National Institute of Electronics & Information Technology

राष्ट्रीय इलेक्ट्रॉनिकी एवं सूचना प्रौद्योगिकी संस्थान

MACHINE LEARNING INTERNSHIP REPORT

Movie Recommendation System

Submitted By:

Name: Sai Yangyadatta Sethy

Program: MCA

Institution: Utkal University

Reg. Number: 15830V24029

Submitted To Organization: NIELIT, Bhubaneswar

Acknowledgment

I take this opportunity to express my profound gratitude to Utkal University and the Department of Computer Science and Applications for facilitating the successful completion of my intern ship titled “Foundation Course on Machine Learning using Python.”

I am especially thankful to Khageswar Behera and Bijaylaxmi Behera, whose expert guidance, constructive feedback, and consistent encouragement were invaluable throughout the course of this internship. Their mentorship played a critical role in enhancing both my theoretical understanding and practi cal application of machine learning techniques using Python.

I would also like to extend my sincere appreciation to the coordinators and faculty members associated with the internship programme for designing a structured, industry-relevant learning experience that effectively bridged the gap between academic knowledge and practical implementation.

Sai Yangyadatta Sethy
Reg. No: 15830V24029

Certificate of Completion

This is to certify that Mr. Sai Yangyadatta Sethy, a student of the Department of Computer Science and Applications, Utkal University, has successfully completed the internship titled “Foundation Course on Machine Learning using Python,” conducted during the period 15-05 2025 to 12-07-2025. This internship was carried out under the guidance and supervision of Khageswar Behera and Bijaylaxmi Behera, Instructors, at NIELIT. We acknowledge the in tern’s active participation, consistent effort, and sincere contribution throughout the internship duration. The student has demonstrated a good understanding of machine learning concepts and hands-on proficiency in Python .

Signature of Guide

Table of Contents

A Project Report On Movie Recommendation System

1. Introduction

The rapid growth of digital media platforms has made it increasingly challenging for users to discover movies that align with their preferences. Recommendation systems, powered by machine learning, address this challenge by analyzing user preferences and movie metadata to suggest relevant content. This project develops a content-based movie recommendation system that leverages movie attributes such as genres, keywords, cast, crew, and plot summaries to recommend similar movies. The system is built as an interactive web application using Streamlit, integrating data from the Movies dataset and the API for fetching movie posters

2. Objective

The primary objective of this project is to design and implement a machine learning-based movie recommendation system that suggests movies similar to a user-selected movie, based on content-based filtering techniques. The system aims to provide personalized recommendations by analyzing movie metadata and displaying results with posters, IMDb ratings, release years, and director information.

3. Technologies Used

The following key technologies and libraries were utilized in the development of the Movie Recommendation System:

- **Python:** The core programming language for the project.
- **NumPy:** Used for numerical operations and array manipulations.
- **Pandas:** Facilitated data manipulation, cleaning, and analysis, particularly for handling DataFrames.
- **Scikit-learn:** Provided tools for text vectorization (CountVectorizer) and similarity compu tation (cosine_similarity) for content-based filtering.
- **Streamlit:** Enabled rapid development and deployment of an interactive web application for user input and result visualization.
- **Requests:** Used to fetch movie poster images from the API.
- **Ast :** Safely parsed string representations of lists for genres, keywords, cast, and crew data.

4. Dataset Description

The dataset used for this project is the TMDb Movies dataset, available from Kaggle , also available in the Movie Recommendation System Repository. It consists of two dataset files: ‘movies‘ and ‘credits‘, containing approximately 5,000 movies with the following key features:

- **Title**: The movie’s title.
- **Overview:** A brief plot summary.
- **Genres:** List of genres associated with the movie.
- **Keywords**: Tags describing key themes or elements.
- **Cast:** List of main actors (top3 used).
- **Crew**: Crew details, from which the director’s name is extracted.
- **Vote Average:** Average user rating on TMDb, used as a proxy for IMDb rating.
- **Release Date:** The movie’s release date, from which the year is extracted.

A sample dataset with 20 movies was created for demonstration purposes when the original CSVfiles are unavailable

5. Methodology

The development process involved several key stages:

5.1. Data Collection and Loading

The TMDb 5000 Movies and Credits datasets are loaded using Pandas and merged on the movie ID. Relevant columns (title, overview, genres, keywords, cast, crew, vote average, release date) are retained. . This ensures easy access and reproducibility. Streamlit's @st.cache_data decorator is used to cache the dataset, preventing redundant loading during application reruns and enhancing performance.

5.2. Data Preprocessing

Data preprocessing is a critical step to prepare the raw data for model training. It ensures the data is suitable for recommendation.The following steps were performed:

- **Feature Extraction:** Genres, keywords, and cast are converted from stringified JSON to lists using a safe parsing function with the ‘ast‘ library. The top 3 actors are retained, and the director’s name is extracted from the crew.
- **Release Year Extraction:** The release year is extracted from the release date using

Pandas’ datetime functionality, with ’Unknown’ for missing or invalid dates.

- **Handling Missing Values**: Missing overviews and ratings are filled with empty strings or 0.0, respectively.
- **Feature Combination:** A ’tags’ column is created by combining overview, genres, key words, cast, and director into a single text string, converted to lowercase for consistency.
- **TextVectorization:** The tags are vectorized using Scikit-learn’s CountVectorizer (max 5,000 features, English stop words removed) to create a numerical representation for similarity computation

5.3. Model Construction & Similarity Computation

In this project, traditional machine learning algorithms like Logistic Regression or Decision Trees were not used. Instead, a content-based filtering approach was applied using natural language processing techniques. The steps involved are as follows:

1. TextVectorization

   The combined feature (tags) from each movie—consisting of genres, cast, overview, keywords, and director—was vectorized using the Bag-of-Words model with Scikit-learn’s CountVectorizer. This transformed the textual data into a matrix of token counts.
2. Cosine Similarity Computation

   The movie vectors were then compared against each other using cosine similarity, which measures the cosine of the angle between two non-zero vectors in a multi-dimensional space. This generates a similarity matrix indicating how close each movie is to another.
3. Caching the Similarity Matrix

   To optimize performance, the similarity matrix was cached using Streamlit’s @st.cache_data decorator. This prevents repeated recomputation during app usage.

While no supervised learning model was trained, this method effectively “learns” which movies are similar based on their content — making it a powerful unsupervised recommendation approach.

Algorithm Used:

- Vector Space Model (Bag-of-Words)
- Cosine Similarity for distance computation
- Content-Based Filtering as the recommendation strategy

5.4. Recommendation Generation

The recommendation generation phase is the core functionality of the Movie Recommendation System, where the system identifies and presents movies similar to a user-selected movie based on content-based filtering. This process leverages the precomputed cosine similarity matrix to recommend movies with similar attributes, enhanced by additional metadata such as IMDb ratings, release years, and director names for a richer user experience.

Steps Involved:

- **Identify Movie Index**: The system locates the index of the user-selected movie in the dataset using Pandas' indexing, ensuring efficient access to its corresponding data.
- **Retrieve Similarity Scores**: The cosine similarity scores for the selected movie are extracted from the precomputed similarity matrix, where each score represents the similarity between the selected movie and all others based on their combined feature tags.
- **Genre Filtering (**Optional**)**: If a genre is specified via the Streamlit interface, the system filters the similarity scores to include only movies that contain the selected genre, ensuring recommendations align with user preferences.
- **Sort and Select Top Recommendations**: The similarity scores are sorted in descending order, and the top 5 movies (excluding the selected movie) are selected to form the recommendation list.
- **Fetch Metadata and Posters**: For each recommended movie, the system retrieves the title, IMDb rating (TMDb vote average), release year, and director name from the dataset. Additionally, it makes API calls to the TMDb API to fetch poster images, enhancing the visual presentation of recommendations.

6. Content-Based Filtering Algorithm

Content-based filtering is a recommendation technique that suggests items (in this case, movies) by analyzing their attributes and identifying items with similar characteristics. This Movie Recommendation System employs content-based filtering to recommend movies based on their metadata, including overview, genres, keywords, cast, director, IMDb ratings (proxied by TMDb's vote_average), and release year. When a user selects a movie, the system identifies and ranks other movies with similar metadata using cosine similarity, presenting the top recommendations with enriched details such as posters, ratings, release years, and directors.

Steps:

1. **Data Preparation** (load_and_process_data, process_movie_features):
   - Loads and merges movies.csv and credits.csv on movie_id.
   - Selects columns: title, overview, genres, keywords, cast, crew, vote_average (IMDb rating proxy), release_date.
   - Processes metadata:
     - **Genres/Keywords**: Parses JSON strings into lists (e.g., [{"name": "Action"}] → ["Action"]).
     - **Cast**: Takes top 3 actors.
     - **Crew**: Extracts director name.
     - **Overview**: Fills missing values with empty strings.
     - **IMDb Rating**: Uses vote_average (e.g., 8.4 for *The Dark Knight*).
     - **Release Year**: Extracts year from release_date (e.g., "2008-07-18" → "2008").
   - Creates a tags column by combining overview, genres, keywords, cast, and director (e.g., "batman battles the joker action crime drama christian bale heath ledger christopher nolan").
   - Converts tags to lowercase.
2. **Vectorization** (compute_similarity_matrix):
   - Uses CountVectorizer (max 5,000 features, no stop words) to convert tags into vectors.
   - Example: *The Dark Knight* → vector [1, 1, ..., 0] for words like "batman", "action".
3. **Similarity Computation** (compute_similarity_matrix):
   - Computes cosine similarity between all movie vectors:

$$cosine\ similarity(A, B) = \frac{A.B}{|A||B|}$$

- Produces a similarity matrix cached with @st.cache_data.
- Example: *The Dark Knight* has high similarity with *Batman Begins* due to shared tags.

4. **Recommendation Generation** (recommend):
    - Finds the selected movie's index (e.g., *The Dark Knight* at index 0).
    - Retrieves similarity scores from the matrix.
    - Filters by genre if selected (e.g., "Action").
    - Sorts scores and picks top 5 movies (excluding the selected movie).
    - Collects title, poster (via TMDb API), IMDb rating, release year, and director.
    - Returns lists for display in Streamlit.

Example

For *The Dark Knight* (Action filter):

- Tags: "batman battles the joker action crime drama batman christian bale christopher nolan".
- Output (sample):
    - *Batman Begins* (Rating: 8.2, Year: 2005, Director: Christopher Nolan).
    - *Inception* (Rating: 8.8, Year: 2010, Director: Christopher Nolan).
    - *The Matrix* (Rating: 8.7, Year: 1999, Director: Lana Wachowski).

7. Streamlit Application Development

A user-friendly, interactive web application was developed using Streamlit to allow users to explore and receive personalized movie recommendations.

- **User Interface :** The application provides a clear and minimalistic interface that enables users to:
    - Select a movie genre (optional) from a dropdown.
    - Choose a movie title based on the selected genre or from the entire list if no genre is selected.
- **Movie Information Display :** Once a movie is selected, the application displays detailed information including:

- Poster image (fetched using the TMDB API)
- Title
- Director
- Genres
- Cast (top 3 actors)
- IMDb rating
- Release year
- Movie overview

• **Recommendation Interface**

After the user selects a movie and clicks the **Recommend** button:

- The app displays the **top 5 most similar movies**.
- Each recommended movie is shown in a **card layout** with:
    - Poster
    - Title
    - Director
    - IMDb rating
    - Release year

• **Caching and Performance**

To optimize performance and responsiveness:

- @st.cache_data is used to cache:
    - Dataset loading
    - Data preprocessing
    - Cosine similarity matrix computation
- This prevents repeated computation and speeds up response time significantly.

• Error Handling

The application includes robust handling for:

- Missing data (e.g., missing posters or genres)
- API failures (fallback to placeholder posters)
- Invalid movie selections (e.g., empty input or unavailable title)

8. Results and Discussion

The developed Movie Recommendation System successfully achieves its primary objective of providing content-based movie recommendations. The system efficiently processes the Movies dataset (or the sample data, if the full dataset is unavailable) to generate relevant movie suggestions.

The Streamlit web interface proved to be highly effective in delivering an intuitive and user-friendly experience. Users can easily select a movie, optionally filter by genre, and immediately receive a set of recommendations. The visual presentation of results, including movie posters, IMDb ratings , release years, and director names, significantly enhances the user experience by providing comprehensive context for each suggested film. The integration of the API for poster retrieval ensures a rich and engaging visual output.

The content-based filtering algorithm effectively leverages movie metadata (overview, genres, keywords, cast, and director) to identify similarities between films. By using CountVectorizer for text representation and cosine similarity for measuring relationships, the system accurately groups movies with shared thematic and structural elements. For instance, when The Dark Knight is selected, the system consistently recommends other critically acclaimed action/crime dramas, often featuring similar cast or director, demonstrating the efficacy of the chosen methodology. The caching mechanisms implemented with @st.cache_data in Streamlit also ensured responsive performance, even with a dataset of 5,000 movies, minimizing wait times for users.

While the content-based approach excels at finding similar movies based on their inherent attributes, a limitation lies in its inability to capture individual user preferences that might diverge from explicit content features (e.g., a user who likes action movies but occasionally enjoys a specific comedy genre that isn't directly related to their typical action choices). This "cold start" problem for new users (or new items) is inherent to content-based systems. Future enhancements could explore integrating collaborative filtering techniques, which analyze user-item interaction data, or a hybrid approach to provide even more personalized recommendations and address this limitation.

Overall, the current system serves as a robust foundation for movie discovery based on content characteristics.

9. Conclusion

The Movie Recommendation System project stands as a successful demonstration of building an end-to-end content-based recommendation engine. It seamlessly integrates essential Python libraries such as Pandas for data handling, Scikit-learn for advanced text processing and similarity computations, and Streamlit for creating a highly interactive and user-friendly web application. The robust connection with the Movie API further enriches the user experience by providing dynamic movie poster visuals.

This system effectively empowers users to discover movies similar to a chosen title by leveraging various content features, including genres, cast, keywords, and plot overviews. The analytical backbone, encompassing meticulous data preprocessing, insightful feature engineering, sophisticated text vectorization, and precise similarity computation, underpins its accuracy and relevance.

While the current implementation provides strong content-based recommendations, the potential for future enhancements is significant. These could include integrating movie trailers, implementing user login functionalities for personalized watchlists, fetching official IMDb ratings for greater accuracy, and incorporating language and runtime filters for more refined search capabilities. Furthermore, exploring advanced Natural Language Processing (NLP) models could lead to deeper semantic understanding of movie overviews, thereby generating even more nuanced and accurate recommendations.

10. References

1. **Scikit-learn Documentation**

   https://scikit-learn.org/stable/

2. **Streamlit Documentation**

   https://docs.streamlit.io/

3. **The Movie Database API**

   https://developer.themoviedb.org/

   https://www.imdb.com/

4. **Dataset**

   https://datasetsearch.research.google.com

   https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata

Code:

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import streamlit as st
import ast
import requests
import warnings
warnings.filterwarnings('ignore')


st.set_page_config(
    page_title="SYS Movie Recommendation System",
    page_icon="🎬",
    layout="wide"
)

st.markdown("""
<style>
    .main-header {
        font-size: 3rem;
        color: #FF6B6B;
        text-align: center;
        margin-bottom: 2rem;
    }
    .movie-card {
        border-radius: 10px;
        padding: 10px;
        text-align: center;
        background: #f0f2f6;
        box-shadow: 0 2px 4px rgba(0,0,0,0.1);
    }
</style>
""", unsafe_allow_html=True)


@st.cache_data
def load_and_process_data():

    try:

        movies = pd.read_csv('movies.csv')
        credits = pd.read_csv('credits.csv')
        movies = movies.merge(credits, left_on='id', right_on='movie_id', how='left')


        required_columns = ['title_x', 'overview', 'genres', 'keywords', 'cast', 'crew', 'vote_average',
'release_date']
        movies = movies[required_columns]
        movies.rename(columns={'title_x': 'title'}, inplace=True)

        return movies, True
    except FileNotFoundError:
        st.error("CSV files not found! Please ensure 'tmdb_5000_movies.csv' and
'tmdb_5000_credits.csv' are in your directory.")
        return create_sample_data(), False
    except Exception as e:
        st.error(f"Error loading data: {str(e)}")
        return create_sample_data(), False

def create_sample_data():

    sample_movies = {
        'title': [
            'The Dark Knight', 'Inception', 'The Matrix', 'Pulp Fiction', 'The Godfather',
            'Forrest Gump', 'The Shawshank Redemption', 'Fight Club', 'Goodfellas', 'The Lord of the
Rings',
            'Star Wars', 'Titanic', 'Avatar', 'The Avengers', 'Jurassic Park',
            'Casablanca', 'Gone with the Wind', 'The Wizard of Oz', 'Citizen Kane', 'Vertigo'
        ],
        'overview': [
            'Batman battles the Joker in Gotham City',
            'A thief enters people\'s dreams to steal secrets',
            'A computer hacker discovers reality is a simulation',
            'Interconnected stories of Los Angeles criminals',
            'The aging patriarch of a crime dynasty transfers control',
            'Life story of a simple man with low IQ',
            'Two imprisoned men bond over years finding solace',
            'An insomniac office worker forms an underground fight club',
            'Story of Henry Hill and his life in the mob',
            'A hobbit journeys to destroy a powerful ring',
            'Rebels battle the evil Galactic Empire',
            'A seventeen-year-old aristocrat falls in love with a poor artist',
            'Marines fight alien creatures on planet Pandora',
            'Superheroes assemble to save the world',
            'Scientists clone dinosaurs for a theme park',
            'A cynical American expatriate meets a former lover',
            'Epic historical romance during American Civil War',
            'A young girl travels to a magical land',
            'A publishing tycoon\'s rise and fall',
            'A detective becomes obsessed with a mysterious woman'
        ],
        'genres': [
            '[{"name": "Action"}, {"name": "Crime"}, {"name": "Drama"}]',
        '[{"name": "Action"}, {"name": "Sci-Fi"}, {"name": "Thriller"}]',
        '[{"name": "Action"}, {"name": "Sci-Fi"}]',
        '[{"name": "Crime"}, {"name": "Drama"}]',
        '[{"name": "Crime"}, {"name": "Drama"}]',
        '[{"name": "Drama"}, {"name": "Romance"}]',
        '[{"name": "Drama"}]',
        '[{"name": "Drama"}, {"name": "Thriller"}]',
        '[{"name": "Crime"}, {"name": "Drama"}]',
        '[{"name": "Adventure"}, {"name": "Fantasy"}]',
        '[{"name": "Adventure"}, {"name": "Sci-Fi"}]',
        '[{"name": "Drama"}, {"name": "Romance"}]',
        '[{"name": "Action"}, {"name": "Sci-Fi"}]',
        '[{"name": "Action"}, {"name": "Adventure"}]',
        '[{"name": "Adventure"}, {"name": "Sci-Fi"}]',
        '[{"name": "Drama"}, {"name": "Romance"}]',
        '[{"name": "Drama"}, {"name": "Romance"}]',
        '[{"name": "Adventure"}, {"name": "Family"}]',
        '[{"name": "Drama"}, {"name": "Mystery"}]',
        '[{"name": "Mystery"}, {"name": "Thriller"}]'
    ],
    'keywords': [
        '[{"name": "batman"}, {"name": "joker"}, {"name": "gotham"}]',
        '[{"name": "dreams"}, {"name": "heist"}, {"name": "mind"}]',
        '[{"name": "virtual reality"}, {"name": "hacker"}, {"name": "simulation"}]',
        '[{"name": "gangster"}, {"name": "nonlinear"}, {"name": "los angeles"}]',
        '[{"name": "mafia"}, {"name": "family"}, {"name": "power"}]',
        '[{"name": "vietnam war"}, {"name": "friendship"}, {"name": "life"}]',
        '[{"name": "prison"}, {"name": "friendship"}, {"name": "hope"}]',
        '[{"name": "insomnia"}, {"name": "club"}, {"name": "violence"}]',
        '[{"name": "mafia"}, {"name": "biography"}, {"name": "violence"}]',
        '[{"name": "ring"}, {"name": "wizard"}, {"name": "quest"}]',
        '[{"name": "space"}, {"name": "rebellion"}, {"name": "empire"}]',
        '[{"name": "ship"}, {"name": "disaster"}, {"name": "love"}]',
        '[{"name": "alien"}, {"name": "planet"}, {"name": "military"}]',
        '[{"name": "superhero"}, {"name": "team"}, {"name": "alien invasion"}]',
        '[{"name": "dinosaur"}, {"name": "cloning"}, {"name": "island"}]',
        '[{"name": "world war ii"}, {"name": "bar"}, {"name": "resistance"}]',
        '[{"name": "civil war"}, {"name": "plantation"}, {"name": "survival"}]',
        '[{"name": "tornado"}, {"name": "magic"}, {"name": "home"}]',
        '[{"name": "newspaper"}, {"name": "power"}, {"name": "mystery"}]',
        '[{"name": "obsession"}, {"name": "detective"}, {"name": "fear"}]'
    ],
    'cast': [
        '[{"name": "Christian Bale"}, {"name": "Heath Ledger"}, {"name": "Aaron Eckhart"}]',
        '[{"name": "Leonardo DiCaprio"}, {"name": "Marion Cotillard"}, {"name": "Tom
Hardy"}]',
        '[{"name": "Keanu Reeves"}, {"name": "Laurence Fishburne"}, {"name": "Carrie-Anne
Moss"}]',
        '[{"name": "John Travolta"}, {"name": "Uma Thurman"}, {"name": "Samuel L.
Jackson"}]',
        '[{"name": "Marlon Brando"}, {"name": "Al Pacino"}, {"name": "James Caan"}]',
        '[{"name": "Tom Hanks"}, {"name": "Robin Wright"}, {"name": "Gary Sinise"}]',
        '[{"name": "Tim Robbins"}, {"name": "Morgan Freeman"}, {"name": "Bob Gunton"}]',
        '[{"name": "Brad Pitt"}, {"name": "Edward Norton"}, {"name": "Helena Bonham
Carter"}]',
        '[{"name": "Robert De Niro"}, {"name": "Ray Liotta"}, {"name": "Joe Pesci"}]',
        '[{"name": "Elijah Wood"}, {"name": "Ian McKellen"}, {"name": "Orlando Bloom"}]',
        '[{"name": "Mark Hamill"}, {"name": "Harrison Ford"}, {"name": "Carrie Fisher"}]',
        '[{"name": "Leonardo DiCaprio"}, {"name": "Kate Winslet"}, {"name": "Billy Zane"}]',
        '[{"name": "Sam Worthington"}, {"name": "Zoe Saldana"}, {"name": "Sigourney
Weaver"}]',
        '[{"name": "Robert Downey Jr."}, {"name": "Chris Evans"}, {"name": "Mark
Ruffalo"}]',
        '[{"name": "Sam Neill"}, {"name": "Laura Dern"}, {"name": "Jeff Goldblum"}]',
        '[{"name": "Humphrey Bogart"}, {"name": "Ingrid Bergman"}, {"name": "Paul
Henreid"}]',
        '[{"name": "Clark Gable"}, {"name": "Vivien Leigh"}, {"name": "Thomas Mitchell"}]',
        '[{"name": "Judy Garland"}, {"name": "Frank Morgan"}, {"name": "Ray Bolger"}]',
        '[{"name": "Orson Welles"}, {"name": "Joseph Cotten"}, {"name": "Dorothy
Comingore"}]',
        '[{"name": "James Stewart"}, {"name": "Kim Novak"}, {"name": "Barbara Bel
Geddes"}]'
    ],
    'crew': [
        '[{"name": "Sai Yangyadatta Sethy", "job": "Director"}]',
        '[{"name": "Christopher Nolan", "job": "Director"}]',
        '[{"name": "Lana Wachowski", "job": "Director"}]',
        '[{"name": "Quentin Tarantino", "job": "Director"}]',
        '[{"name": "Francis Ford Coppola", "job": "Director"}]',
        '[{"name": "Robert Zemeckis", "job": "Director"}]',
        '[{"name": "Frank Darabont", "job": "Director"}]',
        '[{"name": "David Fincher", "job": "Director"}]',
        '[{"name": "Martin Scorsese", "job": "Director"}]',
        '[{"name": "Peter Jackson", "job": "Director"}]',
        '[{"name": "George Lucas", "job": "Director"}]',
        '[{"name": "James Cameron", "job": "Director"}]',
        '[{"name": "James Cameron", "job": "Director"}]',
        '[{"name": "Joss Whedon", "job": "Director"}]',
        '[{"name": "Steven Spielberg", "job": "Director"}]',
        '[{"name": "Michael Curtiz", "job": "Director"}]',
        '[{"name": "Victor Fleming", "job": "Director"}]',
        '[{"name": "Victor Fleming", "job": "Director"}]',
        '[{"name": "Orson Welles", "job": "Director"}]',
        '[{"name": "Alfred Hitchcock", "job": "Director"}]'
    ],
    'vote_average': [
        8.4, 8.3, 8.1, 8.5, 9.0,
        8.2, 9.2, 8.6, 8.5, 8.7,
        8.3, 7.6, 7.7, 7.8, 7.9,
        8.5, 8.0, 7.8, 8.3, 8.1
    ],
    'release_date': [
        '2008-07-18', '2010-07-16', '1999-03-31', '1994-10-14', '1972-03-24',
        '1994-07-06', '1994-09-23', '1999-10-15', '1990-09-19', '2001-12-19',
        '1977-05-25', '1997-12-19', '2009-12-18', '2012-05-04', '1993-06-11',
        '1942-11-26', '1939-12-15', '1939-08-25', '1941-09-05', '1958-07-21'
    ]
  }

  return pd.DataFrame(sample_movies)

def safe_convert(obj):

  if pd.isna(obj) or obj == '':
    return []

  if isinstance(obj, str):
    try:
      parsed = ast.literal_eval(obj)
      if isinstance(parsed, list):
        return [item.get('name', '') for item in parsed if isinstance(item, dict) and 'name' in
item]
    except (ValueError, SyntaxError):
      return []

  return []

def get_director(obj):

  if pd.isna(obj) or obj == '':
    return ''

  if isinstance(obj, str):
    try:
      parsed = ast.literal_eval(obj)
      if isinstance(parsed, list):
        for person in parsed:
          if isinstance(person, dict) and person.get('job') == 'Director':
            return person.get('name', '')
    except (ValueError, SyntaxError):
      return ''

  return ''

def extract_year(release_date):

  if pd.isna(release_date) or release_date == '':
    return 'Unknown'
  try:
    return str(pd.to_datetime(release_date).year)
  except:
    return 'Unknown'

@st.cache_data
def process_movie_features(movies_df):

  movies_df = movies_df.copy()

  movies_df['genres'] = movies_df['genres'].apply(safe_convert)
  movies_df['keywords'] = movies_df['keywords'].apply(safe_convert)
  movies_df['cast'] = movies_df['cast'].apply(lambda x: safe_convert(x)[:3])  # top 3 actors
  movies_df['crew'] = movies_df['crew'].apply(get_director)


  movies_df['release_year'] = movies_df['release_date'].apply(extract_year)


  movies_df['overview'] = movies_df['overview'].fillna('')
  movies_df['vote_average'] = movies_df['vote_average'].fillna(0.0)


  movies_df['tags'] = (
    movies_df['overview'] + ' ' +
    movies_df['genres'].apply(lambda x: ' '.join(x)) + ' ' +
    movies_df['keywords'].apply(lambda x: ' '.join(x)) + ' ' +
    movies_df['cast'].apply(lambda x: ' '.join(x)) + ' ' +
    movies_df['crew']
  )


  movies_df['tags'] = movies_df['tags'].apply(lambda x: x.lower())

  return movies_df

@st.cache_data
def compute_similarity_matrix(movies_df):

  cv = CountVectorizer(max_features=5000, stop_words='english')

  try:
    vectors = cv.fit_transform(movies_df['tags']).toarray()
    similarity = cosine_similarity(vectors)
    return similarity, True
  except Exception as e:
    st.error(f"Error computing similarity: {str(e)}")
    return None, False


API_KEY = "XYZ"

@st.cache_data
def fetch_poster(movie_title):
  if not API_KEY or API_KEY == "XYZ":
    return "https://via.placeholder.com/500x750?text=No+API+Key"

  try:
    url = f"https://api.themoviedb.org/3/search/movie?
api_key={API_KEY}&query={movie_title}"
    response = requests.get(url, timeout=10)
    response.raise_for_status()
    data = response.json()

    if data.get('results'):
        poster_path = data['results'][0].get('poster_path')
        if poster_path:
            return f"https://image.tmdb.org/t/p/w500/{poster_path}"

    return "https://via.placeholder.com/500x750?text=No+Image"

  except requests.exceptions.RequestException:
    return "https://via.placeholder.com/500x750?text=Connection+Error"
  except Exception:
    return "https://via.placeholder.com/500x750?text=Error"

def recommend(movie_title, movies_df, similarity_matrix, genre_filter=None):

  if movie_title not in movies_df['title'].values:
    return [], [], [], []

  try:
    idx = movies_df[movies_df['title'] == movie_title].index[0]

    distances = list(enumerate(similarity_matrix[idx]))

    if genre_filter and genre_filter != "All":
        filtered_distances = []
        for i, score in distances:
            movie_genres = movies_df.iloc[i]['genres']
            if genre_filter in movie_genres:
                filtered_distances.append((i, score))
        distances = filtered_distances

    movie_indices = sorted(distances, key=lambda x: x[1], reverse=True)[1:6]

    recommended_titles = []
    recommended_posters = []
    recommended_ratings = []
    recommended_years = []
    recommended_directors = []

    for i, score in movie_indices:
        title = movies_df.iloc[i]['title']
        recommended_titles.append(title)
        recommended_posters.append(fetch_poster(title))
        recommended_ratings.append(movies_df.iloc[i]['vote_average'])
        recommended_years.append(movies_df.iloc[i]['release_year'])
        recommended_directors.append(movies_df.iloc[i]['crew'])

    return recommended_titles, recommended_posters, recommended_ratings,
recommended_years, recommended_directors

  except Exception as e:
    st.error(f"Error generating recommendations: {str(e)}")
    return [], [], [], [], []

def main():

  st.markdown('<h1 class="main-header"> Movie Recommendation System</h1>',
unsafe_allow_html=True)

  with st.spinner("Loading movie data..."):
    movies_df, data_loaded = load_and_process_data()

    if not data_loaded:
        st.warning("Using sample data for demonstration. Please add your CSV files for full
functionality.")

    movies_df = process_movie_features(movies_df)
    similarity_matrix, similarity_computed = compute_similarity_matrix(movies_df)

  if not similarity_computed:
    st.error("Could not compute similarity matrix. Please check your data.")
    return

  col1, col2 = st.columns([1, 1])

  with col1:
    all_genres = set()
    for genre_list in movies_df['genres']:
        all_genres.update(genre_list)

    selected_genre = st.selectbox(
        " Select Genre (Optional):",
        ["All"] + sorted(list(all_genres))
    )

  with col2:
    if selected_genre == "All":
        filtered_movies = movies_df['title'].values
    else:
        filtered_movies = movies_df[
            movies_df['genres'].apply(lambda x: selected_genre in x)
        ]['title'].values

    selected_movie = st.selectbox(
        " Select a Movie:",
        options=filtered_movies,
        help="Choose a movie to get recommendations"
    )

  if selected_movie:
    movie_info = movies_df[movies_df['title'] == selected_movie].iloc[0]

    st.markdown("### Selected Movie Information")
    col1, col2, col3 = st.columns([1, 2, 1])

    with col1:
        poster_url = fetch_poster(selected_movie)
        st.image(poster_url, width=200)

    with col2:
        st.markdown(f"**Title:** {movie_info['title']}")
        st.markdown(f"**Director:** {movie_info['crew']}")
        st.markdown(f"**Genres:** {', '.join(movie_info['genres'])}")
        st.markdown(f"**Cast:** {', '.join(movie_info['cast'][:3])}")
        st.markdown(f"**IMDb Rating:** {movie_info['vote_average']:.1f}/10")
        st.markdown(f"**Release Year:** {movie_info['release_year']}")
        if len(movie_info['overview']) > 0:
            st.markdown(f"**Overview:** {movie_info['overview'][:200]}...")

  if st.button('Get Recommendations', type="primary", use_container_width=True):
    with st.spinner("Finding similar movies..."):
        names, posters, ratings, years, directors = recommend(selected_movie, movies_df,
similarity_matrix, selected_genre)

    if names:
        st.markdown("### Top 5 Recommendations:")

        cols = st.columns(5)
        for idx, (name, poster, rating, year, director) in enumerate(zip(names, posters, ratings,
years, directors)):
            with cols[idx]:
                st.markdown('<div class="movie-card">', unsafe_allow_html=True)
                st.image(poster, use_container_width=True)
                st.markdown(f"**{name}**")
                st.markdown(f"**Director:** {director}")
                st.markdown(f"**Rating:** {rating:.1f}/10")
                st.markdown(f"**Year:** {year}")
                st.markdown('</div>', unsafe_allow_html=True)
    else:
        st.warning("No recommendations found. Please try a different movie or genre filter.")

  st.markdown("---")
  st.markdown("""
  ###  Instructions:
  1. **Select a genre** (optional) to filter movies by specific genre
  2. **Choose a movie** from the dropdown that you enjoyed
  3. **Click 'Get Recommendations'** to find similar movies
  4. **Enjoy** discovering new movies!

  """)

if __name__ == "__main__":
  main()
```