VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi-590018

A

MINI PROJECT REPORT ON

"MOVIE RECOMMENDATION SYSTEM"

Submitted in partial fulfillment Management of the Bachelor Degree

In

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

V SEMESTER

Mini Project (BAI586)

Submitted by

KESHAV M	**1HK22AI020**
SONVI NORONHA	**1HK22AI053**
SOURABH KHOT	**1HK22AI054**
SUMITH CHOUGALE	**1HK22AI056**

Under the guidance of

SHABEENA LYLATH

Department of Artificial Intelligence and Machine Learning

2024-2025

HKBK COLLEGE OF ENGINEERING

(Approved by AICTE & Affiliated to VTU)

22/1, Opp.Manyata Tech Park Rd, Nagawara, Bangaluru-560045

Email: info@hkbk.edu.in URL: www.hkbk.edu.in

HKBK COLLEGE OF ENGINEERING

Department of Artificial Intelligence and Machine Learning

CERTIFICATE

This is to certify that the mini project entitled **"MOVIE RECOMMENDATION SYSTEM"** is a Bonafide`work conducted by **KESHAV M [1HK22AI020], SONVI NORONHA[1HK22AI053],SOURABH KHOT[1HK22AI054],SUMITH CHOUGALE [1HK22AI056]** in partial fulfilment for the award of Degree of Bachelor of Engineering in Artificial Intelligence and Machine Learning of the Visvesvaraya Technological University, Belagavi during the year 2024-25. It is certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the mini project report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements in respect of **MINI PROJECT (BAI586)** prescribed for the Bachelor of Engineering Degree.

SHABEENA LYLATH	**Dr. Tabassum Ara**	**Dr. Mohammad Riyaz Ahmed**
Guide Department of AI&ML HKBKCE, Bengaluru	H.O. D Department of AIML HKBKCE, Bengaluru	Principal HKBKCE, Bengaluru

DECLARATION

We hereby declare that the entire work embodied in this Project work **"MOVIE RECOMMENDATION SYSTEM"** has been carried out by us during the 5th semester of Bachelor of Engineering in Artificial Intelligence and Machine Learning at HKBK College of Engineering, Bengaluru, affiliated to Visvesvaraya Technological University, Belagavi, under the guidance of **SHABEENA LYLATH**, HKBK College of Engineering, Bengaluru. The work embodied in this project work is original and it has not been submitted in part time or full-time completion for any other degree in any other university.

KESHAV M	**1HK22AI020**
SONVI NORONHA	**1HK22AI053**
SOURABH KHOT	**1HK22AI054**
SUMITH CHOUGALE	**1HK22AI056**

ACKNOWLEDGEMENT

We would like to express our gratitude and appreciation to everyone who contributed to the success of this initiative. The persons named below are just a handful of the many that worked behind the scenes to make it possible.

We would take this opportunity to express our heartfelt gratitude to **Mr. C.M. Ibrahim**, Chairman, HKBKCE, **Mr. C.M. Faiz Mohammed**, Director, HKBKCE, and **Dr. Mohammed Riyaz Ahmed**, Principal, HKBKCE, for the entire infrastructure provided to complete the project in time.

We are deeply indebted to **Dr. Tabassum Ara**, HOD, Artificial Intelligence and Machine Learning for the ineffable encouragement she provided in the successful completion of the project.

A special and earnest word of thanks to the project coordinator **Prof. Shruthi V Kulkarni** and project guide **SHABEENA LYLATH** for their constant assistance, support, patience, endurance, and constructive suggestions for the betterment of the project.

A special and earnest word of thanks to the Project Coordinator and the Committee members for their constant support.

We are extremely thankful to the teaching and non-teaching staff of the Department of Artificial Intelligence and Machine Learning for their valuable guidance and cooperation throughout our dissertation.

We thank our parents for their support and guidance provided to us to finish the project well ahead of time. We thank our friends who lent their support in every way possible to make sure the project has been completed.

ABSTRACT

The recommendation system of today has made getting the stuff we need simple. The Recommendation systems are used to help people make decisions about movies to assist movie fans by making recommendations for movies to watch without the burden reducing the time-consuming process of choosing films. There are different ways, or we can say techniques and most of the OTT platforms and other movies sites depend upon the collaborative filtering technique (CB) which has some problems like cold start problem, scalability issue, etc. In this paper we aim to make movie recommendations based on the user's interests and preferences, we want to reduce the amount of human effort required.

We developed a model based on a content-based approach and sentiment analysis. This system suggests movies by comparing examples supplied by the user to the contents of the movies. It does this without using any human-generated metadata and instead uses information about the director, cast, and genre of the movies as well as information about how positive or negative the reviews are plus also give additional details about the films you searched for. The rating of the film, its premiere date, cast, and genres are among the supplementary information.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATION

1) **MRS** - Movie Recommender System (Title of the project).
2) **ML -** Machine Learning (used for the recommendation logic).
3) **PKL**-Pickle (file format for saving serialized Python objects like models and data).
4) **API -** Application Programming Interface (used for fetching movie posters from the TMDB API).
5) **TMDB -** The Movie Database (API used for movie data and posters).
6) **ST -** Streamlit (Python library used for building the web application).
7) **JSON -** JavaScript Object Notation (format for fetching and parsing data from the TMDB API

CHAPTER 1

INTRODUCTION

INTRODUCTION

1.1 Introduction to Movie Recommendation System

In the era of digital transformation, where multimedia content is abundantly available, navigating the vast ocean of movies has become a challenge for users. The advent of recommendation systems has alleviated this challenge by offering personalized suggestions based on user preferences. A Movie Recommendation System leverages algorithms and data to predict and recommend movies that align with a user's tastes, enhancing their viewing experience.

This project focuses on developing a Movie Recommender System using the K-Nearest Neighbors (KNN) algorithm integrated with an intuitive frontend powered by Streamlit. The system is designed to provide users with movie suggestions based on selected criteria, such as their favorite movie or preferred genres, while also fetching detailed information and visuals for each recommendation.

1.2 Background and Motivation

The movie industry generates an overwhelming amount of content yearly, making it difficult for users to manually discover movies that match their interests. Platforms like Netflix, Amazon Prime, and IMDb have set a benchmark by using advanced recommendation engines that enhance user engagement. Inspired by these systems, the Movie Recommender System aims to simplify the movie selection process by suggesting relevant content based on user input.

The system not only recommends movies but also provides additional metadata such as IMDb ratings, plot summaries, director and cast details, and even movie posters. This comprehensive approach ensures a user-friendly and informative experience, encouraging users to explore movies outside their usual preferences.

1.3 Objectives

1. **Personalized Movie Recommendations**:

- To recommend movies based on user-selected criteria like favorite movie or genres.
- To enhance personalization using the KNN algorithm to find similarities between movies.

2. **Information Enrichment**:

- To provide users with detailed information about each recommended movie, including its director, cast, plot summary, and IMDb rating.
- To fetch and display visually appealing movie posters to create an engaging interface.

3.Interactive User Experience:

- To create an interactive and accessible interface using Streamlit.
- To allow users to customize recommendations by adjusting parameters such as genres, IMDb scores, and the number of results.

1.4 Methodology

The system operates through the following steps:

1. **Data Preparation**:
   - Movie data, including genres, IMDb ratings, and titles, is preprocessed and stored in JSON files.
   - The data acts as the foundation for generating recommendations.
2. **Recommendation Engine**:
   - The KNN algorithm is employed to calculate similarities between movies based on user-defined test points.
   - For movie-based recommendations, the system identifies movies similar to a selected movie.
   - For genre-based recommendations, the system matches movies to selected genres and IMDb score thresholds.
3. **Data Enrichment**:
   - Real-time data, such as movie posters and descriptions, is fetched from IMDb using web scraping with BeautifulSoup.
   - This enriches the recommendations with up-to-date information.
4. **Frontend Implementation**:
   - Streamlit is used to create an interactive and visually appealing interface.
   - Users can easily select options, view recommendations, and explore additional details about each movie.

1.5 Significance

The Movie Recommender System bridges the gap between users and the content they might love but are unaware of. It uses data-driven techniques to simplify decision-making in a world overwhelmed by choices. Additionally, by integrating an interactive frontend with a robust backend algorithm, the system provides both functionality and usability.

This project serves as a foundation for more advanced systems that can incorporate user profiles, real-time feedback, and collaborative filtering techniques. With its modular design and ease of use, the system is a stepping stone toward building scalable and sophisticated recommendation platforms.

CHAPTER 2

LITERATURE SURVEY

Literature Survey on Movie Recommendation System

Recommendation systems have become a cornerstone of digital platforms, especially in the entertainment industry, where they assist users in navigating extensive content libraries. The development of these systems has been guided by advancements in data science, machine learning, and user interface design. This literature survey explores the theoretical foundations, existing methodologies, and their evolution in the context of movie recommendation systems.

2.1 Introduction to Recommendation System

Recommendation systems aim to suggest items, such as movies, books, or products, based on user preferences. They can be categorized into the following types:

- **Content-Based Filtering**: Recommends items similar to what the user has interacted with in the past, based on item attributes.
- **Collaborative Filtering**: Suggests items based on the preferences of other users with similar interests.
- **Hybrid Systems**: Combines content-based and collaborative filtering approaches for enhanced performance.

The choice of methodology depends on the application domain, available data, and specific requirements of the system.

2.2 Content-Based Filtering

Content-based systems rely on item features and user preferences. For movies, features may include:

Genres: Categories like Action, Comedy, Drama, etc.

Cast and Crew: Actors, directors, and writers involved in the movie.

Plot Summaries: Textual descriptions summarizing the movie's story.

IMDb Ratings: Quantitative scores representing the movie's popularity and quality.

Studies:

1. Lops et al. (2011) introduced content-based approaches emphasizing the use of textual and categorical features for personalized recommendations.
2. Pazzani & Billsus (2007) highlighted the importance of attribute selection and similarity metrics, such as cosine similarity, for effective recommendations.

2.3 Collaborative-Based filtering

Collaborative filtering systems predict user preferences based on the preferences of others. Two main approaches are:

- **User-Based Collaborative Filtering**: Finds users with similar behavior patterns.
- **Item-Based Collaborative Filtering:** Identifies items that are frequently rated or interacted with together.

Studies:

1. **Sarwar et al. (2001)** demonstrated that collaborative filtering performs well in scenarios where user-item interaction data is abundant.

2. **Koren et al. (2009)** applied matrix factorization techniques, such as Singular Value Decomposition (SVD), to improve scalability and accuracy.

2.4 Hybrid Recommendation Systems

Hybrid systems combine content-based and collaborative filtering approaches to address limitations like cold-start problems (when a user or item has no prior data). For instance:

- A movie recommendation system may use genres (content-based) and ratings (collaborative filtering) together for suggestions.

Studies:

1. **Burke (2002)** provided a detailed taxonomy of hybrid systems and demonstrated their effectiveness in overcoming the drawbacks of individual techniques.
2. **Zhou et al. (2010)** emphasized that hybrid systems are essential for domains like movie recommendations, where user behavior and item attributes both play critical roles

CHAPTER 3

SYSTEM ANALYSIS AND SPECIFICATION

3.1 SYSTEM ANALYSIS

3.1.1 Existing System

The previously existing systems for movie recommendation relied primarily on basic or standalone approaches, which included the following:

1. **Manual Recommendations**:
   - Users often depended on curated lists, such as IMDb's "Top 250" or movie suggestions shared on social media platforms.
   - These recommendations lacked personalization, as they were not tailored to individual preferences.
2. **Static Filtering Systems**:
   - Platforms used simple category filters (e.g., genre or release year) to allow users to browse movies.
   - While helpful, these systems required manual searching and provided no intelligence in suggesting relevant content.
3. **Rating-Based Systems**:
   - Early systems used global ratings (e.g., IMDb scores) to rank movies.
   - Users could sort movies by rating, but the recommendations were generic and did not account for user-specific preferences.
4. **Basic Algorithmic Systems**:
   - Systems employed basic content-based filtering where movie attributes like genres, actors, or directors were matched with user preferences.
   - Limited to the available data in static databases, they lacked real-time updates or metadata enrichment.
5. **Challenges**:
   - **Lack of Personalization**: Recommendations were generic and failed to adapt to individual tastes.
   - **Limited User Engagement**: Absence of visuals like posters or real-time metadata made the systems less engaging.
   - **Static Nature**: Early systems did not dynamically fetch or update information, leading to outdated results.

3.1.2 Proposed System

The newly proposed **Movie Recommender System** addresses the limitations of existing systems by introducing a combination of advanced recommendation techniques, real-time metadata integration, and an interactive user interface. Key features of the proposed system include:

1. **Personalized Recommendations**:
   - The system uses a **K-Nearest Neighbors (KNN) algorithm** to recommend movies based on:
     - **Movie-Based Approach**: Suggests movies similar to a user-selected movie.
     - **Genre-Based Approach**: Recommends movies based on user-selected genres and IMDb score thresholds.
2. **Dynamic Metadata Enrichment**:
   - The system fetches real-time data from IMDb using web scraping with **BeautifulSoup**. This includes:
     - **Movie Posters**: High-quality posters are retrieved and displayed for each recommendation.
     - **Movie Information**: Metadata such as the director, cast, plot summary, and IMDb ratings is dynamically fetched.
3. **Interactive User Interface**:
   - Developed using **Streamlit**, the frontend allows users to:
     - Select movies or genres as the basis for recommendations.
     - Adjust the number of results and score thresholds using sliders and input fields.
     - View recommendations in an engaging format with embedded visuals and metadata.
4. **Improved Efficiency and Usability**:
   - Users can fetch detailed information about recommended movies without navigating away from the system.
   - Recommendations are enriched with real-time metadata, offering an up-to-date and visually appealing experience.
5. **Use of Structured Data**:
   - Movie data and titles are preprocessed and stored in JSON format, making it easy to query and update.
   - The recommendation engine utilizes these structured datasets to compute similarities effectively.

6. **Scalable and Modular Design**:
   - The system is designed to accommodate future enhancements, such as:
     - Incorporating user profiles for collaborative filtering.
     - Integrating APIs like TMDb for more robust and scalable data fetching.

Advantages of the Proposed System

1. **Personalization**: Tailors recommendations to individual user preferences.
2. **Rich User Experience**: Enhances engagement with posters, ratings, and real-time information.
3. **Dynamic Updates**: Provides the latest information by scraping live data from IMDb.
4. **Ease of Use**: A clean and interactive Streamlit interface makes it accessible for all users.
5. **Customizability**: Users can control parameters like genres, number of recommendations, and IMDb score.

3.2 SYSTEM REQURIMENTS

3.2.1 Hardware Requirements

Laptop Model: Acer Aspire 5

Processor: Intel i5-13420H (High-performance multi-core processor for fast computations)

Graphics Card: NVIDIA RTX 2050 (Capable of accelerating machine learning tasks and handling large-scale computations)

RAM: 16GB (Ensures smooth execution of data-heavy operations and parallel processing)

Storage: 512GB SSD (Fast read/write speeds for efficient data access and storage)

3.2.2 Software Requirements

These outline the tools and technologies required to build and run the system.

Software Dependencies:

- **Python 3.x**: Core language for backend and data processing.
- **Libraries**:
  - **Streamlit**: For building the user interface.
  - **NumPy**: For numerical computations in the KNN algorithm.
  - **BeautifulSoup (bs4)**: For web scraping metadata from IMDb.
  - **Requests**: For sending HTTP requests to IMDb.
  - **Pillow (PIL)**: For processing and displaying movie posters.
  - **io**: For handling data streams (e.g., images).
- **JSON**: For storing and loading movie data and titles.

3.2.3 Environment Setup:

- **Operating System**: Windows/Linux/MacOS.
- **IDE**: Any Python-compatible IDE (e.g., PyCharm, VS Code).

- **Dependencies Installation**:
  - Install required Python libraries using pip:

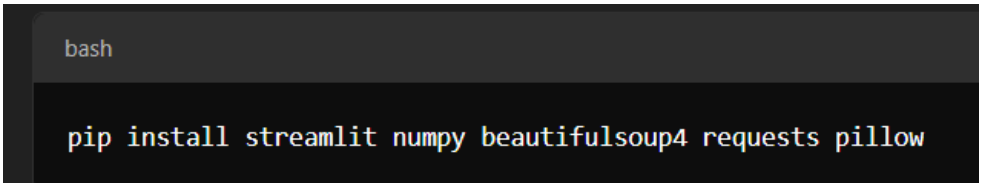

Fig:3.1

3.2.4 Optional Tools:

- **Version Control**: Git for version tracking and collaboration.
- **Browser**: Modern web browser (e.g., Chrome, Firefox) for Streamlit interface.

3.3 SYSTEM ARCHITECTURE

E-R DIAGRAM:

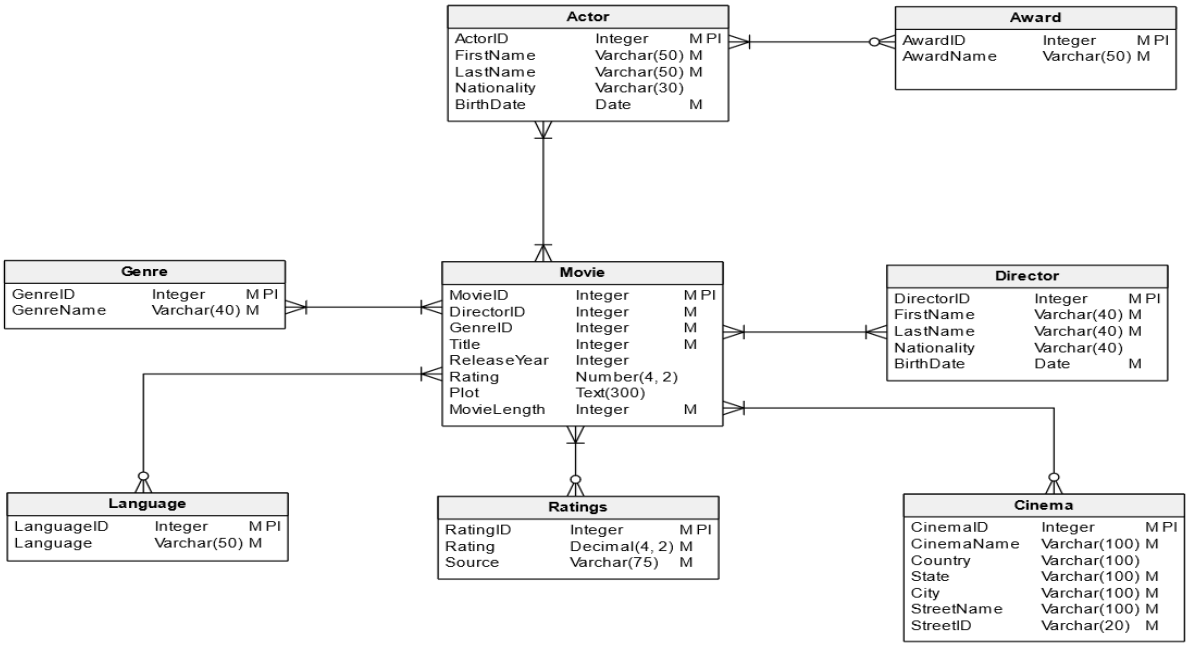

Fig: 3.2

The ER diagram consists of the following

- **Entities:**
  - Entities represent real-world objects or concepts about which you want to store information.
  - They are typically represented by rectangles.
  - Examples of entities include:
    - Customers
    - Products
    - Employees
    - Orders
    - Movies (in our case)

- **Attributes:**
  - Attributes are the properties or characteristics of an entity.
  - They are represented by ovals or ellipses.
  - Examples of attributes:
    - Customer ID
    - Customer Name
    - Address
    - Movie Title
    - Release Year
    - Genre
- **Relationships:**
  - Relationships describe how entities are associated with each other.
  - They are represented by diamonds.
  - Types of relationships:
    - One-to-one: One entity is associated with at most one other entity.
    - One-to-many: One entity is associated with multiple instances of another entity.
    - Many-to-many: Multiple instances of one entity are associated with multiple instances of another entity.

In our Movie Recommendation System example:

- **Entities:** Movie, User, Genre, Director, Actor
- **Attributes:** (See the list of attributes provided earlier for each entity)
- **Relationships:** Directed By, Acted In, Belongs To, Rated By

By combining these components, an ER diagram visually represents the structure of the database, making it easier to understand the relationships between different entities and their attributes.

CHAPTER 4

SYSTEM IMPLEMENTATION

IMPLEMENTATION :

4.1 App.py

```
import streamlit as st
from PIL import Image
import json
from Classifier import KNearestNeighbours
from bs4 import BeautifulSoup
import requests, io
import PIL.Image
from urllib.request import urlopen

with open('./Data/movie_data.json', 'r+',
encoding='utf-8') as f:
    data = json.load(f)
with open('./Data/movie_titles.json', 'r+',
encoding='utf-8') as f:
    movie_titles = json.load(f)
hdr = {'User-Agent': 'Mozilla/5.0'}

def movie_poster_fetcher(imdb_link):

    url_data = requests.get(imdb_link,
headers=hdr).text
    s_data = BeautifulSoup(url_data, 'html.parser')
    imdb_dp = s_data.find("meta",
property="og:image")

    if imdb_dp is not None:
        movie_poster_link = imdb_dp.attrs['content']
        u = urlopen(movie_poster_link)
        raw_data = u.read()
        image =
PIL.Image.open(io.BytesIO(raw_data))
        image = image.resize((850,550), )
        st.image(image, use_container_width=False)
    else:
        print("Meta tag not found")


def get_movie_info(imdb_link):
    url_data = requests.get(imdb_link, headers=hdr).text
    s_data = BeautifulSoup(url_data, 'html.parser')

    imdb_content = s_data.find("meta",
property="og:description")
    if imdb_content is not None:
        movie_descr = imdb_content.attrs['content']
        movie_descr = str(movie_descr).split('.')

        movie_director = movie_descr[0] if
len(movie_descr) > 0 else ""
        movie_cast = str(movie_descr[1]).replace('With',
'Cast: ').strip() if len(movie_descr) > 1 else ""
        movie_story = 'Story: ' +
str(movie_descr[2]).strip() + '.' if len(movie_descr) > 2
else ""
    else:
        movie_director = ""
        movie_cast = ""
        movie_story = ""

    rating = s_data.find("span", class_="sc-bde20123-1
iZlgcd")
    if rating is not None:
        movie_rating = 'Total Rating count: ' +
str(rating.text)
    else:
        movie_rating = "Rating information  available"
    return movie_director, movie_cast, movie_story,
movie_rating
```

```
def KNN_Movie_Recommender(test_point, k):

    target = [0 for item in movie_titles]

    model = KNearestNeighbours(data, target,
test_point, k=k)

    model.fit()

    table = []
    for i in model.indices:

        table.append([movie_titles[i][0],
movie_titles[i][2], data[i][-1]])
    print(table)
    return table

st.set_page_config(
    page_title="Movie Recommender System",
)

def run():
    img1 = Image.open('./meta/logo.jpg')
    img1 = img1.resize((2560, 1709), )
    st.image(img1, use_container_width=False)
    st.title("Movie Recommender System")
    st.markdown('''<h4 style='text-align: left; color:
#d73b5c;'>* Data is based "IMDB 5000 Movie
Dataset"</h4>''',
             unsafe_allow_html=True)
    genres = ['Action', 'Adventure', 'Animation',
'Biography', 'Comedy', 'Crime', 'Documentary',
'Drama', 'Family',
              'Fantasy', 'Film-Noir', 'Game-Show',
'History', 'Horror', 'Music', 'Musical', 'Mystery',
'News',
              'Reality-TV', 'Romance', 'Sci-Fi', 'Short',
'Sport', 'Thriller', 'War', 'Western']
    movies = [title[0] for title in movie_titles]
    category = ['--Select--', 'Movie based', 'Genre based']
    cat_op = st.selectbox('Select Recommendation Type',
category)
    if cat_op == category[0]:
        st.warning('Please select Recommendation
Type!!')
    elif cat_op == category[1]:
        select_movie = st.selectbox('Select movie:
(Recommendation will be based on this selection)',
                                    ['--Select--'] + movies)
        dec = st.radio("Want to Fetch Movie Poster?",
('Yes', 'No'))
        st.markdown(
            '''<h4 style='text-align: left; color: #d73b5c;'>*
Fetching a Movie Posters will take a time."</h4>''',
            unsafe_allow_html=True)
        if dec == 'No':
            if select_movie == '--Select--':
                st.warning('Please select Movie!!')
            else:
                no_of_reco = st.slider('Number of movies
you want Recommended:', min_value=5,
max_value=20, step=1)
                genres = data[movies.index(select_movie)]
                test_points = genres
                table =
KNN_Movie_Recommender(test_points, no_of_reco +
1)
                table.pop(0)
                c = 0
                st.success('Some of the movies from our
Recommendation, have a look below')
                for movie, link, ratings in table:
```

```
                c += 1
                director, cast, story, total_rat =
get_movie_info(link)
                st.markdown(f"({c})[
{movie}]({link})")
                st.markdown(director)
                st.markdown(cast)
                st.markdown(story)
                st.markdown(total_rat)
                st.markdown('IMDB Rating: ' +
str(ratings) + '⭐')
        else:
            if select_movie == '--Select--':
                st.warning('Please select Movie!!')
            else:
                no_of_reco = st.slider('Number of movies
you want Recommended:', min_value=5,
max_value=20, step=1)
                genres =
data[movies.index(select_movie)]
                test_points = genres
                table =
KNN_Movie_Recommender(test_points,
no_of_reco + 1)
                table.pop(0)
                c = 0
                st.success('Some of the movies from our
Recommendation, have a look below')
                for movie, link, ratings in table:
                    c += 1
                    st.markdown(f"({c})[
{movie}]({link})")
                    movie_poster_fetcher(link)
                    director, cast, story, total_rat =
get_movie_info(link)
                    st.markdown(director)
                    st.markdown(cast)
                    st.markdown(story)
                    st.markdown(total_rat)
                    st.markdown('IMDB Rating: ' + str(ratings)
+ '⭐')
    elif cat_op == category[2]:
        sel_gen = st.multiselect('Select Genres:', genres)
        dec = st.radio("Want to Fetch Movie Poster?",
('Yes', 'No'))
        st.markdown(
            '"<h4 style='text-align: left; color: #d73b5c;'>*
Fetching a Movie Posters will take a time."</h4>"',
            unsafe_allow_html=True)
        if dec == 'No':
            if sel_gen:
                imdb_score = st.slider('Choose IMDb score:',
1, 10, 8)
                no_of_reco = st.number_input('Number of
movies:', min_value=5, max_value=20, step=1)
                test_point = [1 if genre in sel_gen else 0 for
genre in genres]
                test_point.append(imdb_score)
                table =
KNN_Movie_Recommender(test_point, no_of_reco)
                c = 0
                st.success('Some of the movies from our
Recommendation, have a look below')
                for movie, link, ratings in table:
                    c += 1
                    st.markdown(f"({c})[ {movie}]({link})")
                    director, cast, story, total_rat =
get_movie_info(link)
                    st.markdown(director)
                    st.markdown(cast)
```

```
                st.markdown(story)
                st.markdown(total_rat)
                st.markdown('IMDB Rating: ' +
str(ratings) + '⭐')
        else:
            if sel_gen:
                imdb_score = st.slider('Choose IMDb
score:', 1, 10, 8)
                no_of_reco = st.number_input('Number of
movies:', min_value=5, max_value=20, step=1)
                test_point = [1 if genre in sel_gen else 0
for genre in genres]
                test_point.append(imdb_score)
                table =
KNN_Movie_Recommender(test_point, no_of_reco)
                c = 0
                st.success('Some of the movies from our
Recommendation, have a look below')
                for movie, link, ratings in table:
                    c += 1
                    st.markdown(f"({c})[
{movie}]({link})")
                    movie_poster_fetcher(link)
                    director, cast, story, total_rat =
get_movie_info(link)
                    st.markdown(director)
                    st.markdown(cast)
                    st.markdown(story)
                    st.markdown(total_rat)
                    st.markdown('IMDB Rating: ' +
str(ratings) + '⭐')

run()
```

4.2 Classifier.py

```
import numpy as np
from operator import itemgetter

class KNearestNeighbours:
    def __init__(self, data, target, test_point, k):
        self.data = data
        self.target = target
        self.test_point = test_point
        self.k = k
        self.distances = list()
        self.categories = list()
        self.indices = list()
        self.counts = list()
        self.category_assigned = None

    @staticmethod
    def dist(p1, p2):
        """Method returns the euclidean distance
between two points"""
        return np.linalg.norm(np.array(p1) -
np.array(p2))

    def fit(self):
        """Method that performs the KNN
classification"""

        self.distances.extend([(self.dist(self.test_point,
point), i) for point, i in zip(self.data, [i for i in
range(len(self.data))])])

        sorted_li = sorted(self.distances,
key=itemgetter(0))
        self.indices.extend([index for (val, index) in
sorted_li[:self.k]])
        for i in self.indices:
            self.categories.append(self.target[i])
        self.counts.extend([(i, self.categories.count(i))
for i in set(self.categories)])
        # Find the highest repeated category among the
K nearest neighbours
        self.category_assigned = sorted(self.counts,
key=itemgetter(1), reverse=True)[0][0]
```

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

The **Movie Recommendation System** represents a significant step forward in delivering personalized entertainment experiences. By leveraging the power of the **K-Nearest Neighbors (KNN)** algorithm, real-time metadata fetching, and an interactive **Streamlit** interface, this project bridges the gap between user preferences and the vast collection of available movies. It not only addresses the challenges faced by traditional systems but also introduces innovative features that enhance user engagement and satisfaction.

5.1.1 Key Achievements

1. **Personalized Recommendations**:
    - The system successfully generates personalized movie suggestions based on either:
        - A user-selected favorite movie.
        - User-selected genres and IMDb score thresholds.
    - This customization ensures that the recommendations align closely with the user's tastes.
2. **Enrichment of User Experience**:
    - The integration of IMDb metadata enriches the recommendations by providing users with:
        - High-quality posters.
        - Movie details such as director, cast, and plot summary.
        - Up-to-date IMDb ratings.
    - This feature creates an immersive experience, enabling users to make informed choices.
3. **Interactive and User-Friendly Interface**:
    - Developed with Streamlit, the system boasts an intuitive interface that simplifies navigation.
    - Users can interactively adjust parameters, such as the number of recommendations, making the application accessible to individuals with varying technical expertise.
4. **Efficient and Scalable Design**:
    - The system is designed to handle a large dataset of movies efficiently.
    - Its modular architecture allows for future enhancements, such as integrating collaborative filtering techniques or real-time user feedback mechanisms.

.

5.1.2Advantages Over Existing Systems

The proposed system offers several improvements over traditional movie recommendation systems:

1. **Dynamic Data Fetching**:
   - Unlike static systems, this project dynamically fetches movie metadata from IMDb, ensuring that users always receive the latest information.
2. **Visual Appeal**:
   - By displaying movie posters alongside textual recommendations, the system makes the interface visually engaging and easy to navigate.
3. **Customizability**:
   - The system allows users to tailor recommendations based on their unique preferences, fostering a sense of personalization and ownership.
4. **Real-Time Insights**:
   - Through web scraping, the system provides real-time insights about movies, a feature often missing in conventional recommender systems.

5.2 Future Scope

The project lays a strong foundation for further development and scalability. Some potential future enhancements include:

1. **Advanced Recommendation Algorithms**:
   - Integration of machine learning techniques such as collaborative filtering, deep learning, or hybrid models to improve recommendation accuracy.
2. **User Profiles**:
   - Addition of user accounts to store preferences, viewing history, and personalized dashboards.
3. **API Integration**:
   - Replacing web scraping with APIs like TMDb to ensure more reliable and scalable data fetching.
4. **Cross-Platform Support**:
   - Expanding the system to mobile platforms or as a browser extension for greater accessibility.
5. **Feedback Mechanisms**:
   - Incorporating user feedback to improve recommendations dynamically.

CHAPTER 6

RESULTS:

RESULTS:

Fig :6.1 Home Page

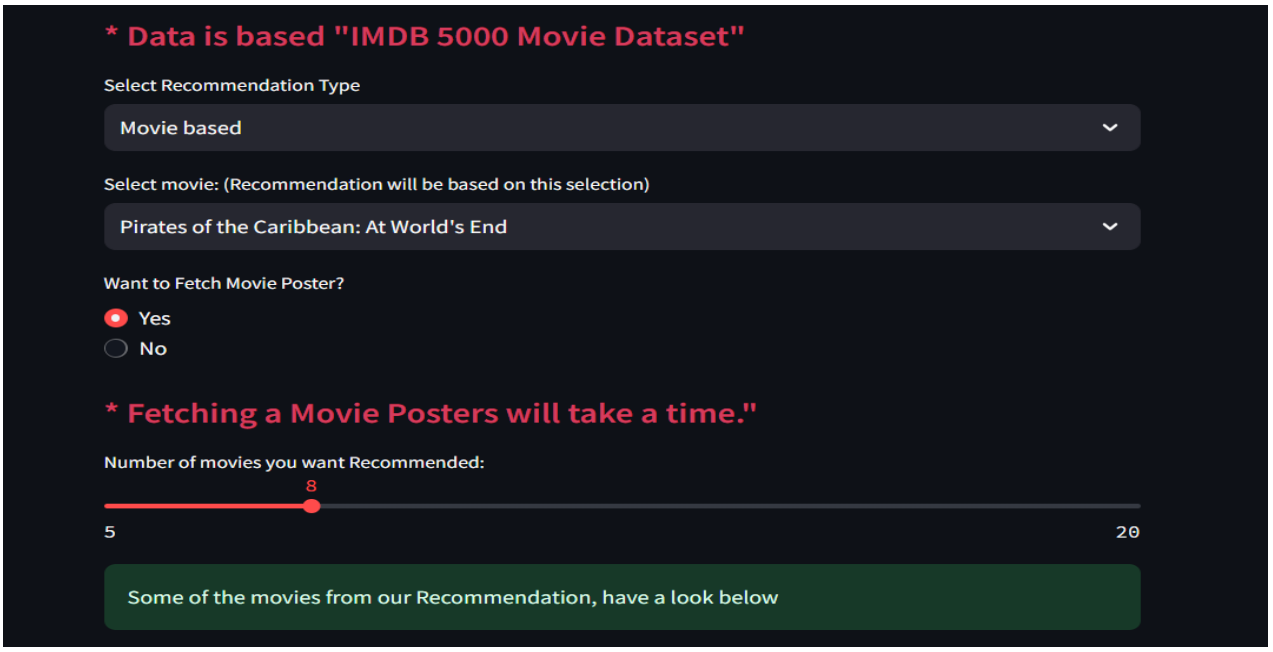

Fig 6.2

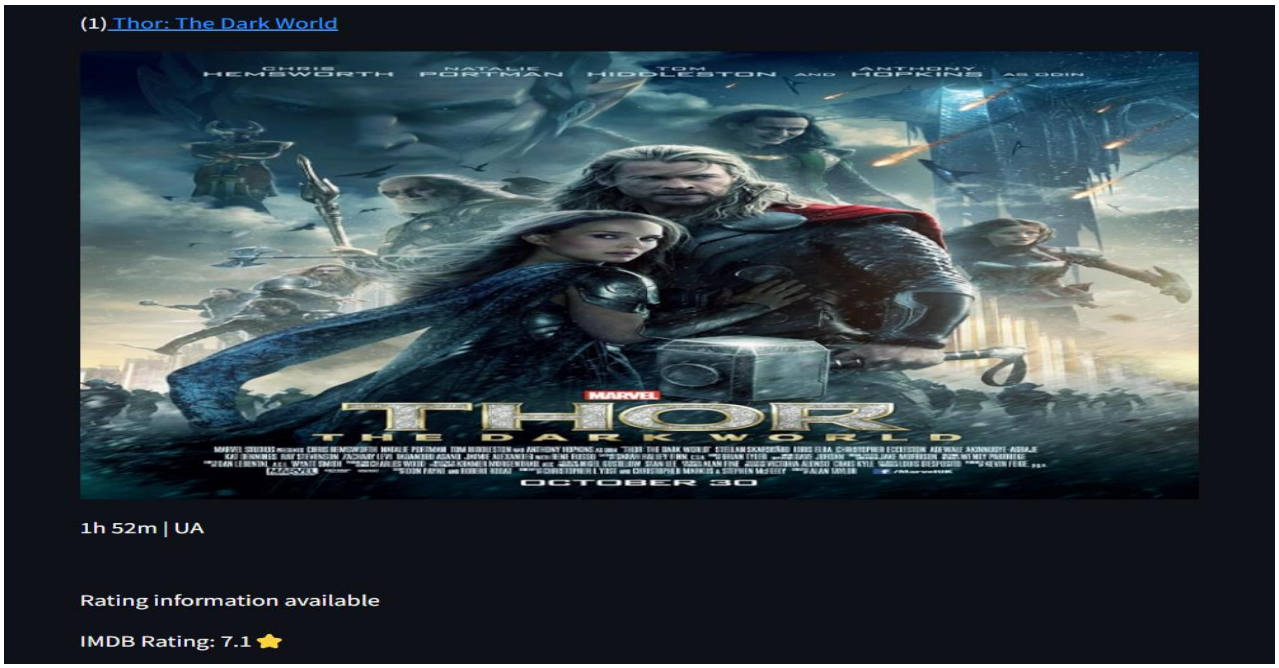

Fig : 6.3.1 Recommended Movie

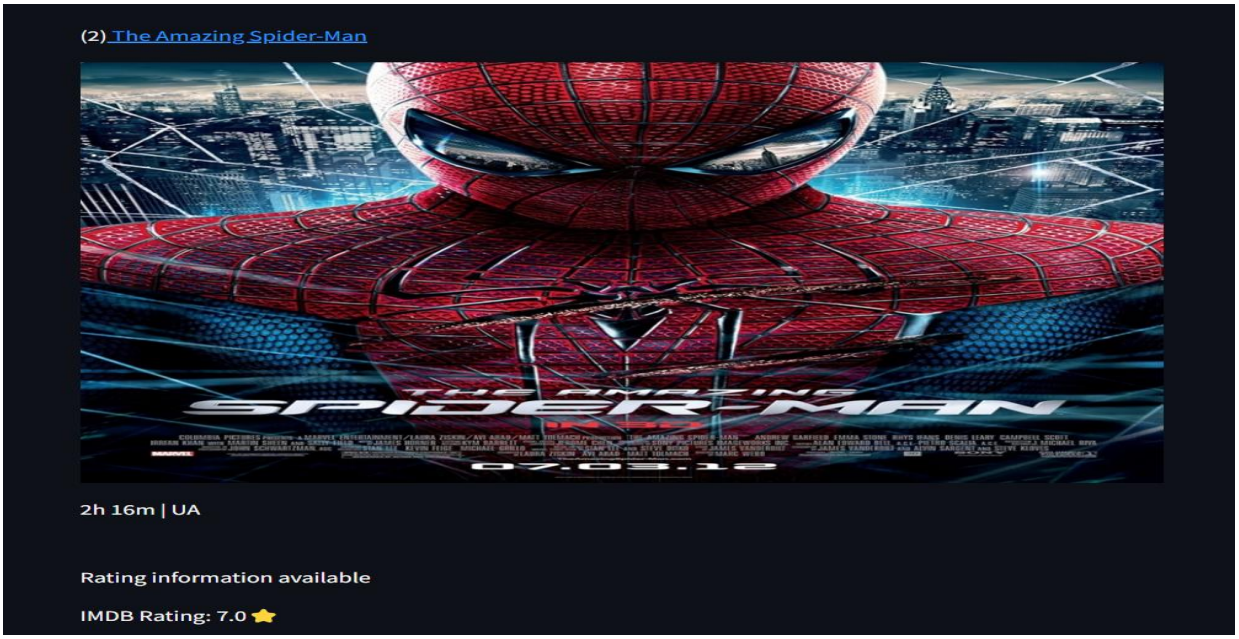

Fig : 6.3.2 Recommended Movie

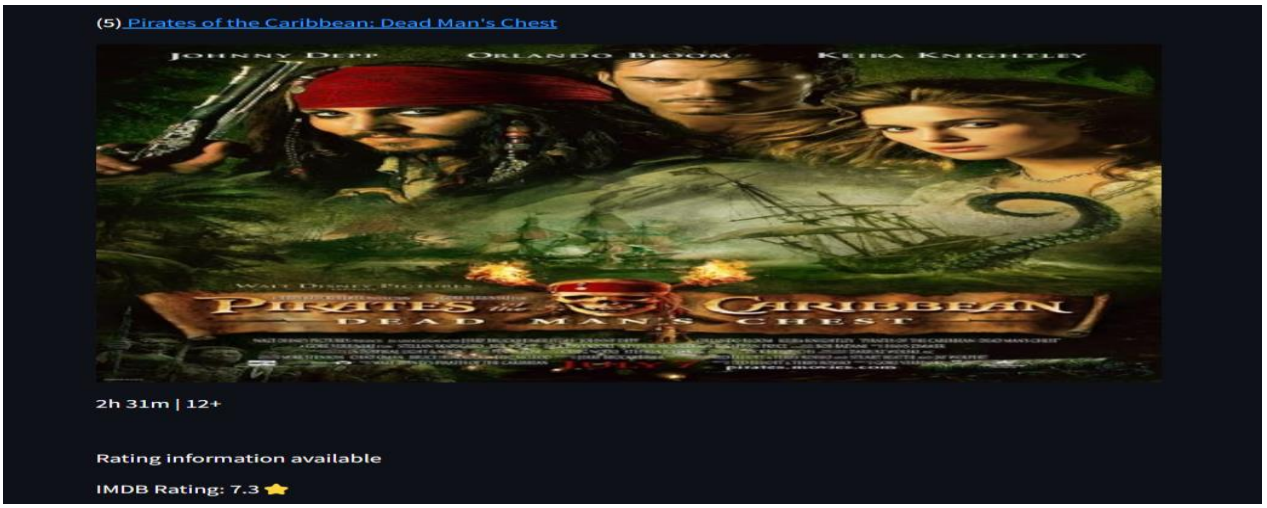

Fig :6.3.3 Recommended Movie

Fig : 6.3.4 Recommended Movie

REFERENCES:

References:

1. Rujhan Singla, Samarth Gupta, Anirudh Gupta, Dinesh Kumar Vishwakarma, FLEX: A Content Based Movie Recommender, 978-1 7281-6221- 8/20/$31.00 ©2020 IEEE NLP

2. N. Kapoor, S. Vishal, and K. K. S., "Movie Recommendation System Using Tools," IEEE Xplore, Jun. 01, 2020. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9137993

3. S. Bhaskaran, R. Marappan, and B. Santhi, "Design and Analysis of a Cluster-Based Intelligent Hybrid Recommendation System for E Learning Applications," Mathematics, vol. 9, no. 2, p. 197, Jan. 2021, doi: https://doi.org/10.3390/math9020197.

4. G. Vibhandik, "Movie Recommendation System using Machine Learning," International Journal for Research in Applied Science and Engineering Technology, vol. 9, no. VI, pp. 4778 4781, Jun. 2021, doi: https://doi.org/10.22214/ijraset.2021.35741.

5. Y.-K. Ng, "MovRec: a personalized movie recommendation system for children based on online movie features," International Journal of Web Information Systems, vol. 13, no. 4, pp. 445–470, Nov. 2017, doi: https://doi.org/10.1108/ijwis-05-2017-0043.

6. A. Yenkikar, N. Babu and S. Sangve, "R-SA: A Rule-based Expert System for Sentiment Analysis," 2019 IEEE Pune Section International Conference (PuneCon), Pune, India, 2019, pp. 1-7, doi: 10.1109/PuneCon46936.2019.9105682.

7. Pradnya Mehta, "Survey on movie rating and review summarization in mobile envioronment," International Journal of Engineering Research and Technology, vol. 2, no. 3, 2017

8. A Nayan Varma and Kedareshwara Petluri, "Movie Recommender System using critic consensus," Dec. 2021, doi: https://doi.org/10.1109/icac353642.2021.9697196.

9. A. Yenkikar and C. N. Babu, "AirBERT: A fine-tuned language representation model for airlines tweet sentiment analysis," Intelligent Decision Technologies, vol. Preprint, no. Preprint, pp. 1–17, Jan. 2022, doi: https://doi.org/10.3233/IDT-220173.

10. J. Hemanth Duraisamy, A. Yenkikar, and N. Babu, "SENTINET: A DEEP SENTIMENT ANALYSIS NETWORK FOR POLITICAL MEDIA BIAS DETECTION," DYNA, vol. 97, no. 6, pp. 645–651, Nov. 2022, doi: https://doi.org/10.6036/10593...

11. R. R. Patil and S. Kumar, "Rice Transformer: A Novel Integrated Management System for Controlling Rice Diseases," in IEEE Access, vol. 10, pp. 87698-87714, 2022, doi: 10.1109/ACCESS.2022.3200688.

12. R. Lavanya, U. Singh and V. Tyagi, "A Comprehensive Survey on Movie Recommendation Systems," 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), 2021,pp.532-536,doi:10.1109/ICAIS50930.2021.9395759.

13. Zhang, Jiang, et al. "Personalized real-time movie recommendation system: Practical prototype and evaluation." Tsinghua Science and Technology 25.2 (2019): 180-191.

14. Rajarajeswari, S., et al."Movie Recommendation System." Emerging Research in Computing, Information, Communication and Applications. Springer, Singapore, 2019. 329-340.