Delivering personalized movie recommendations with an AI driven matchmaking system

**PROGRAM**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.regularizers import l2

# Sample movie ratings data (expanded)

ratings = {

    "user1": {"movie1": 5, "movie2": 3, "movie3": 4, "movie4": 2, "movie5": 5},

    "user2": {"movie1": 4, "movie2": 5, "movie3": 3, "movie4": 5, "movie5": 1},

    "user3": {"movie1": 3, "movie2": 4, "movie3": 5, "movie4": 1, "movie5": 4},

    "user4": {"movie1": 2, "movie2": 3, "movie4": 4, "movie5": 5, "movie6": 3},

    "user5": {"movie2": 1, "movie3": 2, "movie4": 5, "movie5": 4, "movie6": 2},

    "user6": {"movie1": 5, "movie3": 5, "movie5": 3, "movie6": 4, "movie7": 5},

    "user7": {"movie1": 4, "movie2": 4, "movie4": 2, "movie6": 1, "movie7": 4},

    "user8": {"movie2": 2, "movie3": 3, "movie5": 5, "movie7": 2, "movie8": 5},

    "user9": {"movie1": 3, "movie4": 4, "movie6": 3, "movie8": 4, "movie9": 4},

    "user10": {"movie3": 1, "movie5": 2, "movie7": 5, "movie9": 3, "movie10": 5},

}

ratings\_df = pd.DataFrame(ratings).fillna(0)  # Fill missing ratings with 0

# 1. Data Preprocessing

def preprocess\_data(df):

    # Convert DataFrame to a user-movie matrix

    user\_movie\_matrix = df.T  # Transpose for user-centric rows

    # Split data into training and testing sets (80/20 split)

    train\_data, test\_data = train\_test\_split(user\_movie\_matrix, test\_size=0.2, random\_state=42)

    # Scale the data using StandardScaler

    scaler = StandardScaler()

    scaled\_train\_data = scaler.fit\_transform(train\_data)

    scaled\_test\_data = scaler.transform(test\_data)

    return scaled\_train\_data, train\_data, scaled\_test\_data, test\_data, scaler #Return the scaler

# 2. Model Building

def build\_model(input\_dim):

    model = Sequential([

        Dense(128, activation='relu', input\_shape=(input\_dim,), kernel\_regularizer=l2(0.001)),

        Dropout(0.5),

        Dense(64, activation='relu', kernel\_regularizer=l2(0.001)),

        Dropout(0.3),

        Dense(32, activation='relu', kernel\_regularizer=l2(0.001)),

        Dense(input\_dim, activation='linear')  # Output layer with same dimension as input

    ])

    model.compile(optimizer=Adam(learning\_rate=0.001), loss='mse')

    return model

# 3. Model Training

def train\_model(model, train\_data, epochs=50, batch\_size=32, validation\_data=None):

    early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

    history = model.fit(

        train\_data,

        train\_data,  # Autoencoder: input and target are the same

        epochs=epochs,

        batch\_size=batch\_size,

        validation\_data=validation\_data,

        callbacks=[early\_stopping]

    )

    return history

# 4. Recommendation Generation

def get\_user\_recommendations(model, user\_id, all\_movie\_names, user\_movie\_matrix, scaler, num\_recommendations=5):

    if user\_id not in user\_movie\_matrix.index:

        print(f"User '{user\_id}' not found.")

        return []

    user\_ratings = user\_movie\_matrix.loc[user\_id].values.reshape(1, -1)

    # Scale the user's ratings using the same scaler

    scaled\_user\_ratings = scaler.transform(user\_ratings)

    # Predict the user's ratings for all movies

    predicted\_ratings = model.predict(scaled\_user\_ratings)

    # Inverse transform the scaled prediction to get actual ratings

    predicted\_ratings = scaler.inverse\_transform(predicted\_ratings)

    # Create a DataFrame of movie names and predicted ratings

    movie\_ratings\_pred = pd.DataFrame({

        'Movie': all\_movie\_names,

        'Predicted\_Rating': predicted\_ratings[0]

    })

    # Get the movies the user has already rated

    rated\_movies = user\_movie\_matrix.columns[user\_movie\_matrix.loc[user\_id] > 0]

    # Filter out the movies the user has already rated

    recommendations = movie\_ratings\_pred[~movie\_ratings\_pred['Movie'].isin(rated\_movies)]

    # Sort by predicted rating and get top N recommendations

    top\_recommendations = recommendations.nlargest(num\_recommendations, 'Predicted\_Rating')

    return top\_recommendations['Movie'].tolist()

def get\_all\_movie\_names(df):

    movie\_names = set()

    for user\_ratings in df.values():

        movie\_names.update(user\_ratings.keys())

    return list(movie\_names)

def main():

    # 1. Load and Preprocess Data

    all\_movie\_names = get\_all\_movie\_names(ratings)

    scaled\_train\_data, train\_data, scaled\_test\_data, test\_data, scaler = preprocess\_data(ratings\_df)

    input\_dim = scaled\_train\_data.shape[1] # Number of movies

    # 2. Build Model

    model = build\_model(input\_dim)

    # 3. Train Model

    print("Training the model...")

    train\_history = train\_model(model, scaled\_train\_data, epochs=100, batch\_size=32, validation\_data=(scaled\_test\_data, scaled\_test\_data))

    # 5. Generate Recommendations for a User

    user\_id = "user1"

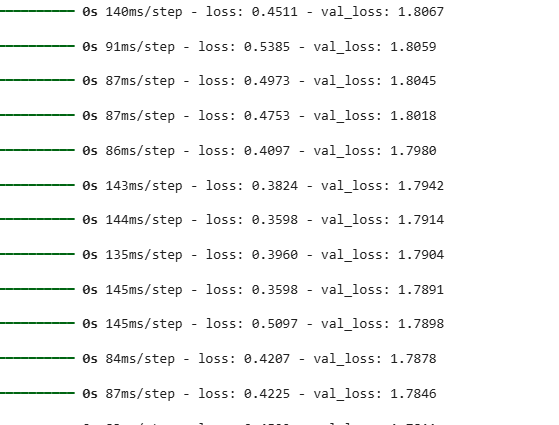
    num\_recommendations = 5

    recommendations = get\_user\_recommendations(model, user\_id, all\_movie\_names, ratings\_df.T, scaler, num\_recommendations) # Pass the scaler

    print(f"\nMovie recommendations for {user\_id}: {recommendations}")

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

    main()

**OUTPUT**