Content Based Filtering

Estimated time needed: **25** minutes

Objectives

After completing this lab you will be able to:

- Create a recommendation system using collaborative filtering

Recommendation systems are a collection of algorithms used to recommend items to users based on information taken from the user. These systems have become ubiquitous, and can be commonly seen in online stores, movies databases and job finders. In this notebook, we will explore Content-based recommendation systems and implement a simple version of one using Python and the Pandas library.

Table of contents

Acquiring the Data

To acquire and extract the data, simply run the following Bash scripts:
Dataset acquired from [GroupLens (http://grouplens.org/datasets/movielens?cm_mmc=Email_Newsletter-_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork-20718538&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newsletter.M12345678&cvo_campaign=000026_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork-20718538&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newsletter.M12345678&cvo_campaign=000026_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork-20718538&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newsletter.M12345678&cvo_campaign=000026_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork-20718538&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newsletter.M12345678&cvo_campaign=000026]
Lets download the dataset. To download the data, we will use `!wget` to download it from IBM Object Storage.

Did you know? When it comes to Machine Learning, you will likely be working with large datasets. As a business, where can you host your data? IBM is offering a unique opportunity for businesses, with 10 Tb of IBM Cloud Object Storage: [Sign up now for free (http://cocl.us/ML0101EN-IBM-Offer-CC)]

In [1]:

```
import wget

url = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/moviedataset.
myfile = wget.download(url)
```

```
100% [##########################################################################]          152M / 152M
```

In [6]:

```
#!wget -O moviedataset.zip https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv
print('unziping ...')
#!unzip -o -j moviedataset.zip
```

```
  File "C:\Users\rohit\AppData\Local\Temp/ipykernel_18252/2010967753.py", line 3
    Expand-Archive -LiteralPath 'I:\COURSES\IBM\IBM Machine Learning with Python\moviedataset.zip' -DestinationPath
I:\COURSES\IBM
                                ^
SyntaxError: invalid syntax
```

Now you're ready to start working with the data!

Preprocessing

First, let's get all of the imports out of the way:

In [7]:

```
#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
```

Now let's read each file into their Dataframes:

In [12]:

```
#Storing the movie information into a pandas dataframe
movies_df = pd.read_csv('I:\COURSES\IBM\IBM Machine Learning with Python\moviedataset\\movies.csv')
#Storing the user information into a pandas dataframe
ratings_df = pd.read_csv('I:\COURSES\IBM\IBM Machine Learning with Python\moviedataset\\ratings.csv')
#Head is a function that gets the first N rows of a dataframe. N's default is 5.
movies_df.head()
```

Out[12]:

	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

Let's also remove the year from the **title** column by using pandas' replace function and store in a new **year** column.

In [13]:

```
#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 titles
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 have appeared
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
movies_df.head()
```

```
C:\Users\rohit\AppData\Local\Temp/ipykernel_18252/1143695627.py:7: FutureWarning: The default value of regex will c
hange from True to False in a future version.
  movies_df['title'] = movies_df.title.str.replace('(\(\d\d\d\d\))', '')
```

Out[13]:

	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

With that, let's also split the values in the **Genres** column into a **list of Genres** to simplify future use. This can be achieved by applying Python's split string function on the correct column.

In [14]:
```
#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[14]:

	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 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. Let's also store this dataframe in another variable since genres won't be important for our first recommendation system.

In [15]:

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

#For every row in the dataframe, iterate through the list of genres and place a 1 into the corresponding column
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[15]:

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

5 rows × 24 columns

Next, let's look at the ratings dataframe.

In [16]:
```
ratings_df.head()
```

Out[16]:

	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

Every row in the ratings dataframe has a user id associated with at least one movie, a rating and a timestamp showing when they reviewed it. We won't be needing the timestamp column, so let's drop it to save on memory.

In [17]:
```
#Drop removes a specified row or column from a dataframe
ratings_df = ratings_df.drop('timestamp', 1)
ratings_df.head()
```

```
C:\Users\rohit\AppData\Local\Temp/ipykernel_18252/3391429438.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[17]:

	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

Now, let's take a look at how to implement **Content-Based** or **Item-Item recommendation systems**. This technique attempts to figure out what a user's favourite aspects of an item is, and then recommends 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.

Let's begin by creating an input user to recommend movies to:

Notice: To add more movies, simply increase the amount of elements in the **userInput**. Feel free to add more in! Just be sure to write it in with capital letters and if a movie starts with a "The", like "The Matrix" then write it in like this: 'Matrix, The' .

In [35]:

```
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':3},
            {'title':'Jade', 'rating':4}
         ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

Out[35]:

	title	rating
0	Breakfast Club, The	5.0
1	Toy Story	3.5
2	Jumanji	2.0
3	Pulp Fiction	5.0
4	Akira	3.0
5	Jade	4.0

Add movieId to input user

With the input complete, let's extract the input movie's ID's from the movies dataframe and add them into it.

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 for the input to save memory space.

In [36]:

```
#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\rohit\AppData\Local\Temp/ipykernel_18252/2071048360.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[36]:

	movieId	title	rating
0	1	Toy Story	3.5
1	2	Jumanji	2.0
2	132	Jade	4.0
3	296	Pulp Fiction	5.0
4	1274	Akira	3.0
5	1968	Breakfast Club, The	5.0

We're going to start by learning the input's preferences, so let's get the subset of movies that the input has watched from the Dataframe containing genres defined with binary values.

In [37]:

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

Out[37]:

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy	Romance	...	Horror	Mystery	Sci-Fi	IMAX	Doc
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995	1.0	1.0	1.0	1.0	1.0	0.0	...	0.0	0.0	0.0	0.0	
1	2	Jumanji	[Adventure, Children, Fantasy]	1995	1.0	0.0	1.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	
130	132	Jade	[Thriller]	1995	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
293	296	Pulp Fiction	[Comedy, Crime, Drama, Thriller]	1994	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
1246	1274	Akira	[Action, Adventure, Animation, Sci-Fi]	1988	1.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	
1885	1968	Breakfast Club, The	[Comedy, Drama]	1985	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	

6 rows × 24 columns

We'll only need the actual genre table, so let's clean this up a bit by resetting the index and dropping the movieId, title, genres and year columns.

In [38]:

```
#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('year', 1)
userGenreTable
```

```
C:\Users\rohit\AppData\Local\Temp/ipykernel_18252/2641803640.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[38]:

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Horror	Mystery	Sci-Fi	IMAX	Documentary	W
0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	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	0.0	0.0	0.0	0.0	0.0	(
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	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	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	(
4	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	(
5	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(

Now we're ready to 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 [39]:
```
inputMovies['rating']
```

Out[39]:
```
0    3.5
1    2.0
2    4.0
3    5.0
4    3.0
5    5.0
Name: rating, dtype: float64
```

In [40]:
```
#Dot produt to get weights
userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
#The user profile
userProfile
```

Out[40]:
```
Adventure              8.5
Animation              6.5
Children               5.5
Comedy                13.5
Fantasy                5.5
Romance                0.0
Drama                 10.0
Action                 3.0
Crime                  5.0
Thriller               9.0
Horror                 0.0
Mystery                0.0
Sci-Fi                 3.0
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.

Let's start by extracting the genre table from the original dataframe:

In [41]:
```
#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', 1)
genreTable.head()
```

```
C:\Users\rohit\AppData\Local\Temp/ipykernel_18252/789887408.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[41]:

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Horror	Mystery	Sci-Fi	IMAX	Document
movieId															
1	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	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	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	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	0.0	0.0	0.0	0.0	
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

In [42]:
```
genreTable.shape
```

Out[42]: (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 [43]:
```
#Multiply the genres by the weights and then take the weighted average
recommendationTable_df = ((genreTable*userProfile).sum(axis=1))/(userProfile.sum())
recommendationTable_df.head()
```

Out[43]:
```
movieId
1    0.568345
2    0.280576
3    0.194245
4    0.338129
5    0.194245
dtype: float64
```

In [44]:
```
#Sort our recommendations in descending order
recommendationTable_df = recommendationTable_df.sort_values(ascending=False)
#Just a peek at the values
recommendationTable_df.head()
```

Out[44]:
```
movieId
5018      0.784173
6902      0.712230
26093     0.712230
117646    0.712230
64645     0.705036
dtype: float64
```

Now here's the recommendation table!

In [45]:
```
#The final recommendation table
movies_df.loc[movies_df['movieId'].isin(recommendationTable_df.head(20).keys())]
```

Out[45]:

	movieId	title	genres	year
2902	2987	Who Framed Roger Rabbit?	[Adventure, Animation, Children, Comedy, Crime...	1988
4625	4719	Osmosis Jones	[Action, Animation, Comedy, Crime, Drama, Roma...	2001
4861	4956	Stunt Man, The	[Action, Adventure, Comedy, Drama, Romance, Th...	1980
4923	5018	Motorama	[Adventure, Comedy, Crime, Drama, Fantasy, Mys...	1991
6793	6902	Interstate 60	[Adventure, Comedy, Drama, Fantasy, Mystery, S...	2002
8605	26093	Wonderful World of the Brothers Grimm, The	[Adventure, Animation, Children, Comedy, Drama...	1962
9296	27344	Revolutionary Girl Utena: Adolescence of Utena...	[Action, Adventure, Animation, Comedy, Drama, ...	1999
9797	31921	Seven-Per-Cent Solution, The	[Adventure, Comedy, Crime, Drama, Mystery, Thr...	1976
12123	55116	Hunting Party, The	[Action, Adventure, Comedy, Drama, Thriller]	2007
13250	64645	The Wrecking Crew	[Action, Adventure, Comedy, Crime, Drama, Thri...	1968
15001	75408	Lupin III: Sweet Lost Night (Rupan Sansei: Swe...	[Action, Animation, Comedy, Crime, Drama, Myst...	2008

Advantages and Disadvantages of Content-Based Filtering

Advantages

- Learns user's preferences
- Highly personalized for the user

Disadvantages

- Doesn't take into account what others think of the item, so low quality item recommendations might happen
- Extracting data is not always intuitive
- Determining what characteristics of the item the user dislikes or likes is not always obvious