**Movie Recommendation System**

**1. Project Overview**

This project is designed to build a **Movie Recommendation System** using various machine learning approaches such as:

* **Collaborative Filtering**
* **Content-Based Filtering**
* **Hybrid Filtering**

The recommendation system aims to suggest movies to users based on their preferences and interactions (e.g., ratings) or item features (e.g., genres).

**2. Preprocessing for Model Building**

**2.1 Genre Encoding**

* The genres column contains multiple genres for each movie, separated by the | symbol.
* **Objective**: Transform genres into a numerical format for model compatibility.
* **Implementation**:
  1. Split the genres string into a list using str.split('|').
  2. Apply MultiLabelBinarizer to encode genres into a one-hot encoded format.
  3. Concatenate the resulting encoded genres with the original dataset.

**Code -**

final\_data['genres\_list'] = final\_data['genres'].apply(lambda x: x.split('|'))

mlb = MultiLabelBinarizer()

genres\_encoded = pd.DataFrame(mlb.fit\_transform(final\_data['genres\_list']), columns=mlb.classes\_)

final\_data = pd.concat([final\_data, genres\_encoded], axis=1)

**2.2 Timestamp Conversion and Scaling**

* The rating\_timestamp column is converted from string to seconds since the epoch using:
  + pd.to\_datetime() for datetime conversion.
  + .astype('int64') // 10\*\*9 to convert nanoseconds to seconds.
* Scaling both rating and rating\_timestamp to a range of [0, 1] using MinMaxScaler.

**Code -**

final\_data['rating\_timestamp'] = pd.to\_datetime(final\_data['rating\_timestamp']).astype('int64') // 10\*\*9

scaler = MinMaxScaler()

final\_data[['rating', 'rating\_timestamp']] = scaler.fit\_transform(final\_data[['rating', 'rating\_timestamp']])

**2.3 User-Item Matrix**

* A matrix where rows represent userId, columns represent movieId, and values represent ratings.
* **Objective**: Prepare input for collaborative filtering models.

**Code -**

user\_item\_matrix = final\_data.pivot\_table(index='userId', columns='movieId', values='rating', fill\_value=0)

**3. Data Splitting**

The dataset is split into training and testing sets using train\_test\_split from the Surprise library.

* **Purpose**:
  + Training set: Build and train the recommendation model.
  + Testing set: Evaluate model performance.

**Code -**

reader = Reader(rating\_scale=(0.5, 5))

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

trainset, testset = train\_test\_split(data, test\_size=0.2)

**4. Recommendation Algorithms**

**4.1 Collaborative Filtering**

**Description**

* Recommends items based on user-item interaction data.
* **User-Based Collaborative Filtering**: Finds similar users to recommend movies.
* **Item-Based Collaborative Filtering**: Finds movies similar to those the user liked.

**Implementation with SVD**

* **SVD**: Matrix factorization technique that reduces dimensionality to find latent features.
* Steps:
  1. Train an SVD model on the training set.
  2. Use the model to predict ratings for the test set.
  3. Recommend top N movies with the highest predicted ratings.

**Code -**

model = SVD()

model.fit(trainset)

def recommend\_movies\_collaborative(user\_id, model, data, n\_recommendations=5):

all\_movie\_ids = set(data['movieId'])

rated\_movie\_ids = set(data[data['userId'] == user\_id]['movieId'])

unrated\_movie\_ids = list(all\_movie\_ids - rated\_movie\_ids)

predicted\_ratings = [

(movie\_id, model.predict(user\_id, movie\_id).est)

for movie\_id in unrated\_movie\_ids

]

predicted\_ratings = sorted(predicted\_ratings, key=lambda x: x[1], reverse=True)

top\_recommendations = [movie\_id for movie\_id, rating in predicted\_ratings[:n\_recommendations]]

return top\_recommendations

**4.2 Content-Based Filtering**

**Description**

* Recommends movies similar to those the user liked based on item features such as genres.

**Implementation**

1. Compute the **TF-IDF Matrix** for the genres column.
2. Calculate pairwise cosine similarity for all movies.
3. For a given movie, recommend the top N most similar movies.

**Code -**

tfidf = TfidfVectorizer(tokenizer=lambda x: x.split('|'))

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

cosine\_sim = cosine\_similarity(tfidf\_matrix, tfidf\_matrix)

def recommend\_movies(movie\_idx, cosine\_sim, data, top\_n=10):

similar\_movies = list(enumerate(cosine\_sim[movie\_idx]))

similar\_movies = sorted(similar\_movies, key=lambda x: x[1], reverse=True)

recommendations = [data.iloc[i[0]].title for i in similar\_movies[1:top\_n+1]]

return recommendations

**4.3 Hybrid Filtering**

**Description**

* Combines collaborative and content-based filtering predictions to leverage the strengths of both methods.

**Implementation**

1. Generate predictions using content-based and collaborative filtering.
2. Combine predictions with equal weights.

**Code -**

content\_pred = [cosine\_sim[idx].mean() for idx in user\_movie\_indices]

collaborative\_pred = [pred.est for pred in model.test(testset) if pred.uid == specific\_user\_id]

final\_pred = [0.5 \* c + 0.5 \* t for c, t in zip(collaborative\_pred, content\_pred)]

**5. Deployment with Gradio**

Gradio is used to create a user-friendly interface for recommending movies based on user input.

* **Interface Setup**:
  + Input: Dropdown menu of movies.
  + Output: Markdown displaying recommended movies.

**Code -**

interface = gr.Interface(

fn=recommend\_movies,

inputs=gr.Dropdown(choices=movie\_list, label="Select a Movie"),

outputs=gr.Markdown(label="Recommended Movies")

)

interface.launch()

**6. Key Results**

* **Collaborative Filtering**: Generates recommendations based on user ratings.
* **Content-Based Filtering**: Suggests movies similar to the genres the user prefers.
* **Hybrid Filtering**: Provides a more balanced recommendation by combining both approaches.

### **7. Conclusion**

The **Movie Recommendation System** successfully demonstrates the application of machine learning and data preprocessing techniques to deliver personalized movie suggestions. By incorporating collaborative filtering, content-based filtering, and hybrid filtering, the project highlights the strengths of each approach and combines them effectively for enhanced accuracy and versatility.

**Key Achievements:**

1. **Comprehensive Preprocessing**:
   * Genres were encoded using one-hot encoding.
   * Ratings and timestamps were normalized to improve model performance.
   * A user-item matrix was constructed to enable collaborative filtering.
2. **Algorithm Implementation**:
   * **Collaborative Filtering**: Accurately recommended movies based on user-item interaction data.
   * **Content-Based Filtering**: Leveraged movie features like genres to suggest similar movies.
   * **Hybrid Filtering**: Combined collaborative and content-based predictions to mitigate limitations and enhance recommendation quality.
3. **Deployment**:
   * A user-friendly interface was built using Gradio, making the system easily accessible and interactive for end-users.

**Insights and Learnings:**

* **Data-Driven Recommendations**:  
  The system shows how user ratings and movie features can be leveraged to build highly personalized recommendation systems.
* **Balancing Techniques**:  
  Combining multiple algorithms (e.g., hybrid filtering) often yields better results than relying on a single technique.
* **Scalability**:  
  The modular design allows for seamless integration of additional features like user demographics or movie metadata.

**Future Enhancements:**

1. Incorporate **user demographics** (age, location, etc.) to improve personalization.
2. Add **real-time updates** for new movies or ratings.
3. Optimize performance for large datasets using distributed computing or deep learning approaches.
4. Implement an **evaluation pipeline** to measure metrics like Mean Absolute Error (MAE) or Root Mean Square Error (RMSE).

This project serves as a foundational step toward understanding and building recommendation systems. With further enhancements, the system can be adapted for larger datasets and real-world applications, offering a scalable and robust solution for personalized recommendations.

Thank you for exploring this project!