**Movie recommendation system**

Creating a movie recommendation system using AI, machine learning, and Python.

1. \*\***Data Collection and Preprocessing**:\*\*

- Obtain a dataset: Use datasets like MovieLens, IMDb, or TMDB.

- Clean and preprocess the data: Handle missing values, normalize ratings, and process textual data (e.g., movie titles, genres).

2. \*\***Exploratory Data Analysis (EDA):**\*\*

- Analyze data distribution (e.g., ratings, genres).

- Visualize relationships and trends in the data.

3. \*\***Recommendation Algorithms**:\*\*

- \*\*Content-Based Filtering:\*\* Recommend movies based on the similarity to movies a user has liked in the past.

- Use features like genres, directors, actors.

- Calculate similarity using methods like cosine similarity.

- \*\*Collaborative Filtering:\*\* Recommend movies based on user-user or item-item interactions.

- User-User Collaborative Filtering: Find users similar to the target user and recommend movies they liked.

- Item-Item Collaborative Filtering: Find movies similar to those the target user liked and recommend them.

- Implement using techniques like Matrix Factorization (e.g., SVD, ALS).

- \*\*Hybrid Models:\*\* Combine both content-based and collaborative filtering to improve recommendations.

4. \*\***Model Evaluation**:\*\*

- Split the data into training and test sets.

- Use metrics like RMSE, MAE, precision, recall, and F1-score to evaluate the model.

5. \*\***Deployment**:\*\*

- Build a user interface (UI) for users to interact with the recommendation system.

- Use frameworks like Flask or Django for web deployment.

6. \*\***Scalability and Optimization**:\*\*

- Optimize the model for faster predictions.

- Use techniques like caching and indexing to improve performance.

- Scale the system to handle large datasets and multiple users.

**Basic framework /Implementation**:

### 1. **Data Collection and Preprocessing**

```python

import pandas as pd

# Load datasets

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

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

# Preprocessing

# Merge datasets if necessary

data = pd.merge(ratings, movies, on='movieId')

# Handle missing values, etc.

data.dropna(inplace=True)

```

### 2. **Content-Based Filtering**

```python

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import linear\_kernel

# Vectorize the genres

tfidf = TfidfVectorizer(stop\_words='english')

tfidf\_matrix = tfidf.fit\_transform(movies['genres'])

# Compute cosine similarity

cosine\_sim = linear\_kernel(tfidf\_matrix, tfidf\_matrix)

# Function to get recommendations

def get\_recommendations(title, cosine\_sim=cosine\_sim):

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

sim\_scores = list(enumerate(cosine\_sim[idx]))

sim\_scores = sorted(sim\_scores, key=lambda x: x[1], reverse=True)

sim\_scores = sim\_scores[1:11] # Get top 10

movie\_indices = [i[0] for i in sim\_scores]

return movies['title'].iloc[movie\_indices]

print(get\_recommendations('Toy Story (1995)'))

```

### 3. **Collaborative Filtering**

```python

import numpy as np

from scipy.sparse.linalg import svds

# Create user-item matrix

user\_movie\_ratings = data.pivot(index='userId', columns='movieId', values='rating').fillna(0)

# Perform matrix factorization

U, sigma, Vt = svds(user\_movie\_ratings, k=50)

sigma = np.diag(sigma)

# Predict ratings

predicted\_ratings = np.dot(np.dot(U, sigma), Vt)

predicted\_ratings\_df = pd.DataFrame(predicted\_ratings, columns=user\_movie\_ratings.columns)

# Function to recommend movies to a user

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

user\_row\_number = user\_id - 1

sorted\_user\_predictions = predicted\_ratings\_df.iloc[user\_row\_number].sort\_values(ascending=False)

user\_data = data[data.userId == user\_id]

recommendations = movies[~movies['movieId'].isin(user\_data['movieId'])].merge(

pd.DataFrame(sorted\_user\_predictions).reset\_index(), how='left',

left\_on='movieId', right\_on='movieId').rename(columns={user\_row\_number: 'Predictions'}).sort\_values('Predictions', ascending=False)

return recommendations.head(num\_recommendations)

print(recommend\_movies(1))

```

### 4. **Model Evaluation**

```python

from sklearn.metrics import mean\_squared\_error

# Evaluate using RMSE

rmse = np.sqrt(mean\_squared\_error(user\_movie\_ratings, predicted\_ratings))

print(f'RMSE: {rmse}')

```

### 5. **Deployment**

```python

from flask import Flask, request, jsonify

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

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

def recommend():

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

recommendations = recommend\_movies(user\_id)

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

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

app.run()

```