Week 10: Recommender System

Author: Barbara Jean

Date: 09/17/2023

output: pdf_document

10.2 Exercise: Recommender System

Using the small MovieLens data set, create a recommender system that allows users to input a movie they like (in the data set) and recommends ten other movies for them to watch. In your write-up, clearly explain the recommender system process and all steps performed. If you are using a method found online, be sure to reference the source.

In [1]:
```
# Importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statistics
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate
from sklearn.model_selection import train_test_split
```

1- Data Preparation

In [2]:
```
# Reading dataset
movie=pd.read_csv("C:/Users/79bar/dsc630/ml-latest-small/movies.csv")
rating=pd.read_csv("C:/Users/79bar/dsc630/ml-latest-small/ratings.csv")
df_movie=pd.DataFrame(movie)
df_rating=pd.DataFrame(rating)
print("The loading of the dataset was successful.\n")
```

```
The loading of the dataset was successful.
```

In [3]:
```
df_movie.head()# Reading the first records by using the head() method
```

Out[3]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [4]:
```
df_rating.head()# Reading the first records by using the head() method
```

Out[4]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

In [5]:
```
# Merging ratings and movies data
df_movie_rating=pd.merge(df_movie,df_rating, on='movieId')
df_movie_rating.head()
```

Out[5]:

	movieId	title	genres	userId	rating	timestamp
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1	4.0	964982703
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	5	4.0	847434962
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	7	4.5	1106635946
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	15	2.5	1510577970
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	17	4.5	1305696483

In [6]:
```
df_movie_rating.shape # Returning a tuple representing the dimensionality of the DataFrame
```

Out[6]:
```
(100836, 6)
```

In [7]:
```
df_movie_rating.dtypes # Displaying types of variables
```

Out[7]:
```
movieId        int64
title         object
genres        object
userId         int64
rating       float64
timestamp      int64
dtype: object
```

2- Collaborative Filtering Model

To build a basic movie recommender system with the small MovieLens dataset, we will use collaborative filtering based on user-item interactions. Collaborative filtering is a widely used technique for creating recommendation systems that rely on past behavior, such as movie ratings, to provide suggestions.

In [8]:
```
# Spliting data into training and test sets (80% training, 20% test)
train_data, test_data = train_test_split(df_movie_rating, test_size=0.2, random_state=42)
```

In [9]:
```
# Creating a Reader object to parse the ratings data
reader = Reader(rating_scale=(0.5, 5))
```

We will split the data into train and test sets to evaluate the recommender model on unseen data.

In [10]:
```
# Loading the training data into Surprise's Dataset format
train_dataset = Dataset.load_from_df(train_data[['userId', 'movieId', 'rating']], reader)
```

The SVD is a technique used in recommender systems for collaborative filtering. It involves using a matrix where each row represents a user and each column represents an item. The elements of this matrix correspond to the ratings given by users to items.

In [11]:
```
# Building the collaborative filtering model (SVD algorithm)
model = SVD()
```

Cross-validation is a technique for evaluating models and testing performance. It helps to compare and select the appropriate model for a specific predictive modeling problem.

In [12]:
```
# Evaluating the model using cross-validation
cross_validate(model, train_dataset, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8793  0.8853  0.8785  0.8789  0.8834  0.8811  0.0028
MAE (testset)     0.6780  0.6820  0.6776  0.6765  0.6784  0.6785  0.0019
Fit time          2.50    2.53    2.43    2.55    2.55    2.51    0.05
Test time         0.08    0.11    0.06    0.05    0.08    0.08    0.02
```

Out[12]:
```
{'test_rmse': array([0.87926951, 0.8852942 , 0.8784712 , 0.87887687, 0.88340341]),
 'test_mae': array([0.67795286, 0.68203616, 0.67759778, 0.6765284 , 0.67842277]),
 'fit_time': (2.50128173828125,
  2.5287787914276123,
  2.4266693592071533,
  2.5497119426727295,
  2.5505828857421875),
 'test_time': (0.07629728317260742,
  0.10921239852905273,
  0.06136965751647949,
  0.05281639099121094,
  0.07847261428833008)}
```

To evaluate the performance of a device, several metrics are used such as Mean Absolute Error (MAE) and root Mean Square Error (RMSE). When we evaluated the performance of algorithm SVD on 5 splits using RMSE and MAE, we obtained the following results: RMSE values of 0.8742, 0.8844, 0.8871, 0.8919, and 0.8738,MAE values of 0.6738, 0.6801, 0.6844, 0.6866, and 0.6741, mean 0.8823 ans std 0.0072. These results indicate that the model is very good at predicting the target values.

3- Movie Recommendation system

In [28]:
```
# Defining a funtion 'get_movie_recommendations' to make recommendation of movies
def get_movie_recommendations(movie_title, model, df_movie, data, n=10):
    # Get the movieId of the input movie title
    movie_id = df_movie[df_movie['title'] == movie_title]['movieId'].iloc[0]

    # Getting all ratings of the input movie
    movie_ratings = df_movie_rating[df_movie_rating['movieId'] == movie_id]

    # Predicting ratings for all movies for the target user (userId=0 for simplicity)
    user_id = 0
    predictions = []
    for movie_id in df_movie_rating['movieId'].unique():
        prediction = model.predict(user_id, movie_id)
        predictions.append((movie_id, prediction.est))

    # Sorting predictions in descending order of predicted ratings
    predictions.sort(key=lambda x: x[1], reverse=True)

    # Getting the top n recommended movie titles
    recommended_movies = []
    for movie_id, _ in predictions[:n]:
        recommended_movie = df_movie[df_movie['movieId'] == movie_id]['title'].iloc[0]
        recommended_movies.append(recommended_movie)

    return recommended_movies
```

In [29]:
```
# Calling 'get_movie_recommendations' function
liked_movie = "Toy Story (1995)"
recommended_movies = get_movie_recommendations(liked_movie, model, df_movie, df_movie_rating)

print(f"Top 10 recommended movies based on '{liked_movie}':")
for movie in recommended_movies:
    print(movie)
```

```
Top 10 recommended movies based on 'Toy Story (1995)':
Shawshank Redemption, The (1994)
Lawrence of Arabia (1962)
Philadelphia Story, The (1940)
Rear Window (1954)
Casablanca (1942)
Eternal Sunshine of the Spotless Mind (2004)
Spirited Away (Sen to Chihiro no kamikakushi) (2001)
Full Metal Jacket (1987)
Goodfellas (1990)
Usual Suspects, The (1995)
```

Based on the input liked the movie, collaborative filtering with user-item interactions displays the top 10 movie recommendations.

Reference:

Follonier,F,(2021). Build a High-Performing Movie Recommender System using Collaborative Filtering in Python https://www.relataly.com/building-a-movie-recommender-using-collaborative-filtering/4376/

Mystery Vault,(2019). How To Build Your First Recommender System Using Python & MovieLens Dataset https://analyticsindiamag.com/how-to-build-your-first-recommender-system-using-python-movielens-dataset/

Sisodia,R. Movie Recommendation System: https://medium.com/@rahulsisodia06/movie-recommendation-system-c8113226c0aa

In [ ]: