Objective:

Create a recommendation system using a Content Based Model.

Get the data

In [1]:
```
# Get the data
!wget -O moviedataset.zip https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
print('unziping ...')
!unzip -o -j moviedataset.zip
```

Pre-Processing the Data

In [372…
```
import pandas as pd
from math import sqrt
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [439…
```
# 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')

movies_df.head()
```

Out[439…

	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

First Pre-Processing Movies DataFrame

In [440…
```
#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 their t
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 ha
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
movies_df.head(20)
```

Out[440…

	movieId	title	genres	year
0	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1995

	movieId	title	genres	year
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
5	6	Heat	Action|Crime|Thriller	1995
6	7	Sabrina	Comedy|Romance	1995
7	8	Tom and Huck	Adventure|Children	1995
8	9	Sudden Death	Action	1995
9	10	GoldenEye	Action|Adventure|Thriller	1995
10	11	American President, The	Comedy|Drama|Romance	1995
11	12	Dracula: Dead and Loving It	Comedy|Horror	1995
12	13	Balto	Adventure|Animation|Children	1995
13	14	Nixon	Drama	1995
14	15	Cutthroat Island	Action|Adventure|Romance	1995
15	16	Casino	Crime|Drama	1995
16	17	Sense and Sensibility	Drama|Romance	1995
17	18	Four Rooms	Comedy	1995
18	19	Ace Ventura: When Nature Calls	Comedy	1995
19	20	Money Train	Action|Comedy|Crime|Drama|Thriller	1995

In [441…

```
#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[441…

	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

Since keeping genres in a list format isn't optimal for the content-based recommendation system technique, we will use the One Hot Encoding technique to convert the list of genres to a vector where each column corresponds to one possible value of the feature. This encoding is needed for feeding categorical data.

In [442…

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

```
#For every row in the dataframe, iterate through the list of genres and place a 1 into
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 genre
moviesWithGenres_df = moviesWithGenres_df.fillna(0)
moviesWithGenres_df.head()
```

Out[442...

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

5 rows × 24 columns

Now Pre-Processing Ratings DataFrame

In [423...

```
ratings_df.head()
```

Out[423...

	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

In [424...

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

Out[424...

	userId	movieId	rating
0	1	169	2.5

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

Content-Based recommandation system model

In [425…

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

Out[425…

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

In [426…

```
#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
inputMovies
```

Out[426…

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

In [427…

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

Out[427…

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy	Roman
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	(

5 rows × 24 columns

In [428…

```
#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).drop(
userGenreTable
```

Out[428…

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Horr
0	1.0	1.0	1.0	1.0	1.0	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	0
2	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0
3	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	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	0

Now we start learning the input's preferences!

To do this, we're going to turn each genre into weights. We can do this by using the input's reviews and multiplying them into the input's genre table and then summing up the resulting table by column. This operation is actually a dot product between a matrix and a vector, so we can simply accomplish by calling Pandas's "dot" function.

In [429…
```
inputMovies['rating']
```

Out[429…
```
0    3.5
1    2.0
2    5.0
3    4.5
4    5.0
Name: rating, dtype: float64
```

In [430…
```
#Dot produt to get weights
userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
#The user profile
userProfile
```

Out[430…
```
Adventure             10.0
Animation              8.0
Children               5.5
Comedy                13.5
Fantasy                5.5
Romance                0.0
Drama                 10.0
Action                 4.5
Crime                  5.0
Thriller               5.0
Horror                 0.0
Mystery                0.0
Sci-Fi                 4.5
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
```

Now, we have the weights for every of the user's preferences. This is known as the User Profile. Using this, we can recommend movies that satisfy the user's preferences.

In [443…
```
#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('year'
genreTable.head()
```

Out[443…

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller
movieId										
1	1.0	1.0	1.0	1.0	1.0	0.0	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	0.0
3	0.0	0.0	0.0	1.0	0.0	1.0	0.0	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	0.0

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller
movieId										
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

In [432...
```
genreTable.shape
```

Out[432...
```
(34208, 20)
```

With the input's profile and the complete list of movies and their genres in hand, we're going to take the weighted average of every movie based on the input profile and recommend the top twenty movies that most satisfy it.

In [444...
```
#Multiply the genres by the weights and then take the weighted average
recommendationTable_df = ((genreTable*userProfile).sum(axis=1))/(userProfile.sum())
recommendationTable_df = recommendationTable_df.rename('rating')
recommendationTable_df
```

Out[444...
```
movieId
1         0.594406
2         0.293706
3         0.188811
4         0.328671
5         0.188811
            ...
151697    0.069930
151701    0.000000
151703    0.139860
151709    0.202797
151711    0.000000
Name: rating, Length: 34208, dtype: float64
```

In [445...
```
recommendationTable_df= recommendationTable_df.to_frame().reset_index()
recommendationTable_df
```

Out[445...

	movieId	rating
0	1	0.594406
1	2	0.293706
2	3	0.188811
3	4	0.328671
4	5	0.188811
...	...	...
34203	151697	0.069930
34204	151701	0.000000
34205	151703	0.139860
34206	151709	0.202797

	movieId	rating
34207	151711	0.000000

34208 rows × 2 columns

In [453…

```
# Merge movie data frame with recommendation series
recommendation_list = pd.merge(movies_df, recommendationTable_df, right_index= True, le

# Drop extra movie id
recommendation_list = recommendation_list.reset_index(drop=True)

recommendation_list.head()
```

Out[453…

	movieId_x	title	genres	year	movieId_y	rating
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995	1	0.594406
1	2	Jumanji	[Adventure, Children, Fantasy]	1995	2	0.293706
2	3	Grumpier Old Men	[Comedy, Romance]	1995	3	0.188811
3	4	Waiting to Exhale	[Comedy, Drama, Romance]	1995	4	0.328671
4	5	Father of the Bride Part II	[Comedy]	1995	5	0.188811

In [454…

```
# drop extra columns
recommendation_list = recommendation_list.drop('movieId_y', 1)
recommendation_list = recommendation_list.rename(columns={'movieId_x' : 'movieId'})
# Sort our recommendations in descending order
recommendation_list = recommendation_list.sort_values('rating', ascending=False)
recommendation_list = recommendation_list.reset_index(drop=True)
#Just a peek at the values
recommendation_list.head()
```

Out[454…

	movieId	title	genres	year	rating
0	5018	Motorama	[Adventure, Comedy, Crime, Drama, Fantasy, Mys...	1991	0.748252
1	26093	Wonderful World of the Brothers Grimm, The	[Adventure, Animation, Children, Comedy, Drama...	1962	0.734266
2	27344	Revolutionary Girl Utena: Adolescence of Utena...	[Action, Adventure, Animation, Comedy, Drama, ...	1999	0.720280
3	148775	Wizards of Waverly Place: The Movie	[Adventure, Children, Comedy, Drama, Fantasy, ...	2009	0.685315
4	6902	Interstate 60	[Adventure, Comedy, Drama, Fantasy, Mystery, S...	2002	0.678322