frontend.
6. The frontend displays the results to the user.

7.2 Diagram

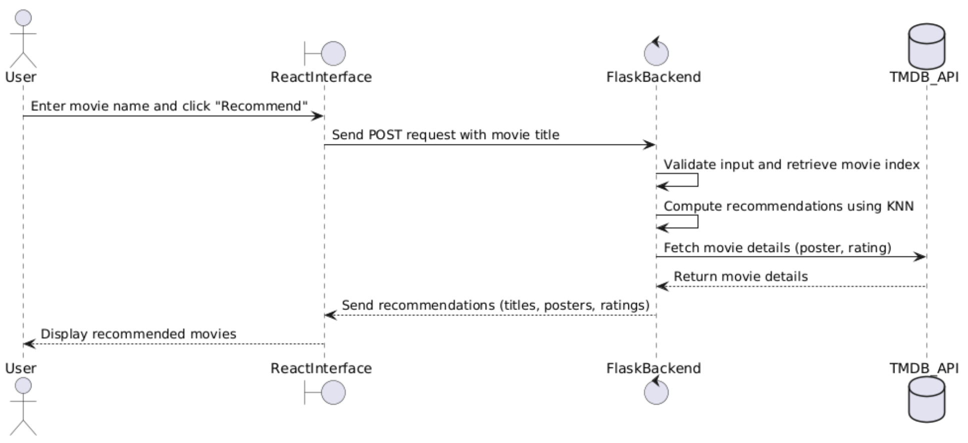
User
ReactInterface
FlaskBackend
TMDB_API
Enter movie name and click "Recommend"
Send POST request with movie title
Validate input and retrieve movie index
Compute recommendations using KNN
Fetch movie details (poster, rating)
Return movie details
Send recommendations (titles, posters, ratings)
Display recommended movies
User
ReactInterface
FlaskBackend
TMDB_API

Figure 3: Sequence Diagram

8 Class Diagram

8.1 Description

The class diagram illustrates the structure of the system, highlighting the relationships between different components. The main classes are:

- **MovieRecommender:** Manages user input, fetches recommendations, and updates the UI.
- **FlaskBackend:** Handles requests from the frontend, processes data using machine learning models, and communicates with external APIs.
- **TMDB API:** Fetches movie details such as posters and ratings.

8.2 Diagram

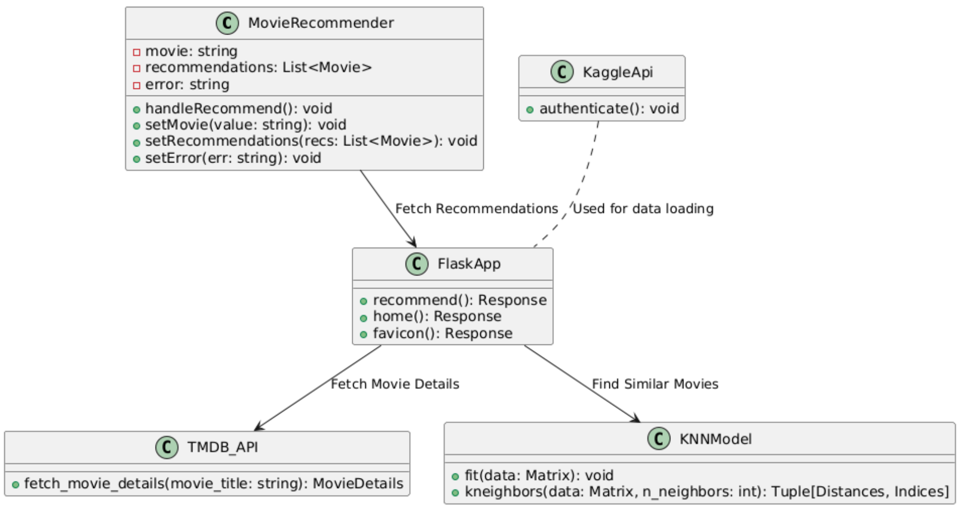

Figure 4: Class Diagram

9 Database Design (ER Diagram)

9.1 Description

The ER Diagram models the data relationships in the movie recommendation system. Key entities include:

- **Movie:** Contains attributes like movie ID, title, genres, and TF-IDF vector.
- **Recommendation:** Stores recommended movies for a given input movie, linked via movie ID.
- **API Response:** Captures data fetched from the TMDB API, such as poster URL and rating.
- **User Input:** Tracks user interactions, including movie titles entered and any errors encountered.

9.2 Diagram

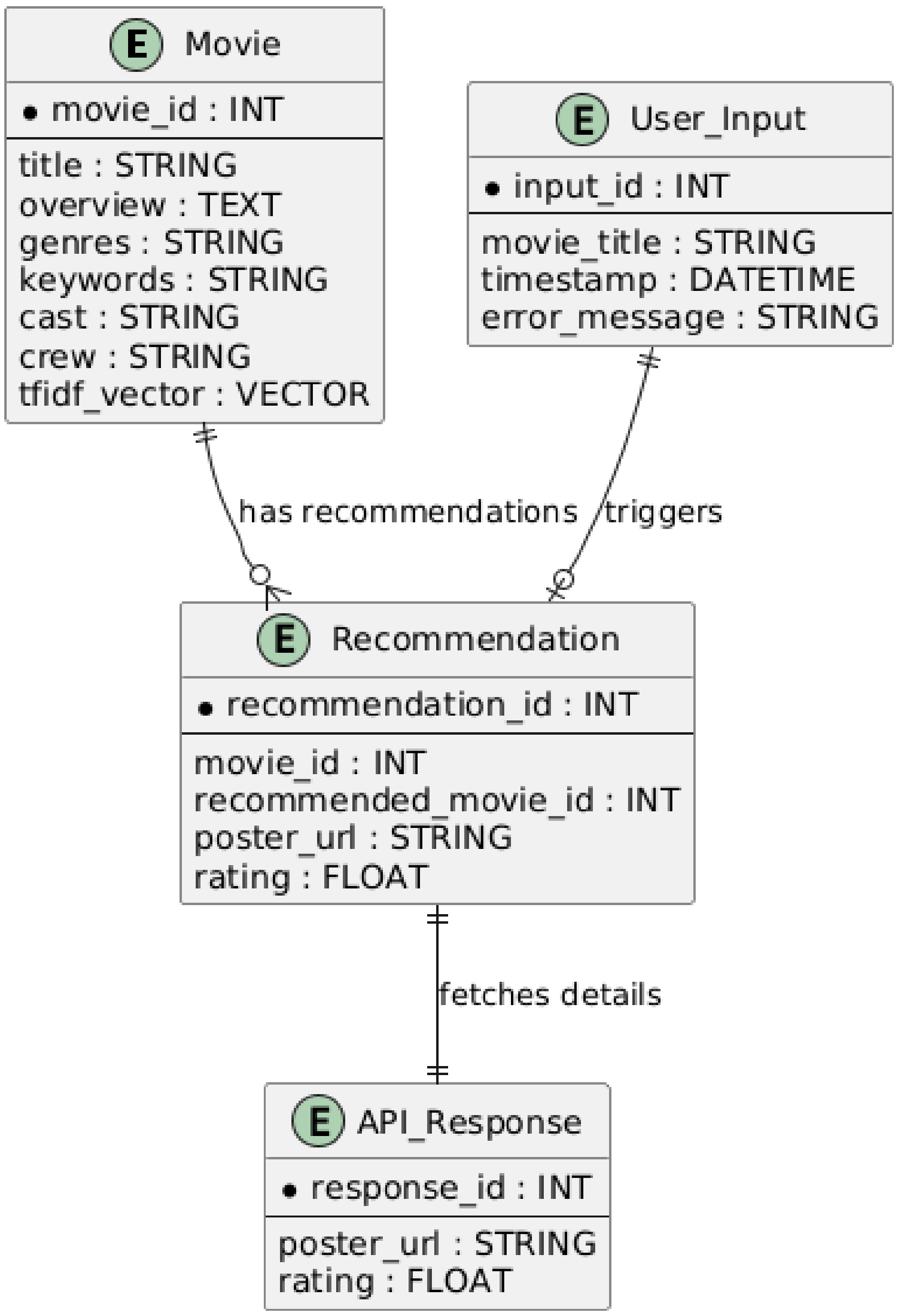

Figure 5: Database Design (ER Diagram)

10 System Architecture Diagram

10.1 Description

The system architecture diagram showcases the interaction between different components:

- **Frontend (React):** User interface for inputs and displaying results.

- **Backend (Flask):** Handles logic, data processing, and communication with APIs.
- **Machine Learning Model:** Processes movie data using TF-IDF and KNN for recommendations.
- **TMDB API:** Provides additional data like posters and ratings for movies.
- **Data Storage:** Includes the TF-IDF matrix and movie metadata stored in .npz and .csv formats.

10.2 Diagram

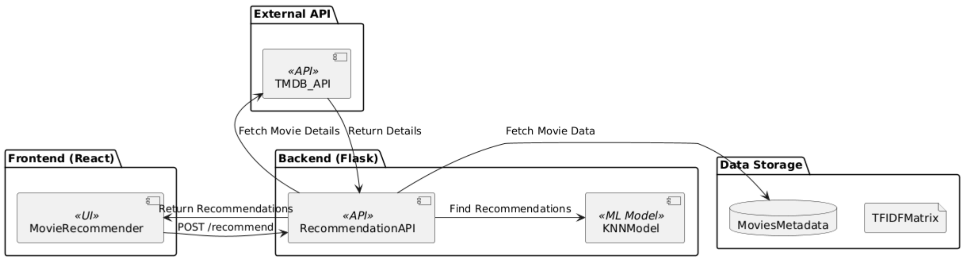

Figure 6: System Architecture Diagram

11 Conclusion

The Movie Recommender System is a comprehensive solution for providing personalized movie recommendations. It integrates state-of-the-art machine learning techniques with a user-friendly interface and third-party APIs to enhance the user experience. The modular design ensures scalability and maintainability, making it suitable for future enhancements like integrating user profiles and advanced recommendation algorithms.