Movie Recomender

Movie Recommendation System Using Python

Description

To build a Movie Recommendations with Movielens Dataset: Almost everyone today uses technology to stream movies and television shows. While figuring out what to stream next can be daunting, recommendations are often made based on a viewer's history and preferences. This is done through machine learning and can be a fun and easy project for beginners to take on. New programmers can practice by coding in either Python or R languages and with data from the Movielens Dataset. Generated by more than 6,000 users, Movielens currently includes more than 1 million movie ratings of 3,900 films.

References

- Concepts
- Hands-on recommendation system
- Dataset

In [1]:

```
import pandas as pd
import os
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import StandardScaler
pd.set_option('display.max_columns', 20)


class DataPreprocessing:
    def __init__(self, dataset_path):
        self.path = dataset_path
        self.dataset = None
        self.data_processed = None
        self.data_processed_scaled = None
        self.ratings = None
        self.movies = None
    def import_dataset(self):
        if self.dataset == None:
            # importing the dataset
            self.ratings = pd.read_csv(self.path + '/ratings.csv')
            self.movies = pd.read_csv(self.path + '/movies.csv')
            self.dataset = self.ratings.merge(self.movies)

    def process_data(self):
        # Data cleaning
        if self.ratings is None:
            self.import_dataset()
        ratings_ = self.ratings.drop(['timestamp'], axis=1)
        no_movies_voted = ratings_.groupby('userId')['rating'].agg('count')
        ratings_ = ratings_.loc[no_movies_voted[no_movies_voted > 10].index, :]
        return ratings_

    def get_movies_data(self):
        return self.movies

    def get_final_data(self):
        # creating the final dataset containing the movies ,user and their ratings
        self.data_processed = self.process_data()
        self.data_processed = self.data_processed.pivot(
            index='movieId', columns='userId', values='rating')
        self.data_processed.fillna(0, inplace=True)
        self.data_processed.reset_index(inplace=True)
        return self.data_processed

    def scale_data(self):
        # Scaling the ratings matrix using Standard Scaler
        scaler = StandardScaler(with_mean=False)
        self.data_processed_scaled = scaler.fit_transform(self.data_processed)
        return self.data_processed_scaled

    def get_csr_matrix(self):
        # Removing Sparsity
        try:
            csr_data = csr_matrix(self.data_processed_scaled)
            return csr_data
        except Exception:
            pass
```

```python
class Model:

    # Using K Nearest Neighbours to find the recomendations using the similarity betwen the movies
    def __init__(self):
        self.model = NearestNeighbors(
            metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
        self.train_data = None

    def train(self, train_data):
        self.train_data = train_data
        self.model.fit(self.train_data)

    def get_model(self):
        return self.model



class Recommender:
    def __init__(self, dataset, movies, model):
        self.dataset = dataset
        self.movies = movies
        self.model = model

    def get_movie_recommendation(self, movie_name, csr_matrix, number_of_recommendations=10):
        try:
            movie_list = self.movies[self.movies['title'].str.lower().str.contains(
                movie_name.lower())]
            if len(movie_list):
                movie_idx = movie_list.iloc[0]['movieId']
                movie_idx = self.dataset[self.dataset['movieId']
                                         == movie_idx].index[0]
                distances, indices = self.model.kneighbors(
                    csr_matrix[movie_idx], n_neighbors=number_of_recommendations+1)
                rec_movie_indices = sorted(list(zip(indices.squeeze().tolist(
                ), distances.squeeze().tolist())), key=lambda x: x[1])[:0:-1]
                recommend_frame = []
                for val in rec_movie_indices:
                    movie_idx = self.dataset.iloc[val[0]]['movieId']
                    idx = self.movies[self.movies['movieId']
                                      == movie_idx].index
                    recommend_frame.append(
                        {'Title': self.movies.iloc[idx]['title'].values[0], 'Distance': val[1]})
                df = pd.DataFrame(recommend_frame, index=range(
                    number_of_recommendations, 0, -1))[::-1]["Title"]
                return df.values
            else:
                return "No movies found. Please check your input"
        except IndexError:
            return "No Recomendations Found"


class RecommendationSystem:

    def __init__(self, dataset_path):
        self.path = dataset_path
        self.data_processor = None
        self.model = None
        self.recommender = None

    def train_model(self):
        if not self.data_processor:
            self.preprocess_data()
        self.model = Model()
        self.model.train(self.data_processor.get_csr_matrix())
        self.model = self.model.get_model()

    def preprocess_data(self):
        self.data_processor = DataPreprocessing(self.path)
        self.data_processor.get_final_data()
        self.data_processor.scale_data()

    def recommend(self, movie_name):
        if not self.model:
            self.train_model()
        self.recommender = Recommender(self.data_processor.get_final_data(
        ), self.data_processor.get_movies_data(), self.model)
        recommendations = self.recommender.get_movie_recommendation(movie_name, self.data_processor.get_csr_matrix())
        if type(recommendations) == str:
            return recommendations
        else :
            return "\n".join(recommendations)
```

In [2]:

```
dataset_path = os.path.abspath('../Dataset/data/')
rm = RecommendationSystem(dataset_path)
```

In [3]:

```
print(rm.recommend("Iron Man"))
```

```
Batman Begins (2005)
Dark Knight, The (2008)
Dark Knight Rises, The (2012)
Avengers, The (2012)
Guardians of the Galaxy (2014)
Inception (2010)
Iron Man 2 (2010)
WALL·E (2008)
Avengers: Age of Ultron (2015)
Avatar (2009)
```

In [4]:

```
print(rm.recommend("Titanic"))
```

```
Truman Show, The (1998)
Sixth Sense, The (1999)
Saving Private Ryan (1998)
Forrest Gump (1994)
Good Will Hunting (1997)
Men in Black (a.k.a. MIB) (1997)
Back to the Future (1985)
As Good as It Gets (1997)
Gladiator (2000)
Catch Me If You Can (2002)
```

In [5]:

```
print(rm.recommend("Your Name"))
```

```
A Silent Voice (2016)
Bruce, King of Kung Fu (1980)
Bruce Lee Fights Back from the Grave (1976)
Dragon Bruce Lee, Part II (1981)
Kubo and the Two Strings (2016)
Personal Shopper (2016)
Sing (2016)
Bloodfisted Brothers (1978)
Finding Dory (2016)
February (2015)
```