Movie Recommendation System

Overview :

- For Movie Recommendation System I basically trained 3 models for the recommendation which are Content/Tag based recommendation system, KNN based recommendation system and Collaborative Filtering based recommendation system.
- Content/Tag based recommendation system :
    - Recommendation is done on the basis of Genre and tags of the movies.
- KNN based recommendation system :
    - Recommendation is done on the basis of user rating similarities for movies.
- Collaborative Filtering based recommendation system :
    - Recommendation is done on the basis of similarity betIen users to recommend a movie to a particular user.

Content/Tag Based Recommendation System :

Preprocessing:

Obtaining all of the tags or comments that specific films have received from various users using the tags dataset. Later introducing tags as a distinct database feature.

The movies dataset after adding the tags data from the dictionary created previously. If a movie hasn't received any tags or comments, I have inserted an empty character; otherwise, I have attached all the tags/comments the movie has gotten.

Also doing some more preprocessing of feature named **genres**.

genres
Adventure|Animation|Children|Comedy|Fantasy
Adventure|Children|Fantasy
Comedy|Romance
Comedy|Drama|Romance
Comedy
...
Thriller
(no genres listed)
Drama
Drama|Sci-Fi
(no genres listed)

genres
Adventure Animation Children Comedy Fantasy
Adventure Children Fantasy
Comedy Romance
Comedy Drama Romance
Comedy
...
Thriller
(no genres listed)
Drama
Drama Sci-Fi
(no genres listed)

Now, I have used a library of NLP **PorterStemmer** to help us find a root word for all of these similar words in order to eliminate words with the same meaning, such as love, loving, and loved, among others. This prevents the differentiation of words with similar meanings.

Then using the **stem() method** to find the **root word** of a word present in the tag of a movie.

The final dataset after preprocessing looks as below:

	movieId	title	tags
0	1	Toy Story (1995)	adventur anim children comedi fantasyanim fun ...
1	2	Jumanji (1995)	adventur children fantasyitaeg fantasi time tr...
2	3	Grumpier Old Men (1995)	comedi romancesequel funniest movi moldi old c...
3	4	Waiting to Exhale (1995)	comedi drama romancechick flick reveng clv cha...
4	5	Father of the Bride Part II (1995)	comedyhumor steve martin dian keaton famili se...
...	...	...	...
34203	151697	Grand Slam (1967)	thriller
34204	151701	Bloodmoney (2010)	(no genr listed)documentari
34205	151703	The Butterfly Circus (2009)	drama
34206	151709	Zero (2015)	drama sci-fi
34207	151711	The 2000 Year Old Man (1975)	(no genr listed)

34208 rows × 3 columns

Model Training:

Using **CountVectorizer** to exclude stop words which will affect the overall performance of the model.

Finding the **count vectors** for each movie depending on the count of words in tags of that movie. Then finding the **cosine similarity** betIen the **vectors** of the movie and storing it in a **matrix.**

To recommend the movies I have made a function named **Recommend1** that **takes movie name** and **number of similar movies requested by the user** as the arguments. The function returns a list of recommended movies.

One of such example is below:

```
Recommend1("Iron Man (2008)",20)
```

```
['Iron Man 2 (2010)',
 'Iron Man 3 (2013)',
 'Avengers, The (2012)',
 'Ant-Man (2015)',
 'X-Men (2000)',
 'Singing Detective, The (2003)',
 'Johnny Be Good (1988)',
 'X2: X-Men United (2003)',
 'Avengers Confidential: Black Widow & Punisher (2014)',
 'Spider-Man 2 (2004)',
 'Avengers: Age of Ultron (2015)',
 'Heart and Souls (1993)',
 'Less Than Zero (1987)',
 'Fantastic Four (2015)',
 'Elektra (2005)',
 'In Dreams (1999)',
 'Spider-Man 3 (2007)',
 'Meteor Man, The (1993)',
 'Chaplin (1992)',
 'Captain America: The First Avenger (2011)']
```

The user gave input as **Iron Man (2008)** and **20**. This describes that the user wants similar movies to **Iron Man (2008)** and needs **20** movie recommendations.

KNN based Recommendation System :

Here the movie recommendation is done using collaborative filtering technique based on K-Nearest Neighbors algorithm.

Preprocessing :

Due to resource restrictions I need to reduce the dataset upto 50,00,000 data points to train the data properly.

I then create the pivot table from this data where movies are represented as rows and users as columns. Here the pivot table shows the user rating for the particular movieId and 0 for the movieId which the user has not rated.

userId	1	2	3	4	5	6	7	8	9	10	...	53583	53584	53585	53586	53587	53588	53589	53590	53591	53592
movieId																					
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
151623	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
151657	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
151661	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
151671	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
151697	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

25599 rows × 53592 columns

'Csr_matrix' function from the **'scipy.sparse'** module is used to create a Compressed Sparse Row matrix (CSR matrix) from the ratings pivot table values. A CSR matrix is a sparse matrix representation that stores non-zero values and their corresponding row and column indices in a compressed format.

Model Training :

The K-Nearest Neighbors **(KNN)** algorithm is applied on the csr_matrix obtained from the pivot table data using cosine similarity as the distance metric and 'brute' algorithm for computing the nearest neighbors. The trained KNN model is stored in the variable 'clf'.

A function named 'recommend_movie' is defined, which takes a movie name, csr_matrix obtained from the ratings pivot table data and the number of recommended movies as input. The function uses the **extractOne** method of the **'fuzzywuzzy'** library to find the closest match of the input movie. Once the closest match is found, the function uses the trained KNN model to find the n closest movies to the selected movie based on their features. The recommended movies are then printed.

```
recommend_movie('batman',mm,10)
```

```
Movie Selected :  Batman Forever (1995) Index : 151
151                                    NaN
586                          Batman (1989)
376                        True Lies (1994)
163    Die Hard: With a Vengeance (1995)
340    Ace Ventura: Pet Detective (1994)
313                         Stargate (1994)
582                          Aladdin (1992)
148                        Apollo 13 (1995)
345      Clear and Present Danger (1994)
325        Star Trek: Generations (1994)
Name: title, dtype: object
```

Collaborative Filtering based Recommendation System :

Preprocessing:

For this Recommendation System I have used movies and ratings dataset labels encoded the MovieId and UserId using **LabelEncoder** and updated the new movieId's in ratings dataset.

Model Training:

All the below mentioned tasks have been done by creating a class **Colaborative_Recommendation.**

Now again due to resource restrictions I could not create a similarity matrix for users hence I used a different approach And tried to train the model for a single user and find his similarity with all other users in the dataset.

To find the similarity betIen users I used **cosine_similarity** metric based on the ratings that these two users gave to different movies

Now after finding the similarities of the given user with other users I find the users which are more similar to the given user and create a new dataset which contains the movies which these similar users have rated.

After that I sorted the dataset in descending order of ratings that users gave to those movies and appended the name of those movies to the list and finally returned the number of movies that Ire asked to be recommended to that given user.

```
Recommender = Colaborative_Recommendation(ratings,1234) # training the model for userId = 1234
Recommender.train()
```

The above screenshot shows how the model trained for a particular user (1234 in this case).

```
GoldenEye (1995)
Braveheart (1995)
Apollo 13 (1995)
Crimson Tide (1995)
Die Hard: With a Vengeance (1995)
Net, The (1995)
Pulp Fiction (1994)
Shawshank Redemption, The (1994)
Star Trek: Generations (1994)
While You Were Sleeping (1995)
Ace Ventura: Pet Detective (1994)
Clear and Present Danger (1994)
Forrest Gump (1994)
Speed (1994)
True Lies (1994)
Cliffhanger (1993)
Firm, The (1993)
Fugitive, The (1993)
Jurassic Park (1993)
Terminator 2: Judgment Day (1991)
Dances with Wolves (1990)
Batman (1989)
Beauty and the Beast (1991)
```

```
Recommender.recommend(10) # getting the recommended movies for the userId 1234
```

```
['Silence of the Lambs, The (1991)',
 'Outbreak (1995)',
 'Batman Forever (1995)',
 'Interview with the Vampire: The Vampire Chronicles (1994)',
 'Natural Born Killers (1994)',
 'Aladdin (1992)',
 'Outbreak (1995)',
 'Dumb & Dumber (Dumb and Dumber) (1994)',
 'Outbreak (1995)',
 'Stargate (1994)']
```

In above figures to the left are the movies that user-1234 has already seen and to the right are the movies recommended to him which are based on the similarity of User with other Users.