Collaborative Filtering

Estimated time needed: **25** minutes

Objectives

After completing this lab you will be able to:

- Create recommendation system based on 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

Acquiring the Data

To acquire and extract the data, simply run the following Bash scripts:
Dataset acquired from GroupLens (http://grouplens.org/datasets/movielens/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01). Let's 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
```

```
!wget -O moviedataset.zip https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%205/data/moviedataset.zip
print('unziping ...')
!unzip -o -j moviedataset.zip
```

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

Preprocessing

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

In [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:

In [5]:
```
#Storing the movie information into a pandas dataframe
movies_df = pd.read_csv('I:\COURSES\IBM\IBM Machine Learning with Python\Week 5\moviedataset\\movies.csv')
#Storing the user information into a pandas dataframe
ratings_df = pd.read_csv('I:\COURSES\IBM\IBM Machine Learning with Python\Week 5\moviedataset\\ratings.csv')
```

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

In [6]:
```
#Head is a function that gets the first N rows of a dataframe. N's default is 5.
movies_df.head()
```

Out[6]:

	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

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 (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.str.extract.html?

utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01#pandas.Series.str.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 [7]:
```
#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())
```

```
C:\Users\rohit\AppData\Local\Temp/ipykernel_13404/2727439841.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('(\(\d\d\d\d\))', '')
```

Let's look at the result!

In [8]:
```
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 drop the genres column since we won't need it for this particular recommendation system.

In [9]:
```
#Dropping the genres column
movies_df = movies_df.drop('genres', 1)
```

```
C:\Users\rohit\AppData\Local\Temp/ipykernel_13404/37422046.py:2: FutureWarning: In a future version of pandas all a
rguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  movies_df = movies_df.drop('genres', 1)
```

Here's the final movies dataframe:

In [10]:
```
movies_df.head()
```

Out[10]:

	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 [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)
```

```
C:\Users\rohit\AppData\Local\Temp/ipykernel_13404/1971122656.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)
```

Here's how the final ratings Dataframe looks like:

In [13]:
```
ratings_df.head()
```

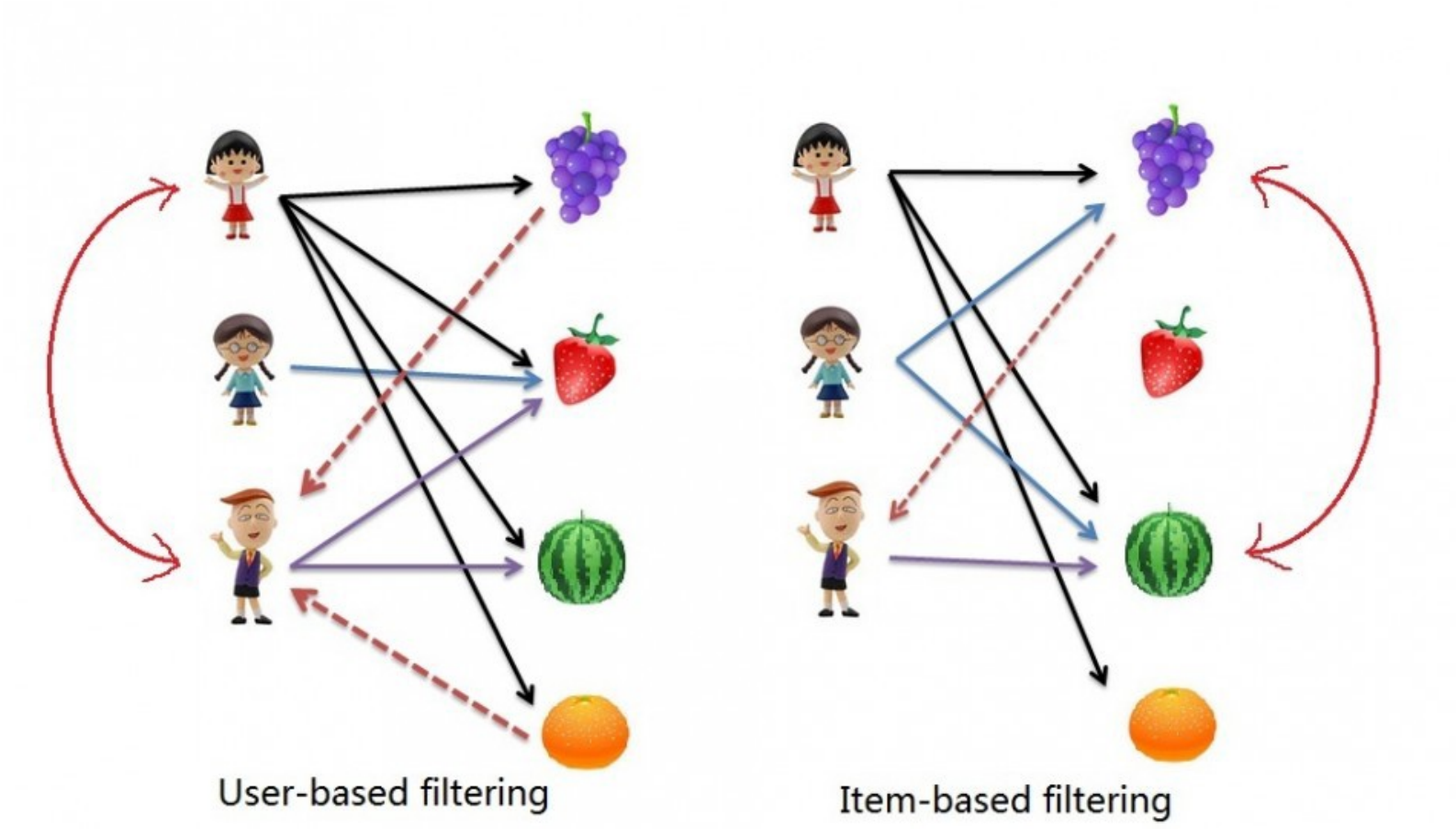
Out[13]:

	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 it's 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 [39]:
```
userInput = [
            {'title':'Jade', 'rating':4},
            {'title':'Saw', 'rating':5},
            {'title':'Get Out', 'rating':5},
            {'title':"Pulp Fiction", 'rating':2},
            {'title':'Wrong Turn', 'rating':3.5},
            {'title':'Lords of Salem, The', 'rating':3}
        ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

Out[39]:

	title	rating
0	Jade	4.0
1	Saw	5.0
2	Get Out	5.0
3	Pulp Fiction	2.0
4	Wrong Turn	3.5
5	Lords of Salem, The	3.0

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 [40]:
```
inputMovies['title'].tolist()
```

Out[40]:
```
['Jade', 'Saw', 'Get Out', 'Pulp Fiction', 'Wrong Turn', 'Lords of Salem, The']
```

In [41]:
```
movies_df['title'].isin(inputMovies['title'].tolist())
```

Out[41]:
```
0        False
1        False
2        False
3        False
4        False
         ...
34203    False
34204    False
34205    False
34206    False
34207    False
Name: title, Length: 34208, dtype: bool
```

In [42]:
```
movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]
```

Out[42]:

	movieId	title	year
130	132	Jade	1995
293	296	Pulp Fiction	1994
6274	6379	Wrong Turn	2003
8275	8957	Saw	2004
21136	102802	Lords of Salem, The	2012
33469	148671	Saw	2003

In [43]:
```
#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('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_13404/2035672847.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('year', 1)
```

Out[43]:

	movieId	title	rating
0	132	Jade	4.0
1	296	Pulp Fiction	2.0
2	6379	Wrong Turn	3.5
3	8957	Saw	5.0
4	148671	Saw	5.0
5	102802	Lords of Salem, The	3.0

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 [44]:
```
#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[44]:

	userId	movieId	rating
19	4	296	4.0
653	14	132	3.0
681	14	296	2.0
776	15	296	3.0
1333	17	296	2.0

We now group up the rows by user ID.

In [45]:
```
#Groupby creates several sub dataframes where they all have the same value in the column specified as the parameter
userSubsetGroup = userSubset.groupby(['userId'])
```

Let's look at one of the users, e.g. the one with userID=1130.

In [46]:
```
userSubsetGroup.get_group(1130)
```

Out[46]:

	userId	movieId	rating
104214	1130	296	4.0

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 [47]:
```
#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 let's look at the first user.

In [48]:
```
userSubsetGroup[0:3]
```

Out[48]:
```
[(58040,
           userId  movieId  rating
 5418089   58040      132     1.0
 5418195   58040      296     5.0
 5420988   58040     6379     0.5
 5421524   58040     8957     2.5
 5423380   58040   102802     1.0),
(204165,
           userId  movieId  rating
 18905180  204165      132     3.5
 18905278  204165      296     5.0
 18908031  204165     6379     3.0
 18908708  204165     8957     4.5
 18911351  204165   102802     2.5),
(815,
          userId  movieId  rating
 73820      815      132     2.5
 73922      815      296     5.0
 76345      815     6379     2.0
 76780      815     8957     3.5)]
```

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.

In [49]:
```
userSubsetGroup = userSubsetGroup[0:100]
```

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.

In [50]:

```
#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='movieId')
    inputMovies = inputMovies.sort_values(by='movieId')
    #Get the N for the formula
    nRatings = len(group)
    #Get the review scores for the movies that they both have in common
    temp_df = inputMovies[inputMovies['movieId'].isin(group['movieId'].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(nRa

    #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
```

In [51]:
```
pearsonCorrelationDict.items()
```

Out[51]:
```
dict_items([(58040, -0.45643546458763845), (204165, -0.05391638660171923), (815, -0.5291502622129182), (4099, 0.233
02720008113567), (5045, -0.4387862045841156), (8863, -0.23946643489586264), (13228, -0.9428090415820634), (15010, -
0.17407765595569785), (15961, -0.7921180343813394), (16382, -0.7224456455979048), (20554, -0.7276068751089989), (27
383, -0.4163331998932266), (29203, -0.4365641250653994), (35743, -0.015032920560056576), (35887, -0.529150262212918
2), (38957, -0.08084520834544433), (44508, -0.09493700744046268), (46750, -0.8433383141984728), (50970, -0.81877649
25226791), (61917, -0.5659164584181102), (67300, -0.4163331998932266), (68204, -0.15555555555555556), (68257, 0.326
59863237109044), (68783, -0.457679369192477), (73419, -0.9382458864878921), (74551, -0.37777777777777777), (80485,
-0.5789407688124092), (81514, 0.17407765595569785), (89331, -0.7442084075352507), (95978, -0.6172133998483676), (10
0664, -0.8392740627357385), (116283, -0.7246573018525413), (121005, -0.8666666666666667), (129179, -0.5887840577551
898), (132717, -0.8640987597877148), (132751, -0.15491933384829668), (135877, -0.15555555555555556), (137409, -0.45
7679369192477), (140845, -0.808290376865476), (141433, -0.662266178532522), (142943, -0.6050580452280905), (146116,
-0.015032920560056576), (146422, -0.8811039019443198), (146607, -0.5033222956847166), (147003, -0.1154700538379251
4), (151300, -0.3443784128369114), (156114, -0.49377071987869414), (157690, -0.28751811537130434), (158419, -0.7811
26577552403), (159400, -0.816496580927726), (161716, -0.5291502622129182), (167342, -0.48989794855663565), (170914,
-0.5773502691896257), (175196, -0.7525121326622725), (176992, -0.9801539442281324), (177209, -0.18516401995451032),
(180362, -0.6092848842499202), (180780, -0.11547005383792514), (184826, 0.4264014327112209), (190449, -0.7666789485
628853), (200118, -0.9797958971132713), (205009, -0.6713171133426189), (215057, 0.2), (217637, -0.626666666666666
7), (218971, -0.40855058468556077), (221732, 0.34575717288130126), (222172, -0.25819888974716115), (222874, 0.03802
345503146868), (228926, 0.09370425713316365), (230909, -0.015032920560056576), (237154, -0.7504787743864564), (2383
50, -0.5033222956847166), (239662, -0.9287432920695055), (242823, 0.0643267520902677), (243688, -0.0150329205600565
76), (106, -0.5), (768, -0.5), (802, -0.944911182523068), (904, -0.9607689228305226), (1313, -0.1889822365046136),
(2676, -0.6185895741317418), (4847, -0.5694947974514993), (5104, -0.7857142857142856), (5133, -0.8934051474415645),
(5630, -0.8660254037844386), (6127, 0.9707253433941508), (6296, 0), (6701, 0.7205766921228925), (7135, 0.8660254037
844448), (7291, 0.8660254037844356), (7315, -0.5), (7440, -1.0), (7618, -0.8386278693775345), (7964, 0.866025403784
4386), (8379, 0.8660254037844356), (8645, -0.8660254037844379), (8793, -0.3273268353539884), (9182, -0.327326835353
9889), (10136, 0.0), (10298, -0.8660254037844356)])
```

In [52]:
```
pearsonDF = pd.DataFrame.from_dict(pearsonCorrelationDict, orient='index')
pearsonDF.columns = ['similarityIndex']
pearsonDF['userId'] = pearsonDF.index
pearsonDF.index = range(len(pearsonDF))
pearsonDF.head()
```

Out[52]:

	similarityIndex	userId
0	-0.456435	58040
1	-0.053916	204165
2	-0.529150	815
3	0.233027	4099
4	-0.438786	5045

The top x similar users to input user

Now let's get the top 50 users that are most similar to the input.

In [53]:
```
topUsers=pearsonDF.sort_values(by='similarityIndex', ascending=False)[0:50]
topUsers.head()
```

Out[53]:

	similarityIndex	userId
85	0.970725	6127
88	0.866025	7135
93	0.866025	7964
94	0.866025	8379
89	0.866025	7291

Now, let's start recommending movies to the input user.

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.

In [54]:
```
topUsersRating=topUsers.merge(ratings_df, left_on='userId', right_on='userId', how='inner')
topUsersRating.head()
```

Out[54]:

	similarityIndex	userId	movieId	rating
0	0.970725	6127	1	3.5
1	0.970725	6127	2	3.0
2	0.970725	6127	3	3.5
3	0.970725	6127	15	3.5
4	0.970725	6127	19	3.5

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.

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:

In [55]:
```
#Multiplies the similarity by the user's ratings
topUsersRating['weightedRating'] = topUsersRating['similarityIndex']*topUsersRating['rating']
topUsersRating.head()
```

Out[55]:

	similarityIndex	userId	movieId	rating	weightedRating
0	0.970725	6127	1	3.5	3.397539
1	0.970725	6127	2	3.0	2.912176
2	0.970725	6127	3	3.5	3.397539
3	0.970725	6127	15	3.5	3.397539
4	0.970725	6127	19	3.5	3.397539

In [56]:
```
#Applies a sum to the topUsers after grouping it up by userId
tempTopUsersRating = topUsersRating.groupby('movieId').sum()[['similarityIndex','weightedRating']]
tempTopUsersRating.columns = ['sum_similarityIndex','sum_weightedRating']
tempTopUsersRating.head()
```

Out[56]:

movieId	sum_similarityIndex	sum_weightedRating
1	-1.114080	-11.119641
2	-1.654284	-1.466968
3	-1.528542	-2.318972
4	-1.618445	-2.478415
5	-1.255754	-4.144140

In [57]:
```
#Creates an empty dataframe
recommendation_df = pd.DataFrame()
#Now we take the weighted average
recommendation_df['weighted average recommendation score'] = tempTopUsersRating['sum_weightedRating']/tempTopUsersRat
recommendation_df['movieId'] = tempTopUsersRating.index
recommendation_df.head()
```

Out[57]:

movieId	weighted average recommendation score	movieId
1	9.981009	1
2	0.886769	2
3	1.517114	3
4	1.531356	4
5	3.300120	5

Now let's sort it and see the top 20 movies that the algorithm recommended!

In [58]:
```
recommendation_df = recommendation_df.sort_values(by='weighted average recommendation score', ascending=False)
recommendation_df.head(10)
```

Out[58]:

movieId	weighted average recommendation score	movieId
130052	9999.948114	130052
94985	641.548439	94985
4732	345.129884	4732
72641	315.193688	72641
6567	218.219862	6567
7408	156.524715	7408
52319	101.659099	52319
59404	84.982711	59404
2380	83.136985	2380
71252	72.607591	71252

In [59]:
```
movies_df.loc[movies_df['movieId'].isin(recommendation_df.head(10)['movieId'].tolist())]
```

Out[59]:

	movieId	title	year
2296	2380	Police Academy 3: Back in Training	1986
4637	4732	Bubble Boy	2001
6458	6567	Buffalo Soldiers	2001
7297	7408	Jack and the Beanstalk	1952
11787	52319	Inglorious Bastards (Quel maledetto treno blin...	1978
12660	59404	Meet Bill	2007
14230	71252	Final Destination, The (Final Destination 4) (...	2009
14530	72641	Blind Side, The	2009
19146	94985	Get the Gringo	2012
28100	130052	Clown	2014

Advantages and Disadvantages of Collaborative Filtering

Advantages

- Takes other user's ratings into consideration
- Doesn't need to study or extract information from the recommended item
- Adapts to the user's interests which might change over time

Disadvantages

- Approximation function can be slow
- There might be a low of amount of users to approximate
- Privacy issues when trying to learn the user's preferences