# RecSys-Content-Based-movies-py-v1

July 15, 2020

CONTENT-BASED 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.

### 0.0.1 Table of contents

```
<ol>
    <li><a href="#ref1">Acquiring the Data</a></li>
    <li><a href="#ref2">Preprocessing</a></li>
    <li><a href="#ref3">Content-Based Filtering</a></li>
</ol>
```

# Acquiring the Data

To acquire and extract the data, simply run the following Bash scripts:
Dataset acquired from GroupLens. 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

[1]:
```
!wget -O moviedataset.zip https://s3-api.us-geo.objectstorage.softlayer.net/
 ↪cf-courses-data/CognitiveClass/ML0101ENv3/labs/moviedataset.zip
print('unziping ...')
!unzip -o -j moviedataset.zip
```

```
--2020-07-15 10:50:05--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-
courses-data/CognitiveClass/ML0101ENv3/labs/moviedataset.zip
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-
geo.objectstorage.softlayer.net)… 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-
geo.objectstorage.softlayer.net)|67.228.254.196|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 160301210 (153M) [application/zip]
Saving to: 'moviedataset.zip'
```

```
moviedataset.zip    100%[===================>] 152.88M  25.1MB/s    in 6.8s

2020-07-15 10:50:12 (22.4 MB/s) - ‘moviedataset.zip’ saved [160301210/160301210]

unziping …
Archive:  moviedataset.zip
  inflating: links.csv
  inflating: movies.csv
  inflating: ratings.csv
  inflating: README.txt
  inflating: tags.csv
```

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

# Preprocessing

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

[2]:
```
#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:

[3]:
```
#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()
```

[3]:
```
   movieId                               title  \
0        1                    Toy Story (1995)
1        2                      Jumanji (1995)
2        3             Grumpier Old Men (1995)
3        4            Waiting to Exhale (1995)
4        5  Father of the Bride Part II (1995)


                                        genres
0  Adventure|Animation|Children|Comedy|Fantasy
1                   Adventure|Children|Fantasy
2                               Comedy|Romance
3                         Comedy|Drama|Romance
4                                       Comedy
```

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

[4]:
```
#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()
```

[4]:
```
   movieId                        title  \
0        1                    Toy Story
1        2                      Jumanji
2        3             Grumpier Old Men
3        4            Waiting to Exhale
4        5  Father of the Bride Part II

                                        genres  year
0  Adventure|Animation|Children|Comedy|Fantasy  1995
1                   Adventure|Children|Fantasy  1995
2                               Comedy|Romance  1995
3                         Comedy|Drama|Romance  1995
4                                       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.

[5]:
```
#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()
```

[5]:
```
   movieId                        title  \
0        1                    Toy Story
1        2                      Jumanji
2        3             Grumpier Old Men
3        4            Waiting to Exhale
4        5  Father of the Bride Part II

                                              genres  year
0  [Adventure, Animation, Children, Comedy, Fantasy]  1995
1                   [Adventure, Children, Fantasy]  1995
2                                [Comedy, Romance]  1995
```

```
3                         [Comedy, Drama, Romance]  1995
4                                         [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.

[6]:
```
#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()
```

[6]:
```
   movieId                        title  \
0        1                    Toy Story
1        2                      Jumanji
2        3             Grumpier Old Men
3        4            Waiting to Exhale
4        5  Father of the Bride Part II

                                          genres  year  Adventure  \
0  [Adventure, Animation, Children, Comedy, Fantasy]  1995        1.0
1                 [Adventure, Children, Fantasy]  1995        1.0
2                              [Comedy, Romance]  1995        0.0
3                       [Comedy, Drama, Romance]  1995        0.0
4                                       [Comedy]  1995        0.0

   Animation  Children  Comedy  Fantasy  Romance  …  Horror  Mystery  \
0        1.0       1.0     1.0      1.0      0.0  …     0.0      0.0
1        0.0       1.0     0.0      1.0      0.0  …     0.0      0.0
2        0.0       0.0     1.0      0.0      1.0  …     0.0      0.0
3        0.0       0.0     1.0      0.0      1.0  …     0.0      0.0
4        0.0       0.0     1.0      0.0      0.0  …     0.0      0.0

   Sci-Fi  IMAX  Documentary  War  Musical  Western  Film-Noir  \
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       0.0         0.0
2     0.0   0.0          0.0  0.0       0.0       0.0         0.0
3     0.0   0.0          0.0  0.0       0.0       0.0         0.0
4     0.0   0.0          0.0  0.0       0.0       0.0         0.0

   (no genres listed)
0                 0.0
1                 0.0
2                 0.0
3                 0.0
4                 0.0

[5 rows x 24 columns]
```

Next, let's look at the ratings dataframe.

[7]:
```
ratings_df.head()
```

[7]:
```
   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.

[8]:
```
#Drop removes a specified row or column from a dataframe
ratings_df = ratings_df.drop('timestamp', 1)
ratings_df.head()
```

[8]:
```
   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' .

[10]:
```python
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
```

[10]:

| | 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 |

**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.

[11]:
```python
#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
```

[11]:

| | 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 |

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.

[12]:
```
#Filtering out the movies from the input
userMovies = moviesWithGenres_df[moviesWithGenres_df['movieId'].
↪isin(inputMovies['movieId'].tolist())]
userMovies
```

[12]:
```
      movieId                  title  \
0           1              Toy Story
1           2                Jumanji
293       296           Pulp Fiction
1246     1274                  Akira
1885     1968  Breakfast Club, The

                                                genres  year  Adventure  \
0     [Adventure, Animation, Children, Comedy, Fantasy]  1995        1.0
1                       [Adventure, Children, Fantasy]  1995        1.0
293                   [Comedy, Crime, Drama, Thriller]  1994        0.0
1246            [Action, Adventure, Animation, Sci-Fi]  1988        1.0
1885                                   [Comedy, Drama]  1985        0.0

      Animation  Children  Comedy  Fantasy  Romance  …  Horror  Mystery  \
0           1.0       1.0     1.0      1.0      0.0  …     0.0      0.0
1           0.0       1.0     0.0      1.0      0.0  …     0.0      0.0
293         0.0       0.0     1.0      0.0      0.0  …     0.0      0.0
1246        1.0       0.0     0.0      0.0      0.0  …     0.0      0.0
1885        0.0       0.0     1.0      0.0      0.0  …     0.0      0.0

      Sci-Fi  IMAX  Documentary  War  Musical  Western  Film-Noir  \
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      0.0        0.0
293      0.0   0.0          0.0  0.0      0.0      0.0        0.0
1246     1.0   0.0          0.0  0.0      0.0      0.0        0.0
1885     0.0   0.0          0.0  0.0      0.0      0.0        0.0

      (no genres listed)
0                    0.0
1                    0.0
293                  0.0
1246                 0.0
1885                 0.0

[5 rows x 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.

[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).drop('year', 1)
userGenreTable
```

[13]:

```
   Adventure  Animation  Children  Comedy  Fantasy  Romance  Drama  Action  \
0        1.0        1.0       1.0     1.0      1.0      0.0    0.0     0.0
1        1.0        0.0       1.0     0.0      1.0      0.0    0.0     0.0
2        0.0        0.0       0.0     1.0      0.0      0.0    1.0     0.0
3        1.0        1.0       0.0     0.0      0.0      0.0    0.0     1.0
4        0.0        0.0       0.0     1.0      0.0      0.0    1.0     0.0

   Crime  Thriller  Horror  Mystery  Sci-Fi  IMAX  Documentary  War  Musical  \
0    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   0.0          0.0  0.0      0.0
2    1.0       1.0     0.0      0.0     0.0   0.0          0.0  0.0      0.0
3    0.0       0.0     0.0      0.0     1.0   0.0          0.0  0.0      0.0
4    0.0       0.0     0.0      0.0     0.0   0.0          0.0  0.0      0.0

   Western  Film-Noir  (no genres listed)
0      0.0        0.0                 0.0
1      0.0        0.0                 0.0
2      0.0        0.0                 0.0
3      0.0        0.0                 0.0
4      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.

[14]:

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

[14]:

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

[15]:

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

[15]:

```
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.

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

[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('year', 1)
genreTable.head()
```

[16]:

```
         Adventure  Animation  Children  Comedy  Fantasy  Romance  Drama  \
movieId
1              1.0        1.0       1.0     1.0      1.0      0.0    0.0
2              1.0        0.0       1.0     0.0      1.0      0.0    0.0
3              0.0        0.0       0.0     1.0      0.0      1.0    0.0
4              0.0        0.0       0.0     1.0      0.0      1.0    1.0
5              0.0        0.0       0.0     1.0      0.0      0.0    0.0

         Action  Crime  Thriller  Horror  Mystery  Sci-Fi  IMAX  Documentary  \
movieId
1           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
3           0.0    0.0       0.0     0.0      0.0     0.0   0.0          0.0
4           0.0    0.0       0.0     0.0      0.0     0.0   0.0          0.0
```

```
5           0.0     0.0       0.0     0.0      0.0     0.0   0.0           0.0

         War  Musical  Western  Film-Noir  (no genres listed)
movieId
1        0.0      0.0      0.0        0.0                 0.0
2        0.0      0.0      0.0        0.0                 0.0
3        0.0      0.0      0.0        0.0                 0.0
4        0.0      0.0      0.0        0.0                 0.0
5        0.0      0.0      0.0        0.0                 0.0
```

[17]:
```
genreTable.shape
```

[17]:
```
(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.

[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()
```

[18]:
```
movieId
1    0.594406
2    0.293706
3    0.188811
4    0.328671
5    0.188811
dtype: float64
```

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

[19]:
```
movieId
5018      0.748252
26093     0.734266
27344     0.720280
148775    0.685315
6902      0.678322
dtype: float64
```

Now here's the recommendation table!

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

[20]:
```
       movieId                                              title  \
664        673                                          Space Jam
1824      1907                                              Mulan
2902      2987                           Who Framed Roger Rabbit?
4923      5018                                           Motorama
6793      6902                                     Interstate 60
8605     26093           Wonderful World of the Brothers Grimm, The
8783     26340  Twelve Tasks of Asterix, The (Les douze travau…
9296     27344  Revolutionary Girl Utena: Adolescence of Utena…
9825     32031                                             Robots
11716    51632                           Atlantis: Milo's Return
11751    51939                TMNT (Teenage Mutant Ninja Turtles)
13250    64645                                   The Wrecking Crew
16055    81132                                             Rubber
18312    91335                                      Gruffalo, The
22778   108540          Ernest & Célestine (Ernest et Célestine)
22881   108932                                     The Lego Movie
25218   117646                     Dragonheart 2: A New Beginning
26442   122787                                      The 39 Steps
32854   146305                             Princes and Princesses
33509   148775                Wizards of Waverly Place: The Movie

                                                  genres  year
664    [Adventure, Animation, Children, Comedy, Fanta…  1996
1824   [Adventure, Animation, Children, Comedy, Drama…  1998
2902   [Adventure, Animation, Children, Comedy, Crime…  1988
4923   [Adventure, Comedy, Crime, Drama, Fantasy, Mys…  1991
6793   [Adventure, Comedy, Drama, Fantasy, Mystery, S…  2002
8605   [Adventure, Animation, Children, Comedy, Drama…  1962
8783   [Action, Adventure, Animation, Children, Comed…  1976
9296   [Action, Adventure, Animation, Comedy, Drama, …  1999
9825   [Adventure, Animation, Children, Comedy, Fanta…  2005
11716  [Action, Adventure, Animation, Children, Comed…  2003
11751  [Action, Adventure, Animation, Children, Comed…  2007
13250  [Action, Adventure, Comedy, Crime, Drama, Thri…  1968
16055  [Action, Adventure, Comedy, Crime, Drama, Film…  2010
18312    [Adventure, Animation, Children, Comedy, Drama]  2009
22778  [Adventure, Animation, Children, Comedy, Drama…  2012
22881  [Action, Adventure, Animation, Children, Comed…  2014
25218  [Action, Adventure, Comedy, Drama, Fantasy, Th…  2000
26442  [Action, Adventure, Comedy, Crime, Drama, Thri…  1959
32854  [Animation, Children, Comedy, Drama, Fantasy, …  2000
33509  [Adventure, Children, Comedy, Drama, Fantasy, …  2009
```

### 0.0.2 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

Want to learn more?

IBM SPSS Modeler is a comprehensive analytics platform that has many machine learning algorithms. It has been designed to bring predictive intelligence to decisions made by individuals, by groups, by systems – by your enterprise as a whole. A free trial is available through this course, available here: SPSS Modeler

Also, you can use Watson Studio to run these notebooks faster with bigger datasets. Watson Studio is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, Watson Studio enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of Watson Studio users today with a free account at Watson Studio

Thanks for completing this lesson!

Author: Saeed Aghabozorgi

Saeed Aghabozorgi, PhD is a Data Scientist in IBM with a track record of developing enterprise level applications that substantially increases clients' ability to turn data into actionable knowledge. He is a researcher in data mining field and expert in developing advanced analytic methods like machine learning and statistical modelling on large datasets.