# **Movie Recommender System**

## **1. Introduction**

### **1.1 Overview**

In the modern age of information overload, recommender systems have become essential tools to help users filter through large amounts of data. They provide personalized suggestions, improving user experience by catering content that is likely to be of interest. One prominent area where recommender systems excel is the entertainment industry, where platforms such as Netflix, Amazon Prime, and Spotify use sophisticated recommendation algorithms to enhance customer satisfaction.

This project focuses on building a **Movie Recommender System** that suggests movies to users based on their previous viewing history and ratings. The system will implement collaborative filtering techniques, which analyze the user's behavior and find patterns among users with similar preferences.

### **1.2 Problem Statement**

The objective of this project is to develop a movie recommendation system capable of suggesting movies that a user may enjoy based on their historical data of movie ratings. The goal is to leverage collaborative filtering to predict ratings for unseen movies and recommend top movies accordingly.

### **1.3 Importance of Recommender Systems**

Recommender systems have transformed how people discover content. In the movie industry, these systems help retain users, increase engagement, and maximize satisfaction by providing personalized recommendations. Recommender systems are particularly beneficial for large streaming services where hundreds of thousands of titles are available.

## **2. Tools and Technologies**

### **2.1 Programming Language**

* **Python**: Chosen for its simplicity, versatility, and the wide array of libraries available for data analysis, machine learning, and system deployment.

### **2.2 Libraries**

* **pandas**: For data manipulation and cleaning.
* **numpy**: To perform numerical operations.
* **matplotlib** & **seaborn**: To visualize data.
* **scikit-learn**: A powerful machine learning library used for building and validating the model.
* **surprise**: A specialized library for building collaborative filtering models.
* **Flask**: A lightweight web framework used for deploying the recommendation system as a service.

### **2.3 Tools Used**

* **Jupyter Notebook**: For initial code development, analysis, and visualization.
* **Visual Studio Code**: As the integrated development environment (IDE).
* **Postman**: For API testing after deploying the Flask application.

## **3. Dataset Overview**

### **3.1 Data Sources**

* **Movie Metadata**: Contains detailed information about movies such as title, genres, release date, etc.
* **User Ratings**: Contains user-generated ratings for movies, which serve as the core data for building collaborative filtering models.

### **3.2 Movie Metadata**

* **Source**: MovieLens Dataset
* **Columns**: 'id', 'title', 'genres', 'release\_date', etc.

### **3.3 Ratings Data**

* **Source**: MovieLens Ratings
* **Columns**: 'userId', 'movieId', 'rating'

### **3.4 Dataset Cleaning**

The datasets provided had missing values in some columns, particularly in release dates and certain metadata fields. The ratings dataset was cleaner, but we ensured that any NaN values were removed before proceeding with further analysis.

python

# Load datasets

movies = pd.read\_csv('movies\_metadata.csv')

ratings = pd.read\_csv('ratings\_small.csv')

# Handling missing values

movies['release\_date'].fillna('Unknown', inplace=True)

ratings.dropna(inplace=True)

# Merge datasets on movieId

movies\_ratings = pd.merge(ratings, movies, left\_on='movieId', right\_on='id')

## **4. Exploratory Data Analysis (EDA)**

Before diving into the modeling, it's important to understand the structure of the dataset, distribution of ratings, and user behaviors.

### **4.1 Distribution of Ratings**

Visualizing the distribution of ratings helps us understand how users rate movies. For example, a concentration of high ratings might indicate a user bias toward positive feedback.

python

# Visualizing rating distribution

plt.figure(figsize=(10,6))

sns.histplot(ratings['rating'], bins=10, kde=False)

plt.title('Distribution of Movie Ratings')

plt.xlabel('Rating')

plt.ylabel('Count')

plt.show()

### **4.2 Popular Movies**

Understanding which movies are frequently rated provides insight into their popularity.

python

# Number of ratings per movie

movie\_rating\_count = ratings.groupby('movieId').size()

plt.figure(figsize=(10,6))

sns.histplot(movie\_rating\_count, bins=30, kde=False)

plt.title('Number of Ratings per Movie')

plt.xlabel('Number of Ratings')

plt.ylabel('Count')

plt.show()

### **4.3 User Activity**

Looking into how many movies users tend to rate can help optimize the recommender system by focusing on users who are more active.

## **5. Collaborative Filtering Model**

### **5.1 Model Selection**

We chose **Singular Value Decomposition (SVD)** as the core algorithm for collaborative filtering. SVD is effective in matrix factorization, which breaks down the user-item interaction matrix into smaller, more manageable dimensions, helping identify latent features that can predict user preferences.

python

from surprise import SVD, Dataset, Reader

from surprise.model\_selection import cross\_validate

# Load ratings into Surprise format

reader = Reader(rating\_scale=(0.5, 5.0))

data = Dataset.load\_from\_df(ratings[['userId', 'movieId', 'rating']], reader)

# Using SVD for collaborative filtering

svd = SVD()

# Cross-validate the model

cross\_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

### **5.2 Model Training**

After cross-validation, we train the model on the entire dataset. This model is now capable of predicting user ratings for movies they haven’t seen.

python

# Training the SVD model on the full dataset

trainset = data.build\_full\_trainset()

svd.fit(trainset)

## **6. Recommendation System Implementation**

After training the model, we built the recommendation system to predict top-rated movies for a user based on the user’s past ratings.

### **6.1 Recommender Function**

python

def recommend\_movies(user\_id, num\_recommendations=10):

movie\_ids = movies['id'].unique()

# Predict ratings for all movies the user hasn't rated yet

movie\_ratings = [svd.predict(user\_id, movie\_id).est for movie\_id in movie\_ids]

# Create a DataFrame for movie ids and predicted ratings

recommendations = pd.DataFrame({

'movieId': movie\_ids,

'predicted\_rating': movie\_ratings

})

# Sort by predicted rating and return top recommendations

recommendations = recommendations.sort\_values(by='predicted\_rating', ascending=False)

top\_recommendations = recommendations.head(num\_recommendations)

# Merge with movies DataFrame to get movie titles

return pd.merge(top\_recommendations, movies, left\_on='movieId', right\_on='id')[['title', 'predicted\_rating']]

# Recommend top 10 movies for user ID 1

recommendations = recommend\_movies(1, 10)

print(recommendations)

## **7. Deployment Using Flask**

To allow users to interact with the recommender system via a web interface, we deployed the system using **Flask**, making it accessible as a service that can receive user requests and return recommendations.

### **7.1 Flask API Implementation**

python

from flask import Flask, jsonify, request

app = Flask(\_\_name\_\_)

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

def recommend():

user\_id = int(request.args.get('user\_id'))

recommendations = recommend\_movies(user\_id, 10)

return jsonify(recommendations.to\_dict('records'))

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

## **8. Evaluation and Performance**

### **8.1 Model Performance**

The performance of the SVD model was evaluated using **Root Mean Squared Error (RMSE)** and **Mean Absolute Error (MAE)**. The cross-validation results indicate that the model is reasonably accurate at predicting user ratings for unseen movies.

* **RMSE**: 0.87
* **MAE**: 0.68

### **8.2 Limitations**

* The model primarily uses collaborative filtering, which means it struggles when there is no prior data on new users or new movies (cold start problem).
* The system does not consider movie metadata, such as genres or directors, for recommendations.

## **9. Conclusion and Future Work**

### **9.1 Conclusion**

The Movie Recommender System was successfully implemented using collaborative filtering (SVD). The system is capable of providing personalized movie recommendations to users based on their previous ratings. The model has been deployed using Flask, providing an easy-to-use API for generating recommendations.

### **9.2 Future Enhancements**

1. **Content-Based Filtering**: Incorporating movie metadata, such as genres, cast, and directors, to enhance the recommendation process.

2. **Hybrid Recommender**: Combining collaborative and content-based filtering for a more robust recommendation system.

3. **Improved User Interface**: Developing a front-end interface for better user interaction with the recommender system.