Filtros basados en contenido

Objetivos

- Crear un sistema de recomendación utilizando filtros colaborativos

Los sistemas de recomendación son una colección de algoritmos utilizados para recomendar ítems a los usuarios basados en información tomada del usuario. Estos sistemas se han vuelto omnipresentes y son vistos comúnmente en tiendas online, bases de datos de películas, etc.

Tabla de contenido

Obteniendo los datos

In [1]:

```
!wget -O moviedataset.zip https://cf-courses-data.s3.us.cloud-object-storage.appdomai
print('unziping ...')
!unzip -o -j moviedataset.zip
```

```
--2021-01-12 12:38:15--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module 5/data/moviedatase
t.zip
Resolviendo cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)... 169.63.118.104
Conectando con cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-
data.s3.us.cloud-object-storage.appdomain.cloud)[169.63.118.104]:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 160301210 (153M) [application/zip]
Guardando como: “moviedataset.zip”

moviedataset.zip    100%[===================>] 152,88M   737KB/s    en 3m 4s

2021-01-12 12:41:22 (848 KB/s) - “moviedataset.zip” guardado [160301210/160301210]

unziping ...
Archive:  moviedataset.zip
  inflating: links.csv
  inflating: movies.csv
  inflating: ratings.csv
  inflating: README.txt
  inflating: tags.csv
```

Pre-procesamiento

In [2]:

```python
#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 un dataframe.

In [8]:

```python
# Información de las películas
movies_df = pd.read_csv('movies.csv')
# Información del usuario
ratings_df = pd.read_csv('ratings.csv')
#
movies_df.head()
```

Out[8]:

	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

Quitemos year de la columna **title** y guardémoslo en una nueva columna **year**

In [9]:

```python
# Usamos expresiones regulares para encontrar el año almacenado entre paréntesis
# Especificamos los paréntesis para no tener conflicto con películas que tienen años e
movies_df['year'] = movies_df.title.str.extract('(\(\d\d\d\d\))',expand=False)
# Quitamos los paréntesis
movies_df['year'] = movies_df.year.str.extract('(\d\d\d\d)',expand=False)
# Quitamos years de la columna "title"
movies_df['title'] = movies_df.title.str.replace('(\(\d\d\d\d\))', '')
# Aplicamos la función strip para deshacernos de cualquier espacio en blanco que pueda
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
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

Dividamos los valores en la columna **Genres** en una **lista de géneros** para simplificar el uso futuro. Esto puede lograrse aplicando la función split de str de Python sobre la columna correcta.

In [10]:

```python
# Cada género es separado por un | así que solo tenemos que llamar a la función split
movies_df['genres'] = movies_df.genres.str.split('|')
movies_df.head()
```

Out[10]:

	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

Ya que mantener géneros en una lista no es óptimo para la técnica utilizaremos One Hot Encoding para convertir la lista de géneros en un vector, donde cada columna corresponde a un posible valor de la característica. Esta codificación es necesaria para alimentar datos categóricos. En este caso, almacenamos cada género diferente en columnas que contienen o bien 1 o bien 0. 1 muestra que la película tiene ese género y 0 que no. Guardemos también este marco de datos en otra variable, ya que los géneros no serán importantes para nuestro primer sistema de recomendación.

In [11]:

```python
# Copiamos el dataframe de películas a uno nuevo ya que en este primer caso no necesi
# de género
moviesWithGenres_df = movies_df.copy()

# Para cada fila en el dataframe, iteramos a través de la lista de géneros y ponemos u
# correspondiente
for index, row in movies_df.iterrows():
    for genre in row['genres']:
        moviesWithGenres_df.at[index, genre] = 1


# Rellenamos los valores NaN con 0 para mostrar que la película no tiene ese género
moviesWithGenres_df = moviesWithGenres_df.fillna(0)
moviesWithGenres_df.head()
```

Out[11]:

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy	Romance	...	Ho
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995	1.0	1.0	1.0	1.0	1.0	0.0	...	
1	2	Jumanji	[Adventure, Children, Fantasy]	1995	1.0	0.0	1.0	0.0	1.0	0.0	...	
2	3	Grumpier Old Men	[Comedy, Romance]	1995	0.0	0.0	0.0	1.0	0.0	1.0	...	
3	4	Waiting to Exhale	[Comedy, Drama, Romance]	1995	0.0	0.0	0.0	1.0	0.0	1.0	...	
4	5	Father of the Bride Part II	[Comedy]	1995	0.0	0.0	0.0	1.0	0.0	0.0	...	

5 rows × 24 columns

Veamos el dataframe de ratings.

In [12]:

```python
ratings_df.head()
```

Out[12]:

	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 indicando cuándo la calificó. No necesitaremos la columna de marca de tiempo, así que borrémosla para ahorrar memoria.

In [13]:

```python
#Drop removes a specified row or column from a dataframe
ratings_df = ratings_df.drop('timestamp', 1)
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

Sistema de recomendación basado en contenido

Veamos cómo implementar un sistema de recomendación basado en contenido. Esta técnica intenta averiguar cuáles son los aspectos favoritos de un usuario de un item, y luego recomendar items que presenten esas características. En nuestro caso intentaremos averiguar los géneros favoritos.

Comencemos creando un usuario de entrada para recomendar películas:

Nota. Para agregar más películas, simplemente incremente la cantidad de elementos en **userInput**.

In [14]:

```python
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 input user

Con la entrada completa, extraigamos los IDs de las películas del dataframe de películas y agreguémoslos.

Podemos lograr esto filtrando las filas que contienen los títulos de las películas y combinándolos con el dataframe de entrada. También eliminaremos las columnas innecesarias.

In [15]:

```python
# Filtramos las películas por título
inputId = movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]
# Combinamos para obtener el movieId. Es implícitamente combinado por title.
inputMovies = pd.merge(inputId, inputMovies)
# Eliminamos información que no vamos a utilizar
inputMovies = inputMovies.drop('genres', 1).drop('year', 1)
# Dataframe de entrada final
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

Vamos a comenzar por aprender las preferencias de entrada, así que obtengamos el subconjunto de películas que la entrada (input) ha visto desde el dataframe que contiene géneros definidos con valores binarios.

In [16]:

```python
#Filtramos las películas de la entrada
userMovies = moviesWithGenres_df[moviesWithGenres_df['movieId'].isin(inputMovies['movi
userMovies
```

Out[16]:

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy	Romance	...
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995	1.0	1.0	1.0	1.0	1.0	0.0	...
1	2	Jumanji	[Adventure, Children, Fantasy]	1995	1.0	0.0	1.0	0.0	1.0	0.0	...
293	296	Pulp Fiction	[Comedy, Crime, Drama, Thriller]	1994	0.0	0.0	0.0	1.0	0.0	0.0	...
1246	1274	Akira	[Action, Adventure, Animation, Sci-Fi]	1988	1.0	1.0	0.0	0.0	0.0	0.0	...
1885	1968	Breakfast Club, The	[Comedy, Drama]	1985	0.0	0.0	0.0	1.0	0.0	0.0	...

5 rows × 24 columns

Sólo necesitamos la tabla de género, así que limpiemos esto un poco resteando el índice y eliminando las columnas movieId, title, genres e year.

In [17]:

```python
# Reseteamos el índice para evitar problemas futuros
userMovies = userMovies.reset_index(drop=True)
# Eliminamos lo innecesario
userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop
userGenreTable
```

Out[17]:

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Horror	Mys
0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	
3	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	

Ahora estamos prontos para aprender las preferencias de la entrada!

Para hacerlo convertiremos cada género en pesos. Hacemos esto utilizando los reviews de la entrada y multiplicandolos en la tabla de géneros y luego resumiendo la tabla resultante por columna. Esta operación es un producto punto entre una matriz y un vector, y puede lograrse con la función "dot" de Pandas.

In [18]:

```python
inputMovies['rating']
```

Out[18]:

```
0    3.5
1    2.0
2    5.0
3    4.5
4    5.0
Name: rating, dtype: float64
```

In [19]:

```python
# Producto punto para obtener los pesos
userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
# El perfil de usuario
userProfile
```

Out[19]:

```
Adventure             10.0
Animation              8.0
Children               5.5
Comedy                13.5
Fantasy                5.5
Romance                0.0
Drama                 10.0
Action                 4.5
Crime                  5.0
Thriller               5.0
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
```

Ahora tenemos los pesos para cada una de las preferencias del usuario. Esto se conoce como Perfil de Usuario. Usando esto, podemos recomendar películas que satisfagan las preferencias del usuario.

Comencemos extrayendo la tabla género del dataframe original:

In [20]:

```python
# Obtengamos los géneros de cada película en nuestro dataframe original
genreTable = moviesWithGenres_df.set_index(moviesWithGenres_df['movieId'])
# Eliminemos la información innecesaria
genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop('ye
genreTable.head()
```

Out[20]:

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Horro
movieId											
1	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.
2	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.
3	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.
4	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.

In [21]:

```python
genreTable.shape
```

Out[21]:

```
(34208, 20)
```

Con el perfil del usuario y una lista completa de películas y su género a mano, vamos a tomar el promedio ponderado de cada película basado en el perfil de entrada y recomendar las primeras 20 películas que más lo satisfagan.

In [22]:

```python
# Multiplicamos los géneros por los pesos y luego tomamos el promedio ponderado
recommendationTable_df = ((genreTable*userProfile).sum(axis=1))/(userProfile.sum())
recommendationTable_df.head()
```

Out[22]:

```
movieId
1    0.594406
2    0.293706
3    0.188811
4    0.328671
5    0.188811
dtype: float64
```

In [23]:

```python
# Ordenamos nuestras recomendaciones en orden descendente
recommendationTable_df = recommendationTable_df.sort_values(ascending=False)
# Un vistazo de los valores
recommendationTable_df.head()
```

Out[23]:

```
movieId
5018      0.748252
26093     0.734266
27344     0.720280
148775    0.685315
6902      0.678322
dtype: float64
```

La tabla de recomendación!

In [24]:

```python
# La tabla de recomendación final
movies_df.loc[movies_df['movieId'].isin(recommendationTable_df.head(20).keys())]
```

Out[24]:

	movieId	title	genres	year
664	673	Space Jam	[Adventure, Animation, Children, Comedy, Fanta...	1996
1824	1907	Mulan	[Adventure, Animation, Children, Comedy, Drama...	1998
2902	2987	Who Framed Roger Rabbit?	[Adventure, Animation, Children, Comedy, Crime...	1988
4923	5018	Motorama	[Adventure, Comedy, Crime, Drama, Fantasy, Mys...	1991
6793	6902	Interstate 60	[Adventure, Comedy, Drama, Fantasy, Mystery, S...	2002
8605	26093	Wonderful World of the Brothers Grimm, The	[Adventure, Animation, Children, Comedy, Drama...	1962
8783	26340	Twelve Tasks of Asterix, The (Les douze travau...	[Action, Adventure, Animation, Children, Comed...	1976
9296	27344	Revolutionary Girl Utena: Adolescence of Utena...	[Action, Adventure, Animation, Comedy, Drama, ...	1999
9825	32031	Robots	[Adventure, Animation, Children, Comedy, Fanta...	2005
11716	51632	Atlantis: Milo's Return	[Action, Adventure, Animation, Children, Comed...	2003
11751	51939	TMNT (Teenage Mutant Ninja Turtles)	[Action, Adventure, Animation, Children, Comed...	2007
13250	64645	The Wrecking Crew	[Action, Adventure, Comedy, Crime, Drama, Thri...	1968
16055	81132	Rubber	[Action, Adventure, Comedy, Crime, Drama, Film...	2010
18312	91335	Gruffalo, The	[Adventure, Animation, Children, Comedy, Drama]	2009
22778	108540	Ernest & Célestine (Ernest et Célestine)	[Adventure, Animation, Children, Comedy, Drama...	2012
22881	108932	The Lego Movie	[Action, Adventure, Animation, Children, Comed...	2014
25218	117646	Dragonheart 2: A New Beginning	[Action, Adventure, Comedy, Drama, Fantasy, Th...	2000
26442	122787	The 39 Steps	[Action, Adventure, Comedy, Crime, Drama, Thri...	1959
32854	146305	Princes and Princesses	[Animation, Children, Comedy, Drama, Fantasy, ...	2000
33509	148775	Wizards of Waverly Place: The Movie	[Adventure, Children, Comedy, Drama, Fantasy, ...	2009

Ventajas y desventajas

Ventajas

- Aprende las preferencias del usuario
- Altamente personalizado por el usuario

Desventajas

- No toma en cuenta lo que otros piensen del item, por lo que pueden llegar a recomendarse items de baja calidad
- Extraer los datos no siempre es intuitivo
- Determinar qué características del item al usuario le gustan o no no siempre es obvio