Phase-3 Submission

Delivering personalized movie recommendations with an AI-driven matchmaking system

Student Name: Hameetha M

Register Number:513523104023

Institution: Annai Mira College of Engineering and Technology

Department: Computer Science and Engineering

Date of Submission: 05/05/2025

Github Repository Link: https://github.com/leenihame/movie-reccomendations.git

---

1. Problem Statement:

With the increasing volume of movies and TV shows released daily, users often find it overwhelming to choose content tailored to their preferences. Traditional recommendation systems often lack personalization and adaptability to individual user behavior. There is a need for an AI-driven movie recommendation system that can deliver highly personalized suggestions using advanced matchmaking algorithms.

2. Abstract:

This project aims to develop an AI-driven matchmaking system for delivering personalized movie recommendations. The system will analyze user behavior, preferences, and movie metadata to generate accurate and adaptive suggestions. By employing machine learning models and data analytics, the system ensures a dynamic and user-centric recommendation process. The project follows a complete lifecycle from data preprocessing to modeldeployment.

3. System Requirements

Hardware:

- Processor: Intel i5 or above
- RAM: 8GB minimum
- Storage: 50GB minimum

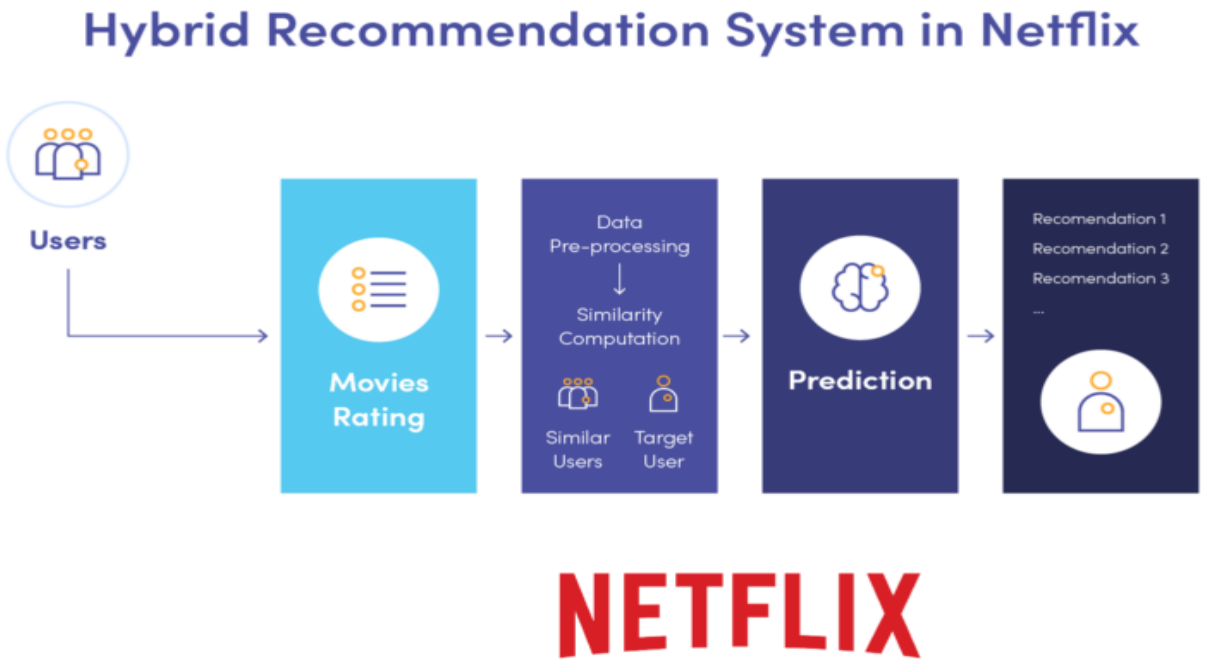
Software:

- OS: Windows/Linux
- Programming Language: Python 3.8+
- Libraries: Pandas, NumPy, Scikit-learn, Matplotlib, Seaborn, Flask/Streamlit
- Tools: Jupyter Notebook, Git, Docker (optional), Heroku or Render for deployment

4. Objectives

- Collect and analyze user and movie data
- Preprocess and clean the dataset
- Perform exploratory data analysis
- Engineer relevant features
- Build and evaluate machine learning models
- Deploy the recommendation system
- Provide an interactive interface for users

5. Flowchart of the Project Workflow



6. Dataset Description

We will utilize the MovieLens dataset, which includes over 100,000 movie ratings by thousands of users. The dataset consists of the following key components:

- **Movies.csv**: Contains movie IDs, titles, and genres
- **Ratings.csv**: Contains user IDs, movie IDs, ratings, and timestamps
- **Users.csv** (optional): May include demographic data about users
  This dataset provides a rich base for collaborative filtering and content-based filtering techniques.

7. Data Preprocessing

Data preprocessing is essential for ensuring data quality and model performance. Steps include:

- Handling missing or duplicate values
- Encoding categorical variables like genres using multi-hot encoding
- Normalizing or standardizing rating values
- Merging datasets (e.g., combining user ratings with movie metadata)
- Creating train-test splits to evaluate the model properly

8. Exploratory Data Analysis (EDA)

EDA helps understand the data patterns and relationships. Key analysis includes:

- Visualizing rating distributions across users and movies
- Identifying highly rated and most rated movies
- Understanding user behavior through histograms and boxplots
- Heatmaps to find correlations between variables
- Analyzing genre popularity trends and time-based viewing habits

9. Feature Engineering

The system will use multiple recommendation techniques:

- **Collaborative Filtering**: User-based and Item-based algorithms to find similarities
- **Matrix Factorization**: Techniques like Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF)
- **Content-Based Filtering**: Based on genres and metadata

- **Hybrid Models**: Combine collaborative and content-based methods for better performance
  Models will be implemented using Scikit-learn and Surprise libraries.

11. Model Evaluation

Evaluation ensures the model's accuracy and usefulness. We will use:

- **RMSE & MAE**: For measuring prediction errors
- **Precision@K & Recall@K**: To evaluate the quality of top-k recommendations
- **Cross-validation**: To validate performance consistency
- **Confusion Matrix**: Optional, for binary classifications like "like/dislike"
  Hyperparameter tuning with Grid Search or Random Search will also be applied.

12. Deployment

The system will be deployed using a lightweight web application:

- **Frontend**: Built using Streamlit or Flask for interactive UI
- **Backend**: Hosts the trained model and handles user requests
- **Deployment Platform**: Heroku, Render, or AWS
- **User Input**: Users will rate a few movies or choose favorite genres
- **Real-time Recommendations**: The app will generate and display recommendations dynamically

13. Source Code

Python File:movie_recommender.py

```
    import pandas as pd

import matplotlib.pyplot as plt

import os

from sklearn.metrics.pairwise import cosine_similarity

from sklearn.feature_extraction.text import TfidfVectorizer

from flask import Flask, request, render_template

# Load datasets
```

```
movies = pd.read_csv('movies.csv')

ratings = pd.read_csv('ratings.csv')

# Clean 'genres'

movies['genres'] = movies['genres'].fillna('').str.replace('|', ' ')

# TF-IDF

tfidf = TfidfVectorizer(stop_words='english')

tfidf_matrix = tfidf.fit_transform(movies['genres'])

# Similarity Matrix

similarity = cosine_similarity(tfidf_matrix)

# Save plot folder

if not os.path.exists('static'):

    os.makedirs('static')

# Top 10 Most Rated

top_movies = ratings['movieId'].value_counts().head(10)

top_titles = movies.set_index('movieId').loc[top_movies.index]['title']

plt.figure(figsize=(10, 5))

top_movies.plot(kind='bar')

plt.xticks(ticks=range(10), labels=top_titles, rotation=45)

plt.title('Top 10 Most Rated Movies')

plt.xlabel('Movie Title')

plt.ylabel('Number of Ratings')

plt.tight_layout()

plt.savefig('static/top_movies.png')
```

```
plt.close()

# Recommend Function

def recommend(title):

    if title not in movies['title'].values:

        return ["Movie not found."]

    idx = movies[movies['title'] == title].index[0]

    sim_scores = list(enumerate(similarity[idx]))

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

    movie_indices = [i[0] for i in sim_scores]

    return movies['title'].iloc[movie_indices].tolist()

# Flask App

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])

def home():

    recommendations, error = [], ''

    if request.method == 'POST':

 movie_name = request.form['movie']

        recommendations = recommend(movie_name)

        if recommendations == ["Movie not found."]:

            error = "Movie not found. Please try another title."

            recommendations = []

    return render_template('index.html', recommendations=recommendations,
image='static/top_movies.png', error=error)

if __name__ == '__main__':
```

```
app.run(debug=True)
```

Sample:movies.csv

```
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
4,Waiting to Exhale (1995),Comedy|Drama|Romance
5,Father of the Bride Part II (1995),Comedy
6,Heat (1995),Action|Crime|Thriller
7,Sabrina (1995),Comedy|Romance
8,Tom and Huck (1995),Adventure|Children
9,Sudden Death (1995),Action
10,GoldenEye (1995),Action|Adventure|Thriller
```

Sample :ratings.csv

```
userId,movieId,rating,timestamp
1,1,4.0,964982703
1,3,4.0,964981247
1,6,4.0,964982224
1,47,5.0,964983815
1,50,5.0,964982931
2,1,5.0,964982931
2,2,3.0,964982224
2,3,4.0,964982703
2,5,2.0,964983815
```

2,7,5.0,964981247

Sample:templates/index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Movie Recommender</title>
</head>
<body>
    <h1>AI Movie Recommendation System</h1>
    <form method="POST">
        <label for="movie">Enter a movie title:</label>
        <input type="text" name="movie" required>
        <button type="submit">Recommend</button>
    </form>
 {% if error %}
        <p style="color:red;">{{ error }}</p>
    {% endif %}
  {% if recommendations %}
        <h2>Recommended Movies:</h2>
        <ul>
            {% for movie in recommendations %}
                <li>{{ movie }}</li>
            {% endfor %}
        </ul>
```

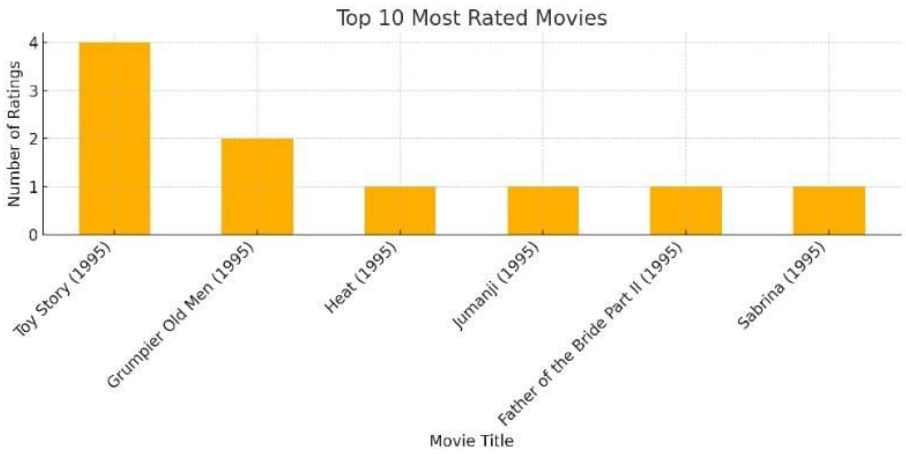
```
 {% endif %}

<img src="{{ image }}" alt="Top 10 Most Rated Movies">

</body>

</html>
```

Output:



14. Future Scope

The current recommendation system lays a strong foundation for personalized content delivery. In the future, this system can be enhanced in several impactful ways:

- **Natural Language Processing (NLP)**: Analyze user-generated reviews and comments to better understand sentiment and intent, leading to more refined recommendations.
- **Real-Time Recommendations**: Implement real-time data ingestion and model updates to adapt instantly to changing user preferences.
- **Streaming Service Integration**: Connect with platforms like Netflix, Prime Video, or YouTube via APIs to offer live suggestions based on real-time activity.
- **Voice and Chatbot Interfaces**: Introduce voice search or AI chatbot interfaces for a more interactive user experience.
- **Social Filtering**: Leverage users' social media data or friends' preferences to improve collaborative filtering.
- **Multilingual Support**: Incorporate support for multiple languages in content and user inputs to broaden the user base.
- **Mobile Application**: Build native mobile apps for wider accessibility.

These future enhancements will help make the recommendation system more intelligent, accessible, and responsive to the evolving demands of users.

15. Team Members and Roles

Names	Contribution
Hameetha	**Project Manager** – Oversees timeline, documentation, and coordination
HariHaran	**Data Analyst** – Performs EDA, feature engineering
Harishini	**ML Engineer** – Builds and optimizes models
Harish kumar	**Frontend Developer** – Designs user interface **Backend Developer** – Manages server, API, and deployment