Delivering personalized movie recommendations with an AI-driven matchmaking system

Source code

df['tagline'].fillna("Unknown", inplace=True)

df['runtime'].fillna(df['runtime'].median(), inplace=True)

df.dropna(subset=['title', 'genres'], inplace=True)

df.drop\_duplicates(inplace=True)

df = df[df['runtime'] > 0]

df = df[df['runtime'] < 300]

from sklearn.preprocessing import MultiLabelBinarizer

df['genres'] = df['genres'].apply(lambda x: [i['name'] for i in eval(x)] if pd.notnull(x) else [])

mlb = MultiLabelBinarizer()

genre\_encoded = mlb.fit\_transform(df['genres'])

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df[['popularity', 'vote\_average', 'runtime']] = scaler.fit\_transform(df[['popularity', 'vote\_average', 'runtime']])

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

plt.title("Distribution of Ratings")

plt.xlabel("Rating")

plt.ylabel("Frequency")

# Explode genres column and merge with ratings

boxplot\_data = merged\_df.explode('genres')

sns.boxplot(data=boxplot\_data, x='genres', y='rating')

plt.xticks(rotation=90)

plt.title("Boxplot of Ratings per Genre")

user\_activity = ratings['user\_id'].value\_counts()

sns.histplot(user\_activity, bins=50, kde=True)

plt.title("User Activity Distribution")

# Create features like average rating, rating count, genre dummies

movie\_features = movies\_df.merge(ratings.groupby('movie\_id').agg({'rating':['mean','count']}), on='movie\_id')

correlation\_matrix = movie\_features.corr(numeric\_only=True)

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

movie\_recommender/

│

├── app.py # Main Gradio app

├── model.py # Model logic: hybrid recommender

├── data\_loader.py # Load and preprocess dataset

├── utils.py # Helper functions

├── requirements.txt # Python dependencies

└── sample\_data/

└── movies.csv # Dataset (movie metadata, ratings, etc.)

import gradio as gr

from model import hybrid\_recommender

def recommend\_movies(user\_id, preferred\_genres, language):

recommendations = hybrid\_recommender(user\_id, preferred\_genres.split(','), language)

return "\n".join([f"{movie['title']} - Predicted Rating: {movie['rating']:.1f}" for movie in recommendations])

iface = gr.Interface(

fn=recommend\_movies,

inputs=[

gr.Number(label="User ID"),

gr.Textbox(label="Preferred Genres (comma-separated)"),

gr.Textbox(label="Language")

],

outputs="text",

title="AI Movie Recommender",

description="Get top 5 personalized movie recommendations using an AI-powered hybrid model."

)

if \_\_name\_\_ == "\_\_main\_\_":

iface.launch()

import pandas as pd

from data\_loader import load\_data

from utils import compute\_similarity\_score

movies, ratings = load\_data()

def hybrid\_recommender(user\_id, genres, language, top\_n=5):

# Simple logic to simulate hybrid recommendation

user\_ratings = ratings[ratings['userId'] == user\_id]

avg\_movie\_ratings = ratings.groupby('movieId')['rating'].mean().reset\_index()

merged = pd.merge(movies, avg\_movie\_ratings, on='movieId')

# Filter by preferred genres and language

filtered = merged[merged['genres'].str.contains('|'.join(genres), case=False)]

filtered = filtered[filtered['language'].str.lower() == language.lower()]

# Simulate recommendation score (could be improved with embeddings)

filtered['score'] = filtered['rating'] + 0.1 \* filtered.index % 5

top\_recommendations = filtered.sort\_values(by='score', ascending=False).head(top\_n)

return top\_recommendations[['title', 'rating']].to\_dict('records')

import pandas as pd

def load\_data():

movies = pd.read\_csv("sample\_data/movies.csv") # Include columns like movieId, title, genres, language

ratings = pd.read\_csv("sample\_data/ratings.csv") # Include columns like userId, movieId, rating

return movies, ratings

def compute\_similarity\_score(movie\_a, movie\_b):

# Placeholder for cosine similarity or other metric

return 0.9

pandas

gradio

scikit-learn

numpy