Netflix_Collaborative_Filtering

January 27, 2021

1 Netflix Recommender System - 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.

1.0.1 Importing necessary libraries

[3]:
```
import pandas as pd
import numpy as np
from math import sqrt

pd.options.display.max_rows = 20
pd.options.display.max_columns = 100
```

1.0.2 Loading the data

The data has been downloaded from the kaggle competition page "Netflix prize data" https://www.kaggle.com/netflix-inc/netflix-prize-data then loaded onto Jupyter Notebook from local hard drive.

Here we use 2 data sets. "combined_data_1" which contains user IDs and their ratings to various movie ID. The second data set "movie_titles" has the movie names and production year for each movie ID.

[4]:
```
df = pd.read_csv("C:/Users/eliec/netflix.csv")
print(df.shape)
df.head()
```

```
(24053764, 3)
```

[4]:

	Cust_Id	Rating	Movie_Id
0	1488844	3.0	1
1	822109	5.0	1
2	885013	4.0	1
3	30878	4.0	1
4	823519	3.0	1

[6]:
```python
df_title = pd.read_csv('C:/Users/eliec/netflix/movie_titles.txt.csv', encoding␣
↪= "ISO-8859-1",
                       header = None, names = ['Movie_Id', 'Year', 'Name'])
df_title.head()
```

[6]:

	Movie_Id	Year	Name
0	1	2003.0	Dinosaur Planet
1	2	2004.0	Isle of Man TT 2004 Review
2	3	1997.0	Character
3	4	1994.0	Paula Abdul's Get Up & Dance
4	5	2004.0	The Rise and Fall of ECW

target user we will randomly choose user 785314 as a target user for whom we will be recommending movies.

for a diversified catalog we will choose 10 movies that this user has rated with the rating values evenly distributed.

[7]:
```python
target = df[df["Cust_Id"]==785314]
target
```

[7]:

	Cust_Id	Rating	Movie_Id
5101	785314	1.0	8
31449	785314	1.0	18
52549	785314	3.0	28
92848	785314	1.0	30
258646	785314	5.0	57
…	…	…	…
23598232	785314	3.0	4418
23818526	785314	5.0	4454
23840420	785314	4.0	4472
23949372	785314	1.0	4479
23977583	785314	3.0	4485

[165 rows x 3 columns]

[8]:
```python
target_movies = [8,18, 28, 4418, 4472, 57, 4454]
```

we need 1 more movie with rating 4 and another 2 with rating 2

[9]:
```python
target[(target["Rating"]==2.0) | (target["Rating"]==4.0)]
```

[9]:

	Cust_Id	Rating	Movie_Id
1497810	785314	4.0	312
2369112	785314	2.0	457
2763214	785314	4.0	494
3013861	785314	4.0	571
3762601	785314	4.0	720

```
…              …       …        …
21964405   785314     4.0      4141
22411417   785314     4.0      4260
22657991   785314     2.0      4302
23591531   785314     4.0      4412
23840420   785314     4.0      4472

[58 rows x 3 columns]
```

update target movies list

[10]:
```
target_movies = [8,18, 28, 4418, 4472, 57, 4454, 312, 457, 4302]
```

Let's get the totles for those target movies on which we will base our recommendations

[11]:
```
target_titles = df_title[df_title["Movie_Id"].isin(target_movies)]
target_titles
```

[11]:

	Movie_Id	Year	Name
7	8	2004.0	What the #$*! Do We Know!?
17	18	1994.0	Immortal Beloved
27	28	2002.0	Lilo and Stitch
56	57	1995.0	Richard III
311	312	2000.0	High Fidelity
456	457	2004.0	Kill Bill: Vol. 2
4301	4302	1982.0	An Officer and a Gentleman
4417	4418	2000.0	Titan A.E.
4453	4454	1944.0	To Have and Have Not
4471	4472	2003.0	Love Actually

[12]:
```
target_titles = target_titles.merge(target, on="Movie_Id")
target_titles
```

[12]:

	Movie_Id	Year	Name	Cust_Id	Rating
0	8	2004.0	What the #$*! Do We Know!?	785314	1.0
1	18	1994.0	Immortal Beloved	785314	1.0
2	28	2002.0	Lilo and Stitch	785314	3.0
3	57	1995.0	Richard III	785314	5.0
4	312	2000.0	High Fidelity	785314	4.0
5	457	2004.0	Kill Bill: Vol. 2	785314	2.0
6	4302	1982.0	An Officer and a Gentleman	785314	2.0
7	4418	2000.0	Titan A.E.	785314	3.0
8	4454	1944.0	To Have and Have Not	785314	5.0
9	4472	2003.0	Love Actually	785314	4.0

Now we will create a dataframe that has similar users to our target who have seen and rated the same movies

[13]:
```
similar_users = df[df["Movie_Id"].isin(target_movies)]
```

[14]:
```
print(similar_users.shape)
similar_users
```

```
(429846, 3)
```

[14]:

	Cust_Id	Rating	Movie_Id
5098	824097	2.0	8
5099	2630686	5.0	8
5100	644003	3.0	8
5101	785314	1.0	8
5102	243963	3.0	8
…	…	…	…
23941067	1573203	2.0	4472
23941068	886903	3.0	4472
23941069	692028	5.0	4472
23941070	2253112	5.0	4472
23941071	2480480	4.0	4472

```
[429846 rows x 3 columns]
```

As you can see this has narrowed down the number of users from 24+ million to 429846.

To proceed we need to create a grouped dataframe that creates several sub dataframes where they all have the same value in the column specified as the parameter which is "Cust_Id"

[15]:
```
userSubsetGroup = similar_users.groupby(['Cust_Id'])
```

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.

[16]:
```
#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)

#this takes the grouped df and returns a sorted list based on the lenght of␣
 ↪each item in the list( key=lambda x: len(x[1]))
#and sorts it in descending order (reverse=True)
```

sorted() can take a maximum of three parameters:

iterable - A sequence (string, tuple, list) or collection (set, dictionary, frozen set) or any other iterator. reverse (Optional) - If True, the sorted list is reversed (or sorted in descending order). Defaults to False if not provided. key (Optional) - A function that serves as a key for the sort comparison. Defaults to None.

Sorted() took a dataframe and returned a grouped list as following:

[17]:
```
print("userSubsetGroup is a : ",type(userSubsetGroup))
userSubsetGroup[0:3]
```

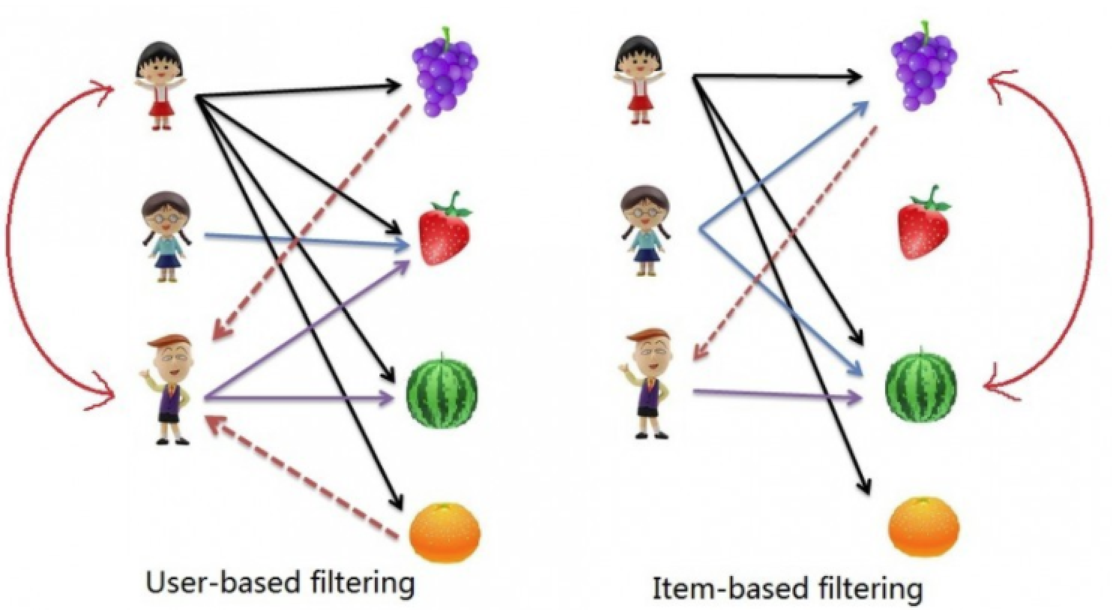
```
userSubsetGroup is a :  <class 'list'>
```

```
[17]: [(305344,
                Cust_Id  Rating  Movie_Id
  14657          305344     1.0         8
  38266          305344     2.0        18
  77908          305344     3.0        28
  260889         305344     2.0        57
  1539639        305344     4.0       312
  2443568        305344     1.0       457
  22693386       305344     5.0      4302
  23607887       305344     1.0      4418
  23822897       305344     4.0      4454
  23904716       305344     1.0      4472),
 (322009,
                Cust_Id  Rating  Movie_Id
  10876          322009     3.0         8
  35520          322009     3.0        18
  67625          322009     3.0        28
  260012         322009     3.0        57
  1522620        322009     4.0       312
  2413385        322009     4.0       457
  22678972       322009     4.0      4302
  23603983       322009     3.0      4418
  23821132       322009     3.0      4454
  23878403       322009     5.0      4472),
 (387418,
                Cust_Id  Rating  Movie_Id
  17522          387418     1.0         8
  40291          387418     2.0        18
  85409          387418     1.0        28
  261564         387418     1.0        57
  1551935        387418     4.0       312
  2465552        387418     4.0       457
  22704004       387418     3.0      4302
  23610757       387418     2.0      4418
  23824162       387418     1.0      4454
  23923705       387418     3.0      4472)]
```

1.1 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

we selected the target movies now let's proceed to finding the top X neighbours based on the rating of those movies

1.1.1 Similarity of users to input user

Next, we are going to compare all users (not really all !!!) to our specified user and find the one that is most similar. we're going to find out how similar each user is to the input through the Pearson Correlation Coefficient. It is used to measure the strength of a linear association between two variables. The formula for finding this coefficient between sets X and Y with N values can be seen in the image below.

Why Pearson Correlation?

Pearson correlation is invariant to scaling, i.e. multiplying all elements by a nonzero constant or adding any constant to all elements. For example, if you have two vectors X and Y,then, pearson(X, Y) == pearson(X, 2 * Y + 3). This is a pretty important property in recommendation systems because for example two users might rate two series of items totally different in terms of absolute rates, but they would be similar users (i.e. with similar ideas) with similar rates in various scales .

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

The values given by the formula vary from r = -1 to r = 1, where 1 forms a direct correlation between the two entities (it means a perfect positive correlation) and -1 forms a perfect negative correlation.

In our case, a 1 means that the two users have similar tastes while a -1 means the opposite.

We will select a subset of users to iterate through. This limit is imposed because we don't want to waste too much time going through every single user.

Now, we calculate the Pearson Correlation between input user and subset group, and store it in a dictionary, where the key is the user Id and the value is the coefficient

[18]:
```
#Store the Pearson Correlation in a dictionary, where the key is the user Id␣
↪and the value is the coefficient
pearsonCorrelationDict = {}

#For every user group in our subset
for name, group in userSubsetGroup:
    #Let's start by sorting the input and current user group so the values␣
↪aren't mixed up later on
    group = group.sort_values(by='Movie_Id')
    target_titles = target_titles.sort_values(by='Movie_Id')
    #Get the N for the formula
    nRatings = len(group)
    #Get the review scores for the movies that they both have in common
    temp_df = target_titles[target_titles['Movie_Id'].isin(group['Movie_Id'].
↪tolist())]
    #And then store them in a temporary buffer variable in a list format to␣
↪facilitate future calculations
    tempRatingList = temp_df['Rating'].tolist()
    #Let's also put the current user group reviews in a list format
    tempGroupList = group['Rating'].tolist()
    #Now let's calculate the pearson correlation between two users, so called,␣
↪x and y
    Sxx = sum([i**2 for i in tempRatingList]) - pow(sum(tempRatingList),2)/
↪float(nRatings)
    Syy = sum([i**2 for i in tempGroupList]) - pow(sum(tempGroupList),2)/
↪float(nRatings)
    Sxy = sum( i*j for i, j in zip(tempRatingList, tempGroupList)) -␣
↪sum(tempRatingList)*sum(tempGroupList)/float(nRatings)

    #If the denominator is different than zero, then divide, else, 0␣
↪correlation.
    if Sxx != 0 and Syy != 0:
        pearsonCorrelationDict[name] = Sxy/sqrt(Sxx*Syy)
    else:
        pearsonCorrelationDict[name] = 0
```

Now we create a dataframe that has the correlation with all the users using the dictionary that we just created

[19]:
```
pearsonDF = pd.DataFrame.from_dict(pearsonCorrelationDict, orient='index')
pearsonDF.columns = ['similarityIndex']
pearsonDF['Cust_Id'] = pearsonDF.index
pearsonDF.index = range(len(pearsonDF))
pearsonDF.head()
```

[19]:

	similarityIndex	Cust_Id
0	0.247537	305344
1	0.105409	322009
2	-0.121268	387418
3	0.322749	603277
4	0.283473	716173

The top x similar users to input user¶ Now let's get the top users that are most similar to the input with similarity index equal 1

[20]:
```
print(pearsonDF.shape)
topUsers=pearsonDF[pearsonDF["similarityIndex"]==1.0]
print(topUsers.shape)
```

```
(219874, 2)
(14908, 2)
```

Rating of selected users to all movies We're going to do this by taking the weighted average of the ratings of the movies using the Pearson Correlation as the weight. But to do this, we first need to get the movies watched by the users in our pearsonDF from the ratings dataframe and then store their correlation in a new column called _similarityIndex". This is achieved below by merging of these two tables.

[21]:
```
topUsersRating=topUsers.merge(df, on="Cust_Id")
topUsersRating.head()
```

[21]:

	similarityIndex	Cust_Id	Rating	Movie_Id
0	1.0	785314	1.0	8
1	1.0	785314	1.0	18
2	1.0	785314	3.0	28
3	1.0	785314	1.0	30
4	1.0	785314	5.0	57

Now all we need to do is simply multiply the movie rating by its weight (The similarity index), then sum up the new ratings and divide it by the sum of the weights. In this case since we sliced on the users with similatity index equal to one there will be no change to the ratings

We can easily do this by simply multiplying two columns, then grouping up the dataframe by movieId and then dividing two columns:

It shows the idea of all similar users to candidate movies for the input user:

[22]:
```
#Multiplies the similarity by the user's ratings
topUsersRating['weightedRating'] =␣
↪topUsersRating['similarityIndex']*topUsersRating['Rating']
topUsersRating.head()
```

[22]:
```
   similarityIndex  Cust_Id  Rating  Movie_Id  weightedRating
0              1.0   785314     1.0         8             1.0
1              1.0   785314     1.0        18             1.0
2              1.0   785314     3.0        28             3.0
3              1.0   785314     1.0        30             1.0
4              1.0   785314     5.0        57             5.0
```

[23]:
```
#Applies a sum to the topUsers after grouping it up by userId
tempTopUsersRating = topUsersRating.groupby('Movie_Id').
↪sum()[['similarityIndex','weightedRating']]
tempTopUsersRating.columns = ['sum_similarityIndex','sum_weightedRating']
tempTopUsersRating.head()
```

[23]:
```
          sum_similarityIndex  sum_weightedRating
Movie_Id
1                        25.0                90.0
2                         5.0                16.0
3                        88.0               315.0
4                         5.0                13.0
5                        41.0               163.0
```

Finally, let's create the dafatrame containing the recommendations together with the ratings

[24]:
```
#Creates an empty dataframe
recommendation_df = pd.DataFrame()
#Now we take the weighted average
recommendation_df['weighted average recommendation score'] =␣
↪tempTopUsersRating['sum_weightedRating']/
↪tempTopUsersRating['sum_similarityIndex']
recommendation_df['movieId'] = tempTopUsersRating.index
recommendation_df.head()
```

[24]:
```
          weighted average recommendation score  movieId
Movie_Id
1                                      3.600000        1
2                                      3.200000        2
3                                      3.579545        3
4                                      2.600000        4
5                                      3.975610        5
```

[25]:
```
recommendation_df = recommendation_df.reset_index().merge(df_title,␣
↪on="Movie_Id")
```

```
recommendation_df = recommendation_df.sort_values(by="weighted average␣
↪recommendation score", ascending=False)
recommendation_df.head(10)
```

[25]:

```
      Movie_Id  weighted average recommendation score  movieId    Year  \
4271      4294                               5.000000     4294  2004.0
2157      2166                               5.000000     2166  2000.0
3002      3019                               5.000000     3019  1990.0
2425      2437                               5.000000     2437  1975.0
2376      2387                               5.000000     2387  2001.0
12          13                               5.000000       13  2003.0
1690      1698                               5.000000     1698  1999.0
710        714                               5.000000      714  1988.0
2270      2280                               5.000000     2280  1976.0
3050      3067                               4.857143     3067  2004.0

                                                  Name
4271                           Ghost Hunters: Season 1
2157                               Cold Feet: Season 3
3002                           Warren Miller's Journey
2425                           A Woman Called Sada Abe
2376                Luther Vandross: Journeys in Black
12    Lord of the Rings: The Return of the King: Ext…
1690                               Escape from Alaska
710                                       Whisper Kill
2270                 Eleanor & Franklin: The Early Years
3050                             Teen Titans: Season 2
```

[ ]: