Filtros colaborativos

Objetivos

- Crear sistemas de recomendación basados en filtros colaborativos

Tabla de contenido

Adquiriendo los datos

In [2]:
```
#!wget -O moviedataset.zip https://cf-courses-data.s3.us.cloud-object-storage.appdoma
import urllib.request
url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDevelope
filename = 'moviedataset.zip'
urllib.request.urlretrieve(url, filename)

print('unziping ...')
#!unzip -o -j moviedataset.zip

from zipfile import ZipFile

# Create a ZipFile Object and load moviedataset.zip in it
with ZipFile('moviedataset.zip', 'r') as zipObj:
   # Extract all the contents of zip file in current directory
   zipObj.extractall()
```
```
unziping ...
```

Pre-procesamiento

In [3]:
```
#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
```

Leamos cada archivo en dataframes:

In [5]:
```
# Información de peliculas
movies_df = pd.read_csv('ml-latest/movies.csv')
# Información de usuarios
ratings_df = pd.read_csv('ml-latest/ratings.csv')
```

Veamos cómo están organizados:

In [6]:
```
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

Cada película tiene un ID único, un título con su año de lanzamiento en él (puede contener caracteres unicode) y varios géneros diferentes en el mismo campo. Quitemos el año del título y pongámoslo en su propio campo.

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
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
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
```

Miremos el resultado.

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

Eliminemos la columna de género, ya que no la necesitaremos para este sistema de recomendación en particular.

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

Aquí el dataframe final de películas:

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

Miremos el dataframe de ratings.

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

Cada fila en el dataframe de ratings tiene un id de usuario asociado con al menos una película, un rating y una marca de tiempo. No necesitaremos la marca de tiempo así que la eliminaremos para ahorrar memoria.

In [12]:
```
ratings_df = ratings_df.drop('timestamp', 1)
```

Aquí el dataframe de ratings final:

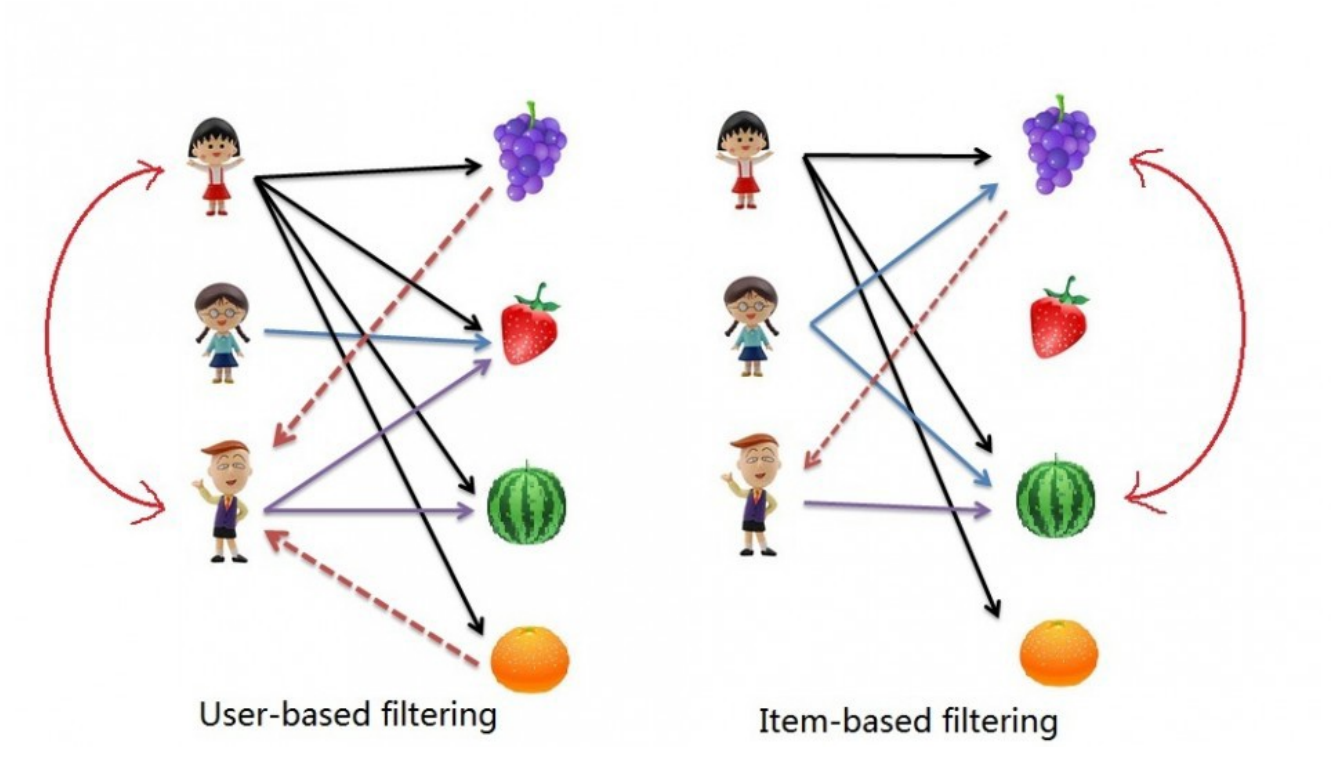
In [13]:
```
ratings_df.head()
```

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

Filtrado colaborativo

La primera técnica que veremos se llama **Filtrado colaborativo** y también es conocida como **User-User Filtering**. La misma usa otros usuarios para recomendar items al usuario de entrada. Intenta encontrar usuarios que tienen preferencias y opiniones similares al de entrada y le recomineda items que les hayan gustado a los primeros a este último. Hay variso métodos de encontrar usuarios similares, aquí usaremos la **función de correlación de Pearson**.



El proceso para crear un sistema de recomendación basado en usuarios es:

- Seleccionar un usuario con las películas que el usuario ha visto.
- Basado en si rating de películas, encontrar los X vecinos superiores.
- Obtener el registro de la película vista del usuario para cada vecino.
- Calcular un puntaje de similaridad usando alguna fórmula.
- Recomendar los items con el mayor puntaje.

Comencemos creando una entrada de usuario para recomendarle películas

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}
         ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

Out[14]:

	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

Agregar movieID a la entrada de usuario

Obtengamos los IDs de películas del dataframe movies y agreguémoslos.

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('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

Los usuarios que han visto las mismas películas

Ahora, con los IDs de películas en nuestra entrada, podemos obtener el subconjunto de usuarios que han visto y calificado las películas en nuestra entrada.

In [16]:
```
# Filtramos los usuarios que han visto las películas que la entrada ha visto y lo gua
userSubset = ratings_df[ratings_df['movieId'].isin(inputMovies['movieId'].tolist())]
userSubset.head()
```

Out[16]:

	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

Agrupamos las filas por user ID.

In [17]:
```
# Groupby crea varios sub-dataframes donde todos tienen el mismo valor en la columna e
userSubsetGroup = userSubset.groupby(['userId'])
```

veamos uno de los usuarios, por ejemplo, el que tiene userID=1130

In [18]:
```
userSubsetGroup.get_group(1130)
```

Out[18]:

	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

Ordenemos estos grupos para que los usuarios que comparten la mayor cantidad de películas en común con la entrada tengan mayor prioridad. Esto brinda una mejor recomendación ya que no recorremos cada usuario.

In [19]:
```
# Ordenando para que los usuarios con más películas en común con el usuario tengan pr
userSubsetGroup = sorted(userSubsetGroup,  key=lambda x: len(x[1]), reverse=True)
```

Miremos el primer usuario

In [20]:
```
userSubsetGroup[0:3]
```

Out[20]:
```
[(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
  9198     106     1274     3.0
  9238     106     1968     3.5),
 (686,
         userId  movieId  rating
  61336     686        1     4.0
  61337     686        2     3.0
  61377     686      296     4.0
  61478     686     1274     4.0
  61569     686     1968     5.0)]
```

Similaridad de usuarios con el usuario de entrada

Luego, vamos a comparar todos los usuarios (no todos realmente !!) con nuestro usuario especificado y encontrar es el que es más similar. Encontraremos que tan similar es cada usuario a la entrada a través del **coeficiente de correlación de Pearson**, que es utilizado para medir la fuerza de la relación lineal entre 2 variables. La fórmula puede verse en la imagen de abajo.

Por qué la correlación de Pearson?

La correlación de Pearson es invariante al escalado, es decir, multiplicar todos los elementos por una constante distinta de 0 o agregar una constante a todos los elementos. Por ejemplo, si tiene 2 vectores X e Y, pearson(X, Y) == pearson(X, 2 * Y + 3). Esta es una propiedad importante en un sistema de recomendación porque por ejemplo 2 usuarios podrían calificar 2 series de items de forma totalmente diferente en términos de tasas absolutas, pero serían usuarios similares (con las mismas ideas) con tasas similares en varias escalas.

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

Los valores dados por la fórmula varían desde r=-1 a r=1, donde 1 significa una correlación directa entre las 2 entidades (una correlación positiva perfecta) y -1 una correlación negativa perfecta.

En nuestro caso, 1 significa que los 2 usuarios tienen gustos similares y -1 que tienen gustos opuestos.

Seleccionaremos un subconjunto de usuarios para iterar. Este límite está impuesto porque no queremos perder demasiado tiempo recorriendo cada usuario.

In [21]:
```
userSubsetGroup = userSubsetGroup[0:100]
```

Ahora calcularemos la correlación de Pearson entre la entrada de usuario y el subconjunto y lo almacenaremos en un diccionario, donde la clave es el user Id y el valor el coeficiente

In [22]:
```
# Almacenamos la correlación de Pearson en un diccionario, la clave es el user Id y e
pearsonCorrelationDict = {}

# Por cada grupo de usuario en nuestro subconjunto
for name, group in userSubsetGroup:
    # Ordenamos la entrada y el grupo de usuario actual para que los valores no se me
    group = group.sort_values(by='movieId')
    inputMovies = inputMovies.sort_values(by='movieId')
    # Obtenemos N para la fórmula
    nRatings = len(group)
    # Obtenemos los puntajes para las películas que ambos tienen en común
    temp_df = inputMovies[inputMovies['movieId'].isin(group['movieId'].tolist())]
    # Lo almacenamos en un buffer temporal en formato lista para facilitar futuros cá
    tempRatingList = temp_df['rating'].tolist()
    # Ponemos las calificaciobes del grupo de usuarios actual en formato de lista
    tempGroupList = group['rating'].tolist()
    # Calculmos la correlación de Pearson entre los 2 usuarios, x e y
    Sxx = sum([i**2 for i in tempRatingList]) - pow(sum(tempRatingList),2)/float(nRat
    Syy = sum([i**2 for i in tempGroupList]) - pow(sum(tempGroupList),2)/float(nRating
    Sxy = sum( i*j for i, j in zip(tempRatingList, tempGroupList)) - sum(tempRatingLis

    # Si el denominador es diferente de 0 dividimos, sino la correlación es 0
    if Sxx != 0 and Syy != 0:
        pearsonCorrelationDict[name] = Sxy/sqrt(Sxx*Syy)
    else:
        pearsonCorrelationDict[name] = 0
```

In [23]:
```
pearsonCorrelationDict.items()
```

Out[23]:
```
dict_items([(75, 0.8272781516947562), (106, 0.5860090386731182), (686, 0.8320502943378
437), (815, 0.5765566601970551), (1040, 0.9434563530497265), (1130, 0.289157465983120
1), (1502, 0.8770580193070299), (1599, 0.4385290096535153), (1625, 0.716114874039432),
(1950, 0.179028718509858), (2065, 0.4385290096535153), (2128, 0.5860090386731196), (24
32, 0.1386750490563073), (2791, 0.8770580193070299), (2839, 0.8204126541423674), (294
8, -0.11720180773462392), (3025, 0.45124262819713973), (3040, 0.89514359254929), (318
6, 0.6784622064861935), (3271, 0.26989594817970664), (3429, 0.0), (3734, -0.1504142093
9904673), (4099, 0.05860090386731196), (4208, 0.29417420270727607), (4282, -0.43852900
96535115), (4292, 0.6564386345361464), (4415, -0.11183835382312353), (4586, -0.9024852
563942795), (4725, -0.08006407690254357), (4818, 0.4885967564883424), (5104, 0.7674257
668936507), (5165, -0.4385290096535153), (5547, 0.17200522903844556), (6082, -0.047287
79924109591), (6207, 0.9615384615384616), (6366, 0.6577935144802716), (6482, 0.0), (65
30, -0.3516054232038709), (7235, 0.6981407669689391), (7403, 0.11720180773462363), (76
41, 0.7161148740394331), (7996, 0.626600514784504), (8008, -0.22562131409856986), (808
6, 0.6933752452815365), (8245, 0.0), (8572, 0.8600261451922278), (8675, 0.537086155529
5773), (9101, -0.08600261451922278), (9358, 0.692178738358485), (9663, 0.1939727250419
52), (9994, 0.5030272728659587), (10248, -0.24806946917841693), (10315, 0.537086155529
574), (10368, 0.4688072309384945), (10607, 0.41602514716892186), (10707, 0.96153846153
84616), (10863, 0.6020183016345595), (11314, 0.8204126541423654), (11399, 0.5172606001
11872), (11769, 0.9376144618769914), (11827, 0.4902903378454601), (12069, 0.0), (1212
0, 0.9292940047327363), (12211, 0.8600261451922278), (12325, 0.9616783115081544), (129
16, 0.5860090386731196), (12921, 0.6611073566849309), (13053, 0.9607689228305227), (13
142, 0.6016568375961863), (13260, 0.7844645405527362), (13366, 0.8951435925492911), (1
3768, 0.8770580193070289), (13888, 0.2508726030021272), (13923, 0.3516054232038718),
(13934, 0.17200522903844556), (14529, 0.7417901772340937), (14551, 0.537086155529574),
(14588, 0.21926450482675766), (14984, 0.716114874039432), (15137, 0.5860090386731196),
(15157, 0.9035841064985974), (15466, 0.7205766921228921), (15670, 0.516015687115336),
(15834, 0.22562131409856986), (16292, 0.6577935144802716), (16456, 0.716114874039433
1), (16506, 0.5481612620668942), (17246, 0.48038446141526137), (17438, 0.7093169886164
387), (17501, 0.8168748513121271), (17502, 0.8272781516947562), (17666, 0.768923834017
6859), (17735, 0.7042381820123422), (17742, 0.3922322702763681), (17757, 0.64657575013
984), (17854, 0.537086155529574), (17897, 0.8770580193070289), (17944, 0.2713848825944
774), (18301, 0.29838119751643016), (18509, 0.1322214713369862)])
```

In [24]:
```
pearsonDF = pd.DataFrame.from_dict(pearsonCorrelationDict, orient='index')
pearsonDF.columns = ['similarityIndex']
pearsonDF['userId'] = pearsonDF.index
pearsonDF.index = range(len(pearsonDF))
pearsonDF.head()
```

Out[24]:

	similarityIndex	userId
0	0.827278	75
1	0.586009	106
2	0.832050	686
3	0.576557	815
4	0.943456	1040

Los primeros x usuarios similares al de entrada

Obtengamos los primeros 50 usuarios que sean más similares al de entrada.

In [25]:
```
topUsers=pearsonDF.sort_values(by='similarityIndex', ascending=False)[0:50]
topUsers.head()
```

Out[25]:

	similarityIndex	userId
64	0.961678	12325
34	0.961538	6207
55	0.961538	10707
67	0.960769	13053
4	0.943456	1040

Comencemos a recomendar películas para el usuario de entrada.

Puntuación de los usuarios seleccionados a todas las películas

Haremos esto tomando el promedio ponderado de los ratings de las películas usando la correlación de Pearson como peso. Para hacer esto, primero necesitamos obtener las películas vistas por los usuarios en nuestro **pearsonDF** del dataframe de ratings y luego almacenar su correlación en una nueva columna llamada _similarityIndex. Esto se logra combinando las 2 tablas.

In [26]:
```
topUsersRating=topUsers.merge(ratings_df, left_on='userId', right_on='userId', how='in
topUsersRating.head()
```

Out[26]:

	similarityIndex	userId	movieId	rating
0	0.961678	12325	1	3.5
1	0.961678	12325	2	1.5
2	0.961678	12325	3	3.0
3	0.961678	12325	5	0.5
4	0.961678	12325	6	2.5

Ahora sólo necesitamos multiplicar los ratings de películas por sus pesos (índice de similaridad), luego sumar los nuevos ratings y dividirlos por la suma de los pesos.

Podemos hacer esto multiplicando 2 columnas, luego agrupando el dataframe por movieId y luego dividiendo 2 columnas:

Muestra las idea de todos los usuarios similares para las películas candidatas para el usuario de entrada:

In [27]:
```
# Multiplica la similitud por las calificaciones del usuario
topUsersRating['weightedRating'] = topUsersRating['similarityIndex']*topUsersRating['r
topUsersRating.head()
```

Out[27]:

	similarityIndex	userId	movieId	rating	weightedRating
0	0.961678	12325	1	3.5	3.365874
1	0.961678	12325	2	1.5	1.442517
2	0.961678	12325	3	3.0	2.885035
3	0.961678	12325	5	0.5	0.480839
4	0.961678	12325	6	2.5	2.404196

In [28]:
```
# Aplica una suma a los topUsers después de agruparlo por userId
tempTopUsersRating = topUsersRating.groupby('movieId').sum()[['similarityIndex','weigh
tempTopUsersRating.columns = ['sum_similarityIndex','sum_weightedRating']
tempTopUsersRating.head()
```

Out[28]:

movieId	sum_similarityIndex	sum_weightedRating
1	38.376281	140.800834
2	38.376281	96.656745
3	10.253981	27.254477
4	0.929294	2.787882
5	11.723262	27.151751

In [29]:
```
# Crea un dataframe vacío
recommendation_df = pd.DataFrame()
# Ahora podemos tomar el promedio ponderado
recommendation_df['weighted average recommendation score'] = tempTopUsersRating['sum_w
recommendation_df['movieId'] = tempTopUsersRating.index
recommendation_df.head()
```

Out[29]:

movieId	weighted average recommendation score	movieId
1	3.668955	1
2	2.518658	2
3	2.657941	3
4	3.000000	4
5	2.316058	5

Ahora ordenemos y veamos las 20 películas recomendadas por el algoritmo:

In [30]:
```
recommendation_df = recommendation_df.sort_values(by='weighted average recommendation
recommendation_df.head(10)
```

Out[30]:

movieId	weighted average recommendation score	movieId
5073	5.0	5073
3329	5.0	3329
2284	5.0	2284
26801	5.0	26801
6776	5.0	6776
6672	5.0	6672
3759	5.0	3759
3769	5.0	3769
3775	5.0	3775
90531	5.0	90531

In [31]:
```
movies_df.loc[movies_df['movieId'].isin(recommendation_df.head(10)['movieId'].tolist()
```

Out[31]:

	movieId	title	year
2200	2284	Bandit Queen	1994
3243	3329	Year My Voice Broke, The	1987
3669	3759	Fun and Fancy Free	1947
3679	3769	Thunderbolt and Lightfoot	1974
3685	3775	Make Mine Music	1946
4978	5073	Son's Room, The (Stanza del figlio, La)	2001
6563	6672	War Photographer	2001
6667	6776	Lagaan: Once Upon a Time in India	2001
9064	26801	Dragon Inn (Sun lung moon hak chan)	1992
18106	90531	Shame	2011

Ventajas y desventajas del filtro colaborativo

Ventajas

- Toma los ratings de otros usuarios en consideración
- No necesita estudiar o extraer información del item recomendado
- Se adapta a los intereses del usuario, que pueden cambiar con el tiempo

Desventajas

- La función de aproximación puede ser lenta
- Puede haber poca cantidad de usuarios para aproximar
- Problemas de privacidad cuando se intenta aprender las preferencias del usuario