Movie Reccomendation System

Recommender system is a system that seeks to predict or filter presences according to the user's choice. Recommender system are utilized in a variety of areas including movies, music, news, books, research articles, search quaries, social tags, and products in general. Recommender systems produce a list of recommendations in any of the two ways -

Collaborative filtering : COllaborative filtering approaches build a model from the user's past behavior (i.e. items purchased or searched by the user) as well as similar decisions made by other users. This model is then used to predict items (or ratings for items) that users may have an interest in.

Content based filtering : Content-based filtering approaches uses a series or discrete characteristics of an item in order to recommend additional items with similar properties. Content-based filtering methods are totally based on a description of the item and a profile of the users's preferences. It recommends items based on the user's past preferences. Let's develop a basic recommendation system using python and pandas.

Let's develop a basic recommendation system by suggesting items that are most similar to a particular item, inn this case, movies. It just tells what movies/items are most similar to the user's movie choice

Import Library

```
import pandas as pd
```

```
import numpy as np
```

Import Dataset

```
df = pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/refs/heads/main/Movies%20
```

```
df .head()
```

ID	Movie_Title	Movie_Genre	Movie_Language	Movie_Budget	Movie_Popularity
1	Four Rooms	Crime Comedy	en	4000000	22.876230
2	Star Wars	Adventure Action Science Fiction	en	11000000	126.393695
3	Finding Nemo	Animation Family	en	94000000	85.688789
4	Forrest Gump	Comedy Drama Romance	en	55000000	138.133331
5	American Beauty	Drama	en	15000000	80.878605

columns

```
df .info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4760 entries, 0 to 4759
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Movie_ID              4760 non-null   int64  
 1   Movie_Title           4760 non-null   object 
 2   Movie_Genre           4760 non-null   object 
 3   Movie_Language        4760 non-null   object 
 4   Movie_Budget          4760 non-null   int64  
 5   Movie_Popularity      4760 non-null   float64
 6   Movie_Release_Date    4760 non-null   object 
 7   Movie_Revenue         4760 non-null   int64  
 8   Movie_Runtime         4758 non-null   float64
 9   Movie_Vote            4760 non-null   float64
 10  Movie_Vote_Count      4760 non-null   int64  
 11  Movie_Homepage        1699 non-null   object 
 12  Movie_Keywords        4373 non-null   object 
```

```
 13  Movie_Overview           4757 non-null   object
 14  Movie_Production_House   4760 non-null   object
 15  Movie_Production_Country 4760 non-null   object
 16  Movie_Spoken_Language    4760 non-null   object
 17  Movie_Tagline            3942 non-null   object
 18  Movie_Cast               4733 non-null   object
 19  Movie_Crew               4760 non-null   object
 20  Movie_Director           4738 non-null   object
dtypes: float64(3), int64(4), object(14)
memory usage: 781.1+ KB
```

```
df .shape
```

```
(4760, 21)
```

```
df .columns
```

```
Index(['Movie_ID', 'Movie_Title', 'Movie_Genre', 'Movie_Language',
       'Movie_Budget', 'Movie_Popularity', 'Movie_Release_Date',
       'Movie_Revenue', 'Movie_Runtime', 'Movie_Vote', 'Movie_Vote_Count',
       'Movie_Homepage', 'Movie_Keywords', 'Movie_Overview',
       'Movie_Production_House', 'Movie_Production_Country',
       'Movie_Spoken_Language', 'Movie_Tagline', 'Movie_Cast', 'Movie_Crew',
       'Movie_Director'],
      dtype='object')
```

Get Feature Collection

```
df_features = df[['Movie_Genre','Movie_Keywords','Movie_Tagline','Movie_Cast','Movie_Dire
```

```
df_features.shape
```

```
(4760, 5)
```

```
df_features
```

	Movie_Genre	Movie_Keywords	Movie_Tagline	Movie_Cast	Movie_Director
0	Crime Comedy	hotel new year's eve witch bet hotel room	Twelve outrageous guests. Four scandalous requ...	Tim Roth Antonio Banderas Jennifer Beals Madon...	Allison Anders
1	Adventure Action Science Fiction	android galaxy hermit death star lightsaber	A long time ago in a galaxy far, far away...	Mark Hamill Harrison Ford Carrie Fisher Peter ...	George Lucas
2	Animation Family	father son relationship harbor underwater fish...	There are 3.7 trillion fish in the ocean, they...	Albert Brooks Ellen DeGeneres Alexander Gould ...	Andrew Stanton
3	Comedy Drama Romance	vietnam veteran hippie mentally disabled runni...	The world will never be the same, once you've ...	Tom Hanks Robin Wright Gary Sinise Mykelti Wil...	Robert Zemeckis

```
x = df_features['Movie_Genre']+' '+df_features['Movie_Keywords']+' '+df_features['Movie_T
```

```
x
```

	0
0	Crime Comedy hotel new year's eve witch bet ho...
1	Adventure Action Science Fiction android galax...
2	Animation Family father son relationship harbo...
3	Comedy Drama Romance vietnam veteran hippie me...
4	Drama male nudity female nudity adultery midli...
...	...
4755	Horror The hot spot where Satan's waitin'. Li...
4756	Comedy Family Drama It's better to stand out ...
4757	Thriller Drama christian film sex trafficking ...
4758	Family
4759	Documentary music actors legendary perfomer cl...

4760 rows × 1 columns

dtype: object

```
x .shape
```

```
(4760,)
```

Get Feature Text Conversion To Tokens

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tfidf=TfidfVectorizer()
```

```
x=tfidf.fit_transform(x)
```

```
x.shape
```

```
(4760, 17258)
```

```
print(x)
```

```
  (0, 3583)	0.06486754376295062
  (0, 3240)	0.04527089872278055
  (0, 7213)	0.25146675849405775
  (0, 10898)	0.17625708810661284
  (0, 17052)	0.26079573581490934
  (0, 5059)	0.29553419178998613
  (0, 16862)	0.12768803549311025
  (0, 1595)	0.15687561633854538
  (0, 13052)	0.1465525095337543
  (0, 15708)	0.17654247479915475
  (0, 11362)	0.18801785343006192
  (0, 6463)	0.18801785343006192
  (0, 5662)	0.1465525095337543
  (0, 13467)	0.19712637387361423
  (0, 12731)	0.19712637387361423
  (0, 614)	0.07642616241686973
  (0, 11244)	0.08262965296941757
  (0, 9206)	0.15186283580984414
  (0, 1495)	0.19712637387361423
  (0, 7454)	0.14745635785412262
  (0, 7071)	0.19822417598406614
  (0, 5499)	0.11454057510303811
  (0, 3878)	0.11998399582562203
  (0, 11242)	0.07277788238484746
  (0, 15219)	0.09800472886453934
  :	:
  (4757, 3485)	0.199161573117024
  (4757, 1184)	0.18890726729447022
  (4757, 14568)	0.24255077606762876
  (4757, 15508)	0.24255077606762876
  (4757, 5802)	0.24255077606762876
  (4757, 819)	0.27474840155297187
  (4757, 14195)	0.28805858134028367
```

```
  (4757, 2227)  0.28805858134028367
  (4757, 7691)  0.28805858134028367
  (4757, 1932)  0.28805858134028367
  (4758, 5238)  1.0
  (4759, 10666) 0.15888268987343043
  (4759, 1490)  0.21197258705292082
  (4759, 15431) 0.19628653185946862
  (4759, 5690)  0.19534291014627303
  (4759, 14051) 0.20084315377640435
  (4759, 4358)  0.18306542312175342
  (4759, 10761) 0.3126617295732147
  (4759, 7130)  0.26419662449963793
  (4759, 3058)  0.2812896191863103
  (4759, 14062) 0.3237911628497312
  (4759, 8902)  0.3040290704566037
  (4759, 205)   0.3237911628497312
  (4759, 11708) 0.33947721804318337
  (4759, 11264) 0.33947721804318337
```

Get Similarity Score Using Cosine Similarity

Cosine_similarity computers the L2-normalized dot product of vectors. Euclidean (L2) normalization projects the vectors onto the unit sphere, and their dot product is then the cosine of the angle between the points denoted by the vectors.

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
similarity_score = cosine_similarity(x)
```

```
similarity_score
```

```
array([[1.        , 0.01351235, 0.03570468, ..., 0.        , 0.        ,
        0.        ],
       [0.01351235, 1.        , 0.00806674, ..., 0.        , 0.        ,
        0.        ],
       [0.03570468, 0.00806674, 1.        , ..., 0.        , 0.08014876,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 1.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.08014876, ..., 0.        , 1.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        1.        ]])
```

```
similarity_score.shape
```

```
(4760, 4760)
```

Get Movie Name as Input from User and Validate for Closest Spelling

```
Favourite_Movie_Name = input('Enter your favourite movie name :')
```

```
Enter your favourite movie name :Avtaar
```

```
All_Movies_Title_List = df['Movie_Title'].tolist()
```

```
import difflib
```

```
Movie_Recommendation = difflib.get_close_matches(Favourite_Movie_Name, All_Movies_Title_L
print(Movie_Recommendation)
```

```
['Avatar']
```

```
Close_Match = Movie_Recommendation[0]
print(Close_Match)
```

```
Avatar
```

```
Index_of_Close_Match_Movie = df[df.Movie_Title == Close_Match]['Movie_ID'].values[0]
print(Index_of_Close_Match_Movie)
```

```
2692
```

```
#getting a list of similar movies
Recommendation_Score = list(enumerate(similarity_score[Index_of_Close_Match_Movie]))
print(Recommendation_Score)
```

```
[(0, 0.009805093506053453), (1, 0.0), (2, 0.0), (3, 0.00800429043895183), (4, 0.00267
```

```
len(Recommendation_Score)
```

```
4760
```

Get All Movies Sort Based on Recommendation Score wrt Favourite Movie

```
#sorting the movies based on their similarity score

Sorted_Similar_Movies = sorted(Recommendation_Score, key = lambda x:x[1], reverse = True)
print(Sorted_Similar_Movies)
```

```
[(2692, 1.0), (3276, 0.11904275527845871), (3779, 0.10185805797079384), (62, 0.101535
```

```
#print the name of similar movies based on the index

print('Top 30 Movies Suggested for You : \n')

i = 1

for movie in Sorted_Similar_Movies :
  index = movie[0]
  title_from_index = df[df.index==index]['Movie_Title'].values[0]
  if (i<31):
    print(i, '.', title_from_index)
    i+=1
```

```
Top 30 Movies Suggested for You :

1 . Niagara
2 . Caravans
3 . My Week with Marilyn
4 . Brokeback Mountain
5 . Harry Brown
6 . Night of the Living Dead
7 . The Curse of Downers Grove
8 . The Boy Next Door
9 . Back to the Future
10 . The Juror
11 . Some Like It Hot
12 . Enough
13 . The Kentucky Fried Movie
14 . Eye for an Eye
15 . Welcome to the Sticks
16 . Alice Through the Looking Glass
17 . Superman III
18 . The Misfits
19 . Premium Rush
20 . Duel in the Sun
21 . Sabotage
22 . Small Soldiers
23 . All That Jazz
24 . Camping Sauvage
25 . The Raid
26 . Beyond the Black Rainbow
27 . To Kill a Mockingbird
28 . World Trade Center
29 . The Dark Knight Rises
30 . Tora! Tora! Tora!
```

```
Movie_Name = input('Enter your favourite movie name :')

list_of_all_titles = df['Movie_Title'].tolist()

Find_Close_Match = difflib.get_close_matches(Movie_Name, list_of_all_titles)

close_Match = Find_Close_Match[0]

Index_of_Movie = df[df.Movie_Title == Close_Match]['Movie_ID'].values[0]

Recommendation_Score = list(enumerate(similarity_score[Index_of_Movie]))

sorted_similar_movies = sorted(Recommendation_Score, key = lambda x:x[1], reverse = True)

print('Top 10 Movies Suggested for you : \n')

i = 1

for movie in sorted_similar_movies :
  index = movie[0]
  title_from_index = df[df.Movie_ID==index]['Movie_Title'].values
  if (i<11) :
    print(i, '.', title_from_index)
    i+=1
```

```
Enter your favourite movie name :Avtaar
Top 10 Movies Suggested for you :

1 . ['Avatar']
2 . ['The Girl on the Train']
3 . ['Act of Valor']
4 . ['Donnie Darko']
5 . ['Precious']
6 . ['Freaky Friday']
7 . ['The Opposite Sex']
8 . ['Heaven is for Real']
9 . ['Run Lola Run']
10 . ['Elizabethtown']
```