Movie Recommendation System

We are going to implement a Content based recommendation system using the scikit-learn library

Content based recommendation engine

This type of recommendation systems, takes in a movie that a user currently likes as input.

Then it analyzes the contents (storyline, genre, cast, director etc.) of the movie to find out other movies which have similar content.

Then it ranks similar movies according to their similarity scores and recommends the most relevant movies to the user.

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

import warnings
warnings.filterwarnings('ignore')
```

Data Pre-Processing

Load the dataset - Read the csv file

In [2]:

```
df = pd.read_csv("F:\\ITVEDANT\\Recommendation System\\RS Project\\movie_dataset.csv")
```

To check the head of the list - First 5 values

In [4]:

```
df.head(5)
```

Out[4]:

	index	budget	genres	homepage	id	keywords	c
0	0	237000000	Action Adventure Fantasy Science Fiction	http://www.avatarmovie.com/	19995	culture clash future space war space colony so...	
1	1	300000000	Adventure Fantasy Action	http://disney.go.com/disneypictures/pirates/	285	ocean drug abuse exotic island east india trad...	
2	2	245000000	Action Adventure Crime	http://www.sonypictures.com/movies/spectre/	206647	spy based on novel secret agent sequel mi6	
3	3	250000000	Action Crime Drama Thriller	http://www.thedarkknightrises.com/	49026	dc comics crime fighter terrorist secret ident...	
4	4	260000000	Action Adventure Science Fiction	http://movies.disney.com/john-carter	49529	based on novel mars medallion space travel pri...	

5 rows × 24 columns

To check the tail of the list - Last 5 values

In [6]:

```
df.tail(5)
```

Out[6]:

	index	budget	genres	homepage	id	
4798	4798	220000	Action Crime Thriller	NaN	9367	state b
4799	4799	9000	Comedy Romance	NaN	72766	
4800	4800	0	Comedy Drama Romance TV Movie	http://www.hallmarkchannel.com/signedsealeddel...	231617	
4801	4801	0	NaN	http://shanghaicalling.com/	126186	
4802	4802	0	Documentary	NaN	25975	c

5 rows × 24 columns

In [7]:

```
df.columns
```

Out[7]:

```
Index(['index', 'budget', 'genres', 'homepage', 'id', 'keywords',
       'original_language', 'original_title', 'overview', 'popularity',
       'production_companies', 'production_countries', 'release_date',
       'revenue', 'runtime', 'spoken_languages', 'status', 'tagline', 'tit
le',
       'vote_average', 'vote_count', 'cast', 'crew', 'director'],
      dtype='object')
```

Checking the data shape

In [11]:

```
print(f"The dataset contains {df.shape[0]} rows and {df.shape[1]} columns")
```

```
The dataset contains 4803 rows and 24 columns
```

In [14]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   index                 4803 non-null   int64  
 1   budget                4803 non-null   int64  
 2   genres                4775 non-null   object 
 3   homepage              1712 non-null   object 
 4   id                    4803 non-null   int64  
 5   keywords              4391 non-null   object 
 6   original_language     4803 non-null   object 
 7   original_title        4803 non-null   object 
 8   overview              4800 non-null   object 
 9   popularity            4803 non-null   float64
 10  production_companies  4803 non-null   object 
 11  production_countries  4803 non-null   object 
 12  release_date          4802 non-null   object 
 13  revenue               4803 non-null   int64  
 14  runtime               4801 non-null   float64
 15  spoken_languages      4803 non-null   object 
 16  status                4803 non-null   object 
 17  tagline               3959 non-null   object 
 18  title                 4803 non-null   object 
 19  vote_average          4803 non-null   float64
 20  vote_count            4803 non-null   int64  
 21  cast                  4760 non-null   object 
 22  crew                  4803 non-null   object 
 23  director              4773 non-null   object 
dtypes: float64(3), int64(5), object(16)
memory usage: 900.7+ KB
```

In [15]:

```
# Statistic about the dataset

df.describe()
```

Out[15]:

	index	budget	id	popularity	revenue	runtime	vo
count	4803.000000	4.803000e+03	4803.000000	4803.000000	4.803000e+03	4801.000000	4
mean	2401.000000	2.904504e+07	57165.484281	21.492301	8.226064e+07	106.875859	
std	1386.651002	4.072239e+07	88694.614033	31.816650	1.628571e+08	22.611935	
min	0.000000	0.000000e+00	5.000000	0.000000	0.000000e+00	0.000000	
25%	1200.500000	7.900000e+05	9014.500000	4.668070	0.000000e+00	94.000000	
50%	2401.000000	1.500000e+07	14629.000000	12.921594	1.917000e+07	103.000000	
75%	3601.500000	4.000000e+07	58610.500000	28.313505	9.291719e+07	118.000000	
max	4802.000000	3.800000e+08	459488.000000	875.581305	2.787965e+09	338.000000	

To check if there are any duplicate values in the dataset

In [16]:

```
duplicate_values = df.duplicated().sum()
print(f'The data contains {duplicate_values} duplicate values')
```

```
The data contains 0 duplicate values
```

Select features

If we go through the dataset, we can see that the dataset contains many more extra info columns and we don't need all of them.

So, we choose keywords, cast, genres, director columns to use as features set which is the main content of movie

In [17]:

```
#Create a column in 'df' which combines all selected features

features = ['keywords','cast','genres','director']


for feature in features:
    df[feature] = df[feature].fillna('')
```

In [20]:

```
df.original_title[0:20]
```

Out[20]:

```
0                                      Avatar
1    Pirates of the Caribbean: At World's End
2                                     Spectre
3                      The Dark Knight Rises
4                                   John Carter
5                                  Spider-Man 3
6                                      Tangled
7                    Avengers: Age of Ultron
8     Harry Potter and the Half-Blood Prince
9         Batman v Superman: Dawn of Justice
10                             Superman Returns
11                           Quantum of Solace
12    Pirates of the Caribbean: Dead Man's Chest
13                              The Lone Ranger
14                                 Man of Steel
15    The Chronicles of Narnia: Prince Caspian
16                                 The Avengers
17    Pirates of the Caribbean: On Stranger Tides
18                              Men in Black 3
19    The Hobbit: The Battle of the Five Armies
Name: original_title, dtype: object
```

Create a function for combining the values of these columns into a single string.

In [21]:

```
def combine_features(row):
    try:
        return row['keywords'] +" "+row['cast']+" "+row["genres"]+" "+row["director"]
    except:
        print ("Error:"), row

df["combined_features"] = df.apply(combine_features,axis=1)
```

In [22]:

```
print ("Combined Features:")
df["combined_features"].head()
```

```
Combined Features:
```

Out[22]:

```
0    culture clash future space war space colony so...
1    ocean drug abuse exotic island east india trad...
2    spy based on novel secret agent sequel mi6 Dan...
3    dc comics crime fighter terrorist secret ident...
4    based on novel mars medallion space travel pri...
Name: combined_features, dtype: object
```

Now, we need to call function over each row of our dataframe.

But, before doing that we need to clean and preprocess the data for our use.

We will fill all the NaN values with blank string in the dataframe.

In [23]:

```
 #filling all NaNs with blank string

for feature in features:
    df[feature] = df[feature].fillna('')

df["combined_features"] = df.apply(combine_features,axis=1)
```

In [24]:

```
#applying combined_features() method over each rows of dataframe and storing the combined
df.iloc[0].combined_features
```

Out[24]:

```
'culture clash future space war space colony society Sam Worthington Zoe S
aldana Sigourney Weaver Stephen Lang Michelle Rodriguez Action Adventure F
antasy Science Fiction James Cameron'
```

We have obtained the combined strings, we can now feed these strings to a CountVectorizer() object for getting the count matrix.

In [26]:

```
#creating new CountVectorizer() object
cv = CountVectorizer()


#feeding combined strings(movie contents) to CountVectorizer() object
count_matrix = cv.fit_transform(df["combined_features"])
```

Now, we need to obtain the cosine similarity matrix from the count matrix.

In [27]:

```
#Compute the Cosine Similarity based on the count_matrix
cosine_sim = cosine_similarity(count_matrix)
```

Now, we will define two helper functions to get movie title from movie index and other.

In [28]:

```
def get_title_from_index(index):
    return df[df.index == index]["title"].values[0]
def get_index_from_title(title):
    return df[df.title == title]["index"].values[0]
```

The next step is to get the title of the movie that the user currently likes.

Then we will find the index of that movie. After that, we will access the row corresponding to this movie in the similarity matrix.

We will get the similarity scores of all other movies from the current movie. Then we will get all the similarity scores of that movie to make a tuple of movie index and similarity score

In [29]:

```
#Get index of this movie from its title

movie_user_likes = "Avatar"
movie_index = get_index_from_title(movie_user_likes)
similar_movies = list(enumerate(cosine_sim[movie_index]))
```

In [41]:

```
#Print titles of first 25 movies

i=0
for element in sorted_similar_movies:
        print (get_title_from_index(element[0]))
        i=i+1
        if i>25:
            break
```

```
Guardians of the Galaxy
Aliens
Star Wars: Clone Wars: Volume 1
Star Trek Into Darkness
Star Trek Beyond
Alien
Lockout
Jason X
The Helix... Loaded
Moonraker
Planet of the Apes
Galaxy Quest
Gravity
Alien³
Jupiter Ascending
The Wolverine
Silent Running
Zathura: A Space Adventure
Trekkies
Cargo
Wing Commander
Star Trek
Lost in Space
Babylon A.D.
The Fifth Element
Oblivion
```

We will sort the list similar_movies according to similarity scores in descending order.

Since the most similar movie to a given movie will be itself, we will discard the first element after sorting the movies

In [36]:

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

We will sort our sorted_similar_movies with respect to vote_average.

x[0] has the index of the movie in the data frame.

In [43]:

```
df["vote_average"].unique()
```

Out[43]:

```
array([ 7.2,  6.9,  6.3,  7.6,  6.1,  5.9,  7.4,  7.3,  5.7,  5.4,  7. ,
        6.5,  6.4,  6.2,  7.1,  5.8,  6.6,  7.5,  5.5,  6.7,  6.8,  6. ,
        5.1,  7.8,  5.6,  5.2,  8.2,  7.7,  5.3,  8. ,  4.8,  4.9,  7.9,
        8.1,  4.7,  5. ,  4.2,  4.4,  4.1,  3.7,  3.6,  3. ,  3.9,  4.3,
        4.5,  3.4,  4.6,  8.3,  3.5,  4. ,  2.3,  3.2,  0. ,  3.8,  2.9,
        8.5,  1.9,  3.1,  3.3,  2.2,  0.5,  9.3,  8.4,  2.7, 10. ,  1. ,
        2. ,  2.8,  9.5,  2.6,  2.4])
```

In [42]:

```
sort_by_average_vote = sorted(sorted_similar_movies,key=lambda x:df["vote_average"][x[0]]
print(sort_by_average_vote)
```

```
[(3519, 0.0), (4045, 0.0), (4247, 0.0), (4662, 0.0), (3992, 0.0), (2386,
0.08399210511316162), (1881, 0.0), (2970, 0.0), (2796, 0.0392837100659193
1), (3337, 0.036369648372665396), (2294, 0.08025723539051281), (1818, 0.0
7698003589195011), (662, 0.0), (2731, 0.0), (3232, 0.0), (3865, 0.0), (47
55, 0.0), (1990, 0.23570226039551587), (1987, 0.08399210511316162), (224
7, 0.08206099398622181), (690, 0.07548513560963972), (4535, 0.04303314829
119352), (2947, 0.041030496993110906), (1663, 0.040128617695256406), (65,
0.03774256780481986), (809, 0.0), (1847, 0.0), (2170, 0.0), (3057, 0.0),
(3719, 0.0), (3723, 0.0), (4602, 0.0), (2912, 0.16412198797244362), (96,
0.15097027121927944), (329, 0.11785113019775793), (95, 0.1111111111111111
3), (3573, 0.04199605255658081), (3454, 0.040128617695256406), (1553, 0.
0), (2091, 0.0), (2284, 0.0), (2453, 0.0), (2760, 0.0), (3041, 0.0), (362
2, 0.0), (3788, 0.0), (3866, 0.0), (3906, 0.0), (4238, 0.0), (4302, 0.0),
(3208, 0.3464101615137755), (262, 0.12309149097933272), (330, 0.117851130
19775793), (2285, 0.11785113019775793), (2638, 0.11111111111111113), (152
5, 0.08606629658238704), (1850, 0.08025723539051281), (3886, 0.0802572353
9051281), (1405, 0.07698003589195011), (2522, 0.07698003589195011), (417
3, 0.07407407407407408), (3432, 0.051434449987363975), (4678, 0.046676002
80093366), (463, 0.04415107856883479), (494, 0.03849001794597506), (4076,
```

In [44]:

```
i=0
print("Suggesting top 5 movies in order of Average Votes:\n")
for element in sort_by_average_vote:
    print(get_title_from_index(element[0]))
    i=i+1
    if i>5:
        break
```

```
Suggesting top 5 movies in order of Average Votes:

Stiff Upper Lips
Dancer, Texas Pop. 81
Me You and Five Bucks
Little Big Top
Sardaarji
One Man's Hero
```

Above are the Top 5 Movies with respect to Average Voting.

Now, we will run a loop to print first 5 entries from sorted_similar_movies list

In [37]:

```
i=0
print("Top 5 similar movies to "+movie_user_likes+" are:\n")
for element in sorted_similar_movies:
    print(get_title_from_index(element[0]))
    i=i+1
    if i>5:
        break
```

```
Top 5 similar movies to Avatar are:

Guardians of the Galaxy
Aliens
Star Wars: Clone Wars: Volume 1
Star Trek Into Darkness
Star Trek Beyond
Alien
```

The above are the recommended similar movies from user's like (interest)

Here we have used Movie named 'Avatar' as the user's interest. If we search on google for similar movies as 'Avatar' we can find above similar interest listed movies name.

Thank You