Project Report on

Movie Recommendation System based on User-Input using Machine Learning

by

Dhiraj Dharmadip Raut

Table of Contents

Introduction

The Project aims at building a **Movie Recommendation System** from the `content` of Movies-dataset that contains around **1700+ Hindi Movies** extracted from WikiPedia using `Webscrapping` and **4800+ English movies** collected from `'English_Movies.csv'` file downloaded from `Kaggle` site (https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata (https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata)).

Basically, The **System will work as follows**:
After the user has provided the name of a film he liked, the engine should be able to select in the database a list of 30 films that the user will enjoy based on Content.

Movie Recommendation System is extensively used Now-a-days by all the **OTT Platforms** such as `NETFLIX` , `AMAZON PRIME` , etc to lure the users and make them spend more time on Platform.

There are `Three main types` of Movie Recommendation System:

- **Content Based Recommendation System -** Promotes movies based on content (Genre,Storyline,etc) of the movies watched by user
- **Popularity Based Recommendation System -** Promotes movies based on popularity (Highest Viewed, Highest Rated, IMDb ratings,etc) of the movies according to regions, Genre, etc
- **Collaborative Recommendation System -** Promotes movies based on Watching History Pattern of other People who follows same watching history as yours

In the current case, since the dataset only describe the content of the films, collaborative filtering and Popularity filtering is excluded and I will thus build an system that uses the content of the entries.

Import the Requied Modules and the Functions

Install **Python libraries** required for this project, which are `Scikit-Learn` , `BeautifulSoup4` , `urllib` , `difflib` ,and `pandas` .

Once Installed all libraries, **import** all Modules and Functions requied from these libraries.

In [1]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from bs4 import BeautifulSoup
import urllib
import pandas as pd
import difflib
import warnings
warnings.filterwarnings('ignore')
```

Data Extraction of Hindi Movies Using Webscrapping

First, Created a **Pandas DataFrame** named `movie_data` to Store all the movies data.

The dataframe have columns of `Title` , `Year` , `Genre` , `Director` , and `Cast`

In [2]:
```
movie_data = pd.DataFrame(columns=['Title','Year','Genre','Director','Cast'])
display(movie_data)
```

Title	Year	Genre	Director	Cast

Used the `urllib` library and downloaded the webpage

https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2000 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2000). Did the same for all other years of Bollywood films data. Changed the year in the url.

I have **grouped** the urls according to their **content similarity**, to extract the data from tables.

Stored it as a string in the variable `url`. Saved the text of the response as a variable named `html_data`.

In [3]:
```
URL = {2000:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2000',
       2001:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2001',
       2002:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2002',
       2003:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2003',
       2004:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2004',
       2005:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2005',
       2006:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2006',
       2007:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2007'
      }
for key, url in URL.items():
    print(key,":",url)
```

```
2000 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2000 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2000)
2001 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2001 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2001)
2002 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2002 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2002)
2003 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2003 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2003)
2004 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2004 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2004)
2005 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2005 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2005)
2006 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2006 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2006)
2007 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2007 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2007)
```

Now, Parse the html_data using `beautiful_soup` .

BeautifulSoup is a Python library for pulling data out of HTML and XML files. This is accomplished by representing the HTML as a set of objects with methods used to parse the HTML. We can navigate the HTML as a tree and filter out what we are looking for.

To parse a document, pass it into the `BeautifulSoup` constructor, the BeautifulSoup object, which represents the document as a nested data structure.

First, the document is converted to Unicode, (similar to ASCII), and HTML entities are converted to Unicode characters. Beautiful Soup transforms a complex HTML document into a complex tree of Python objects.

In [4]:
```
for key, url in URL.items():
    html_data = urllib.request.urlopen(url).read()
    soup = BeautifulSoup(html_data,"html.parser") # or soup = BeautifulSoup(html_data,"lxml")
    table = soup.find_all('table', class_='wikitable')[1].tbody
    rows = table.find_all('tr')
    for row in rows:
        col = row.find_all("td")
        if (col != []):
            Title = col[0].get_text(strip=True)
            Year = str(key)
            Genre = col[3].get_text(strip=True)
            Director = col[1].get_text(strip=True)
            Cast = col[2].get_text(strip=True)
            # Finally we append the data of each row to the table
            movie_data = movie_data.append({"Title":Title, "Year":Year, "Genre":Genre, "Directo

display(movie_data)
```

	Title	Year	Genre	Director	Cast
0	Aaghaaz	2000	Thriller	Yogesh Ishwar	Sunil Shetty,Sushmita Sen,Namrata Shirodkar
1	Aaj Ka Ravan	2000	Drama	Imran Khalid	Kasam Ali,Mithun Chakraborty
2	Anjaane	2000	Romance	Ravi Rai	Raveena Tandon,Vivek Mushran
3	Anokha Moti	2000	Family	Anshul Singla	Sanjay Sharma,Arjun Chakraborty,Nayab Aftab
4	Apradhi Kaun	2000	Thriller	Mohan Bhakri	Ishrat Ali,Shagufta Ali
...	...	...	...	...	...
772	The Train	2007	Drama Thriller	Hasnain Hyderabadwala,Raksha Mistry	Emraan Hashmi,Geeta Basra,Rajat Bedi,Sayali Bh...
773	Victoria No. 203	2007	Comedy	Anant Mahadevan	Om Puri,Anupam Kher,Jimmy Sheirgill,Preeti Jha...
774	Welcome	2007	Comedy	Anees Bazmee	Akshay Kumar,Nana Patekar,Anil Kapoor,Katrina ...

	Title	Year	Genre	Director	Cast
775	Yatra	2007	Drama	Gautam Ghose	Rekha,Nana Patekar,Deepti Naval
776	Zamaanat	2007	Drama	S. Ramanathan	Amitabh Bachchan,Arshad Warsi,Karisma Kapoor

777 rows × 5 columns

Now, **new urls** are group together based on the difference in html text of these pages.

Followed same steps as above to extract the data from these webpages using **Webscrapping**. You will find changes in the code which is **necessary to locate** the required `table` , `rows` , and `columns` from html text.

In [5]:
```
URL1 = {2008:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2008',
        2009:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2009',
        2010:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2010',
         2012:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2012'
       }
for key, url in URL1.items():
    print(key,":",url)
```

```
2008 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2008 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2008)
2009 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2009 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2009)
2010 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2010 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2010)
2012 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2012 (https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2012)
```

In [6]:
```
for key, url in URL1.items():
    html_data = urllib.request.urlopen(url).read()
    soup = BeautifulSoup(html_data,"html.parser") # or soup = BeautifulSoup(html_data,"lxml")
    table = soup.find_all('table', class_='wikitable')[1:]
    #print(len(table))
    for t in table:
        rows = t.tbody.find_all('tr')[1:]
        for row in rows:
            col = row.find_all("td")
            if (col != []):
                Title = col[-4].get_text(strip=True)
                Year = str(key)
                Genre = col[-1].get_text(strip=True)
                Director = col[-3].get_text(strip=True)
                Cast = col[-2].get_text(strip=True)
                # Finally we append the data of each row to the table
                movie_data = movie_data.append({"Title":Title, "Year":Year, "Genre":Genre, "Dir
```

In [7]:
```
URL2 = {2011:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2011'
       }
for key, url in URL2.items():
    print(key,":",url)
```

```
2011 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2011 (https://en.wikipedia.or
g/wiki/List_of_Bollywood_films_of_2011)
```

In [8]:
```
for key, url in URL2.items():
    html_data = urllib.request.urlopen(url).read()
    soup = BeautifulSoup(html_data,"html.parser") # or soup = BeautifulSoup(html_data,"lxml")
    table = soup.find_all('table', class_='wikitable')[1:]
#     print(len(table))
    for t in table:
        rows = t.tbody.find_all('tr')[2:]
        for row in rows:
            col = row.find_all("td")
            if (col != []):
                Title = col[-5].get_text(strip=True)
                Year = str(key)
                Genre = col[-4].get_text(strip=True)
                Director = col[-3].get_text(strip=True)
                Cast = col[-2].get_text(strip=True)
                # Finally we append the data of each row to the table
                movie_data = movie_data.append({"Title":Title, "Year":Year, "Genre":Genre, "Dir
```

In [9]:
```
URL3 = {2013:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2013',
        2015:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2015',
       }
for key, url in URL3.items():
    print(key,":",url)
```

```
2013 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2013 (https://en.wikipedia.or
g/wiki/List_of_Bollywood_films_of_2013)
2015 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2015 (https://en.wikipedia.or
g/wiki/List_of_Bollywood_films_of_2015)
```

In [10]:
```
for key, url in URL3.items():
    html_data = urllib.request.urlopen(url).read()
    soup = BeautifulSoup(html_data,"html.parser") # or soup = BeautifulSoup(html_data,"lxml")
    table = soup.find_all('table', class_='wikitable')[1:]
#     print(len(table))
    for t in table:
        rows = t.tbody.find_all('tr')[1:]
        for row in rows:
            col = row.find_all("td")
            if (col != []):
                Title = col[-5].get_text(strip=True)
                Year = str(key)
                Genre = col[-2].get_text(strip=True)
                Director = col[-4].get_text(strip=True)
                Cast = col[-3].get_text(strip=True)
                # Finally we append the data of each row to the table
                movie_data = movie_data.append({"Title":Title, "Year":Year, "Genre":Genre, "Dir
```

In [11]:
```
URL4 = {2014:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2014',
      }
for key, url in URL4.items():
    print(key,":",url)
```

```
2014 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2014 (https://en.wikipedia.or
g/wiki/List_of_Bollywood_films_of_2014)
```

In [12]:
```
for key, url in URL4.items():
    html_data = urllib.request.urlopen(url).read()
    soup = BeautifulSoup(html_data,"html.parser") # or soup = BeautifulSoup(html_data,"lxml")
    table = soup.find_all('table', class_='wikitable')[2:]
#     print(len(table))
    for t in table:
        rows = t.tbody.find_all('tr')[1:]
        for row in rows:
            col = row.find_all("td")
            if (col != []):
                Title = col[-5].get_text(strip=True)
                Year = str(key)
                Genre = col[-2].get_text(strip=True)
                Director = col[-4].get_text(strip=True)
                Cast = col[-3].get_text(strip=True)
                # Finally we append the data of each row to the table
                movie_data = movie_data.append({"Title":Title, "Year":Year, "Genre":Genre, "Dir
```

In [13]:
```
URL5 = {2016:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2016',
        2016:'https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2016'
      }
for key, url in URL5.items():
    print(key,":",url)
```

```
2016 : https://en.wikipedia.org/wiki/List_of_Bollywood_films_of_2016 (https://en.wikipedia.or
g/wiki/List_of_Bollywood_films_of_2016)
```

In [14]:
```
for key, url in URL5.items():
    html_data = urllib.request.urlopen(url).read()
    soup = BeautifulSoup(html_data,"html.parser") # or soup = BeautifulSoup(html_data,"lxml")
    table = soup.find_all('table', class_='wikitable')[1:]
#     print(len(table))
    for t in table:
        rows = t.tbody.find_all('tr')[1:]
        for row in rows:
            col = row.find_all("td")
            if (col != []):
                if (col[-5].get_text(strip=True) == 'Force 2'):
                    continue
                Title = col[-6].get_text(strip=True)
                Year = str(key)
                Genre = col[-3].get_text(strip=True)
                Director = col[-5].get_text(strip=True)
                Cast = col[-4].get_text(strip=True)
                # Finally we append the data of each row to the table
                movie_data = movie_data.append({"Title":Title, "Year":Year, "Genre":Genre, "Dir
```

Once saved all data into `movie_data` dataframe. We need to remove the **comma** and **slash** sign from the `Genre` and `Cast` columns.

In [15]:
```
movie_data["Genre"] = movie_data['Genre'].str.replace('/'," ")
movie_data["Cast"] = movie_data['Cast'].str.replace(','," ")
display(movie_data)
```

	Title	Year	Genre	Director	Cast
0	Aaghaaz	2000	Thriller	Yogesh Ishwar	Sunil Shetty Sushmita Sen Namrata Shirodkar
1	Aaj Ka Ravan	2000	Drama	Imran Khalid	Kasam Ali Mithun Chakraborty
2	Anjaane	2000	Romance	Ravi Rai	Raveena Tandon Vivek Mushran
3	Anokha Moti	2000	Family	Anshul Singla	Sanjay Sharma Arjun Chakraborty Nayab Aftab
4	Apradhi Kaun	2000	Thriller	Mohan Bhakri	Ishrat Ali Shagufta Ali
...	...	...	...	...	...
1749	Moh Maya Money	2016	Crime drama	Munish Bharadwaj	Ranvir Shorey Neha Dhupia Vidhushi Mehra Ash...
1750	Kahaani 2	2016	Thriller	Sujoy Ghosh	Vidya Balan Arjun Rampal Jugal Hansraj
1751	Befikre	2016	Romance	Aditya Chopra	Ranveer Singh Vaani Kapoor
1752	Wajah Tum Ho	2016	Romance drama	Vishal Pandya	Sana Khan Sharman Joshi Gurmeet Choudhary
1753	Dangal	2016	Biopic	Nitesh Tiwari	Aamir Khan Sakshi Tanwar Fatima Sana Shaikh Sa...

1754 rows × 5 columns

Data Collection of English Movies From 'English_Movies.csv' File

Credit for English Movies data : https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata (https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata)

Load the data from the `English_Movies.csv` file to the pandas dataframe named `english_movies_data` .

In [16]:
```
english_movies_data = pd.read_csv('English_Movies.csv')
```

Now, Displayed the **First 5 rows** of the `english_movie_data` dataframe using the `head` function. For Last 5 rows, use `tail` function.

In [17]:
```
display(english_movies_data.head())
```

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

5 rows × 24 columns

Displayed the number of rows and columns available in `english_movie_data` dataframe.

This will be used in the next step **for iterating the data** through rows and columns to collect requied data for the movies and neglect other unnecessary data from the dataframe.

In [18]:
```
display(english_movies_data.shape)
```

```
(4803, 24)
```

The `3rd column` in the dataframe provides **Genre** of all the Films. It is important to build the recommendation system since it describes the content of the film (i.e. Drama, Comedy, Action, ...).

Similarly, The `13th column` in the dataframe provides **date-month-Year** of movie released. We are extracting only year from it.
The `19th column` in the dataframe provides **Title** of movie.
The `22th column` in the dataframe provides **Cast** (i.e. Actors and Actresses) worked in the Film.
The `24th column` (i.e. last column) in the dataframe provides **Director** name of the movie.

Now this all data, **iterated** starting from 1st row till the end row of `english_movies_data` dataframe using **for loop** and Saved into the `movie_data` dataframe that already includes all Bollywood movies Title, Year, Genre, Director, and Cast.

In [19]:
```
for i in range(0,len(english_movies_data)):
    Genre = english_movies_data.iloc[i,2]
    Year = str(english_movies_data.iloc[i,12]).split('-')[0]
    Title = english_movies_data.iloc[i,18]
    Cast = english_movies_data.iloc[i,21]
    Director = english_movies_data.iloc[i,23]
    movie_data = movie_data.append({"Title":Title, "Year":Year, "Genre":Genre, "Director":Direc

display(movie_data)
```

	Title	Year	Genre	Director	Cast
0	Aaghaaz	2000	Thriller	Yogesh Ishwar	Sunil Shetty Sushmita Sen Namrata Shirodkar
1	Aaj Ka Ravan	2000	Drama	Imran Khalid	Kasam Ali Mithun Chakraborty
2	Anjaane	2000	Romance	Ravi Rai	Raveena Tandon Vivek Mushran
3	Anokha Moti	2000	Family	Anshul Singla	Sanjay Sharma Arjun Chakraborty Nayab Aftab
4	Apradhi Kaun	2000	Thriller	Mohan Bhakri	Ishrat Ali Shagufta Ali
...	...	...	...	...	...
6552	El Mariachi	1992	Action Crime Thriller	Robert Rodriguez	Carlos Gallardo Jaime de Hoyos Peter Marquardt...
6553	Newlyweds	2011	Comedy Romance	Edward Burns	Edward Burns Kerry Bish\u00e9 Marsha Dietlein ...
6554	Signed, Sealed, Delivered	2013	Comedy Drama Romance TV Movie	Scott Smith	Eric Mabius Kristin Booth Crystal Lowe Geoff G...
6555	Shanghai Calling	2012	NaN	Daniel Hsia	Daniel Henney Eliza Coupe Bill Paxton Alan Ruc...
6556	My Date with Drew	2005	Documentary	Brian Herzlinger	Drew Barrymore Brian Herzlinger Corey Feldman ...

6557 rows × 5 columns

selected all relevant features for recommendation and Replaced **null values** from all columns with **null string**.

In [20]:
```
selected_features = ['Title','Year','Genre','Director','Cast']

for feature in selected_features:
  movie_data[feature] = movie_data[feature].fillna('')
```

Now, **Concatenated** all columns data of a particular movie into a single row. This will be used for **feature extraction** and them ML algorithm will be fitted for the whole data of `combined_features` variable.

In [21]:
```
# combining all the 5 selected features
combined_features = movie_data['Title']+' '+ \
                    movie_data['Year']+' '+ \
                    movie_data['Genre']+' '+ \
                    movie_data['Director']+' '+ \
                    movie_data['Cast']
```

In [22]:
```
print(combined_features)
```

```
0       Aaghaaz 2000 Thriller Yogesh Ishwar Sunil Shet...
1       Aaj Ka Ravan 2000 Drama Imran Khalid Kasam Ali...
2       Anjaane 2000 Romance Ravi Rai Raveena Tandon V...
3       Anokha Moti 2000 Family Anshul Singla Sanjay S...
4       Apradhi Kaun 2000 Thriller Mohan Bhakri Ishrat...
                              ...
6552    El Mariachi 1992 Action Crime Thriller Robert ...
6553    Newlyweds 2011 Comedy Romance Edward Burns Edw...
6554    Signed, Sealed, Delivered 2013 Comedy Drama Ro...
6555    Shanghai Calling 2012  Daniel Hsia Daniel Henn...
6556    My Date with Drew 2005 Documentary Brian Herzl...
Length: 6557, dtype: object
```

Feature Extraction of movies data

Now, Feature extraction is done on the `combined_features` variable. That is, All the **textual data** is converted into respective **numerical values** and stored into the variable named `feature_vectors` .

This is required for the use of `Cosine_Similarity Algorithm` , as it is best to use on numerical data than textual data.

In [23]:
```
# converting the text data to feature vectors
vectorizer = TfidfVectorizer()
```

In [24]:
```
feature_vectors = vectorizer.fit_transform(combined_features)
```

In [25]:
```
print(feature_vectors)
```

```
  (0, 15508)    0.3212168918471364
  (0, 11946)    0.3212168918471364
  (0, 15155)    0.22984093616686824
  (0, 16471)    0.29891232380015276
  (0, 15456)    0.2277890128059397
  (0, 16410)    0.2543586126639824
  (0, 8167)     0.4086452972083719
  (0, 18772)    0.3904280286213679
  (0, 16969)    0.10976573054416208
  (0, 115)      0.19082276158496245
  (0, 246)      0.4086452972083719
  (1, 3267)     0.275549755044569
  (1, 11464)    0.2876508093071839
  (1, 690)      0.22462070380693686
  (1, 8936)     0.3585400849116652
  (1, 9136)     0.3378765021211304
  (1, 8006)     0.31288994718397617
  (1, 5036)     0.08270743498818127
  (1, 13883)    0.41581314536561087
  (1, 8720)     0.2920081425195199
  (1, 247)      0.3841242613755517
  (1, 115)      0.19416989071951987
  (2, 11826)    0.39385528146594107
  (2, 18125)    0.3187795790159615
  (2, 16682)    0.3421942072235046
  :     :
  (6555, 18927) 0.2927398263319963
  (6555, 5374)  0.25218082303255973
  (6555, 7499)  0.2927398263319963
  (6555, 12915) 0.22131315003878832
  (6555, 2217)  0.16279936091809258
  (6555, 4259)  0.3465818206788711
  (6555, 15314) 0.26419636040687494
  (6555, 127)   0.13606599701744893
  (6555, 2951)  0.28304849602633114
  (6555, 625)   0.18341692591160594
```

```
(6556, 7557)  0.5555328847124797
(6556, 5817)  0.22570415340630234
(6556, 5154)  0.2354295962691884
(6556, 6923)  0.21426117322566274
(6556, 4337)  0.23950758877632605
(6556, 14276) 0.1687544167633868
(6556, 1754)  0.19406234206082199
(6556, 5062)  0.384305993365825
(6556, 3912)  0.21426117322566274
(6556, 5501)  0.16084134616330062
(6556, 2656)  0.31076834834710654
(6556, 18546) 0.1776838220579077
(6556, 120)   0.120921320525212
(6556, 11853) 0.1719243271386113
(6556, 4882)  0.15429335897004393
```

Cosine Similarity

When builing the system, the first step consists in defining a criteria that would tell us how close two films are.

To do so, start from the **description** of the film that was selected by the user. From it, will get the `director` name, the names of the `actors` and a other `keywords`. Then, build a **matrix** where each row corresponds to a film of the database and where the **columns** correspond to the previous quantities (director + actors + genre)

Title	Year	Genre 1	Genre 2	....	director	actor 1	actor 2	actor 3	...
Film 1	a_{11}	a_{12}			...				
...					...				
Film i	a_{i1}	a_{i2}			a_{ij}				
...					...				

Title	Year	Genre 1	Genre 2	....	director	actor 1	actor 2	actor 3	...
Film p	a_{p1}	a_{p2}			...				

In this matrix, the a_{ij} coefficients take either the value 0 or 1 depending on the correspondance between the significance of column j and the content of film i. Once this matrix has been defined, we **determine the distance between two films according to**:

$$d_{m,n} = \sqrt{\sum_{i=1}^{N} \left(a_{m,i} - a_{n,i}\right)^2}$$

At this stage, we just have to select the N films which are the **closest** from the entry selected by the user. According to similarities between entries, we get a list of N films.

Now, Our Primary aim is to get **Similarity Score** using `Cosine_similarity` .

Cosine_Similarity will compare first movie with all other movies, 2nd movie with all other movies and so on.

This will find which `feature_vectors` values are similar to each other i.e. In other way, it will find which movies are related to each other and accordingly, it will score them.

In [26]:
```
similarity = cosine_similarity(feature_vectors)
```

In [27]:
```
print(similarity)
```

```
[[1.         0.03705203 0.04319075 ... 0.         0.         0.        ]
 [0.03705203 1.         0.04394834 ... 0.00479505 0.         0.        ]
 [0.04319075 0.04394834 1.         ... 0.01171512 0.         0.        ]
 ...
 [0.         0.00479505 0.01171512 ... 1.         0.         0.02714674]
 [0.         0.         0.         ... 0.         1.         0.        ]
 [0.         0.         0.         ... 0.02714674 0.         1.        ]]
```

In [28]:
```
print(similarity.shape)
```

```
(6557, 6557)
```

Steps for Recommendation System on a Predefined movie

Before Getting input movie name from user, lets check our system with a movie name `kahaani`

In [29]:
```
movie_name = 'kahaani'
```

Create a `List` with all the movie names given in the dataset to **compare** it with the similarity score of movie `kahaani`

In [30]:
```
list_of_all_titles = movie_data['Title'].tolist()
print(list_of_all_titles)
```

```
i Kismat', 'Kismat', 'Kiss Kis Ko[4]', 'Krishna Cottage', 'Kyun! Ho Gaya Na...', 'Lakeer – F
orbidden Lines', 'Lakshya', "Let's Enjoy[5]", 'Madhoshi', 'Main Hoon Na', 'Maqbool', 'Mast
i', 'Meenaxi: A Tale of Three Cities', 'Meri Biwi Ka Jawaab Nahin', 'Mughal-e-Azam(Colorized
Version)', 'Mujhse Shaadi Karogi', 'Murder', 'Musafir', 'Muskaan', 'Naach', 'Netaji Subhas C
handra Bose: The Forgotten Hero', 'Paap', 'Paisa Vasool', 'Phir Milenge', 'Plan', 'Police Fo
rce: An Inside Story', 'Popcorn Khao! Mast Ho Jao', 'Raincoat', 'Rakht', 'Rok Sako To Rok L
o', 'Rudraksh', 'Run', 'Shaadi Ka Laddoo', 'Shart: The Challenge', 'Sheen', 'Shikaar', 'Shuk
riya: Till Death Do Us Apart', 'Silence Please...The Dressing Room[6]', 'Stop!', 'Suno Sasur
jee', 'Swades', 'Taarzan: The Wonder Car', 'Thoda Tum Badlo Thoda Hum', 'Tumsa Nahin Dekha:
A Love Story', 'Uff Kya Jaadoo Mohabbat Hai[7]', 'Vaastu Shastra (film)', 'Veer-Zaara', 'Waj
ahh: A Reason to Kill', "Where's the Party Yaar?", 'Woh[8]', 'Yeh Lamhe Judaai Ke', 'Yuva',
'7½ Phere', '99.9 FM', 'Aashiq Banaya Aapne', 'Amavas[2]', 'Anjaane', 'Apaharan', 'Bachke Re
hna Re Baba', 'Barsaat', 'Bewafaa', 'Bhagmati', 'Bhola in Bollywood', 'Black', 'Blackmail',
'Bluffmaster!', 'Brides Wanted', 'Bullet: Ek Dhamaka', 'Bunty Aur Babli', 'Chaahat – Ek Nash
a', 'Chand Sa Roshan Chehra', 'Chehraa', 'Chetna: The Excitement[3]', 'Chocolate', 'Chor Man
di[4]', 'Classic – Dance of Love', 'D', 'Dansh', 'Deewane Huye Paagal', 'Dil Jo Bhi Kahe
y...', 'Dosti: Friends Forever', 'Double Cross - Ek Dhoka', 'Dreams', 'Dus', 'Ek Ajnabee',
'Ek Khiladi Ek Haseena', 'Elaan', 'Fareb', 'Film Star', 'Fun – Can Be Dangerous Sometimes',
'Garam Masala', 'Hanuman', 'Hazaaron Khwaishein Aisi', 'Home Delivery', 'Hum Dum', 'Insan',
'Iqbal', 'Jackpot[5]', 'Jai Chiranjeeva', 'Jalwa - Fun in Love[6]', 'James', 'Jo Bole So Nih
```

Now, Find the **best match** for the movie name given by the user using `get_close_matches` function of the library `difflib` .

In [31]:
```
find_close_match = difflib.get_close_matches(movie_name, list_of_all_titles)
print(find_close_match)
```

```
['Kahaani', 'Yahaan', 'Tahaan']
```

In [32]:
```
close_match = find_close_match[0]
print(close_match)
```

```
Kahaani
```

Find the `index` of of the movie based on the Genre of the movie of `close_match` .

In [33]:
```
index_of_Genre_of_the_movie = movie_data[movie_data.Title == close_match]['Genre'].index.values
print(index_of_Genre_of_the_movie)
```

```
1108
```

Now, we need to get a `list` of Similar movies based on the **index found** above using the similarity variable that we have defined earlier which contains all Similarity Score.

First, It will find the movies that has same similarity score with `kahaani` movie. **Then**, These movies will be labelled as Highest similarity score. The movies which are not similar will be labelled a less similarity score.

In [34]:
```
similarity_score = list(enumerate(similarity[index_of_Genre_of_the_movie]))
print(similarity_score)
```

```
0463761433900153033), (152, 0.0), (153, 0.0), (154, 0.007561121363248064), (155, 0.007668358968765916), (156, 0.0), (157, 0.0), (158, 0.0), (159, 0.004210827982983537), (160, 0.005121509523360875), (161, 0.0), (162, 0.0), (163, 0.0), (164, 0.0045154781648319795), (165, 0.008305881226944463), (166, 0.0), (167, 0.00450248397189414), (168, 0.0040669331361413 57), (169, 0.0038651853548593595), (170, 0.0), (171, 0.004551805016219385), (172, 0.005621741036692465 4), (173, 0.004654439585094154), (174, 0.009509159130718415), (175, 0.004475807359993243), (176, 0.005537320049824148), (177, 0.0), (178, 0.0), (179, 0.004011265730909305), (180, 0.00 9514197952946696), (181, 0.0), (182, 0.0), (183, 0.0), (184, 0.00717599393449078), (185, 0.0 03991331874345362), (186, 0.008249383659277333), (187, 0.0), (188, 0.003922000970141539), (1 89, 0.003995665576756499), (190, 0.005044114666404466), (191, 0.0), (192, 0.0), (193, 0.0), (194, 0.0), (195, 0.0046357946188858926), (196, 0.0), (197, 0.0), (198, 0.07530283832392975), (199, 0.0), (200, 0.0055148729226062706), (201, 0.0), (202, 0.004667272752757332), (203, 0.0), (204, 0.004168694169138823), (205, 0.006769425267027014), (206, 0.0034810319692764814), (207, 0.005262126554167085), (208, 0.0), (209, 0.005800122399797953), (210, 0.0), (211, 0.010975096370016495), (212, 0.005492123235775489), (213, 0.009956629746166472), (214, 0.004325248140708979), (215, 0.004337497488543137), (216, 0.004564464761853026), (217, 0.004696909189462), (218, 0.009635323418400706), (219, 0.0), (220, 0.004723833231117671), (221, 0.003990491464499908), (222, 0.003925755294766058), (223, 0.0), (224, 0.0), (225, 0.0), (226, 0.004057601657041678), (227, 0.0), (228, 0.0), (229, 0.0), (230, 0.005490080770032394), (231, 0.008389089838031168), (232, 0.0046752433154388885), (233, 0.006022475107826216), (234, 0.0),
```

In the **Output of above cell**, the `1st value` represents the **index of the movie** from the dataset and the `2nd value` represent **similarity score** of that movie **based** on the movie_name entered by user (here **according** to `kahaani`). The same is repeated for all the movies and that movie score is relative to `kahaani` movie.

In [35]:
```
display(len(similarity_score))
```

```
6557
```

Now, we need to `sort` the movies in decreasing order based on their `similarity score`. This will help us to recommend movies with highest similarity score first.

In [36]:
```
sorted_similar_movies = sorted(similarity_score, key = lambda x:x[1], reverse = True)
print(sorted_similar_movies)
```

```
(1340, 0.08561483949065432), (374, 0.08360247220617906), (421, 0.0829986178454969), (757, 0.
08268313563349189), (5557, 0.08024876413170944), (321, 0.0788294843943297), (1399, 0.0787420
3822329301), (1152, 0.076967367310209), (1315, 0.07684360001580519), (1183, 0.07670365778525
629), (198, 0.07530283832392975), (767, 0.07486126777899821), (1040, 0.0721431595526483), (7
19, 0.07050716193413781), (925, 0.06990471553853432), (5322, 0.06940252121961471), (1580, 0.
0690838666278676), (1185, 0.06612667224715793), (6117, 0.06494686529164272), (271, 0.0648370
8733736837), (1024, 0.06440264184999966), (675, 0.06364531770410695), (1129, 0.0627142736455
2251), (1217, 0.06152586132972497), (1571, 0.059685441266296116), (466, 0.0595938324253118),
(838, 0.05912815075031422), (861, 0.05823355430836662), (6494, 0.05788104006935482), (6423,
0.05760095675344854), (348, 0.0572337158722908), (998, 0.05645258331547405), (558, 0.0564345
4236621456), (1067, 0.055674884849967306), (999, 0.055646507050943934), (6068, 0.05555838908
526451), (1586, 0.05418935967046449), (1255, 0.053527409956768926), (1422, 0.053160039828257
906), (555, 0.052431333698417626), (1335, 0.05241864269588819), (926, 0.05219588997848937),
(1019, 0.0508373514028261 3), (968, 0.04990615890729402), (1407, 0.04888955757230185), (698,
0.04868423301847023), (1005, 0.04708381221162844), (1457, 0.04553087380288707), (1278, 0.044
19925644273492), (6412, 0.0364189249292856 1), (4578, 0.03557377992308224), (5763, 0.03552578
9284603454), (5228, 0.03501624668108182), (4842, 0.03490916880400684), (1124, 0.034427385408
59374), (1757, 0.0342782618974168), (3598, 0.0339678300992797), (3224, 0.03347386321338444),
(2471, 0.03331616517591499), (3607, 0.033256940251226594), (4618, 0.03309349905102344), (285
6, 0.033042424839246465), (4299, 0.03296404656150578), (2638, 0.0325461887349143), (5261, 0.
```

In [37]:
```
display(movie_data.Genre.index)
```

```
RangeIndex(start=0, stop=6557, step=1)
```

Now, we can recommend the First few movies according to **similarity score** of movies relative to `kahaani` movie.

We need to go through **every index** of the `sorted_similar_movies` in the **decreasing order** and Find the **Title** of the movies related to that index. Print the movies serial wise. These are **recommended movies for** `kahaani` movie.

In [38]:
```
i = 1
for movie in sorted_similar_movies:
  index = movie[0]
  title_from_index = movie_data[movie_data.Genre.index == index]['Title'].values[0]
  if (i<=5):
    print(i,title_from_index, sep=') ')
    i+=1
```

```
1) Kahaani
2) Kahaani 2
3) Te3n
4) Traffic
5) Utthaan
```

Live Movie Recommendation System Based on User Input

The above steps are combined to one cell. Now, **User** can Input his favourite **movie** and get the **recommendation** of 30 movies which are **similar in content** to the movie_name he/she entered. `For that Run below-cell .`

In [39]:
```
movie_name = input('Enter your favourite movie name : ')

list_of_all_titles = movie_data['Title'].tolist()

find_close_match = difflib.get_close_matches(movie_name, list_of_all_titles)

close_match = find_close_match[0]

index_of_Genre_of_the_movie = movie_data[movie_data.Title == close_match]['Genre'].index.values

similarity_score = list(enumerate(similarity[index_of_Genre_of_the_movie]))

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

print('\nMovies suggested for you : \n')

i = 1
for movie in sorted_similar_movies:
  Genre_index = movie[0]
  title_from_index = movie_data[movie_data.Genre.index == Genre_index]['Title'].values[0]
  if (i<=30):
    print(i,title_from_index, sep=') ')
    i+=1
```

```
Enter your favourite movie name : Iron man

Movies suggested for you :

1) Iron Man
2) Iron Man 2
3) Iron Man 3
4) Made
5) Duets
6) The Good Night
7) The Avengers
8) The Last Airbender
9) Charlie Bartlett
10) Contagion
```

```
11) Captain America: Civil War
12) Zathura: A Space Adventure
13) Avengers: Age of Ultron
14) Sky Captain and the World of Tomorrow
15) The Iron Giant
16) Mortdecai
17) The Judge
18) Red Tails
19) The Best Man
20) A Scanner Darkly
21) Tropic Thunder
22) Holy Man
23) Lucky You
24) The Nativity Story
25) Sherlock Holmes
26) The Kite Runner
27) R.I.P.D.
28) The Best Man Holiday
29) Two Lovers
30) Se7en
```

NOTE: Execute the last cell to input your favourite movie name and you will get movies recommendation based on the content of movie entered (i.e. superhero movie, Space related movie, drame, thriller, comedy, etc)