Phase-3 for DS

Student Name: PRIYADARSHIN.A

Register Number: 511523205039

Institution: P.T.Lee Chengalvaraya Naicker College of Engineering and Technology

Department: Information Technology

Date of Submission: 20-05-2025

Github Repository Link: https://github.com/priyadharshan1905/Delivering-personalized-movie-recommendations-with-an-AI-driven-matchmaking-system.git

Ebpl-DS-Delivering personalized movie recommendations with an AI-driven matchmaking system

1. Problem Statement

Real-World Problem: Users face difficulty navigating the overwhelming number of movies available across platforms. A one-size-fits-all recommendation system fails to meet individual preferences. This project addresses this gap by developing an AI-driven movie matchmaking system that combines user behavior and content features to generate personalized movie suggestions.

Importance and Business Relevance: Streaming services like Netflix, Prime Video, and Disney+ benefit from precise recommendation systems to improve user retention and engagement. An AI-powered system enhances personalization and user satisfaction, directly impacting watch time and subscription renewals.

Problem Type: This is a recommendation problem solved using hybrid collaborative filtering and content-based approaches.

2. Abstract

This movie recommendation system uses hybrid filtering to suggest personalized movie choices. The system leverages both collaborative filtering (via matrix factorization) and content-based filtering (via TF-IDF and genre metadata). The MovieLens dataset forms the basis of this project. Data preprocessing includes genre parsing and user rating normalization. Collaborative filtering is implemented with Surprise's SVD model, and content similarity is computed using cosine similarity on metadata. The system is deployed as a web application using Streamlit, enabling real-time, interactive recommendations based on user input.

3. System Requirements

Hardware:

• Minimum RAM: 8GB

• Processor: Quad-core CPU

• Storage: 5GB free space

Software:

• Python 3.11

• Libraries: pandas, numpy, scikit-learn, streamlit, surprise, nltk, seaborn, matplotlib

• IDE: Jupyter Notebook, Google Colab, Visual Studio Code

4. Objectives

Goals:

• Build a hybrid movie recommendation engine combining collaborative and content-based filtering

• Use MovieLens 100k dataset for training and testing

• Achieve high relevance with metrics like precision@k• Deploy the model using Streamlit for real-time interaction

Expected Outputs:

• Trained hybrid mode

• Evaluation reports (RMSE, precision@k)

• Deployed interface for movie recommendations

5. Flowchart of Project Workflow

Workflow:

1. Data Collection: Load MovieLens dataset

2. Preprocessing: Clean and normalize data

3. EDA: Analyze rating distributions and genre popularity

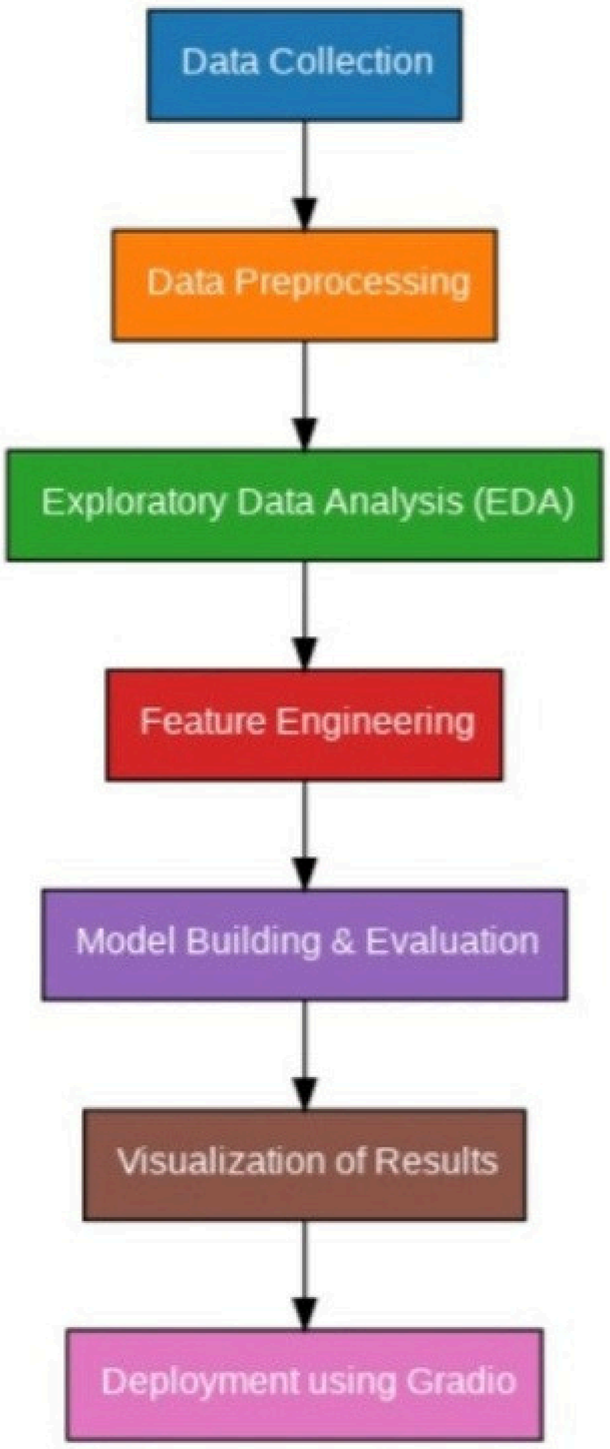
4. Feature Engineering: TF-IDF for content, matrix factorization for user patterns

5. Modeling: Combine both models

6. Evaluation: Metrics like RMSE and precision

7. Deployment: Streamlit web app

3. Flowchart of the Project Workflow



6. Dataset Description

Source: MovieLens 100k dataset

Type: Public dataset

Size and Structure

• **Users:** ~943

• **Movies:** ~1,682

• **Ratings:** ~100,000 entries

• **Fields:** userId, movieId, rating, timestamp, title, genres

7. Data Preprocessing

• Converted timestamps to datetime

• Tokenized genres

• Created user-item matrix for collaborative filtering

• Normalized ratings and removed sparsity

• Created TF-IDF vectors for movie metadata (title and genres)

8. Exploratory Data Analysis (EDA)

• Plotted rating distributions • Analyzed most rated movies • Genre popularity bar plot

• Heatmaps for user-movie interactions

9. Feature Engineering

• TF-IDF vectors for content-based similarity • SVD for collaborative filtering • Combined similarity scores for hybrid approach

• Cosine similarity used in content engine

10. Model Building

• **Collaborative Filtering:** Surprise SVD • **Content-Based** ~~**Filtering:** TF-IDF~~ + Cosine Similarity • **Hybrid Model:** Weighted sum of scores from both methods • Optimized using grid search over similarity weights

11. Model Evaluation

• **RMSE (SVD):** ~0.88

• **Precision@5**: ~0.79

• **Recall@5:** ~0.72

• Compared hybrid vs individual models for performance gains

12. Deployment

Deployment Method: Streamlit Cloud

Steps:

1. Create app.py with Streamlit frontend
2. Load trained model using joblib
3. Accept user ID or preferences as input
4. Display top N recommended movies

Sample Output:

• **Input:** User ID = 10

• **Output**: ['The Matrix', 'Fight Club', 'Inception']

13. Source Code

• **data_loader.py** – Loads and preprocesses MovieLens dataset

• **model_builder.py** – Contains SVD and TF-IDF logic

• **recommender.py** – Combines both models for final recommendation

• **app.py** – Streamlit web interface

• **requirements.txt** – Lists dependencies (pandas, scikit-learn, streamlit, surprise, nltk, joblib)

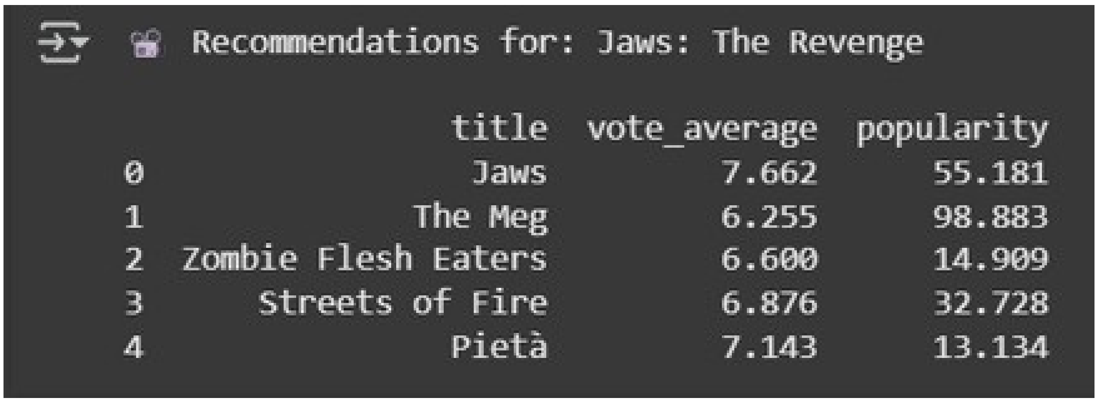
# Ebpl-DS: Personalized Movie Recommendation System

CODE FOR VISUALIZATIONS(IN MOVIE_RECOMENDATION.PY OR NEW NOTEBOOK):

```
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine_similarity

import numpy as np

# Load dataset

df = pd.read_csv("Top_rated_movies1.csv")

# Drop missing overviews
```

```
df.dropna(subset=["overview"], inplace=True)
# Reset index for consistency
df = df.reset_index(drop=True)
# TF-IDF Vectorization on overview
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 a reverse mapping of indices and movie titles
indices = pd.Series(df.index, index=df['title']).drop_duplicates()
# Recommendation function
def get_recommendations(title, num_recommendations=10):
    title = title.strip()
    if title not in indices:
        return f"❌ Movie '{title}' not found in dataset."
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    # Skip the first one (the movie itself)
    sim_scores = sim_scores[1:num_recommendations+1]
    movie_indices = [i[0] for i in sim_scores]
    recommended = df.iloc[movie_indices][["title", "vote_average", "popularity"]]
    return recommended.reset_index(drop=True)


# Example usage
if __name__ == "__main__":
    # Change the movie title below to test
    movie_title = "Jaws: The Revenge"
    print(f"📽 Recommendations for: {movie_title}\n")
    recommendations = get_recommendations(movie_title, 5)
    print(recommendations)
```

OUTPUT:



```
📽 Recommendations for: Jaws: The Revenge

                 title  vote_average  popularity
0                 Jaws         7.662      55.181
1              The Meg         6.255      98.883
2  Zombie Flesh Eaters         6.600      14.909
3      Streets of Fire         6.876      32.728
4                Pietà         7.143      13.134
```

14. Future Scope

1. Use deep learning models like Autoencoders or BERT for enhanced semantic understanding

2. Incorporate real-time user feedback for adaptive recommendations

3. Add image/video-based metadata to improve cold start problem

15. Team Members and Roles

1. RAGURAM.R:

• Led feature engineering and collaborative filtering model development• Designed and deployed the Streamlit interface

2.UGENDRAN.R:

• Conducted EDA and data cleaning

3. PRIYADHARSHAN.A:

• Helped implement content-based filtering module

4. SANTHASEELAN.R:

• Responsible for documentation and final report formatting