Movie_Recommendation_System

December 6, 2023

1 Group 27: Movie Recommendation System

1.0.1 1. Movie Dataset Files Overview

`movies_metadata.csv`

- **Description**: The main Movies Metadata file.
- **Content**: Contains information on 45,000 movies featured in the Full MovieLens dataset.
- **Features**:
  - Posters
  - Backdrops
  - Budget
  - Revenue
  - Release dates
  - Languages
  - Production countries
  - Production companies

`keywords.csv`

- **Description**: Contains the movie plot keywords for MovieLens movies.
- **Format**: Stringified JSON Object.

`credits.csv`

- **Description**: Consists of Cast and Crew Information for all movies.
- **Format**: Stringified JSON Object.

`links.csv`

- **Description**: Contains the TMDB and IMDB IDs of all the movies in the Full MovieLens dataset.

`links_small.csv`

- **Description**: Contains the TMDB and IMDB IDs of a small subset of 9,000 movies from the Full Dataset.

`ratings_small.csv`

- **Description**: A subset of 100,000 ratings from 700 users on 9,000 movies.

1.0.2 2. Goal:

Our goal is to build a recommendation system that recommends movies to the user based on their preference. Input of the system is a movie and the output is supposed to be a recommendation list consisting similar movies to the given one.

We plan to do this in **two** different approaches as suggested in our proposal.

1. Collaborative Filtering with Deep Learning

- **Techniques**:
  - **User-Based Collaborative Filtering**:
    - User-based collaborative filtering makes recommendations based on user-product interactions in the past. The assumption behind the algorithm is that similar users like similar products.
  - **Item-Based Collaborative Filtering**:
    - Item-based collaborative filtering makes recommendations also based on user-product interactions in the past. The assumption behind the algorithm is that users like similar products and dislike similar products, so they give similar ratings to similar products.

2. Content-Based Filtering with Machine Learning

- **Techniques**:
  - **Natural Language Processing (NLP)**:
    - Utilizes methods like TF-IDF or word embeddings for textual data analysis.
  - **Convolutional Neural Networks (CNNs)**:
    - Employed for analyzing visual content in movie posters.

1.0.3 3. Implementation

[234]:
```
# lib
import numpy as np
import pandas as pd

from surprise import Dataset, Reader
from surprise.prediction_algorithms.matrix_factorization import SVD
from surprise.prediction_algorithms.knns import KNNBasic
```

[235]:
```
# load dataset
movies_metadata = pd.read_csv('data/Movie/movies_metadata.csv')
ratings = pd.read_csv('data/Movie/ratings_small.csv')
```

```
C:\Users\yi_ch\AppData\Local\Temp\ipykernel_3636\488603508.py:2: DtypeWarning:
Columns (10) have mixed types. Specify dtype option on import or set
low_memory=False.
  movies_metadata = pd.read_csv('data/Movie/movies_metadata.csv')
```

[236]:
```
# Data Preprocessing and Cleaning
movies_metadata = movies_metadata[movies_metadata['id'].str.isnumeric()]
```

```
movies_metadata = movies_metadata.drop_duplicates()
movies_metadata['id'] = pd.to_numeric(movies_metadata['id'], errors='coerce')

ratings['movieId'] = pd.to_numeric(ratings['movieId'], errors='coerce')

# Drop NaN values
movies_metadata.dropna(subset=['id'], inplace=True)
ratings.dropna(subset=['movieId'], inplace=True)

# Check
print(movies_metadata.shape, ratings.shape)
```

```
(45450, 24) (100004, 4)
```

[237]:
```
# To ensure the quality of recommendation
# we select movies with votes more than 50
movie_md = movies_metadata[movies_metadata['vote_count'] > 50][['id', 'title']]

# ID
movie_ids = [int(x) for x in movie_md['id'].values]

# Select ratings of movies with more than 50 counts
ratings = ratings[ratings['movieId'].isin(movie_ids)]

# Reset Index
ratings.reset_index(inplace=True, drop=True)

# Check
ratings.head(5)
```

[237]:

	userId	movieId	rating	timestamp
0	1	1371	2.5	1260759135
1	1	2105	4.0	1260759139
2	1	2294	2.0	1260759108
3	2	17	5.0	835355681
4	2	62	3.0	835355749

[238]:
```
movie_md.head(5)
```

[238]:

	id	title
0	862	Toy Story
1	8844	Jumanji
2	15602	Grumpier Old Men
4	11862	Father of the Bride Part II
5	949	Heat

[239]:
```
# Initialize a surprise reader object
```

```
reader = Reader(line_format='user item rating', sep=',', rating_scale=(0,5),␣
 ↪skip_lines=1)

# Load the data
data = Dataset.load_from_df(ratings[['userId','movieId','rating']],␣
 ↪reader=reader)

# Split the dataset into the train and test set
trainset, testset = train_test_split(data, test_size=0.2)
```

[240]:
```
# Initialize model
svd = SVD()
svd.fit(trainset)
```

[240]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x21544613e50>
```

[241]:
```
def get_recommendations(data, movie_md, user_id, top_n, algo):
    recommendations = []

    # creating an user-item interactions matrix
    user_movie_interactions_matrix = data.pivot(index='userId',␣
 ↪columns='movieId', values='rating')

    # extracting product ids which the user_id has not interacted yet
    non_interacted_movies = user_movie_interactions_matrix.
 ↪loc[user_id][user_movie_interactions_matrix.loc[user_id].isnull()].index.
 ↪tolist()

    # looping through each of the product ids which user_id has not interacted␣
 ↪yet
    for item_id in non_interacted_movies:
        # predicting the ratings for those non interacted product ids by this␣
 ↪user
        est = algo.predict(user_id, item_id).est

        # appending the predicted ratings
        # movie_name =␣
 ↪movies_metadata[movies_metadata['movieId']==str(item_id)]['title']
        movie_info = movie_md[movie_md['id'] == item_id]['title']
        movie_name = movie_info.iloc[0] if not movie_info.empty else 'Unknown␣
 ↪Movie'
        recommendations.append((movie_name, est))

    # sorting the predicted ratings in descending order
    recommendations.sort(key=lambda x: x[1], reverse=True)
```

```
    return recommendations[:top_n] # returing top n highest predicted rating␣
↪products for this user
```

[242]:
```
get_recommendations(data=ratings, movie_md=movies_metadata, user_id=5,␣
↪top_n=10, algo=svd)
```

[242]:
```
[('Galaxy Quest', 4.788345823329939),
 ('The Million Dollar Hotel', 4.74863849558638),
 ('The Thomas Crown Affair', 4.733076817812155),
 ('Laura', 4.704455451027601),
 ('Hard Target', 4.702198803036662),
 ('The Thomas Crown Affair', 4.644921737398932),
 ('Once Were Warriors', 4.639610744366078),
 ('The Dreamers', 4.637313840097374),
 ('Point Break', 4.631094276526474),
 ('Dead Man', 4.61354343528863)]
```

1.0.4 User-Based

[243]:
```
sim_options = {'name': 'cosine',
               'user_based': True}

# KNN algorithm to find similar items
sim_user = KNNBasic(sim_options=sim_options, verbose=False, random_state=50)

# Train on the trainset, and predict ratings for the testset
sim_user.fit(trainset)
```

[243]:
```
<surprise.prediction_algorithms.knns.KNNBasic at 0x21529f433d0>
```

[244]:
```
get_recommendations(ratings, movie_md, 10, 10, sim_user)
```

[244]:
```
[('The Wizard', 5),
 ('Rio Bravo', 5),
 ('The Celebration', 5),
 ('A Streetcar Named Desire', 5),
 ('Gentlemen Prefer Blondes', 5),
 ('The Evil Dead', 5),
 ('Strangers on a Train', 5),
 ("Singin' in the Rain", 5),
 ("Frank Herbert's Dune", 5),
 ('The General', 5)]
```

[245]:
```
predictions_user_based = sim_user.test(testset)
rmse_user_based = accuracy.rmse(predictions_user_based)
```

```
RMSE: 0.9596
```

1.0.5 Item-Based

[246]:
```
sim_options = {'name': 'cosine',
               'user_based': False}

# KNN algorithm to find similar items
sim_item = KNNBasic(sim_options=sim_options, verbose=False, random_state=50)

# Train on the trainset, and predict ratings for the testset
sim_item.fit(trainset)
```

[246]:
```
<surprise.prediction_algorithms.knns.KNNBasic at 0x21529f40160>
```

[247]:
```
get_recommendations(ratings, movie_md, 10, 10, sim_item)
```

[247]:
```
[('Fear and Loathing in Las Vegas', 5),
 ('Flirting with Disaster', 5),
 ('The Princess Bride', 5),
 ('Sin City', 4.660455804854959),
 ('Drugstore Cowboy', 4.5),
 ('Ratatouille', 4.5),
 ("Mr. Magorium's Wonder Emporium", 4.5),
 ('Breach', 4.5),
 ('Scary Movie 2', 4.5),
 ('Elizabeth', 4.5)]
```

[248]:
```
predictions_item_based = sim_item.test(testset)
rmse_item_based = accuracy.rmse(predictions_item_based)
```

```
RMSE: 0.9796
```