ML0101EN-RecSys-Collaborative-Filtering-movies-py-v1

March 6, 2019

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 can be commonly seen in online stores, movies databases and job finders. In this notebook, we will explore recommendation systems based on Collaborative Filtering and implement simple version of one using Python and the Pandas library.

Table of contents

```
<ol>
    <li><a href="#ref1">Acquiring the Data</a></li>
    <li><a href="#ref2">Preprocessing</a></li>
    <li><a href="#ref3">Collaborative 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 [9]: !wget -O moviedataset.zip https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-d
        print('unziping ...')
        !unzip -o -j moviedataset.zip
```

```
--2019-03-06 20:41:48--  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  35.7MB/s   in 4.4s

2019-03-06 20:41:53 (34.4 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:

```
In [10]: #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('movies.csv')
         #Storing the user information into a pandas dataframe
         ratings_df = pd.read_csv('ratings.csv')
```

Let's also take a peek at how each of them are organized:

```
In [14]: #Head is a function that gets the first N rows of a dataframe. N's default is 5.
         movies_df.head()
```

```
Out[14]:    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
```

So each movie has a unique ID, a title with its release year along with it (Which may contain unicode characters) and several different genres in the same field. Let's remove the year from the title column and place it into its own one by using the handy extract function that Pandas has.

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

```
In [15]: #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())
```

Let's look at the result!

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

```
Out[16]:    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 drop the genres column since we won't need it for this particular recommendation system.

```
In [17]: #Dropping the genres column
         movies_df = movies_df.drop('genres', 1)
```

Here's the final movies dataframe:

```
In [18]: movies_df.head()
```

```
Out[18]:    movieId                        title  year
         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
```

Next, let's look at the ratings dataframe.

```
In [19]: ratings_df.head()
```

```
Out[19]:    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 [20]: #Drop removes a specified row or column from a dataframe
         ratings_df = ratings_df.drop('timestamp', 1)
```

Here's how the final ratings Dataframe looks like:

```
In [21]: ratings_df.head()
```

```
Out[21]:    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
```

Collaborative Filtering

Now, time to start our work on recommendation systems.

The first technique we're going to take a look at is called **Collaborative Filtering**, which is also known as **User-User Filtering**. As hinted by its alternate name, this technique uses other users to recommend items to the input user. It attempts to find users that have similar preferences and opinions as the input and then recommends items that they have liked to the input. There are several methods of finding similar users (Even some making use of Machine Learning), and the one we will be using here is going to be based on the **Pearson Correlation Function**.

The process for creating a User Based recommendation system is as follows: - Select a user with the movies the user has watched - Based on his rating to movies, find the top X neighbours - Get the watched movie record of the user for each neighbour. - Calculate a similarity score using some formula - Recommend the items with the highest score

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 [30]: 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[30]:    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
```

Add movieId to input user With the input complete, let's extract the input movies'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 movies' title and then merging this subset with the input dataframe. We also drop unnecessary columns for the input to save memory space.

```
In [31]: #Filtering out the movies by title
         inputId = movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]
         print(inputId)
         #Then merging it so we can get the movieId. It's implicitly merging it by title.
         inputMovies = pd.merge(inputId, inputMovies)
         print(inputMovies)
         #Dropping information we won't use from the input dataframe
         inputMovies = inputMovies.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
```

```
      movieId                title  year
0           1            Toy Story  1995
1           2              Jumanji  1995
293       296          Pulp Fiction  1994
1246     1274                Akira  1988
1885     1968  Breakfast Club, The  1985
   movieId                title  year  rating
0        1            Toy Story  1995     3.5
1        2              Jumanji  1995     2.0
2      296          Pulp Fiction  1994     5.0
3     1274                Akira  1988     4.5
4     1968  Breakfast Club, The  1985     5.0
```

```
Out[31]:    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
```

The users who has seen the same movies Now with the movie ID's in our input, we can now get the subset of users that have watched and reviewed the movies in our input.

```
In [32]: #Filtering out users that have watched movies that the input has watched and storing it
         userSubset = ratings_df[ratings_df['movieId'].isin(inputMovies['movieId'].tolist())]
         userSubset.head()
```

```
Out[32]:      userId  movieId  rating
         19        4      296     4.0
         441      12     1968     3.0
         479      13        2     2.0
         531      13     1274     5.0
         681      14      296     2.0
```

We now group up the rows by user ID.

```
In [34]: #Groupby creates several sub dataframes where they all have the same value in the colum
         userSubsetGroup = userSubset.groupby(['userId'])
```

lets look at one of the users, e.g. the one with userID=1130

```
In [35]: userSubsetGroup.get_group(1130)
```

```
Out[35]:         userId  movieId  rating
         104167    1130        1     0.5
         104168    1130        2     4.0
         104214    1130      296     4.0
         104363    1130     1274     4.5
         104443    1130     1968     4.5
```

Let's also sort these groups so the users that share the most movies in common with the input have higher priority. This provides a richer recommendation since we won't go through every single user.

```
In [38]: #Sorting it so users with movie most in common with the input will have priority
         userSubsetGroup = sorted(userSubsetGroup,  key=lambda x: len(x[1]), reverse=True)
```

Now lets look at the first user

```
In [39]: userSubsetGroup[0:3]
```

```
Out[39]: [(75,        userId  movieId  rating
           7507      75        1     5.0
           7508      75        2     3.5
           7540      75      296     5.0
           7633      75     1274     4.5
           7673      75     1968     5.0), (106,        userId  movieId  rating
           9083     106        1     2.5
           9084     106        2     3.0
           9115     106      296     3.5
```