Web Analytics - Final Project

Movie recommendations based on text from Wikipedia

July 12, 2017

Group 1 Members:

- Mauricio Alarcon
- Sekhar Mekala
- Aadi Kalloo
- Srinivasa Illapani
- Param Singh

Background

Movies recommendation is one of the classic application of recommendation systems, and there are several ways to achieve this. In this project, our goal is to apply Natural Language Processing (NLP) techniques to the movie plot obtained from Wikipedia and determine relevant movies for a given movie. We scraped text related to 4037 movies from Wikipedia. These movies are American movies released since the year 2000. The key deliverables of this project are:

- Text corpus of 4037 movies
- Movie posters of 3749 movies
- Movie recommender based on movie's plot

Technologies used:

We used the following software/packages to develop the core logic of this project:

- Python 3
- Pandas
- Numpy
- Sklearn
- BeautifulSoup
- urllib

NOTE: We scraped movie release posters to render the recommendations in a more aesthetic fashion. However, we could not get all the movie posters, since some of them are not available, and some of them were not easily downloadable by our crawler since the webpage's HTML IDs are not consistent.

This project is divided into 4 logical phases:

1. Phase-1: Build a web crawler to download the movies text and release posters
2. Phase-2: Cleanse the text data to build the recommender system
3. Phase-3: Build the recommender system using the text data
4. Phase-4: Get the recommendations

The subsequent sections will have a detailed explanation of these phases.

I. Phase-1: Build a Web Crawler

The main goal of **Phase-1** is to build a web crawler and scrape the text related to American movies, which were released between 2000 and 2016 years. Along with the text, we will also crawl the Movies posters. The major deliverables of this phase are:

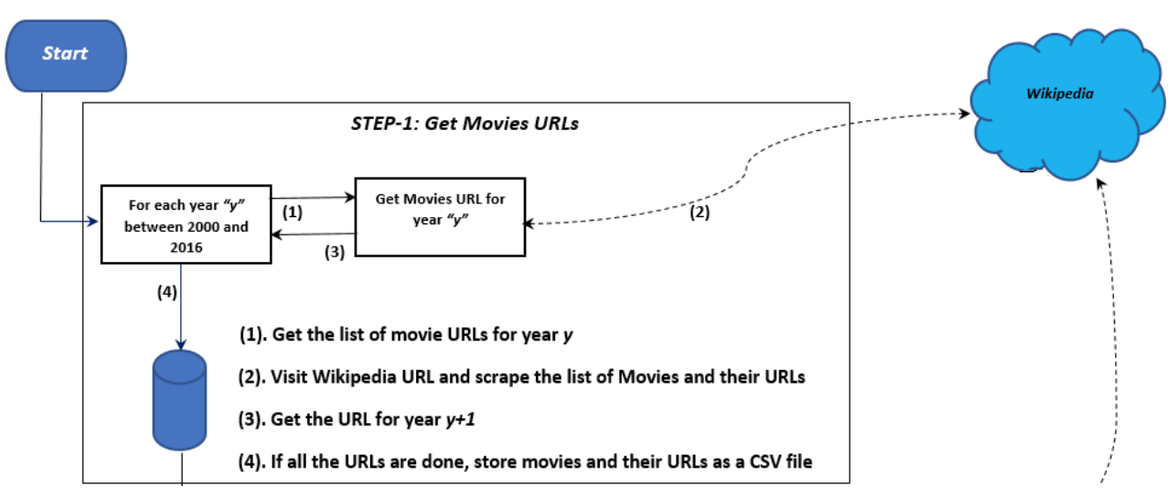
1. Text (or plot) of American movies, which were released between the years 2000 and 2016
2. Movies release posters

NOTE: All the data will be obtained from Wikipedia.

I.I Design

Wikipedia maintains list of movies, released in each year. The list of American movies released in each year are present at https://en.wikipedia.org/wiki/List_of_American_films_of_XXXX (https://en.wikipedia.org/wiki/List_of_American_films_of_XXXX), where XXXX is the year. For each year between 2000 and 2016 (XXXX = 2000 to 2016), we have to recurrsively visit each year's URL to obtain the movies URLs, along with movie details such as cast, director, genre etc. Once the URLs related to all the movies are obtained, the web crawler will visit the movie URLs to scrape the plot of the movie.

The following flowchart will provide an overview of the web crawling process:



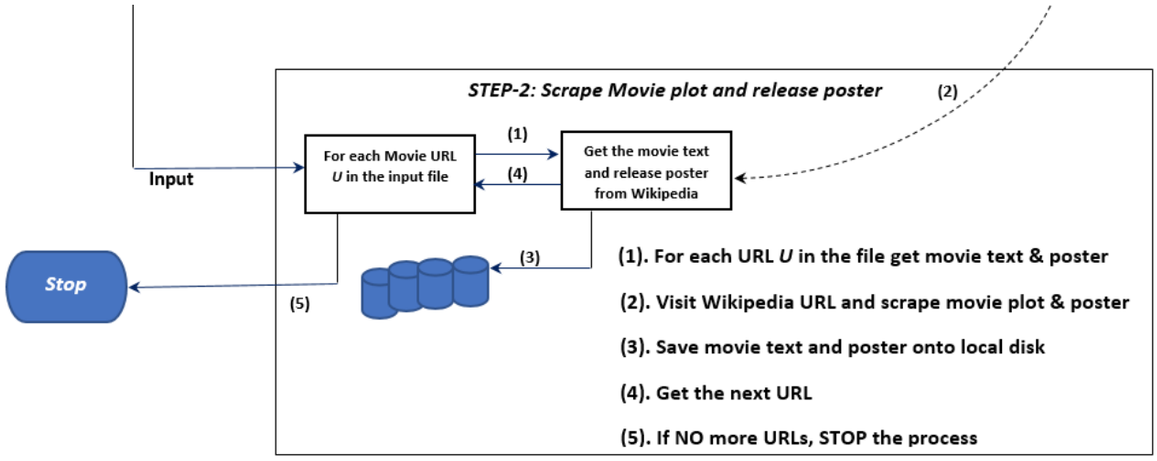


Figure-1: Web crawling process

The whole web crawling process is divided into 2 steps. In Step-1, we will visit Wikipedia to obtain a list of all the URLs related to the American movies which were released between the years 2000-2016. In Step-2, we will visit each of the URLs obtained in Step-1, to download movie text and release poster. There must be some delay of 2 to 3 seconds between successive requests to Wikipedia website.

The output of Step-1 will be a comma separated file (Movie_Details.csv), with the following details:

Movie - Movie Name

URL - Wikipedia web page for the movie

Year - Year of release

Director - Director of the movie

Cast - Cast of the movie

Genre - Movie's genre

Movie_ID - Unique key to distinguish each movie

The output of Step-2 will be a set of text files, each file named using the Movie_ID, and a set of image files, each of the image files will also be named using the Movie_ID. Naming the files using the Movie_ID will help us to refer movie's text and image files uniquely.

I.II Implementation

I.II.I Import the required packages

Let us import all the required packages:

In [3]:
```
import pandas as pd
import numpy as np
from IPython.display import display # Allows the use of display() for DataFrames
import time
import pickle #To save the objects that were created using webscraping
import pprint
from lxml import html
import requests
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
from urllib.request import urlopen
from bs4 import BeautifulSoup
from IPython.display import HTML
import re
import urllib
import os
```

I.II.II Step-1 Implementation

In Step-1 of the web crawling process, we will obtain the list of movies, along with the URLs of the movies, for all the movies which were released between the years 2000 and 2016.

But the challenge was, the format of the HTML file. Wikipedia used one format for the movies released between 2000 and 2013, and another format for the movies released in 2014-16. So we will sub-divide the Step-1 process further to scrape the list of movies between 2000 and 2013 in phase 1 and 2015-16 in phase2.

In [60]:
```
##PHASE-1: Get the movies and URLs for the years 2000-2013
#Define the lists to hold the details of the movies
URL = list()
Movie_Name = list()
Director = list()
Cast = list()
Genre = list()
year = list()

#Create a beautiful soup object
bs = BeautifulSoup(html)

#Iterate over the years 2000 to 2014.
for y in list(range(2000,2014)):

    #Prepare the URL String and open the URL
    url = "https://en.wikipedia.org/wiki/List_of_American_films_of "+str(y)
```

```
    html = urlopen(url)

    #Mandatory wait of 3 seconds
    time.sleep(3)

    #Get the web page as HTML document
    bs = BeautifulSoup(html)

    #Parse and get the required data
    for table in bs.find_all('table', {"class":"wikitable"}):
        for row in table.find_all('tr'):
            columns = row.find_all('td')
            if len(columns) > 4:
                Movie_Name.append(columns[0].get_text())
                Director.append(columns[1].get_text())
                Cast.append(columns[2].get_text())
                Genre.append(columns[3].get_text())
                year.append(y)

                #Handle exceptions, so that the process continues
                try:
                    a = columns[0].find('a',href=True)['href']
                    URL.append("https://en.wikipedia.org"+a)
                except:
                    URL.append("NA")
                    continue
```

In [33]:
```
##PHASE-2: Get the movies details for the years 2014 to 2016
#Declare the lists
URL1 = list()
Movie_Name1 = list()
Director1 = list()
Cast1 = list()
Genre1 = list()
year1 = list()

#For the years between 2014 and 2016
for y in range(2014,2017):

    #Prepare wiki URL
    url = "https://en.wikipedia.org/wiki/List_of_American_films_of_"+str(y)

    #Exception handling to ignore the failures and continue processing
    try:
        html = urlopen(url)
    except:
        print("problem with the following URL...continuining...:")
        print(url)
        continue
    #Sleep for 3 secs
    time.sleep(3)

    #Declare beautiful soup object
    bs = BeautifulSoup(html)
    for table in bs.find_all('table', {"class":"wikitable"}):
        for row in table.find_all('tr'):
            columns = row.find_all('td')
            if len(columns) > 3: #To make sure that we are accessing the movies tables only
                if len(columns) == 6:
                    #print(columns[0].get_text())
                    Movie_Name1.append(columns[0].get_text())
                    Director1.append(columns[1].get_text())
                    Cast1.append(columns[2].get_text())
                    Genre1.append(columns[3].get_text())
                    year1.append(y)
                    try:
                        a=columns[0].find('a',href=True)['href']
                        URL1.append("https://en.wikipedia.org"+a)
                    except:
                        URL1.append("NA")
                        continue

                if len(columns) == 7:
                    #print(columns[1].get_text())
                    Movie_Name1.append(columns[1].get_text())
                    Director1.append(columns[2].get_text())
                    Cast1.append(columns[3].get_text())
                    Genre1.append(columns[4].get_text())
                    year1.append(y)

                    try:
                        a=columns[1].find('a',href=True)['href']
                        URL1.append("https://en.wikipedia.org"+a)
                    except:
                        URL1.append("NA")
                        continue

                if len(columns) > 7:
                    #print("col len:{}".format(len(columns)))
                    #print(columns[2].get_text())
                    Movie_Name1.append(columns[2].get_text())
                    Director1.append(columns[3].get_text())
                    Cast1.append(columns[4].get_text())
                    Genre1.append(columns[5].get_text())
                    year1.append(y)
                    try:
                        a=columns[2].find('a',href=True)['href']
                        URL1.append("https://en.wikipedia.org"+a)
                    except:
                        URL1.append("NA")
                        continue
```

I.II.III Save the results

We will save the movies details as a CSV file named Movie_Details.csv. This helps us to avoid running Step-1 again.

In [ ]:
```
#Create a data frame:
df = pd.DataFrame(list(zip(Movie_Name+Movie_Name1,URL+URL1,year+year1,Director+Director1,Cast+Cast
1,Genre+Genre1)),\
                  columns=["Movie","URL","Year","Director","Cast","Genre"])

#Remove the rows which do not have URL information
df = df[df["URL"] != "NA"]

df["Movie_ID"] = df.index + 1

#Write the file
df.to_csv("Movies_Details.csv",encoding='utf-8',index=False)
```

I.II.IV Read the saved file to a data frame.

We will read back the saved file in Step-1, to a data frame. You can download the Movies_Details.csv file from *https://goo.gl/RDhVtf (https://goo.gl/RDhVtf)*.

In [4]:
```
URL = pd.read_csv("Movie_Details.csv")

print("Initial rows of the file Movie_Details.csv")
display(URL.head())

print("The Movie_Details.csv has {} rows and {} columns".format(URL.shape[0],URL.shape[1]))
```

```
Initial rows of the file Movie_Details.csv
```

	Movie	URL	Year	Director	Cast	Genre	Movie_ID
0	102 Dalmatians	https://en.wikipedia.org/wiki/102_Dalmatians	2000	Kevin Lima	Glenn Close, Gérard Depardieu, Alice Evans	Comedy, family	1
1	28 Days	https://en.wikipedia.org/wiki/28_Days_(film)	2000	Betty Thomas	Sandra Bullock, Viggo Mortensen	Drama	2
2	3 Strikes	https://en.wikipedia.org/wiki/3_Strikes_(film)	2000	DJ Pooh	Brian Hooks, N'Bushe Wright	Comedy	3
3	The 6th Day	https://en.wikipedia.org/wiki/The_6th_Day	2000	Roger Spottiswoode	Arnold Schwarzenegger, Robert Duvall	Science fiction	4
4	Across the Line	https://en.wikipedia.org/wiki/Across_the_Line_...	2000	Martin Spottl	Brad Johnson, Adrienne Barbeau, Brian Bloom	Thriller	5

```
The Movie_Details.csv has 4045 rows and 7 columns
```

There are 4045 movie URLs that have to be scraped from Wikipedia. Our goal is to scrape the image of the movie (if exists), along with the plot and initial introduction texts.

I.II.V Step-2 Implementation

In Step-2 we will use the file Movie_Details.csv, which has the list of all movie URLs, along with some other details. Each URL of the movie will be crawled to extract the movies text and the images. In Step-2, we will build a set of functions to perform the web crawling. These functions are explained below:

I.II.V.I Functions

We will code the following functions to obtain the plot information of the movie, along with the release poster image of the movie.

- **Open_URL(url)** Gets the HTML content, prepares Beautiful Soup object and returns the Beautiful Soup object. The *url* parameter represents the complete URL of the webpage to be scraped. If an error occurs while opeing the URL, then -1 is returned. If an error occurs while preparing the beautiful soup object, then -2 is returned.
- **Get_Plot(bs)** Takes a beautiful soup object as input. It extracts the introductory text, and the text in the section *plot* If plot section is NOT present, then it return a negative code. The function returns the extracted text (in the *plot* section and the initial paragraph). If an error occurs, then a negative code is returned. Return code = -1: If an error has occurred while getting the paragraphs from bs object Return code = -2: If there is NO *Plot* section in the document Return code = -3: If an error occurred while extracting the first paragrapg in HTML doc
- **Get_All_Text(bs)** This function is called only when **Get_Plot(bs)** returns a -2 (No section with the heading *Plot* is found). This function will take a beautiful object as input and gives all the text (present in < p > tags) as output. It will return -1, if any error occurs.
- **Save_Text_File(text,text_file_name)** It will save the text string (*text*) as a text file (with the name contained in *text_file_name*). The file is saved into the *data* directory. Returns 0, if sucessfully saved. Returns -1, when the change directory command fails (while changing to *data* sub-directory), and -2 when the change to parent directory from *data* fails.
- **Get_And_Save_Image(bs,image_file_name)** It will get the movies poster (image file) and saves the image in the *images* directory. It will take beautiful soup object as input and extracts the image URL. The image URL will be downloaded and saved to *images* directory with the file name as the value present in *image_file_name*. Returns 0, if successfully downloaded and saved. Returns -1 if image is not found, and -2 if the an error occurs when saving the image.
- **Write_Error(url,msg,file)** Will write a error/warning message (contained in the parameter *msg*), while parsing the URL (present in the *url* parameter). The *file* parameter contains the name of the error file.

The source code of these functions is given below:

In [9]:
```
def Open_URL(url):
    '''
    Gets the HTML content, prepares Beautiful Soup object and
```

```
    returns the  Beautiful Soup object.
    The url parameter represents the complete URL of the webpage.
    '''
    try:
         html = urlopen(url)
    except:
        return -1
    try:
        #bs = BeautifulSoup(html).encode("ascii")
        bs = BeautifulSoup(html)

        return bs
    except:
        return -2

def Get_Plot(bs):
    """
    Takes a beautiful soup object as input.
    Extracts the introductory text, and the text in the section plot
    If plot section is NOT present, just get all the available text in the webpage
    Returns the extracted text (if NO error, else returns a negative error code).
    -1: If an error has occurred while getting the paragraphs from bs object
    -2: If an error occurred while extracting the plot text
    -3: If an error occurred while extracting the first paragrapg in HTML doc
    """
    try:
        p = bs.find("p")
        initial_paragraph = p.getText()
    except:
        return -1

    # collect plot in this list
    plot = []

    # find the node with id of "Plot"
    try:
        mark = bs.find(id="Plot")
        # walk through the siblings of the parent (H2) node
        # until we reach the next H2 node
        for elt in mark.parent.nextSiblingGenerator():
            if elt.name == "h2":
                break
            if hasattr(elt, "text"):
                plot.append(elt.text)
    except:
         return -2

    try:
        plot="".join(plot)
        text = initial_paragraph + plot
        return text
    except:
        return -3



def Get_All_Text(bs):
    try:
        p = bs.find_all("p")
        l = list()
        for i in p:
            l.append(i.getText())
        return " ".join(l)
    except:
        return -1

def Save_Text_File(text,text_file_name):
    try:
        os.chdir("./data")
    except:
        return -1

    with open(text_file_name, 'w',encoding='utf-8') as f:
         f.write(text)

    try:
        os.chdir("..")
        return 0
    except:
        return -2

def Get_And_Save_Image(bs,image_file_name):
    try:
        img=bs.findAll("img",{"class":"thumbborder"})
        img_URL="https:"+img[0]['src']
    except:
        return -1

    try:
        os.chdir("./images")
        ignore=urllib.request.urlretrieve(img_URL,image_file_name)
        os.chdir("..")
        return 0
    except:
        return -2

def Write_Error(url,msg,file):
    with open(file,'a') as f:
        f.write("\n"+msg)
        f.write("\n"+url)
```

I.III Beginning the major crawling process

The following code block will crawl the movies text from Wikipedia. This code ran for approximately 7 hours. So do NOT execute this code, unless you really want to start the download process. The output of this code is a series of text files and image files. The text files are saved to the *data* directory and images to the *image* directory. You can download these files directly from the location: *https://goo.gl/RDhVtf (https://goo.gl/RDhVtf)*

In [57]:
```
tracker = 0
term=1
start = time.time() # Get start time
print("Beginning the files download...")
#k = 1
for movie, url, year,Movie_ID in zip(list(URL["Movie"]),list(URL["URL"]),list(URL["Year"]),list(UR
L["Movie_ID"])):
    #if k < 30:
    #    k = k+1
    #    continue
    #print("{},{},{}".format(movie,url,year))
    #Open the URL

    bs=Open_URL(url)

    if bs == -1:
        Write_Error(url,"Error in opening the URL","error.txt")
        #print("Error in opening the URL: {}".format(url))
        continue


    if bs == -2:
        Write_Error(url,"Error in the creation of bs object for the URL","error.txt")
        #print("Error in the creation of bs object for the URL: {}".format(url))
        continue

    time.sleep(3)

    #create a name for the files
    #image_file_name = str(year)+"_"+movie.strip()+".jpg"
    #text_file_name =  str(year)+"_"+movie.strip()+".txt"
    image_file_name = str(Movie_ID)+".jpg"
    text_file_name =  str(Movie_ID)+".txt"

    text=Get_Plot(bs)

    if text == -1:
        Write_Error(url,"No paragraphs are found","error.txt")
        #print("No paragraphs are found")
        #print(url)
        continue

    if text == -2:
        Write_Error(url,"Warning: No Plot ID found","error.txt")
        #print("Warning: No Plot ID found")
        #print(url)

        text = Get_All_Text(bs)
        if text == -1:
            Write_Error(url,"No paragraphs are found","error.txt")
            #print("No paragraphs are found")
            #print(url)
            continue

    if text == -3:
        Write_Error(url,"Error while appending the main plot with the initial paragraph","error.tx
t")
        #print("Error while appending the main plot with the initial paragraph")
        #print(url)
        continue

    status = Save_Text_File(text,text_file_name)

    if status == -1:
        Write_Error(url,"Not able to change the directory to ./data","error.txt")
        #print("Not able to change the directory to ./data")
        #print(url)
        continue

    if status == -2:
        Write_Error(url,"Not able to change the directory to .. (parent directory) from ./data","e
rror.txt")
        #print("Not able to change the directory to .. (parent directory) from ./data")
        #print(url)
        continue

    #Downloading Image files
    status = Get_And_Save_Image(bs,image_file_name)

    if status == -1:
        Write_Error(url,"Not able to find the image","error.txt")
        #print("Not able to find the image")
        #print(url)
        continue

    if status == -2:
        Write_Error(url,"Not able to save the image","error.txt")
        #print("Not able to save the image")
        #print(url)
        continue

    #Check the status of the webbot
    tracker = tracker + 1
    if (tracker % 100 == 0):
        print("Processed {} URLs".format(tracker))
        end = time.time() # Get end time
```

```
        end = time.time() # Get end time
        elapsed_time = end - start
        print("Elapsed time to process 100 URLs:{} secs".format(elapsed_time))
        start = time.time() # Get end time
        #break

    #if term == 1:
    #    break
```

```
Beginning the files download...
Processed 100 URLs
Elapsed time to process 100 URLs:446.38204622268677 secs
Processed 200 URLs
Elapsed time to process 100 URLs:433.5029966831207 secs
Processed 300 URLs
Elapsed time to process 100 URLs:461.3205850124359 secs
Processed 400 URLs
Elapsed time to process 100 URLs:458.53809428215027 secs
Processed 500 URLs
Elapsed time to process 100 URLs:476.4255542755127 secs
Processed 600 URLs
Elapsed time to process 100 URLs:473.6112816333771 secs
Processed 700 URLs
Elapsed time to process 100 URLs:458.1378722190857 secs
Processed 800 URLs
Elapsed time to process 100 URLs:434.91878509521484 secs
Processed 900 URLs
Elapsed time to process 100 URLs:569.373973608017 secs
Processed 1000 URLs
Elapsed time to process 100 URLs:533.2878279685974 secs
Processed 1100 URLs
Elapsed time to process 100 URLs:493.94636368751526 secs
Processed 1200 URLs
Elapsed time to process 100 URLs:560.7728536128998 secs
Processed 1300 URLs
Elapsed time to process 100 URLs:519.7430667877197 secs
Processed 1400 URLs
Elapsed time to process 100 URLs:661.1075065135956 secs
Processed 1500 URLs
Elapsed time to process 100 URLs:468.4266812801361 secs
Processed 1600 URLs
Elapsed time to process 100 URLs:482.55657744407654 secs
Processed 1700 URLs
Elapsed time to process 100 URLs:468.59105467796326 secs
Processed 1800 URLs
Elapsed time to process 100 URLs:448.06637740135193 secs
Processed 1900 URLs
Elapsed time to process 100 URLs:454.26227259635925 secs
Processed 2000 URLs
Elapsed time to process 100 URLs:437.51902770996094 secs
Processed 2100 URLs
Elapsed time to process 100 URLs:437.7170066833496 secs
Processed 2200 URLs
Elapsed time to process 100 URLs:432.09807682037354 secs
Processed 2300 URLs
Elapsed time to process 100 URLs:429.547082901001 secs
Processed 2400 URLs
Elapsed time to process 100 URLs:431.0166103839874 secs
Processed 2500 URLs
Elapsed time to process 100 URLs:426.8530662059784 secs
Processed 2600 URLs
Elapsed time to process 100 URLs:434.2797124385834 secs
Processed 2700 URLs
Elapsed time to process 100 URLs:432.32950735092163 secs
Processed 2800 URLs
Elapsed time to process 100 URLs:462.65187335014343 secs
Processed 2900 URLs
Elapsed time to process 100 URLs:475.1674907207489 secs
Processed 3000 URLs
Elapsed time to process 100 URLs:446.28495264053345 secs
Processed 3100 URLs
Elapsed time to process 100 URLs:486.55276560783386 secs
Processed 3200 URLs
Elapsed time to process 100 URLs:478.56283259391785 secs
Processed 3300 URLs
Elapsed time to process 100 URLs:509.5021858215332 secs
Processed 3400 URLs
Elapsed time to process 100 URLs:468.18198013305664 secs
Processed 3500 URLs
Elapsed time to process 100 URLs:441.2250940799713 secs
Processed 3600 URLs
Elapsed time to process 100 URLs:445.52249574661255 secs
Processed 3700 URLs
Elapsed time to process 100 URLs:511.63219952583313 secs
```

Out of 4045 movies we obtained 4037 movies text successfully. But we were able to obtain only 3749 images, since images were not available to some of the movies. The errors and warnings are logged into a file named *error.txt*. You can find this file at *https://goo.gl/RDhVtf (https://goo.gl/RDhVtf)*

II. Phase-2: Data Cleansing

Now that we downloaded the data, let us clean the data to make the data ready for applying text analytics algorithms. To perform data cleansing, we created the following 3 functions:

- **Read_File(p)** It will open the input file, reads the text in the file, converts all the test to lower case, removes the punctuation (if any), and returns a list of tokens.
- **Remove_Stop_Words(tokens)** It will remove all stop words from the list of input tokens. Returns a refined list of tokens, with no stop words
- **Clean_Text(tokens)** It will clean all the text by removing any square brackets "[...]", braces "(" and ")", commas, colons, apostrophes etc.

Returns a list of tokens that are just alphanumeric.

The source code of these functions is given below:

In [3]:
```
def Read_File(p):
  with open(p, 'r',encoding='utf-8') as f:
    text = f.read()
    #Convert all the text to lower case
    #
    lowers = text.lower()
    #remove the punctuation using the character deletion step of translate
    no_punctuation = lowers.translate(string.punctuation)
    tokens = nltk.word_tokenize(no_punctuation)
    return tokens

def Remove_Stop_Words(tokens):
    filtered = [w for w in tokens if not w in stopwords.words('english')]
    return filtered



def Clean_Text(tokens):
    text = " ".join(tokens)
    #Remove punctuation marks, text in [], (, ), :
    filtered1 = re.sub('\.|\`|\'|\[.*\]|\(|\)|,|:', " ",text)

    #Remove any single characters
    filtered1 = re.sub('(^| ).( |$)', " ",filtered1)
    #Remove any contiguous spaces
    filtered1 = re.sub(' +'," ",filtered1)

    #Include only alpha numeric characters
    filtered1=" ".join([i for i in filtered1.split() if re.search('[0-9 a-z]*',i)])
    return filtered1
```

The following code block will use the above functions to clean the text. Do NOT run this code, unless you want to test it, since it will run for some time. To save time, we saved the results as *processed_data.csv*. This file can be downloaded from *https://goo.gl/RDhVtf (https://goo.gl/RDhVtf)*.

In [6]:
```
import re
import string
import nltk
from collections import Counter
from nltk.corpus import stopwords

#List the files in the directory./data
file_names = os.listdir("./data")

#Process each file
file_names = [i for i in file_names if re.search('[1-9]*\.txt',i)]
y = list()
x = list()
k = 0
start = time.time()
for i in file_names:
    y.append(int(i.split(".")[0]))
    tokens = Read_File("./data/"+i)
    tokens = Remove_Stop_Words(tokens)
    cleaned_text = Clean_Text(tokens)
    x.append(cleaned_text)
    k = k+1
    if(k%100 == 0):
        temp_time = time.time() - start
        print("Processed {} files. Elapsed time:{} seconds".format(k, temp_time))

temp_time = time.time() - start
print("Processed {} files. Elapsed time:{} seconds".format(k, temp_time))


print("Now saving the result as processed_data.csv file...")
df=pd.DataFrame(list(zip(y,x)),columns = ["Movie_ID","Plot"])
df.to_csv("processed_data.csv",encoding='utf-8',index=False)
```

```
Processed 100 files. Elapsed time:34.47499084472656 seconds
Processed 200 files. Elapsed time:68.88607907295227 seconds
Processed 300 files. Elapsed time:102.14953780174255 seconds
Processed 400 files. Elapsed time:134.21351504325867 seconds
Processed 500 files. Elapsed time:160.64091873168945 seconds
Processed 600 files. Elapsed time:190.43764638900757 seconds
Processed 700 files. Elapsed time:219.01563954353333 seconds
Processed 800 files. Elapsed time:251.53410053253174 seconds
Processed 900 files. Elapsed time:282.54603719711304 seconds
Processed 1000 files. Elapsed time:316.63187885284424 seconds
Processed 1100 files. Elapsed time:351.59121203422546 seconds
Processed 1200 files. Elapsed time:387.9956316947937 seconds
Processed 1300 files. Elapsed time:429.9995768070221 seconds
Processed 1400 files. Elapsed time:471.7074546813965 seconds
Processed 1500 files. Elapsed time:509.83835220336914 seconds
Processed 1600 files. Elapsed time:549.4117841720581 seconds
Processed 1700 files. Elapsed time:585.4907121658325 seconds
Processed 1800 files. Elapsed time:620.8212609291077 seconds
Processed 1900 files. Elapsed time:658.4339408874512 seconds
Processed 2000 files. Elapsed time:694.9503221511841 seconds
Processed 2100 files. Elapsed time:735.9942619800568 seconds
Processed 2200 files. Elapsed time:765.2880411148071 seconds
Processed 2300 files. Elapsed time:795.2281031608582 seconds
Processed 2400 files. Elapsed time:833.5187902450562 seconds
Processed 2500 files. Elapsed time:869.070404291153 seconds
Processed 2600 files. Elapsed time:903.4122550487518 seconds
Processed 2700 files. Elapsed time:934.0018377304077 seconds
Processed 2800 files. Elapsed time:966.8597326278687 seconds
Processed 2900 files. Elapsed time:1001.9057586193085 seconds
```

```
Processed 3000 files. Elapsed time:1038.1752934455872 seconds
Processed 3100 files. Elapsed time:1074.702586889267 seconds
Processed 3200 files. Elapsed time:1111.477680683136 seconds
Processed 3300 files. Elapsed time:1148.6706750392914 seconds
Processed 3400 files. Elapsed time:1187.1972496509552 seconds
Processed 3500 files. Elapsed time:1220.9358689785004 seconds
Processed 3600 files. Elapsed time:1253.176054239273 seconds
Processed 3700 files. Elapsed time:1288.4286420345306 seconds
Processed 3800 files. Elapsed time:1321.8880531787872 seconds
Processed 3900 files. Elapsed time:1352.051034450531 seconds
Processed 4000 files. Elapsed time:1383.7957265377045 seconds
Processed 4037 files. Elapsed time:1394.2528154850006 seconds
Now saving the result as processed_data.csv file...
```

We can see that the data cleaninsing process has ran for approximately 23 minutes. However the results of this process are saved as a CSV file processed_data.csv. This file is located at *https://goo.gl/RDhVtf (https://goo.gl/RDhVtf)*

Reading the processed_data.csv file into a data frame.

In [5]:
```
df = pd.read_csv("processed_data.csv")
print("Initial records of processed_data.csv file")
df.head()
```

```
Initial records of processed_data.csv file
```

Out[5]:

	Movie_ID	Plot
0	1	102 dalmatians 2000 american family comedy fil...
1	10	american psycho 2000 american black comedy hor...
2	100	legacy 2000 american documentary film directed...
3	1000	lemony snicket series unfortunate events 2004 ...
4	1001	life death peter sellers 2004 british-american...

The above display shows that our final data frame, which will be used to build movies recommender is composed of two columns. The first column *Movie_ID* will uniquely identify the movie, and the *Plot* will identify the cleansed text of the movie plot. The movie's release poster will be present in the "./images" directory (you can download it from *https://goo.gl/RDhVtf (https://goo.gl/RDhVtf)*. But after download save it in the ./data directory or else this Jupyter notebook will not find the images)

III. Phase-3: Building the recommender

Our recommender system is based on text analytics of the movies plot. We will use TF-IDF (Term Frequency - Inverse Document Frequency) score for each unique word in each document. All the unique words in the combined text of all the documents will form the *features*.

Once the TFIDF is computed, we will obtain the cosine similarity between each pair of movies.

III.I. TF-IDF Algorithm:

TFIDF (Term Frequency - Inverse Document Frequency) is one of the most popular text processing algorithms that helps us to accurately assign importance scores to each word in a document.

At a very high level, the algorithm follows the below logic:

Let $D = d_1, d_2...d_n$ be a set of documents.

For each document d in D perform the following:

a. Get the frequencies of all the words in d. Call this as TF (Term Frequency) vector for document d

b. Get the list of all unique words in all the documents, and for each unique word, get the number of documents containing the word. Let DF (Document Frequency) be the vector containing these counts.

For each word w in DF, get the following:

$$IDF_w = log(n / (1 + w))$$

The log can have any valid base. IDF stands for Inverse Document Frequency. "n" represents the total number of documents

For each document d, multiply the elements of TF_d with the corresponding elements of IDF, to obtain TFIDF vector for document d.

In sklearn package, we have TfidfVectorizer class, which implements the TF-IDF algorithm. Using this class, we are able to obtain the TF-IDF scores of all the unique words in all the movies plot.

III.II Get the TFIDF scores

Using the data frame (*processed_data*), The below code block will get the TFIDF scores for all the words in each of the document.

In [6]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(df["Plot"])

print("The TF-IDF matrix has {} rows and {} columns".format(tfidf_matrix.shape[0],tfidf_matrix.sha
pe[1]))
```

```
The TF-IDF matrix has 4037 rows and 54075 columns
```

The TF-IDF matrix for the movies text has 4037 rows (representing the number of movies) and 54075 columns (representing the unique words in all the movies text). Internally python represents this matrix as a sparse matrix, since most of the elements of this matrix have a value of 0.

III.III. Get the cosine similarity measure between each pair of movie

To obtain the cosine similarity between each pair of movies, we will use cosine_similarity class of sklearn package. The below code block will

compute the cosine similarity between each pair of movies.

In [7]:
```
from sklearn.metrics.pairwise import cosine_similarity
cos_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
cos_sim_df = pd.DataFrame(cos_sim,columns=df["Movie_ID"].tolist(),index=df["Movie_ID"].tolist())
```

Let us display some rows and columns of the cosine similarity measure.

In [8]:
```
display(cos_sim_df)
```

	1	10	100	1000	1001	1002	1003	1004	1005	1006	...	990	991
1	1.000000	0.013110	0.025153	0.016308	0.011577	0.007299	0.010271	0.010509	0.006320	0.009129	...	0.011846	0.0
10	0.013110	1.000000	0.009444	0.012178	0.004680	0.005145	0.009484	0.008657	0.011202	0.007491	...	0.013249	0.0
100	0.025153	0.009444	1.000000	0.016322	0.059142	0.027863	0.014048	0.036151	0.007757	0.006688	...	0.042228	0.0
1000	0.016308	0.012178	0.016322	1.000000	0.004812	0.024725	0.011021	0.018253	0.011957	0.012848	...	0.034581	0.0
1001	0.011577	0.004680	0.059142	0.004812	1.000000	0.006460	0.007578	0.032768	0.004027	0.009007	...	0.009151	0.0
1002	0.007299	0.005145	0.027863	0.024725	0.006460	1.000000	0.006180	0.015325	0.006960	0.030447	...	0.015545	0.0
1003	0.010271	0.009484	0.014048	0.011021	0.007578	0.006180	1.000000	0.009379	0.007544	0.010232	...	0.027237	0.0
1004	0.010509	0.008657	0.036151	0.018253	0.032768	0.015325	0.009379	1.000000	0.013861	0.013198	...	0.013786	0.0
1005	0.006320	0.011202	0.007757	0.011957	0.004027	0.006960	0.007544	0.013861	1.000000	0.018479	...	0.011095	0.0
1006	0.009129	0.007491	0.006688	0.012848	0.009007	0.030447	0.010232	0.013198	0.018479	1.000000	...	0.006813	0.0
1007	0.012161	0.006834	0.031098	0.019809	0.016764	0.008200	0.005869	0.028042	0.008751	0.008400	...	0.011436	0.0
1008	0.005168	0.017228	0.004783	0.014488	0.003851	0.007814	0.025487	0.013173	0.012126	0.008498	...	0.015284	0.0
1009	0.006860	0.006283	0.002877	0.010189	0.008098	0.005978	0.008856	0.014339	0.005008	0.005503	...	0.007776	0.0
101	0.007217	0.006228	0.016035	0.013586	0.006717	0.004289	0.014910	0.016476	0.007636	0.007138	...	0.014681	0.0
1010	0.012828	0.011837	0.005898	0.011563	0.002839	0.007153	0.011027	0.013573	0.008332	0.008729	...	0.009084	0.0
1011	0.002369	0.007200	0.004407	0.014103	0.060245	0.012932	0.013007	0.011664	0.008578	0.006249	...	0.018062	0.0
1012	0.002352	0.005225	0.004948	0.003813	0.019987	0.005936	0.005744	0.010136	0.011901	0.005184	...	0.010874	0.0
1013	0.009021	0.008068	0.009987	0.013779	0.005803	0.008468	0.014385	0.013991	0.012237	0.006871	...	0.033674	0.0
1014	0.006049	0.003348	0.042446	0.008618	0.028980	0.018278	0.010611	0.016392	0.016902	0.017354	...	0.036515	0.0
1015	0.011783	0.028707	0.004563	0.024459	0.007988	0.012881	0.014013	0.018540	0.013141	0.022323	...	0.005712	0.0
1016	0.006527	0.011595	0.007311	0.012065	0.009269	0.017188	0.013721	0.016822	0.009899	0.023049	...	0.021691	0.0
1017	0.013588	0.004501	0.004908	0.012455	0.005210	0.014692	0.012983	0.008832	0.012551	0.005058	...	0.008093	0.0
1018	0.004493	0.005355	0.075642	0.008378	0.028305	0.027055	0.011851	0.022230	0.008660	0.006233	...	0.044089	0.0
1019	0.008922	0.001772	0.088560	0.008358	0.029135	0.023489	0.017657	0.021178	0.004580	0.012630	...	0.023838	0.0
102	0.010463	0.016268	0.008494	0.012740	0.017658	0.005935	0.008634	0.013455	0.005932	0.009994	...	0.009232	0.0
1020	0.004388	0.010141	0.007081	0.011539	0.010282	0.004343	0.007069	0.009985	0.007576	0.009854	...	0.019900	0.0
1021	0.005920	0.019551	0.007954	0.017934	0.021514	0.009532	0.013417	0.012348	0.007245	0.010113	...	0.017899	0.0
1022	0.008803	0.008500	0.015811	0.025692	0.010491	0.011157	0.027993	0.026839	0.012209	0.008641	...	0.035492	0.0
1023	0.009560	0.015559	0.029976	0.014922	0.013450	0.011879	0.014758	0.034183	0.016698	0.015088	...	0.013746	0.0
1024	0.006224	0.011056	0.013350	0.014525	0.006918	0.013057	0.007531	0.017613	0.010216	0.016489	...	0.015705	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
972	0.009681	0.007004	0.004076	0.013946	0.009610	0.006150	0.016548	0.011914	0.005906	0.004668	...	0.008714	0.0
973	0.010037	0.010688	0.030123	0.005557	0.045752	0.000975	0.001440	0.017814	0.005781	0.004993	...	0.020029	0.0
974	0.003963	0.004449	0.005801	0.007455	0.001955	0.006100	0.006789	0.006730	0.004720	0.007324	...	0.009394	0.0
975	0.001507	0.005981	0.005906	0.001234	0.003822	0.010084	0.003405	0.002593	0.002076	0.005428	...	0.010683	0.0
976	0.005268	0.007586	0.005233	0.013073	0.018016	0.005943	0.009229	0.014910	0.012079	0.009841	...	0.005081	0.0
977	0.002228	0.020208	0.003584	0.011665	0.007240	0.007243	0.009560	0.004291	0.008988	0.008407	...	0.013713	0.0
978	0.014239	0.013449	0.055582	0.017650	0.055002	0.063739	0.010727	0.113403	0.010396	0.022355	...	0.020644	0.0
979	0.011750	0.008395	0.009136	0.013261	0.002887	0.006216	0.010433	0.012739	0.006609	0.004368	...	0.010544	0.0
98	0.002561	0.009913	0.004767	0.010953	0.025717	0.005133	0.017264	0.003866	0.002850	0.003310	...	0.010271	0.0
980	0.006966	0.080107	0.021908	0.013319	0.006779	0.006613	0.004489	0.012278	0.014525	0.017829	...	0.015832	0.0
981	0.003511	0.008100	0.025254	0.009503	0.037620	0.013184	0.012637	0.009581	0.007887	0.017866	...	0.015634	0.0
982	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.006439	0.000000	0.002168	...	0.000000	0.0
983	0.008722	0.015061	0.013522	0.010571	0.006744	0.006470	0.008954	0.013496	0.007023	0.008841	...	0.009325	0.0
984	0.014441	0.014108	0.043739	0.021406	0.015499	0.010308	0.007174	0.045110	0.017182	0.010931	...	0.030611	0.0
985	0.011299	0.013814	0.028086	0.046124	0.007003	0.010765	0.010802	0.010696	0.010400	0.009144	...	0.018938	0.0
986	0.007509	0.018454	0.013864	0.015821	0.005327	0.008527	0.014062	0.037035	0.011097	0.007404	...	0.017779	0.0
987	0.009902	0.014218	0.005377	0.011105	0.004166	0.006638	0.010225	0.017329	0.008726	0.009962	...	0.007410	0.0
988	0.002860	0.011601	0.027264	0.004472	0.010228	0.009590	0.009133	0.016486	0.004346	0.004346	...	0.010503	0.0
989	0.022378	0.011796	0.039465	0.016096	0.010164	0.012404	0.024392	0.025727	0.010568	0.012238	...	0.024324	0.0
99	0.007429	0.005828	0.024114	0.006961	0.014845	0.014686	0.027022	0.012402	0.007944	0.010018	...	0.012748	0.0
990	0.011846	0.013249	0.042228	0.034581	0.009151	0.015545	0.027237	0.013786	0.011095	0.006813	...	1.000000	0.0
991	0.008216	0.017382	0.001339	0.014664	0.002705	0.013500	0.018090	0.027088	0.013903	0.011289	...	0.011686	1.0

992	0.011399	0.007149	0.009732	0.008955	0.014901	0.031731	0.007262	0.012796	0.015171	0.014773	...	0.020587	0.0
993	0.005237	0.007385	0.007993	0.009943	0.004565	0.008173	0.006958	0.010316	0.006398	0.010392	...	0.007312	0.0
994	0.007315	0.011362	0.013280	0.010786	0.016529	0.010223	0.008695	0.018577	0.006556	0.007992	...	0.008677	0.0
995	0.006427	0.004854	0.009319	0.008122	0.044348	0.007692	0.009016	0.009816	0.006096	0.007115	...	0.022572	0.0
996	0.008360	0.019540	0.021211	0.021269	0.009097	0.015265	0.010088	0.023062	0.010419	0.015426	...	0.012851	0.0
997	0.006734	0.013429	0.006464	0.020851	0.007988	0.008178	0.012814	0.009664	0.014563	0.007499	...	0.007790	0.0
998	0.002983	0.000961	0.023891	0.005010	0.014264	0.010762	0.011469	0.007165	0.004381	0.003727	...	0.022799	0.0
999	0.006313	0.004964	0.016957	0.009184	0.025655	0.013937	0.015597	0.012340	0.005944	0.018923	...	0.019821	0.0

4037 rows × 4037 columns

We can see that the cosine similarity matrix has 4037 rows and 4037 columns, and the elements represent the cosine similarity between each pair of movies. The diagonal elements of this matrix will be 1 since similarity score between the same move is always 1.

IV. Phase-4: Getting recommendations

Let us build the required functions to make movie recommendations, given that the user has liked a movie.

IV.I Functions

- **Get_Recommendations(Movie_ID,cos_sim_df)** This function will accept *Movie_ID*, and *cos_sim_df* as inputs. The *Movie_ID* is a number unique to a movie, and *cos_sim_df* is a pandas data frame containing the cosine similarity scores between all pairs of movies. This function will get top 6 Movies (based on the cosine similarity score between the input Movie_ID and other movies. Higher cosine similarity measure, better the match). The result is returned in the form of a dictionary.
- **Get_Available_Images()** This function will not accept any input. It returns the list of all movie IDs, for which we have an available image.
- **Display_Recommendations(Recommended_Movies_Dict,Movie_Map,Source_Movie_ID)** This function will accept 3 inputs. The *Recommended_Movies_Dict* is the dictionary of recommended movies (output of *Get_Recommendations(Movie_ID,cos_sim_df)* function). The *Movie_Map* is a data frame with the columns: "*Movie*" *(Movie name),*"*Movie_ID*" *(Unique ID),*"*URL*" *(Movie URL)*. This data frame is obtained by joining the *Movie_Details.csv* and *processed_data.csv* files data (using movie ID). This joining is needed, since it will help to map the movies which are successfully downloaded (4037 movies) and all the available movie names (4045 movies). The *Source_Movie_ID* is the movie ID, which is assumed to be liked by the user. The function does NOT return any value. It just renders the recommended movies along with the cosine similarity scores. The user can click on the movie to read visit the wikipedia site or hover on the image to get the text, which was used for building cosine similarity matrix.

The source code of these functions is given below:

In [9]:
```
#Get the mapping between available Movie plots and movie IDs
Movie_Map=pd.merge(URL[["Movie","Movie_ID","URL"]],df,how='inner',on=["Movie_ID"])[["Movie","Movie
_ID","Plot","URL"]]


def Get_Recommendations(Movie_ID,cos_sim_df):
    #Get the indices (movie IDs) with highest cosine sim scores
    recommended_idx=np.argpartition(np.array(cos_sim_df[Movie_ID].tolist()), -6)[-6:]

    #Convert to a list
    Recommended_Movie_IDs = cos_sim_df.columns[recommended_idx].tolist()

    #Prepare a dict and return the recommended movies list
    return dict(zip(Recommended_Movie_IDs,np.array(cos_sim_df[Movie_ID].tolist())
[recommended_idx]))


def Get_Available_Images():
    #Get all the available image names (movie IDs which have images)
    image_files = os.listdir("./images")

    #Make sure that we are dealing with movie data files only
    image_files = [i for i in image_files if re.search('[1-9]*\.jpg',i)]

    #Define a list to collect the movie IDs
    y = list()
    for i in image_files:
        y.append(int(i.split(".")[0]))
    #Return the list
    return y


def Display_Recommendations(Recommended_Movies_Dict,Movie_Map,Source_Movie_ID):
    #The following statement will make sure that we sort the movies in the descending order of sim
ilarity
    Recommended_Movies = pd.DataFrame(sorted(Recommended_Movies_Dict.items(), key=lambda x: -
x[1]))[0].tolist()

    #Delete the liked movie from the list (since cosine sim with itself is 1)
    Recommended_Movies = Recommended_Movies[1:]

    Recommended_Movies_Plot = dict()
    Recommended_Movies_URL = dict()

    for i in Recommended_Movies:
        Recommended_Movies_Plot[i] = Movie_Map[Movie_Map["Movie_ID"] == i]["Plot"].tolist()[0]
        Recommended_Movies_URL[i] = Movie_Map[Movie_Map["Movie_ID"] == i]["URL"].tolist()[0]

    #Get the available movies with images
    Available_Images_List = Get_Available_Images()

    Source_Movie_Name = Movie_Map[Movie_Map["Movie_ID"] == Source_Movie_ID]["Movie"].tolist()[0]
```

```
    Source_Plot = Movie_Map[Movie_Map["Movie_ID"] == Source_Movie_ID]["Plot"].tolist()[0]
    Source_URL = Movie_Map[Movie_Map["Movie_ID"] == Source_Movie_ID]["URL"].tolist()[0]
    print("Assuming that the user liked {}:".format(Source_Movie_Name))

    #Prepare HTML for display:
    if Source_Movie_ID in Available_Images_List:
        display(HTML("<table><tr><td><a href='"+str(Source_URL)+\
                     "' target='_blank'><img src='./images/"+str(Source_Movie_ID)+".jpg'
title='"+\
                     str(Source_Plot)+"'></a></td></tr></table>" \
            ))

    display_html = ""
    display_values = ""
    for i in Recommended_Movies:
        if i in Available_Images_List:
            display_html = display_html + "<td><a href='"+str(Recommended_Movies_URL[i])+\
            "' target='_blank'><img src='./images/"+str(i)+".jpg' title='"+\
            str(Recommended_Movies_Plot[i])+"'></a></td>"
            display_values = display_values + "<td> Similarity:"+\
            str(Recommended_Movies_Dict[i])+"</td>"
    print("The following movies are recommended:")
    display(HTML("<table><tr>"+display_html+"</tr><tr>"+display_values+"</tr></table>" \
            ))
```

IV.II Demonstration of the system

We will get recommended movies given that the user has liked some movies. The cosine similarity measure is also displayed, along with the movie recommendations. The recommended movies are sorted in the descending order of similarity score. Also the top 5 movies are displayed. At some places you may find less than 5 movies, since we avoided the display of the movie, if an associated image is not available (as the web robot did not download the picture due to unavailability or some other reason). Also if you hover over the image, you will see the text (cleansed) used for building the recommender, and if you click the image, you will redirected to the Wikipedia URL:

In [11]:
```
Recommended_Movies = Get_Recommendations(3974,cos_sim_df)
Recommended_Movies
Display_Recommendations(Recommended_Movies,Movie_Map,3974)
```

```
Assuming that the user liked X-Men: Apocalypse:
```

(https://en.wikipedia.org/wiki/X-Men:_Apocalypse)

```
The following movies are recommended:
```

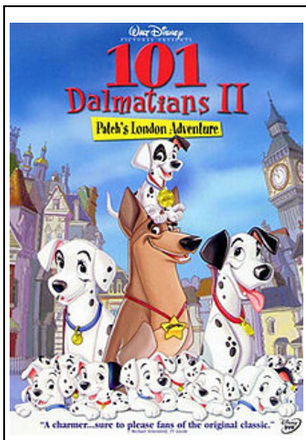
 (https://en.wikipedia.org/wiki/X2_(film))	 (https://en.wikipedia.org/wiki/X-Men:_First_Class)	 (https://en.wikipedia.org/wiki/X-Men_(film))	 (https://en.wikipedia.org/w
Similarity:0.268575850171	Similarity:0.238367317257	Similarity:0.229481952653	Similarity:0.221129217079

In [33]:
```
Recommended_Movies = Get_Recommendations(1,cos_sim_df)
Display_Recommendations(Recommended_Movies,Movie_Map,1)
```

```
Assuming that the user liked  102 Dalmatians:
```

(https://en.wikipedia.org/wiki/102_Dalmatians)

```
The following movies are recommended:
```

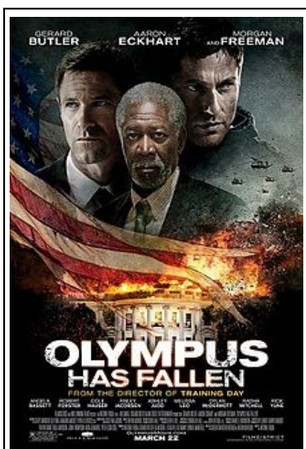
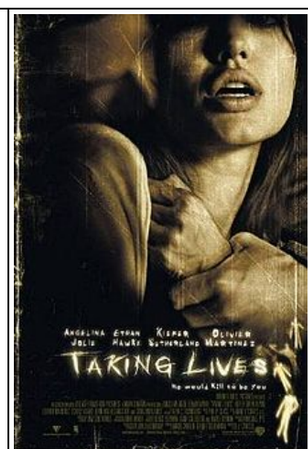
 (https://en.wikipedia.org/wiki/101_Dalmatians_II:_Patch%27s_London_Adventure)	 (https://en.wikipedia.org/wiki/Cold_Comes_the_N
Similarity:0.306608854172	Similarity:0.185453340218

```
In [34]: Recommended_Movies = Get_Recommendations(3934,cos_sim_df)
         Display_Recommendations(Recommended_Movies,Movie_Map,3934)
```

```
Assuming that the user liked London Has Fallen:
```

(https://en.wikipedia.org/wiki/London_Has_Fallen)

```
The following movies are recommended:
```

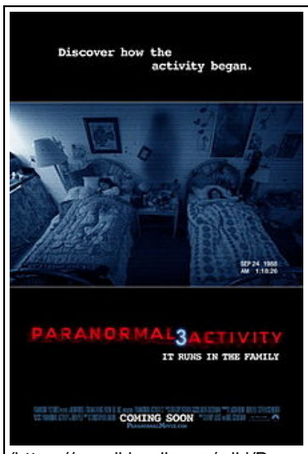
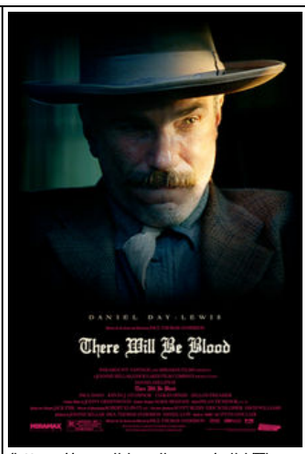
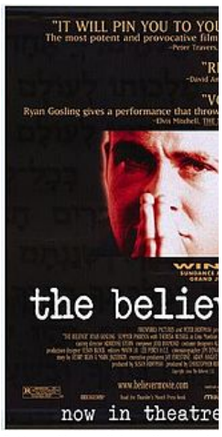
 (https://en.wikipedia.org/wiki/Olympus_Has_Fallen)	 (https://en.wikipedia.org/wiki/Taking_Lives_(film))	 (https://en.wikipedia.org/wiki/
Similarity:0.543469898779	Similarity:0.301055015997	Similarity:0.175642267065

```
In [35]: Recommended_Movies = Get_Recommendations(2635,cos_sim_df)
         Display_Recommendations(Recommended_Movies,Movie_Map,2635)
```

```
Assuming that the user liked Paranormal Activity 2:
```

(https://en.wikipedia.org/wiki/Paranormal_Activity_2)

```
The following movies are recommended:
```

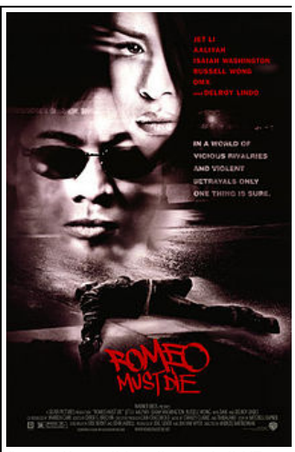
 (https://en.wikipedia.org/wiki/Paranormal_Activity_3)	 (https://en.wikipedia.org/wiki/There_Will_Be_Blood)	 (https://en.wikipedia.org/w
Similarity:0.526099643661	Similarity:0.242463753141	Similarity:0.234107937331

In [36]:
```
Recommended_Movies = Get_Recommendations(2810,cos_sim_df)
Display_Recommendations(Recommended_Movies,Movie_Map,2810)
```

```
Assuming that the user liked Kung Fu Panda 2:
```

(https://en.wikipedia.org/wiki/Kung_Fu_Panda_2)

```
The following movies are recommended:
```

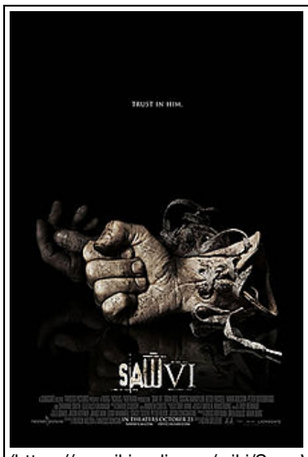
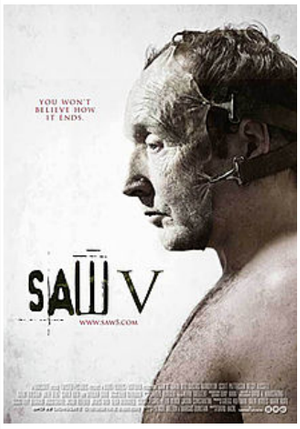
 (https://en.wikipedia.org/wiki/Kung_Fu_Panda_3)	 (https://en.wikipedia.org/wiki/Kung_Fu_Panda)	 (https://en.wikipedia.org/wiki/Rom
Similarity:0.477916709463	Similarity:0.472015971221	Similarity:0.234033843151

In [37]:
```
Recommended_Movies = Get_Recommendations(2656,cos_sim_df)
Display_Recommendations(Recommended_Movies,Movie_Map,2656)
```

```
Assuming that the user liked Saw VII:
```

(https://en.wikipedia.org/wiki/Saw_VII)

```
The following movies are recommended:
```

 (https://en.wikipedia.org/wiki/Saw_VI)	 (https://en.wikipedia.org/wiki/Saw_V)	 (https://en.wikipedia.org/wiki/Saw_IV)	 (https://en.wikip
Similarity:0.299457229304	Similarity:0.263637194848	Similarity:0.214968641642	Similarity:0.1988

```
In [38]: Recommended_Movies = Get_Recommendations(3176,cos_sim_df)
         Display_Recommendations(Recommended_Movies,Movie_Map,3176)
```

```
Assuming that the user liked Titanic 3D:
```

(https://en.wikipedia.org/wiki/Titanic_(1997_film))

```
The following movies are recommended:
```

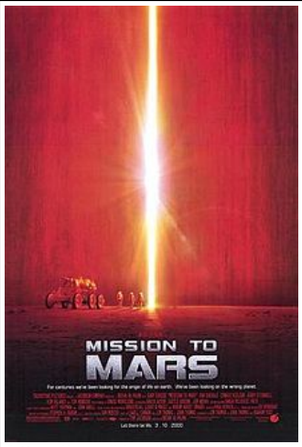
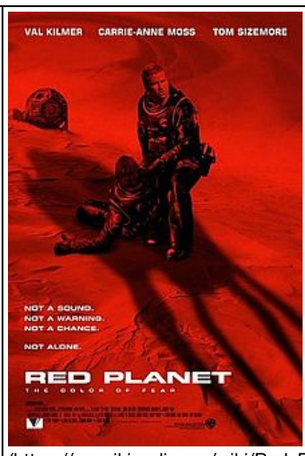
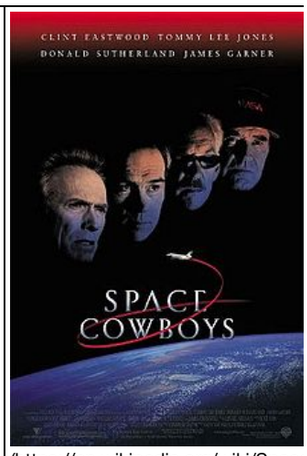
 (https://en.wikipedia.org/wiki/The_Ballad_of_Jack_and_Rose)	 (https://en.wikipedia.org/wiki/The_Greatest_(2009_film))	 (https://en.wi
Similarity:0.292896298323	Similarity:0.171477544925	Similarity:0.1

```
In [39]: Recommended_Movies = Get_Recommendations(3893,cos_sim_df)
         Display_Recommendations(Recommended_Movies,Movie_Map,3893)
```

```
Assuming that the user liked The Martian:
```

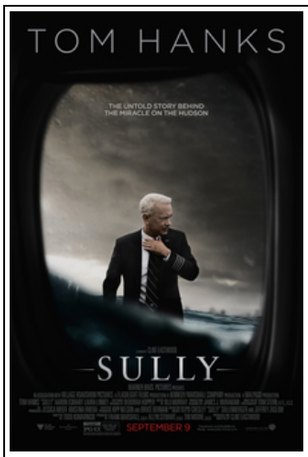
(https://en.wikipedia.org/wiki/The_Martian_(film))

The following movies are recommended:

 (https://en.wikipedia.org/wiki/Mission_to_Mars)	 (https://en.wikipedia.org/wiki/Red_Planet_(film))	 (https://en.wikipedia.org/wiki/Space
Similarity:0.175169934667	Similarity:0.152099978063	Similarity:0.0842259518701

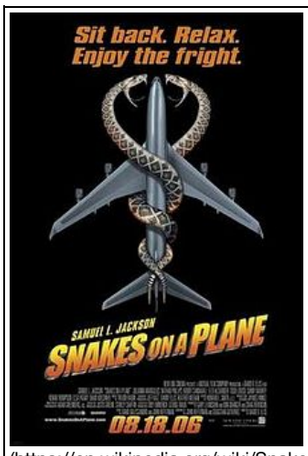
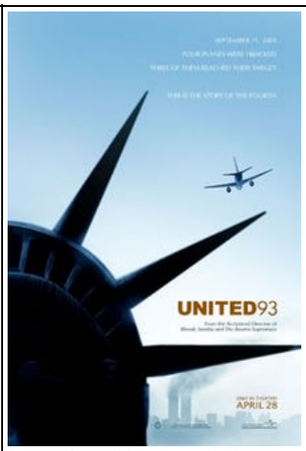
In [40]:
```
Recommended_Movies = Get_Recommendations(4015,cos_sim_df)
Display_Recommendations(Recommended_Movies,Movie_Map,4015)
```

Assuming that the user liked Sully:



(https://en.wikipedia.org/wiki/Sully_(film))

The following movies are recommended:

 (https://en.wikipedia.org/wiki/Snakes_on_a_Plane)	 (https://en.wikipedia.org/wiki/United_93_(film))	 (https://en.wikipedia.org/wiki/Sou
Similarity:0.104853207218	Similarity:0.0937654970696	Similarity:0.091478167585

In [41]:
```
Recommended_Movies = Get_Recommendations(3077,cos_sim_df)
Display_Recommendations(Recommended_Movies,Movie_Map,3077)
```

Assuming that the user liked The Hunger Games:

(https://en.wikipedia.org/wiki/The_Hunger_Games_(film))

```
The following movies are recommended:
```

 (https://en.wikipedia.org/wiki/The_Hunger_Games:_Catching_Fire)	 (https://en.wikipedia.org/wiki/The_Hunger_Games:_Mockingjay_9
Similarity:0.756288369602	Similarity:0.626990711657

In [42]:
```
Recommended_Movies = Get_Recommendations(616,cos_sim_df)
Display_Recommendations(Recommended_Movies,Movie_Map,616)
```

```
Assuming that the user liked Spider-Man:
```

(https://en.wikipedia.org/wiki/Spider-Man_(2002_film))

```
The following movies are recommended:
```

 (https://en.wikipedia.org/wiki/The_Amazing_Spider-Man_2)	 (https://en.wikipedia.org/wiki/Spider-Man_3)	 (https://en.wikipedia.org/wiki/Spider-Man_2)	 (htt Ma
Similarity:0.532694697879	Similarity:0.515483578747	Similarity:0.45961557517	Sin

In [11]:
```
Recommended_Movies = Get_Recommendations(2708,cos_sim_df)
Recommended_Movies
```

```
Display_Recommendations(Recommended_Movies,Movie_Map,2708)
```

```
Assuming that the user liked The Adventures of Tintin: The Secret of the Unicorn:
```

(https://en.wikipedia.org/wiki/The_Adventures_of_Tintin:_The_Secret_of_the_Unicorn)

```
The following movies are recommended:
```

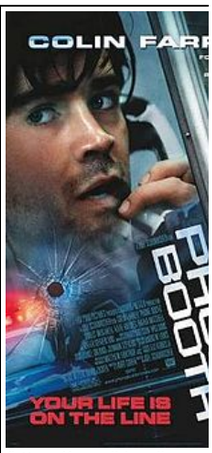
 (https://en.wikipedia.org/wiki/Meet_Dave)	 (https://en.wikipedia.org/wiki/Treasure_Planet)	 (https://en.wikipedia.org/wiki/Your_Highnes
Similarity:0.0602222780375	Similarity:0.0600969469384	Similarity:0.0597959791667

In [15]:
```
Recommended_Movies = Get_Recommendations(2354,cos_sim_df)
Recommended_Movies
Display_Recommendations(Recommended_Movies,Movie_Map,2354)
```

```
Assuming that the user liked The Hangover:
```

(https://en.wikipedia.org/wiki/The_Hangover)

```
The following movies are recommended:
```

 (https://en.wikipedia.org/wiki/The_Hangover:_Part_II)	 (https://en.wikipedia.org/wiki/The_Hangover_Part_III)	 (https://en.wikipedia.org/
Similarity:0.573032072419	Similarity:0.553884362519	Similarity:0.3368367249

Similarity:0.573032072419	Similarity:0.553884363519	Similarity:0.3368387249

In [17]:
```
Recommended_Movies = Get_Recommendations(3166,cos_sim_df)
Recommended_Movies
Display_Recommendations(Recommended_Movies,Movie_Map,3166)
```

Assuming that the user liked Taken 2: