DSBDA Mini Project Report

Title: Movie Recommendation System Using Content-Based Filtering (Scikit-Learn)

INDEX

1. Problem Statement

The increasing volume of movies available across different platforms has made it challenging for users to choose what to watch. As more content is produced globally, users often miss out on relevant, engaging, or similar films simply because of a lack of proper recommendation tools. This project aims to solve this issue by designing and developing a movie recommendation system that provides personalized movie suggestions to users. The recommendation engine uses a content-based filtering approach that analyzes the textual metadata of movies to find and recommend similar titles.

2. Objective

The main objective of this project is to:

- Build a recommendation model that understands the content of a movie and recommends similar movies.
- Enable an easy-to-use system that requires minimal user input to deliver meaningful movie suggestions.
- Integrate machine learning models with a simple graphical user interface using Streamlit.
- Enhance user viewing experience and help them discover new and relevant content.

Technology Used

Machine Learning Libraries:

- **Pandas:** Used for data manipulation and analysis.
- **Numpy:** Supports matrix operations and data arrays.
- **Scikit-learn:** Key library used to apply TF-IDF and cosine similarity for modeling.

Frontend:

- **Streamlit:** Chosen for its simplicity in creating data-driven web apps.

Development Tools:

- **Python 3+:** Primary programming language.
- **VS Code / Jupyter Notebook:** IDEs used for development and testing.

3. Introduction

With digital content consumption on the rise, the demand for personalized and efficient recommendation systems has become crucial. A movie recommendation system offers viewers suggestions based on their preferences, helping them discover content that aligns with their tastes.

Recommendation systems are categorized mainly into collaborative filtering, content-based filtering, and hybrid approaches. In this project, we utilize content-based filtering, which examines the attributes of movies, such as their descriptions (overviews), genres, cast, and more, to identify similarities.

Scikit-learn plays a significant role in this model. It offers a range of tools for feature extraction and similarity measurement, which are core to implementing content-based recommendation systems.

This system takes a user-selected movie as input and returns a list of ten most similar movies based on textual similarities in their overviews. It offers a highly interactive UI through Streamlit, making it both functional and user-friendly.

4. Architecture

The architecture of the recommendation system includes several modular components that work together to deliver personalized results. Below is a breakdown of the system design:

1. **Data Collection & Preparation:**
   - The dataset contains a list of movies with associated metadata such as titles, genres, overviews, and more. For this project, we use the `overview` column as the primary feature.
2. **Data Preprocessing:**
   - The overviews are cleaned and null values are handled. Preprocessing ensures that all descriptions are suitable for vectorization.
3. **Feature Extraction:**
   - TF-IDF Vectorizer is used to convert textual overviews into numerical vectors. TF-IDF helps emphasize words that are important in one document but not frequent in others.
4. **Similarity Measurement:**
   - Cosine similarity is used to measure how similar one movie is to others in the dataset. This helps identify and rank movies with the most similar content.
5. **Recommendation Engine:**
   - Based on the similarity matrix, the system selects the top 10 most similar movies to the selected title.

6. **Frontend Interface:**
    - The model is deployed using Streamlit. Users can interact via dropdown menus and receive recommendations dynamically.

5. Code

```
import streamlit as st

import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine_similarity


# Load dataset from local file

@st.cache_data

def load_data():

    return pd.read_csv(r"C:\Users\Lenovo\Desktop\VSC1\dsbda-lab\movie-rec-
model\movie_dataset.csv")  # adjust to actual location

  # Make sure this file is in the same folder


df = load_data()


# Handle missing values

df['overview'] = df['overview'].fillna('')


# Vectorize overview text using TF-IDF

tfidf = TfidfVectorizer(stop_words='english')

tfidf_matrix = tfidf.fit_transform(df['overview'])


# Compute cosine similarity matrix

cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)


# Create reverse mapping of movie titles to DataFrame indices
```

```
indices = pd.Series(df.index, index=df['title']).drop_duplicates()

# Recommendation function
def recommend(title):
    idx = indices.get(title)
    if idx is None:
        return ["Movie not found."]

    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)[1:11]
    movie_indices = [i[0] for i in sim_scores]
    return df['title'].iloc[movie_indices].tolist()

# Streamlit UI
st.title("Movie Recommendation System ")

selected_movie = st.selectbox("Choose a movie you like:", df['title'].dropna().unique())

if st.button("Recommend"):
    recommendations = recommend(selected_movie)
    st.subheader("Recommended Movies:")
    for i, movie in enumerate(recommendations, 1):
        st.write(f"{i}. {movie}")
```

6. Output

Upon selecting a movie, users are presented with a list of top 10 recommended titles. For example:

Input: `Inception`

Output:

1. Interstellar
2. The Prestige
3. Memento
4. The Matrix
5. Shutter Island
6. Source Code
7. Minority Report
8. Tenet
9. Edge of Tomorrow
10. Looper

These outputs are visually displayed through the Streamlit interface, offering an intuitive experience to the user.

7. Conclusion

The movie recommendation system built in this project successfully implements a content-based filtering approach using TF-IDF and cosine similarity. It accurately identifies similar movies based on textual overviews and provides meaningful suggestions to users. The integration of Streamlit as a user interface makes the model accessible and easy to interact with.

By automating the discovery of similar movies, the system not only improves user engagement but also contributes to the broader scope of recommender systems in real-world applications.

8. Future Scope

The system can be enhanced in the following ways:

1. **Use of Hybrid Filtering:** Combine content-based filtering with collaborative filtering to improve recommendation quality.
2. **API Integration:** Integrate with external APIs such as TMDB to fetch live data, posters, trailers, and user ratings.

3. **Fuzzy Matching for Search:** Add fuzzy matching to handle typos and make movie title selection more flexible.
4. **Genre and Actor Filters:** Let users apply filters for more refined recommendations.
5. **Model Deployment:** Deploy the application online using Heroku or Streamlit Cloud to allow public access.
6. **User Feedback Loop:** Incorporate a feedback mechanism to continuously learn and improve recommendations.