ML0101EN-RecSys-Content-Based-movies-py-v1

December 10, 2018

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

```
In [5]: !wget -O moviedataset.zip https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-d
        print('unziping ...')
        print('Download Successful')
        !unzip -o -j moviedataset.zip
```

```
--2018-12-10 14:43:58--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/Cogni
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net).
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.n
HTTP request sent, awaiting response... 200 OK
Length: 160301210 (153M) [application/zip]
Saving to: moviedataset.zip

moviedataset.zip    100%[===================>] 152.88M  34.8MB/s   in 4.3s

2018-12-10 14:44:03 (35.6 MB/s) - moviedataset.zip saved [160301210/160301210]
```

```
unziping ...
Download Successful
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:

```
In [6]: #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 [7]: #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[7]:    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.

```
In [8]: #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 ti
        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 hav
        movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
        movies_df.head()
```

Out[8]:

	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 [9]: #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[9]:

	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 [10]:
```
#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[10]:

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy	Romance	...	Horror	Mystery	Sci-Fi	IMAX	Documentary	War	Musical	Western	Film-Noir	(no genres listed)
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	0.0	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	0.0	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	0.0	0.0	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	0.0	0.0	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	0.0	0.0	0.0	0.0	0.0	0.0

```
3                  0.0
4                  0.0

[5 rows x 24 columns]
```

Next, let's look at the ratings dataframe.

```
In [11]: ratings_df.head()
```

```
Out[11]:    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 [12]: #Drop removes a specified row or column from a dataframe
         ratings_df = ratings_df.drop('timestamp', 1)
         ratings_df.head()
```

```
Out[12]:    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 [14]: 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},
                      {'title':'Warriors, The', 'rating':5}
                  ]
         inputMovies = pd.DataFrame(userInput)
         inputMovies
```

```
Out[14]:    rating                title
         0     5.0  Breakfast Club, The
         1     3.5            Toy Story
         2     2.0              Jumanji
         3     5.0         Pulp Fiction
         4     4.5                Akira
         5     5.0        Warriors, The
```

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 [15]: #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
```

```
Out[15]:    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
         5     7802        Warriors, 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 [16]: #Filtering out the movies from the input
         userMovies = moviesWithGenres_df[moviesWithGenres_df['movieId'].isin(inputMovies['movie
         userMovies
```

```
Out[16]:       movieId                title  \
         0            1            Toy Story
         1            2              Jumanji
         293        296         Pulp Fiction
         1246      1274                Akira
         1885      1968  Breakfast Club, The
         7486      7802        Warriors, 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
7486               [Action, Adventure, Crime, Thriller]  1979        1.0

      Animation  Children  Comedy  Fantasy  Romance         ...          \
0           1.0       1.0     1.0      1.0      0.0         ...
1           0.0       1.0     0.0      1.0      0.0         ...
293         0.0       0.0     1.0      0.0      0.0         ...
1246        1.0       0.0     0.0      0.0      0.0         ...
1885        0.0       0.0     1.0      0.0      0.0         ...
7486        0.0       0.0     0.0      0.0      0.0         ...

      Horror  Mystery  Sci-Fi  IMAX  Documentary  War  Musical  Western  \
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
293      0.0      0.0     0.0   0.0          0.0  0.0      0.0      0.0
1246     0.0      0.0     1.0   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      0.0
7486     0.0      0.0     0.0   0.0          0.0  0.0      0.0      0.0

      Film-Noir  (no genres listed)
0           0.0                0.0
1           0.0                0.0
293         0.0                0.0
1246        0.0                0.0
1885        0.0                0.0
7486        0.0                0.0

[6 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.

```
In [17]: #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[17]:    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
5        1.0        0.0       0.0     0.0      0.0      0.0    0.0     1.0

   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
5    1.0       1.0     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
5      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 [18]: inputMovies['rating']

Out[18]: 0    3.5
         1    2.0
         2    5.0
         3    4.5
         4    5.0
         5    5.0
         Name: rating, dtype: float64

In [19]: #Dot produt to get weights
         userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
         #The user profile
         userProfile

Out[19]: Adventure            15.0
         Animation             8.0
         Children              5.5
         Comedy               13.5
         Fantasy               5.5
         Romance               0.0
         Drama                10.0
         Action                9.5
         Crime                10.0
```

```
Thriller              10.0
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:

In [20]:
```
#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('yea
genreTable.head()
```

Out[20]:

movieId	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	\
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	

movieId	Action	Crime	Thriller	Horror	Mystery	Sci-Fi	IMAX	Documentary	\
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	

movieId	War	Musical	Western	Film-Noir	(no genres listed)
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

In [21]:
```
genreTable.shape
```

```
Out[21]: (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 [22]: #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[22]: movieId
         1    0.519126
         2    0.284153
         3    0.147541
         4    0.256831
         5    0.147541
         dtype: float64
```

```
In [23]: #Sort our recommendations in descending order
         recommendationTable_df = recommendationTable_df.sort_values(ascending=False)
         #Just a peek at the values
         recommendationTable_df.head()
```

```
Out[23]: movieId
         5018      0.748634
         81132     0.743169
         64645     0.743169
         122787    0.743169
         117646    0.693989
         dtype: float64
```

Now here's the recommendation table!

```
In [24]: #The final recommendation table
         movies_df.loc[movies_df['movieId'].isin(recommendationTable_df.head(20).keys())]
```

```
Out[24]:        movieId                                              title  \
         4625      4719                                     Osmosis Jones
         4861      4956                                   Stunt Man, The
         4923      5018                                        Motorama
         5559      5657                                       Flashback
         6793      6902                                    Interstate 60
         9296     27344  Revolutionary Girl Utena: Adolescence of Utena...
         9459     27735                                      Unstoppable
         9697     31367                                       Chase, The
         9797     31921                 Seven-Per-Cent Solution, The
         12123    55116                           Hunting Party, The
         13250    64645                            The Wrecking Crew
         15001    75408  Lupin III: Sweet Lost Night (Rupan Sansei: Swe...
```

```
15073    76153  Lupin III: First Contact (Rupan Sansei: Faasut...
15825    80219                                            Machete
16055    81132                                             Rubber
24528   115333                          Charlie Chan in Panama
24565   115479                                  Whip Hand, The
25218   117646                  Dragonheart 2: A New Beginning
26442   122787                                    The 39 Steps
26806   124681                                         Raffles

                                                    genres  year
4625   [Action, Animation, Comedy, Crime, Drama, Roma...  2001
4861   [Action, Adventure, Comedy, Drama, Romance, Th...  1980
4923   [Adventure, Comedy, Crime, Drama, Fantasy, Mys...  1991
5559           [Action, Adventure, Comedy, Crime, Drama]  1990
6793   [Adventure, Comedy, Drama, Fantasy, Mystery, S...  2002
9296   [Action, Adventure, Animation, Comedy, Drama, ...  1999
9459        [Action, Adventure, Comedy, Drama, Thriller]  2004
9697   [Action, Adventure, Comedy, Crime, Romance, Th...  1994
9797   [Adventure, Comedy, Crime, Drama, Mystery, Thr...  1976
12123       [Action, Adventure, Comedy, Drama, Thriller]  2007
13250  [Action, Adventure, Comedy, Crime, Drama, Thri...  1968
15001  [Action, Animation, Comedy, Crime, Drama, Myst...  2008
15073  [Action, Animation, Comedy, Crime, Drama, Myst...  2002
15825       [Action, Adventure, Comedy, Crime, Thriller]  2010
16055  [Action, Adventure, Comedy, Crime, Drama, Film...  2010
24528  [Adventure, Comedy, Crime, Drama, Mystery, Thr...  1940
24565  [Action, Adventure, Crime, Drama, Sci-Fi, Thri...  1951
25218  [Action, Adventure, Comedy, Drama, Fantasy, Th...  2000
26442  [Action, Adventure, Comedy, Crime, Drama, Thri...  1959
26806  [Adventure, Comedy, Crime, Drama, Romance, Thr...  1939
```

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.