A Movie-Based Content Based Filtering

This project aimed to create two types of recommenders. One being a content-based filter and the other being user-user (or collaborative) filters. Collaborative filtering is when a recommendation is made based upon what other people liked whose tastes are similar to your own. Ex. If many people who rated 'Rambo' highly also rated 'The Expendables' highly, recommending to you "The expandables" based on your high rating of 'Rambo'. Content-based filter is based upon recommending movies based of the movies properties which are similar to past content you've liked. Ex. Recommending highly rated 1980s sci-fi films 'Aliens' or 'The Fly' to a retro sci-fi film fanatic.

The source used for this project was as listed below. These researchers in 2015 gathered 22884377 ratings and 586994 tag applications across 34208 movies created by 247753 users between January 09, 1995 and January 29, 2016 from the movie website MovieLens.org. Users were selected at random for inclusion from a pool of users who had rated at least 1 movie.

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

Table of contents

Acquiring the Data

The source for the data can be acquired from IBM Cloud link below and extracted into the same directory as this Jupyter Notebook to be executed. If using a Linux shell, consider adding !wget and !zip before the url/filename to extract the data. Data can also be manually extracted with any zipping archive program on Windows through file explorer.

In [2]:

```
print('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDevelop
```

```
https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkill
sNetwork-ML0101EN-SkillsNetwork/labs/Module%205/data/moviedataset.zip
```

Preprocessing

Some proprocessing tasks are necessary to perform as part of this project. Including importing necessary packages, loading the data into a dataframe and dropping columns not needed for this analysis.

In [3]:

```
#Dataframe manipulation library
import pandas as pd
#Math functions, we'll only need the sqrt function so let's import only that
from math import sqrt
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

This process requires up to 1GB of Ram as the ratings-file extracted is over 600MB and movies file over 100MB plus overhead. Loading into memory from a solid state disk will take about 5-10 seconds. On a mechanical Hard Drive, there may be 1-2 minute load time.

In [4]:

```
#Storing the movie information into a pandas dataframe
movies_df = pd.read_csv('movies.csv')
#Storing the user information into a pandas dataframe
ratings_df = pd.read_csv('ratings.csv')
#Head is a function that gets the first N rows of a dataframe. N's default is 5.
movies_df.head()
```

Out[4]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

The Transform pipeline below removes excess parenthesis and year tags from titles and creates a new colum called 'Year' to store the year-data. Ex. The 'Blob' (1996) would become The Blob as would The Blob (2010). However, both seperate entries will have an entry of Year colum of 1996 and 2010 respectively.

In [5]:

```
#Using regular expressions to find a year stored between parentheses
#We specify the parantheses so we don't conflict with movies that have years in thei
movies_df['year'] = movies_df.title.str.extract('(\(\d\d\d\d\))',expand=False)
#Removing the parentheses
movies_df['year'] = movies_df.year.str.extract('(\d\d\d\d)',expand=False)
#Removing the years from the 'title' column
movies_df['title'] = movies_df.title.str.replace('(\(\d\d\d\d\))', '')
#Applying the strip function to get rid of any ending whitespace characters that may
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
movies_df.head()
```

```
C:\Users\WBurc\AppData\Local\Temp\ipykernel_13228\1143695627.py:7: FutureWarning: Th
e default value of regex will change from True to False in a future version.
  movies_df['title'] = movies_df.title.str.replace('(\(\d\d\d\d\))', '')
```

Out[5]:

	movieId	title	genres	year
0	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1995
1	2	Jumanji	Adventure|Children|Fantasy	1995
2	3	Grumpier Old Men	Comedy|Romance	1995
3	4	Waiting to Exhale	Comedy|Drama|Romance	1995
4	5	Father of the Bride Part II	Comedy	1995

The **Genres** column can contain multiple genres seperated by a |. This next transform process seperates a string containing multiple genres deliminated by '|' into a **list of Genres** to simplify for future use. This can be achieved by applying Python's split string function on the correct column.

In [6]:

```
#Every genre is separated by a | so we simply have to call the split function on |
movies_df['genres'] = movies_df.genres.str.split('|')
movies_df.head()
```

Out[6]:

	movieId	title	genres	year
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995
1	2	Jumanji	[Adventure, Children, Fantasy]	1995
2	3	Grumpier Old Men	[Comedy, Romance]	1995
3	4	Waiting to Exhale	[Comedy, Drama, Romance]	1995
4	5	Father of the Bride Part II	[Comedy]	1995

Data Science often prefers "numbers" for encoding/mathematical operations over characters or strings. A process known as One-Hot Encoding including can be used to replace each unique genre with a corresponding binary vallue for if a genre is present or not present. So, an Action Adventure movie, might be [1,1,0,0,0,0,0]. Assuming the first-column represents "Action", the second column represents "Adventure" and the third column represents Comedy "3". So an action-comedy would be [1,0,1,0,0,0,0] and so-forth. There are likely more than 8 genres so the actual-array probably [much,much,larger] in column count.

This converts our data into an optimal format the content-based recommendation system technique. This encoding is needed for feeding categorical data. In this case, we store every different genre in columns that contain either 1 or 0. 1 shows that a movie has that genre and 0 shows that it doesn't.

In [7]:

```
#Copying the movie dataframe into a new one since we won't need to use the genre inf
moviesWithGenres_df = movies_df.copy()
```

```
#For every row in the dataframe, iterate through the list of genres and place a 1 in
for index, row in movies_df.iterrows():
    for genre in row['genres']:
        moviesWithGenres_df.at[index, genre] = 1
#Filling in the NaN values with 0 to show that a movie doesn't have that column's ge
moviesWithGenres_df = moviesWithGenres_df.fillna(0)
moviesWithGenres_df.head()
```

Out[7]:

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy	Romai
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995	1.0	1.0	1.0	1.0	1.0	
1	2	Jumanji	[Adventure, Children, Fantasy]	1995	1.0	0.0	1.0	0.0	1.0	
2	3	Grumpier Old Men	[Comedy, Romance]	1995	0.0	0.0	0.0	1.0	0.0	
3	4	Waiting to Exhale	[Comedy, Drama, Romance]	1995	0.0	0.0	0.0	1.0	0.0	
4	5	Father of the Bride Part II	[Comedy]	1995	0.0	0.0	0.0	1.0	0.0	

5 rows × 24 columns

Next, examining the ratings dataframe columns via running the head function, we can see some data isn't necessary to a recommender system. The actual "timestamp" or "when" a rating was made is not important.

In [8]:

```
ratings_df.head()
```

Out[8]:

	userId	movieId	rating	timestamp
0	1	169	2.5	1204927694
1	1	2471	3.0	1204927438
2	1	48516	5.0	1204927435
3	2	2571	3.5	1436165433
4	2	109487	4.0	1436165496

We won't be needing the timestamp column, so we wil drop it to safe memory and storage space if we were to output our "cleaned and transformed" dataset.

In [9]:

```
#Drop removes a specified row or column from a dataframe
ratings_df = ratings_df.drop('timestamp', 1)
ratings_df.head()
```

```
C:\Users\WBurc\AppData\Local\Temp\ipykernel_13228\3391429438.py:2: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.
  ratings_df = ratings_df.drop('timestamp', 1)
```

Out[9]:

	userId	movieId	rating
0	1	169	2.5
1	1	2471	3.0
2	1	48516	5.0
3	2	2571	3.5
4	2	109487	4.0

Content-Based recommendation system

As mentioned earlier, content-based filters examines "content" you rated higly and attempts to match it to similar content. This technique attempts to figure out what a user's favourite aspects of an item is, and then make recommendations items that present those aspects. In our case, we're going to try to figure out the input's favorite genres from the movies and ratings given. Intutively, humans can figure this out quickly and our local "video store rental dude or dudette" used to do this in the 80s. They'd ask, what movies do you like? and if you said 'Rambo!' they might recommend 'Oh, you're going to love this Arnold movie called Commando'. However, this is now done by content-based reocmmendation systems on Netflix and performed by an algorithm.

To test the system, I've created a sample user input of someone who rates sci-fi/action films highly and children's movies so-so. We'll test the recommendation this system gives to this user:

In [10]:

```
userInput = [
            {'title':'Breakfast Club, The', 'rating':3.5},
            {'title':'Toy Story', 'rating':2.5},
            {'title':'Jumanji', 'rating':2},
            {'title':"Pulp Fiction", 'rating':5},
            {'title':'Akira', 'rating':4.5},
            {'title':'Matrix, The', 'rating':5},
            {'title':'Predator','rating':5},
            {'title':'Commando','rating':5}
         ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

Out[10]:

	title	rating
0	Breakfast Club, The	3.5
1	Toy Story	2.5
2	Jumanji	2.0
3	Pulp Fiction	5.0
4	Akira	4.5
5	Matrix, The	5.0
6	Predator	5.0
7	Commando	5.0

Add movieId to input user

With the input complete, we want to get the movie's ID using the title. This data is contained within movies dataframe.

We can achieve this by first filtering out the rows that contain the input movie's title and then merging this subset with the input dataframe. We also drop unnecessary columns before we merge the movie titles into the inputMovies dataframe to save memory space.

In [11]:

```
#Filtering out the movies by title
inputId = movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]
#Then merging it so we can get the movieId. It's implicitly merging it by title.
inputMovies = pd.merge(inputId, inputMovies)
#Dropping information we won't use from the input dataframe
inputMovies = inputMovies.drop('genres', 1).drop('year', 1)
#Final input dataframe
#If a movie you added in above isn't here, then it might not be in the original
#dataframe or it might spelled differently, please check capitalisation.
inputMovies
```

```
C:\Users\WBurc\AppData\Local\Temp\ipykernel_13228\2071048360.py:6: FutureWarning: In
a future version of pandas all arguments of DataFrame.drop except for the argument
'labels' will be keyword-only.
  inputMovies = inputMovies.drop('genres', 1).drop('year', 1)
C:\Users\WBurc\AppData\Local\Temp\ipykernel_13228\2071048360.py:6: FutureWarning: In
a future version of pandas all arguments of DataFrame.drop except for the argument
'labels' will be keyword-only.
  inputMovies = inputMovies.drop('genres', 1).drop('year', 1)
```

Out[11]:

	movieId	title	rating
0	1	Toy Story	2.5
1	2	Jumanji	2.0
2	296	Pulp Fiction	5.0
3	1274	Akira	4.5
4	1968	Breakfast Club, The	3.5

	movieId	title	rating
5	2571	Matrix, The	5.0
6	3527	Predator	5.0
7	6664	Commando	5.0

Next, we are going to grab out one-hot encoding (binary) values from our earlier data transformation process to find out which genres are present in the films the user inputted.

From the output, we can see 'The Breakfast Club', 'Pulp Fiction' and 'The Matrix' are not considered Adventure films while 'Jumanji', 'Toy Story' and 'Akira' are.

In [12]:

```
#Filtering out the movies from the input
userMovies = moviesWithGenres_df[moviesWithGenres_df['movieId'].isin(inputMovies['mc
userMovies
```

Out[12]:

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995	1.0	1.0	1.0	1.0	1.0
1	2	Jumanji	[Adventure, Children, Fantasy]	1995	1.0	0.0	1.0	0.0	1.0
293	296	Pulp Fiction	[Comedy, Crime, Drama, Thriller]	1994	0.0	0.0	0.0	1.0	0.0
1246	1274	Akira	[Action, Adventure, Animation, Sci-Fi]	1988	1.0	1.0	0.0	0.0	0.0
1885	1968	Breakfast Club, The	[Comedy, Drama]	1985	0.0	0.0	0.0	1.0	0.0
2487	2571	Matrix, The	[Action, Sci-Fi, Thriller]	1999	0.0	0.0	0.0	0.0	0.0
3438	3527	Predator	[Action, Sci-Fi, Thriller]	1987	0.0	0.0	0.0	0.0	0.0
6555	6664	Commando	[Action, Adventure]	1985	1.0	0.0	0.0	0.0	0.0

8 rows × 24 columns

The one-hot encoded genre-table is the most crucil part so we'll drop the unnecessary columns and then reset the index to it is from 0 to n.

In [13]:

```
#Resetting the index to avoid future issues
userMovies = userMovies.reset_index(drop=True)
#Dropping unnecessary issues due to save memory and to avoid issues
userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1).dr
userGenreTable
```

```
C:\Users\WBurc\AppData\Local\Temp\ipykernel_13228\2641803640.py:4: FutureWarning: In
a future version of pandas all arguments of DataFrame.drop except for the argument
'labels' will be keyword-only.
  userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1).
drop('year', 1)
C:\Users\WBurc\AppData\Local\Temp\ipykernel_13228\2641803640.py:4: FutureWarning: In
a future version of pandas all arguments of DataFrame.drop except for the argument
'labels' will be keyword-only.
  userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1).
drop('year', 1)
C:\Users\WBurc\AppData\Local\Temp\ipykernel_13228\2641803640.py:4: FutureWarning: In
a future version of pandas all arguments of DataFrame.drop except for the argument
'labels' will be keyword-only.
  userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1).
drop('year', 1)
C:\Users\WBurc\AppData\Local\Temp\ipykernel_13228\2641803640.py:4: FutureWarning: In
a future version of pandas all arguments of DataFrame.drop except for the argument
'labels' will be keyword-only.
  userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1).
drop('year', 1)
```

Out[13]:

	Adventure	**Animation**	**Children**	**Comedy**	**Fantasy**	**Romance**	**Drama**	**Action**	**Crime**	**Thriller**
0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
1	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0
3	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
7	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0

Each genre is next going to be turned into weights by multiplying the inputted user rating for our "customer" and multiplying it by the genre's binary value. Next, we'll be summing up the resulting table by column. This operation is called the dot product of a matrix and a vector. Pandas dot function can be used or a custom for loop could be written. Below, I've printed the ratings for each film as a reminder.

In [14]:

```
inputMovies['rating']
```

Out[14]:

```
0    2.5
1    2.0
2    5.0
3    4.5
4    3.5
5    5.0
6    5.0
7    5.0
Name: rating, dtype: float64
```

In [15]:

```
#Dot produt to get weights
userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
#The user profile
userProfile
```

Out[15]:

```
Adventure             14.0
Animation              7.0
Children               4.5
Comedy                11.0
Fantasy                4.5
Romance                0.0
Drama                  8.5
Action                19.5
Crime                  5.0
Thriller              15.0
Horror                 0.0
Mystery                0.0
Sci-Fi                14.5
IMAX                   0.0
Documentary            0.0
War                    0.0
Musical                0.0
Western                0.0
Film-Noir              0.0
(no genres listed)     0.0
dtype: float64
```

We can see that films with comedy, tended to be more watched (or higher rated) as the value is 13.5. We can presume our hypothetical users likes lean towards "Action, Crime, Sci-Fi, Thriller, Drama", and lean away from "musicals" and "Film-noir" as she or he's never watched or rated these types of films. Now, we have the weights for every of the user's preferences. This is known as the User Profile. We can multiply these weights against all the movies in our movie database and sort. We just need to make our genre table with a bit of cleaning first. Using this, we can recommend movies that satisfy the user's preferences.

Let's start by extracting the genre table from the original dataframe. Please note I'm using head(10) to show ten entries to make the list longer and distinguish it from user's inputted/ranked movies of our hypothetical "customer". Using shape, we can see there are actually 34,208 movies.

In [16]:

```
#Now let's get the genres of every movie in our original dataframe
```

```
genreTable = moviesWithGenres_df.set_index(moviesWithGenres_df['movieId'])
#And drop the unnecessary information
genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop('
genreTable.head(10)
```

```
C:\Users\WBurc\AppData\Local\Temp\ipykernel_13228\1648705702.py:4: FutureWarning: In
a future version of pandas all arguments of DataFrame.drop except for the argument
'labels' will be keyword-only.
  genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop
('year', 1)
C:\Users\WBurc\AppData\Local\Temp\ipykernel_13228\1648705702.py:4: FutureWarning: In
a future version of pandas all arguments of DataFrame.drop except for the argument
'labels' will be keyword-only.
  genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop
('year', 1)
C:\Users\WBurc\AppData\Local\Temp\ipykernel_13228\1648705702.py:4: FutureWarning: In
a future version of pandas all arguments of DataFrame.drop except for the argument
'labels' will be keyword-only.
  genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop
('year', 1)
C:\Users\WBurc\AppData\Local\Temp\ipykernel_13228\1648705702.py:4: FutureWarning: In
a future version of pandas all arguments of DataFrame.drop except for the argument
'labels' will be keyword-only.
  genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop
('year', 1)
```

Out[16]:

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thr
movieId										
1	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	
2	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	
7	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	
8	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
10	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	

In [17]:

```
genreTable.shape
```

Out[17]:

```
(34208, 20)
```

Now, we have our hypothetical customer's input profile and our one-hot encoding (binary) genre information for our 32,000 movies. So we are going to take the weighted average of every movie based on the input profile multiplied by the genre table. We will then sort the result so the highest-weights are at the top (most higly recommended) and print the top twenty movies that most satisfy it.

In [18]:
```
#Multiply the genres by the weights and then take the weighted average
recommendationTable_df = ((genreTable*userProfile).sum(axis=1))/(userProfile.sum())
recommendationTable_df.head(10)
```

Out[18]:
```
movieId
1     0.396135
2     0.222222
3     0.106280
4     0.188406
5     0.106280
6     0.381643
7     0.106280
8     0.178744
9     0.188406
10    0.468599
dtype: float64
```

In [19]:
```
#Sort our recommendations in descending order
recommendationTable_df = recommendationTable_df.sort_values(ascending=False)
#Just a peek at the values
recommendationTable_df.head()
```

Out[19]:
```
movieId
115479    0.739130
71999     0.734300
116758    0.714976
27032     0.714976
122787    0.705314
dtype: float64
```

Now here's the recommendation table! What do you think, will our "Action, Crime, Sci-Fi, Thriller, Drama" fan enjoy these films?

In [20]:
```
#The final recommendation table
movies_df.loc[movies_df['movieId'].isin(recommendationTable_df.head(20).keys())]
```

Out[20]:

	movieId	title	genres	year
4923	5018	Motorama	[Adventure, Comedy, Crime, Drama, Fantasy, Mys...	1991
7763	8361	Day After Tomorrow, The	[Action, Adventure, Drama, Sci-Fi, Thriller]	2004
9180	27032	Who Am I? (Wo shi shei)	[Action, Adventure, Comedy, Sci-Fi, Thriller]	1998
9403	27618	Sound of Thunder, A	[Action, Adventure, Drama, Sci-Fi, Thriller]	2005
10575	40339	Chicken Little	[Action, Adventure, Animation, Children, Comed...	2005
11410	48774	Children of Men	[Action, Adventure, Drama, Sci-Fi, Thriller]	2006

	movieId	title	genres	year
11785	52287	Meet the Robinsons	[Action, Adventure, Animation, Children, Comed...	2007
12464	58025	Jumper	[Action, Adventure, Drama, Sci-Fi, Thriller]	2008
13109	62956	Futurama: Bender's Game	[Action, Adventure, Animation, Comedy, Fantasy...	2008
13250	64645	The Wrecking Crew	[Action, Adventure, Comedy, Crime, Drama, Thri...	1968
14397	71999	Aelita: The Queen of Mars (Aelita)	[Action, Adventure, Drama, Fantasy, Romance, S...	1924
16055	81132	Rubber	[Action, Adventure, Comedy, Crime, Drama, Film...	2010
16884	85261	Mars Needs Moms	[Action, Adventure, Animation, Children, Comed...	2011
18347	91500	Hunger Games, The	[Action, Adventure, Drama, Sci-Fi, Thriller]	2012
22145	106240	Free Birds	[Action, Adventure, Animation, Children, Comed...	2013
24565	115479	Whip Hand, The	[Action, Adventure, Crime, Drama, Sci-Fi, Thri...	1951
24946	116758	Death Racers	[Action, Adventure, Comedy, Sci-Fi, Thriller]	2008
25218	117646	Dragonheart 2: A New Beginning	[Action, Adventure, Comedy, Drama, Fantasy, Th...	2000
26442	122787	The 39 Steps	[Action, Adventure, Comedy, Crime, Drama, Thri...	1959
31427	141385	Humanity's End	[Action, Adventure, Drama, Sci-Fi, Thriller]	2009

Now just for fun, let's sort ascending order for the LEAST recommended films. Do you think our "Action, Crime, Sci-Fi, Thriller, Drama" will dislike these films?

In [22]:

```
#Sort our recommendations in descending order
recommendationTable_df = recommendationTable_df.sort_values(ascending=True)
#The final "do not watch" anti-recommendation table
movies_df.loc[movies_df['movieId'].isin(recommendationTable_df.keys())]
```

Out[22]:

	movieId	title	genres	year
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995
1	2	Jumanji	[Adventure, Children, Fantasy]	1995
2	3	Grumpier Old Men	[Comedy, Romance]	1995
3	4	Waiting to Exhale	[Comedy, Drama, Romance]	1995
4	5	Father of the Bride Part II	[Comedy]	1995
...	...	...	...	...
34203	151697	Grand Slam	[Thriller]	1967

	movieId	title	genres	year
34204	151701	Bloodmoney	[(no genres listed)]	2010
34205	151703	The Butterfly Circus	[Drama]	2009
34206	151709	Zero	[Drama, Sci-Fi]	2015
34207	151711	The 2000 Year Old Man	[(no genres listed)]	1975

34208 rows × 4 columns

I have some family members who refuse to watch old-movies or movies with subtitles. So how about we improve this recommender output for them by doing two more stages?

In [31]:

```
#Sort our recommendations in descending order
recommendationTable_df = recommendationTable_df.sort_values(ascending=False)

first_recommendation_stage_df = movies_df.loc[movies_df['movieId'].isin(recommendati

#Let's drop years < 1998. This date was chosen at random, you could update it easily
minYearToRecommend = 1998
first_recommendation_stage_df['year'] = first_recommendation_stage_df['year'].apply(
first_recommendation_stage_df.drop(first_recommendation_stage_df[first_recommendatic
#Let's drop the foreign films. Note each foreign film tends to have a '(alternative 
#The str.find() function will return -1 when a string is not-found. So we are negati
#We will now drop these indexes from our recommender system.
first_recommendation_stage_df.drop(first_recommendation_stage_df[first_recommendatic
#One can also achieve this with an alternative algorithm to capture, foreign titled 
#The downside is while it is more precise than removing all ( possibly, it would req
first_recommendation_stage_df.drop(first_recommendation_stage_df[first_recommendatic
first_recommendation_stage_df.drop(first_recommendation_stage_df[first_recommendatic
first_recommendation_stage_df.drop(first_recommendation_stage_df[first_recommendatic
#etc. Ideally, use a "for" loop and a list of foreign-accent characters. for c in [é

first_recommendation_stage_df.loc[movies_df['movieId'].isin(recommendationTable_df.c
```

Out[31]:

	movieId	title	genres	year
2533	2617	Mummy, The	[Action, Adventure, Comedy, Fantasy, Horror, T...	1999.0
4625	4719	Osmosis Jones	[Action, Animation, Comedy, Crime, Drama, Roma...	2001.0
4686	4781	Megiddo: The Omega Code 2	[Action, Adventure, Fantasy, Sci-Fi, Thriller]	2001.0
6394	6503	Charlie's Angels: Full Throttle	[Action, Adventure, Comedy, Crime, Thriller]	2003.0
6793	6902	Interstate 60	[Adventure, Comedy, Drama, Fantasy, Mystery, S...	2002.0
7763	8361	Day After Tomorrow, The	[Action, Adventure, Drama, Sci-Fi, Thriller]	2004.0
8286	8968	After the Sunset	[Action, Adventure, Comedy, Crime, Thriller]	2004.0

	movieId	title	genres	year
9218	27155	Batman/Superman Movie, The	[Action, Adventure, Animation, Children, Fanta...	1998.0
9403	27618	Sound of Thunder, A	[Action, Adventure, Drama, Sci-Fi, Thriller]	2005.0
9459	27735	Unstoppable	[Action, Adventure, Comedy, Drama, Thriller]	2004.0
10231	34048	War of the Worlds	[Action, Adventure, Sci-Fi, Thriller]	2005.0
10382	36509	Cave, The	[Action, Adventure, Horror, Mystery, Sci-Fi, T...	2005.0
10575	40339	Chicken Little	[Action, Adventure, Animation, Children, Comed...	2005.0
11410	48774	Children of Men	[Action, Adventure, Drama, Sci-Fi, Thriller]	2006.0
11785	52287	Meet the Robinsons	[Action, Adventure, Animation, Children, Comed...	2007.0
11806	52462	Aqua Teen Hunger Force Colon Movie Film for Th...	[Action, Adventure, Animation, Comedy, Fantasy...	2007.0
11838	52722	Spider-Man 3	[Action, Adventure, Sci-Fi, Thriller, IMAX]	2007.0
12021	54278	Underdog	[Action, Adventure, Children, Comedy, Fantasy,...	2007.0
12123	55116	Hunting Party, The	[Action, Adventure, Comedy, Drama, Thriller]	2007.0
12464	58025	Jumper	[Action, Adventure, Drama, Sci-Fi, Thriller]	2008.0
12951	61248	Death Race	[Action, Adventure, Sci-Fi, Thriller]	2008.0
13056	62331	Dead Leaves	[Action, Adventure, Animation, Comedy, Sci-Fi]	2004.0
13109	62956	Futurama: Bender's Game	[Action, Adventure, Animation, Comedy, Fantasy...	2008.0
14432	72165	Cirque du Freak: The Vampire's Assistant	[Action, Adventure, Comedy, Fantasy, Horror, T...	2009.0
14515	72601	Teenage Mutant Ninja Turtles: Turtles Forever	[Action, Adventure, Animation, Comedy, Thriller]	2009.0
15825	80219	Machete	[Action, Adventure, Comedy, Crime, Thriller]	2010.0
16055	81132	Rubber	[Action, Adventure, Comedy, Crime, Drama, Film...	2010.0
16504	83266	Kaho Naa... Pyaar Hai	[Action, Adventure, Comedy, Drama, Mystery, Ro...	2000.0
16884	85261	Mars Needs Moms	[Action, Adventure, Animation, Children, Comed...	2011.0

	movieId	title	genres	year
17317	87232	X-Men: First Class	[Action, Adventure, Sci-Fi, Thriller, War]	2011.0
17544	88140	Captain America: The First Avenger	[Action, Adventure, Sci-Fi, Thriller, War]	2011.0
17742	89002	Spy Kids: All the Time in the World in 4D	[Action, Adventure, Children, Comedy, Sci-Fi]	2011.0
18347	91500	Hunger Games, The	[Action, Adventure, Drama, Sci-Fi, Thriller]	2012.0
18361	91542	Sherlock Holmes: A Game of Shadows	[Action, Adventure, Comedy, Crime, Mystery, Th...	2011.0
20681	101076	G.I. Joe: Retaliation	[Action, Adventure, Sci-Fi, Thriller, IMAX]	2013.0
22145	106240	Free Birds	[Action, Adventure, Animation, Children, Comed...	2013.0
24946	116758	Death Racers	[Action, Adventure, Comedy, Sci-Fi, Thriller]	2008.0
25218	117646	Dragonheart 2: A New Beginning	[Action, Adventure, Comedy, Drama, Fantasy, Th...	2000.0
25485	118782	Fat Pizza	[Action, Adventure, Comedy, Crime, Thriller]	2003.0
25898	120799	Terminator Genisys	[Action, Adventure, Sci-Fi, Thriller]	2015.0
25915	120833	Super Capers	[Action, Adventure, Comedy, Fantasy, Sci-Fi]	2009.0
26301	122280	Sabretooth	[Action, Adventure, Horror, Sci-Fi, Thriller]	2002.0
28219	130518	The Amazing Screw-On Head	[Action, Adventure, Animation, Comedy, Sci-Fi]	2006.0
29140	133759	Eyeborgs	[Action, Adventure, Sci-Fi, Thriller]	2009.0
30066	136618	Pokémon the Movie: Genesect and the Legend Awa...	[Action, Adventure, Animation, Children, Fanta...	2013.0
31427	141385	Humanity's End	[Action, Adventure, Drama, Sci-Fi, Thriller]	2009.0
32006	143055	Jett Jackson: The Movie	[Action, Adventure, Children, Comedy, Sci-Fi]	2001.0
32313	144350	Under the Mountain	[Action, Adventure, Children, Drama, Fantasy, ...	2009.0
32630	145493	The Challenge	[Action, Adventure, Children, Comedy, Sci-Fi]	2003.0
33692	149488	Christmas Town	[Action, Children, Comedy, Drama, Fantasy, Thr...	2008.0

Does this algorithm/list seem more like something an Action, Sci-Fi, Adventure, Thriller film fan might enjoy?

A second recommender system is available based upon user-rating being evaluated using Pearson Correlation by collaborative filter. IMO, this item-item based recommender performs much better than the collaborate filter. However, feel free to check it out on my substack.

Advantages and Disadvantages of Content-Based Filtering

Advantages

- Learns user's preferences and improves with more ratings
- Highly personalized for the user
- Fairly fast to calculate. Using a laptop, I can calculate recommendations lists from user-input to movie-output in about 0.5s total. A dataware house server could easily make these ~500ms down to 5ms for near instant-recommendation.

Disadvantages

- Doesn't take into account what others think of the item, so low quality item recommendations might happen.
- Rarer items people don't regularly watch show up. In a user-user recommendation system, you'd see items people have watched and rated more often so rarely-watched/ranked films would show less prevalently.
- Extracting data is not always intuitive
- Determining what characteristics of the item the user dislikes or likes is not always obvious
- Recommender could be improved by adding more features. We could take min(year) and set to zero and max(year) and set to "1" and the scale all values between "0 to 1" and add a float to our weight multiplication to take into account if someone prefers newer or older movies.