



Universidad  
Internacional  
Menéndez Pelayo

Máster Universitario en Investigación en Inteligencia Artificial

Curso 2020-2021

**Sistemas de Recomendación**

# **Recomendación para grupos en Python**

10 de mayo de 2021

**Laura Rodríguez Navas**

**DNI: 43630508Z**

**e-mail:** [rodrigueznava@posgrado.uimp.es](mailto:rodrigueznava@posgrado.uimp.es)

# Índice

<b>1. Introducción</b>	<b>3</b>
1.1. Filtrado Colaborativo . . . . .	3
1.2. Ventajas y Desventajas del Filtrado Colaborativo . . . . .	4
<b>2. Conjunto de datos</b>	<b>5</b>
<b>3. Sistema de Recomendación</b>	<b>7</b>
3.1. Coeficiente de Correlación Pearson . . . . .	9
3.2. Predicción y Resultado . . . . .	12
<b>4. Conclusiones</b>	<b>15</b>
<b>Bibliografía</b>	<b>16</b>

# 1. Introducción

El crecimiento de Internet y de la información disponible en línea ha hecho que sea mucho más difícil extraer información útil de manera efectiva. La abrumadora cantidad de datos requiere de mecanismos de filtrado de información eficientes. Y uno de los sistemas utilizados para hacer frente a este problema son los sistemas de recomendación.

Los sistemas de recomendación pueden definirse como herramientas diseñadas para interactuar con conjuntos de información grandes y complejos con la finalidad de proporcionar al usuario información o elementos que sean de su interés, todo ello de forma automatizada. Su funcionamiento se basa en el empleo de métodos matemáticos y estadísticos capaces de explotar la información previamente almacenada y crear recomendaciones adaptadas a cada usuario. En la actualidad, los sistemas de recomendación son una tecnología implementada en la mayoría de plataformas online como *Amazon*, *Neflix*, *Spotify*, etc; ya que han dado muy buenos resultados incrementando las ventas. Los sistemas de recomendación también están presentes en muchos otros ámbitos, como por ejemplo el de las noticias, mostrando al usuario noticias que le interesan. La mayoría de los sistemas de recomendación se pueden clasificar en tres grupos: **basados en contenido**, **filtrado colaborativo** y **híbridos** (combinación de los dos anteriores).

En este documento se describe el desarrollo práctico en Python de un sistema de recomendación. El sistema de recomendación elegido aplica la técnica de filtrado colaborativo (*user-based*). La implementación se puede encontrar en [1], la cual está formada por el desarrollo del programa que implementa un sistema de recomendación de **filtrado colaborativo** basado en usuarios.

El documento se divide en diferentes secciones donde se va describiendo paso a paso el trabajo realizado. En la primera parte del documento se describe la técnica de filtrado colaborativo elegida, con sus ventajas y desventajas (ver secciones 1.1 y 1.2). El documento sigue con la descripción del conjunto de datos que usamos con el sistema de recomendación y cómo podemos adquirirlo (ver sección 2).

En la sección 3 del documento se describe paso a paso la implementación en Python del sistema de recomendación de filtrado colaborativo basado en usuarios. En esta sección también se describe el **coeficiente de Correlación de Pearson** y porqué se ha elegido como medida de similitud para encontrar las correlaciones entre las valoraciones de los usuarios representados por el conjunto de datos y un nuevo usuario que se añade para recomendarle elementos de su interés respecto a las valoraciones de los otros usuarios (ver sección 3.1). En la siguiente sección, la 3.2, se presentan las recomendaciones que crea el sistema de recomendación para el nuevo usuario y finalmente, en la parte final del documento se añaden las conclusiones y la bibliografía.

## 1.1. Filtrado Colaborativo

La técnica de filtrado colaborativo basada en usuarios (*collaborative filtering user-based*) se fundamenta en la idea de *muéstrame cosas que le hayan gustado a gente parecida a mí*. Para que este tipo de sistema de recomendación funcione se necesita disponer de las valoraciones de los usuarios para poder agruparlos por similitud. Sin embargo, no se necesita conocer la descripción detallada de los elementos que se quieren recomendar. La limitación de este tipo de sistema aparece en el momento en que se une un nuevo usuario, ya que, hasta que no se dispone de suficiente información sobre sus valoraciones, no se pueden hacer recomendaciones. Este problema se denomina el problema del **comienzo frío** (*cold start*).

En esta práctica el Filtrado Colaborativo (FC) nos permitirá crear recomendaciones personalizadas a un nuevo usuario, encontrando elementos de su interés con el análisis de datos de otros usuarios; asumiendo que los usuarios con intereses parecidos tienden a hacer valoraciones de forma similar. Es decir, para predecir la valoración que un usuario *A* hará de un elemento *X* que todavía no ha visto, se buscan usuarios con intereses similares a *A* y se utilizan las valoraciones de estos otros usuarios sobre el elemento *X* como estimación de la valoración

del usuario A. En resumen, el filtrado colaborativo basado en usuarios es un método para hacer predicciones automáticas (filtrado) sobre los intereses de un usuario mediante la recopilación de las valoraciones de muchos usuarios (colaboradores). Estos sistemas usualmente siguen una metodología que puede reducirse en:

1. La búsqueda de usuarios que comparten los mismos patrones de valoración con el usuario activo (el usuario para el que se está haciendo la predicción).
2. Utilizar las valoraciones por parte de aquellos usuarios afines que se encuentran en el paso 1 al fin de calcular una predicción para el usuario activo.

En la sección 3 se describe detalladamente la metodología desarrollada en Python del sistema de recomendación de filtrado colaborativo basado en usuarios.

## 1.2. Ventajas y Desventajas del Filtrado Colaborativo

Los SR colaborativos tienen importantes ventajas respecto a los no colaborativos:

- Dan un mayor soporte para el filtrado de elementos cuyo contenido no es fácil de analizar por procesos automatizados.
- Dan la posibilidad de filtrar elementos basándose en su calidad o preferencias.
- Se adaptan a los intereses del usuario si estos cambian con el tiempo.
- Realizan recomendaciones válidas, pero que no esperábamos, lo cual puede resultar de gran utilidad.

Sin embargo, también presentan inconvenientes:

- Son difíciles de aplicar en grandes cantidades de usuarios.
- A veces no encuentran suficientes usuarios para realizar recomendaciones a un nuevo usuario, cuando el número de usuarios del conjunto de datos es bajo.
- No trabajan bien a la hora de filtrar información para necesidades de contenido específicas, o cuando tenemos pocos datos de un nuevo usuario. Este problema se ha comentado en la sección anterior, se denomina problema del comienzo frío.

Para aliviar los inconvenientes se suele recurrir a sistemas de recomendación híbridos, que utilizan a la vez sistemas de recomendación de filtrado colaborativo y sistemas de recomendación basados en contenido (*content-based filtering*). Los sistemas de recomendación basados en contenido se fundamentan en la idea de *muéstrame más cosas como las que me han gustado*. Es de esperar que, a partir de los elementos que han sido valorados positivamente por un usuario, se puedan recomendar a otros usuarios cuyas características se asemejen a los anteriores.

En esta práctica no me ha parecido adecuado el desarrollo de un sistema de recomendación híbrido, porque se me podría haber complicado el desarrollo en Python. Y al ser mi primera implementación en Python de un sistema de recomendación, mi decisión ha sido centrarme en un tipo de SR.

## 2. Conjunto de datos

El conjunto de datos que se ha usado en esta práctica se encuentra disponible públicamente para su descarga en el siguiente enlace: <https://www.kaggle.com/abhiakjha/movielens-100k/download>. Este conjunto de datos se llama *ml-latest-small*, contiene valoraciones de películas con valores entre 1 y 5 estrellas, y la actividad del etiquetado de MovieLens [2], un servicio de recomendación de películas. Exactamente, el conjunto de datos contiene 100836 valoraciones y 3683 etiquetas de 9742 películas. Los datos fueron creados por 610 usuarios que se seleccionaron al azar. Cada usuario clasificó al menos 20 películas diferentes y está representado por un identificador.

Una vez descargado el conjunto de datos podemos ver que está contenido en los archivos *links.csv*, *movies.csv*, *ratings.csv* y *tags.csv*. Aunque para esta práctica solo se utilizan los archivos *movies.csv* y *ratings.csv*. Estos archivos se pueden encontrar en la carpeta *dataset* y en la carpeta con el mismo nombre en el repositorio [1].

Con la finalidad de mejorar las recomendaciones del sistema de recomendación, realizaremos un seguido de transformaciones en el conjunto de datos. Para empezar, cargamos los archivos *movies.csv* y *ratings.csv* dentro de un dataframe (ver Definición 1) con el uso de la librería pandas [3] de Python. Se ha decidido trabajar con esta librería porque permite leer y escribir fácilmente ficheros en formato CSV, acceder a los datos mediante índices o nombres para filas y columnas, ofrece métodos para reordenar, dividir y combinar conjuntos de datos, etc. y todo ello de forma muy eficiente.

```
movies_df = pd.read_csv('dataset/movies.csv')
ratings_df = pd.read_csv('dataset/ratings.csv')
```

Miramos el contenido de *movies\_df* para ver cómo ha quedado organizado:

movieId	title	genres
1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
2	Jumanji (1995)	Adventure Children Fantasy
3	Grumpier Old Men (1995)	Comedy Romance
4	Waiting to Exhale (1995)	Comedy Drama Romance
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action Crime Thriller
7	Sabrina (1995)	Comedy Romance
8	Tom and Huck (1995)	Adventure Children
9	Sudden Death (1995)	Action
10	GoldenEye (1995)	Action Adventure Thriller

Tabla 1: Contenido preliminar del dataframe *movies\_df*.

Observando la Tabla 1, vemos que cada película tiene un único identificador (*movieId*), un título, su año de estreno y diferentes géneros. Como los años del estreno de las películas contienen caracteres *unicode*, los sacaremos de la columna de los títulos y los ubicaremos en una nueva columna que nombraremos *year*. Para ello, primero creamos una expresión regular con el fin de seleccionar los años guardados entre paréntesis, y con la función *extract* de la librería pandas los extraemos para guardarlos en su propia columna. Después borramos los años en la columna de los títulos con la función *replace*, y para acabar, con la función *strip* nos aseguramos de sacar los espacios extra que pudieran haber quedado durante el proceso de transformación. Además, eliminamos la columna de los géneros con la función *drop*, ya que no la tendremos en cuenta más adelante en el sistema de recomendación.

```
regular_expression = r'\((.*?)\)'
movies_df['year'] = movies_df.title.str.lower().str.extract(regular_expression)
movies_df['title'] = movies_df.title.str.replace(regular_expression, '', regex=True)
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
movies_df = movies_df.drop('genres', 1)
```

Vemos el resultado:

movieId	title	year
1	Toy Story	1995
2	Jumanji	1995
3	Grumpier Old Men	1995
4	Waiting to Exhale	1995
5	Father of the Bride Part II	1995
6	Heat	1995
7	Sabrina	1995
8	Tom and Huck	1995
9	Sudden Death	1995
10	GoldenEye	1995

Tabla 2: Contenido del dataframe *movies\_df*.

**Definición 1.** *Un DataFrame es una estructura de datos bidimensional y etiquetada que acepta diferentes tipos de datos de entrada organizados en columnas. Se puede pensar en un DataFrame como una hoja de cálculo o una tabla SQL.*

Ahora, miramos el contenido de *ratings\_df* para ver cómo ha quedado organizado. Pero antes, eliminamos la columna *timestamp*, porque tampoco la tendremos en cuenta más adelante en el sistema de recomendación.

```
ratings_df = ratings_df.drop('timestamp', 1)
```

Así queda organizado el contenido en *ratings\_df*:

userId	movieId	rating
1	1	4.0
1	3	4.0
1	6	4.0
1	47	5.0
1	50	5.0
1	70	3.0
1	101	5.0
1	110	4.0
1	151	5.0
1	157	5.0

Tabla 3: Contenido del dataframe *ratings\_df*.

Observando la Tabla 3, vemos que cada fila del dataframe *ratings\_df* contiene un identificador de usuario (asociado con al menos una película) y la valoración que este ha realizado de cada una de ellas. Por ejemplo, vemos que el usuario 1 ha visto y valorado diferentes películas (1, 3, 6, 47, 50, etc.).

### 3. Sistema de Recomendación

En esta sección describimos detalladamente el sistema de recomendación desarrollado en Python [1], que como se ha comentado en la sección 1.1, utiliza la técnica de filtrado colaborativo basado en usuarios. Con esta técnica entrenaremos al sistema para predecir recomendaciones de películas a un nuevo usuario acordes a sus intereses. Para predecir estas recomendaciones, el sistema de recomendación buscará las similitudes entre las valoraciones introducidas por el nuevo usuario con las valoraciones de los otros usuarios ya existentes en el sistema. Es decir, el sistema de recomendación intentará encontrar usuarios que tengan valoraciones parecidas a las del nuevo usuario y entonces recomendarle películas acordes a sus valoraciones.

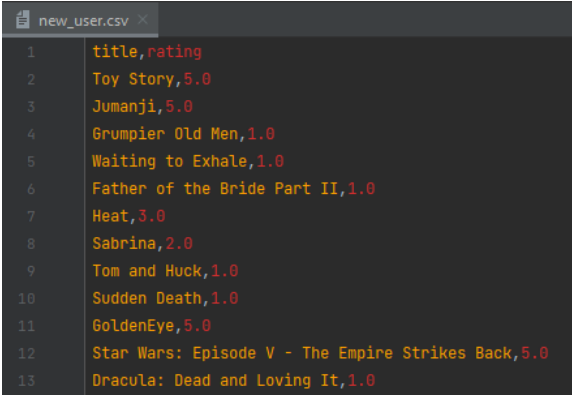
A la hora de entrenar el sistema de recomendación, es crítico entender de qué forma se están haciendo las valoraciones, porque dependiendo de ello, se pueden emplear varios métodos para definir y cuantificar la similitud entre los usuarios. En este caso, según el conjunto de datos anteriormente visto en la sección 2, las valoraciones son numéricas, es decir, las valoraciones se hacen empleando una escala numérica de 1 a 5 estrellas. Y personalmente el método que me pareció más adecuado para calcular la similitud entre los usuarios de este conjunto de datos fue un método que se basa en [correlación Pearson](#) (ver sección 3.1). También porque es uno de los métodos que más he utilizado con anterioridad a la realización de esta práctica.

El objetivo de este sistema de recomendación es recomendar 10 películas a un nuevo usuario, mediante la estrategia siguiente:

1. Crear un nuevo usuario identificando algunas de las películas del conjunto de datos que haya visto. Se asume que estas películas serán valoradas por el nuevo usuario.
2. Basándose en la selección de las películas vistas por el nuevo usuario, obtener las películas donde el nuevo usuario y los otros usuarios coinciden, es decir, encontrar las películas que tanto el nuevo usuario como sus vecinos (usuarios similares a él) hayan visto.
3. Calcular la similitud entre el nuevo usuario y sus vecinos en base a sus perfiles de valoración, es decir, utilizando los vectores formados por sus valoraciones. Se emplea la correlación de Pearson como medida de similitud.
4. Para cada una de las películas obtenidas en el paso 2:
  - Seleccionar los K=50 usuarios más parecidos al nuevo usuario (selección de vecindario), cuyo valor de similitud es positivo. En la práctica, el número óptimo de usuarios debería identificarse mediante validación cruzada, sin embargo, para no añadir una capa de complejidad extra a la práctica, he decidido emplear este valor.
  - Calcular la media ponderada de las valoraciones que los K=50 usuarios han dado a la película. Este valor se almacena como el valor predicho de la película.
5. Mostrar como recomendaciones las 10 películas con mayor valor predicho.

Afin de seguir la estrategia anterior, comenzamos creando un nuevo usuario a quien recomendar películas. Para ello, hemos creado el archivo *new\_user.csv* dentro de la carpeta *dataset*, y le hemos añadido los títulos de 100 películas elegidas aleatoriamente del conjunto de datos con una nueva valoración, que, en este caso, he valorado según mi criterio. El archivo *textitnew\_user.csv* se puede modificar como se desee para realizar tantas recomendaciones como se quiera, solo debemos asegurarnos de escribir bien los títulos de las películas, igual que aparecen en el archivo *dataset/movies.csv*, y que valoramos las películas con un valor entre 1 y 5 estrellas. En la Figura 1, podemos observar cómo se organiza parte del contenido del archivo *new\_user.csv*, y que cargamos en un nuevo dataframe que nombramos *user\_df*.

```
user_df = pd.read_csv('dataset/new_user.csv')
```



	title,rating
1	Toy Story,5.0
2	Jumanji,5.0
3	Grumpier Old Men,1.0
4	Waiting to Exhale,1.0
5	Father of the Bride Part II,1.0
6	Heat,3.0
7	Sabrina,2.0
8	Tom and Huck,1.0
9	Sudden Death,1.0
10	GoldenEye,5.0
11	Star Wars: Episode V - The Empire Strikes Back,5.0
12	Dracula: Dead and Loving It,1.0
13	

Figura 1: Datos del nuevo usuario.

Después de crear un nuevo dataframe con los datos del nuevo usuario, extraemos los títulos de las películas del conjunto de datos que el nuevo usuario haya visto, los guardamos en la variable *titles*, y los unimos a los datos del nuevo usuario almacenados en el dataframe *user\_df*. En este punto para ahorrar un poco de espacio en memoria, aprovechamos a eliminar la columna *year* de *user\_df*, ya que no se utilizará más adelante en el sistema de recomendación.

```
titles = movies_df[movies_df['title'].isin(user_df['title'].tolist())]
user_df = pd.merge(titles, user_df)
user_df = user_df.drop('year', 1)
```

Así queda organizado el contenido en *user\_df*:

movielf	title	rating
1	Toy Story	5.0
2	Jumanji	5.0
3	Grumpier Old Men	1.0
4	Waiting to Exhale	1.0
5	Father of the Bride Part II	1.0
6	Heat	3.0
7	Sabrina	2.0
915	Sabrina	2.0
8	Tom and Huck	1.0
9	Sudden Death	1.0

Tabla 4: Contenido del dataframe *user\_df*.

Observando la Tabla 4, vemos que cada fila del dataframe *user\_df* contiene un identificador de película, el título de la película y la valoración de esta realizada por el nuevo usuario.

Siguiendo con la estrategia y una vez añadidos los títulos de las películas a los datos del nuevo usuario, podemos obtener todos los usuarios que hayan visto las mismas películas, que además agruparemos por su identificador de usuario (*userId*) con el método *groupby*. En la Tabla 5, se observan todas las películas valoradas por uno de los usuarios, concretamente las películas valoradas por el usuario 525.

```
movies = ratings_df[ratings_df['movielf'].isin(user_df['movielf'].tolist())]
users = movies.groupby(['userId'])
```



userId	movieId	rating
525	1	4.0
525	2	3.5
525	34	3.0
525	39	4.5
525	48	3.0
525	62	3.5
525	107	3.5
525	150	4.0
525	223	3.5
525	377	3.5
525	480	4.0
525	595	3.5
525	915	3.5
525	1196	4.5

Tabla 5: Películas que ha visto y valorado el usuario 525.

Vemos que en total el usuario 525 ha valorado 14 películas, 12 de las cuales no han sido valoradas por el nuevo usuario (ver Tabla 4). Este hecho podría entorpecer la predicción de las recomendaciones, así que, para una mejor predicción, ordenaremos a los usuarios de tal forma que los que compartan la mayor cantidad de películas tengan más prioridad (usuarios comunes).

```
common_users = sorted(users, key=lambda x: len(x[1]), reverse=True)
```

### 3.1. Coeficiente de Correlación Pearson

Una vez encontrados los usuarios más comunes del conjunto de datos, ya podemos medir la similitud entre ellos para predecir las recomendaciones. El sistema de recomendación buscará correlaciones entre patrones de valoración sobre las películas que han valorado los usuarios y viendo cómo los usuarios se relacionan entre sí. Es decir, el sistema buscará las similitudes entre las valoraciones de las películas del nuevo usuario con el resto de usuarios para encontrar aquellos que se parecen más a este (usuarios similares). Para ello usaremos el [coeficiente de correlación Pearson](#). La fórmula para calcular el coeficiente de correlación Pearson sobre un estadístico muestral (ver Definición 2) se muestra a continuación:

$$r = \frac{\sum_1^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_1^n (x_i - \bar{x})^2} \sqrt{\sum_1^n (y_i - \bar{y})^2}} \quad (1)$$

Los valores dados por la fórmula (1) pueden variar entre [-1,1].

- Si  $r = 1$  (correlación positiva perfecta), entonces las valoraciones estarán perfectamente correlacionadas. Los usuarios tendrán los mismos intereses.
- Si  $0 < r < 1$ , entonces existe una correlación positiva. Los usuarios tendrán intereses parecidos.
- Si  $-1 < r < 0$ , entonces existe una correlación negativa. Los usuarios tendrán pocos intereses parecidos.
- Si  $r = -1$  (correlación negativa perfecta), entonces las valoraciones estarán inversamente correlacionadas. Los usuarios no tendrán los mismos intereses.

Se ha decidido usar el coeficiente de correlación Pearson como medida de similitud por una de sus propiedades. Esta propiedad nos dice que, si se multiplican todos los elementos por una constante distinta a cero o si se agrega cualquier constante a todos los elementos del coeficiente, este no cambiará con la escala. Por ejemplo, si tenemos dos vectores  $X$  e  $Y$ , entonces  $pearson(X, Y) = pearson(X, 2 \cdot Y + 3)$ . Esta es una propiedad muy importante en los sistemas de recomendación porque si dos usuarios valoran dos elementos de manera completamente diferente, pero son usuarios muy parecidos (con intereses similares) contaríamos con valoraciones muy parecidas en escalas variadas y esto crearía grandes problemas en la predicción de las recomendaciones. El coeficiente de correlación Pearson es una de las medidas de similitud más utilizadas en un sistema de recomendación cuando hay variaciones entre magnitudes y escala en las valoraciones, aunque pudiéramos obtener valores altos de correlación si existen pocos usuarios con valoraciones en común.

**Definición 2.** *En estadística un estadístico muestral es una medida cuantitativa, derivada de un conjunto de datos de una muestra, con el objetivo de estimar o inferir características de una población o modelo estadístico.*

A continuación, mostramos el código para calcular las similitudes entre los usuarios que se basa en los coeficientes de correlación Pearson. Los coeficientes los almacenaremos en un diccionario que llamaremos *pearsonCorrelationDict*, donde las claves serán los identificadores de los usuarios y los valores de cada clave serán los coeficientes. Elegimos un subconjunto de usuarios (*usersSubset*) para realizar menos iteraciones y no añadir demasiado sobrecoste computacional. El subconjunto de usuarios está formado por 100 usuarios.

```
usersSubset = common_users[0:100]
pearsonCorrelationDict = {}

for id, group in usersSubset:
    # The current user and the new user are ordered in the same way
    user = group.sort_values(by='movieId')
    movies = user_df.sort_values(by='movieId')

    # Number of ratings for user
    nRatings = len(user)

    # Common ratings of the current user with the new user
    temp_df = movies[movies['movieId'].isin(user['movieId'].tolist())]
    tempRatingList = temp_df['rating'].tolist()

    # Ratings of the current user
    tempUserList = user['rating'].tolist()

    # Calculate the Pearson Correlation between the current user and new user
    Uxx = sum([i ** 2 for i in tempRatingList]) - pow(sum(tempRatingList), 2) / float(nRatings)
    Uyy = sum([i ** 2 for i in tempUserList]) - pow(sum(tempUserList), 2) / float(nRatings)
    Uxy = sum(i * j for i, j in zip(tempRatingList, tempUserList)) - sum(tempRatingList) * sum(
        tempUserList) / float(nRatings)

    # If the denominator is nonzero, then we divide, otherwise the correlation is 0
    if Uxx != 0 and Uyy != 0:
        pearsonCorrelationDict[id] = Uxy / sqrt(Uxx * Uyy)
    else:
        pearsonCorrelationDict[id] = 0
```

Inicialmente en cada iteración, se ordenan los datos de cada usuario instanciado y los datos del nuevo usuario de la misma manera, para que los valores no se mezclen más adelante. Después se obtiene el número de valoraciones del usuario instanciado y se obtienen las valoraciones en común de este con el nuevo usuario, y todo se almacena en una lista temporal llamada *tempRatingList*. Guardamos también en una lista temporal las valoraciones del usuario instanciado (*tempUserList*).

En definitiva, la finalidad de esta parte del código es calcular los coeficientes de correlación Pearson de cada usuario instanciado con el nuevo usuario basándonos en la fórmula (1), siempre comprobando que el denominador sea diferente a cero, sino el valor del coeficiente será 0. El contenido del diccionario *pearsonCorrelationDict* después de realizar todas las iteraciones lo podemos ver en la página siguiente.

```
dict_items([(414, 0.1710335374313876), (599, 0.40123525382704545), (6, 0.20329349941761546), (474,
0.21044663004441924), (274, 0.3530769675354643), (68, 0.33059489678211385), (608,
0.3260682075724168), (182, -0.23109387695394956), (19, 0.3133595303614617), (307,
-0.18843150346003273), (91, 0.31180055592445977), (314, -0.1855577860558229), (380,
0.45012727283065757), (84, 0.012277750762966397), (117, -0.06953551887224728), (480,
0.16458728710301657), (217, 0.3252127201443157), (483, -0.16673896789941262), (372,
0.05020839625891806), (470, 0.32705689114535186), (489, -0.025392998342644958), (600,
0.042693983860009506), (240, 0.05131981034870924), (337, 0.3214285714285713), (448,
0.3145474121592105), (559, 0.17567351114860863), (602, -0.01953661662911384), (603,
-0.12466125165644129), (604, -0.11070047639148746), (226, -0.08878401871687391), (43,
-0.4295675016629953), (57, 0.05739773181955021), (160, -0.23053561726821561), (181,
-0.025450425803285343), (411, 0.4184472097595221), (492, 0.009531160645787254), (524,
0.554930806561758), (45, 0.38256520195621246), (58, 0.06698696164390111), (109,
-0.08767933833518694), (288, 0.040138052643118996), (304, -0.05079155413724917), (438,
0.1689463508123878), (592, 0.06429719695335717), (597, -0.04354691441125118), (40,
-0.046609537669447546), (64, 0.027500104500595552), (446, -0.29551933110701833), (501,
-0.021239769762143604), (590, 0.1779059797601214), (191, -0.5019578060538685), (219,
0.48620274802831803), (284, -0.00550473542118167), (353, 0.20638197410277853), (386,
-0.1111123018786146), (140, -0.08363570955493667), (357, -0.009026976808426818), (373,
-0.13235807087846027), (42, -0.010032434901514608), (136, -0.1711753075974906), (177,
0.1690569419933152), (179, 0.09896799408051585), (249, 0.25782317138405986), (330,
-0.0750008147037369), (368, 0.2405974233356921), (437, 0.04624821484746095), (477,
0.31049173032627525), (594, 0.008390572815346994), (18, 0.019351013185103308), (103,
-0.003498129001342704), (112, 0.38778864063592566), (385, 0.5296480205051889), (387,
0.43198485996958375), (425, 0.09288282368076137), (541, 0.25728140367086977), (566,
0.0679837639353997), (584, 0.573224999318389), (32, -0.04999999999999922), (235,
-0.30774536098405614), (266, 0.177355756021643), (391, -0.3687156994355351), (428,
-0.24008629652200658), (458, 0.0843349010400103), (469, 0.10400998543792594), (555,
-0.006953000128056482), (570, 0.2575275846563296), (580, 0.31742024784266926), (94,
0.5774257546144353), (144, -0.25560087159134015), (294, 0.4777456720827587), (318,
-0.40396888282756127), (323, 0.2786429232355919), (381, 0.07479684684631624), (453, 0.0), (486,
0.3959797974644654), (534, 0.23281019568466926), (588, -0.16115829864721526), (606,
-0.4177121530460993), (82, 0.3263157894736842), (121, -0.1532100435348196)])])
```

Aunque hemos incluido los usuarios más comunes a la hora de calcular la similitud entre usuarios, vemos que la estimación no es muy realista, los valores de los coeficientes son bajos. Puede que el valor de iteraciones límite determinado en función de los datos disponibles y de su robustez no sea el adecuado. Esto puede deberse a que, para algunas películas, no haya suficientes usuarios que las hayan valorado. A pesar de esta estimación se sigue con la construcción del sistema de recomendación y guardamos el contenido del diccionario *pearsonCorrelationDict*, para una mejor visualización, en un nuevo dataframe que llamaremos *pearson\_df*. El nuevo dataframe tiene dos columnas, la columna *similarityIndex* que contiene los coeficientes de correlación Pearson y la columna *userId* que contiene los identificadores de los usuarios. Ver Tabla 6.

```
pearson_df = pd.DataFrame.from_dict(pearsonCorrelationDict, orient='index')
pearson_df.columns = ['similarityIndex']
pearson_df['userId'] = pearson_df.index
pearson_df.index = range(len(pearson_df))
```

similarityIndex	userId
0.171034	414
0.401235	599
0.203293	6
0.210447	474
0.353077	274
0.330595	68
0.326068	608
-0.231094	182
0.313360	19
-0.188432	307

Tabla 6: Contenido del dataframe *pearson\_df*.

### 3.2. Predicción y Resultado

Como tenemos ordenado el conjunto de datos de tal forma que los usuarios que comparten la mayor cantidad de películas tienen más prioridad, podemos realizar una selección de vecindario. La selección de vecindario o la selección de los  $K$  vecinos más cercanos, es la aproximación más común para seleccionar los  $K$  usuarios más similares de un nuevo usuario.  $K$  es el número de vecinos que vamos a seleccionar, en teoría cuantos más vecinos seleccionemos para nuestro vecindario mejores recomendaciones realizará el sistema de recomendación. Porque aplicando una selección de vecindario conseguiremos penalizar aquellas situaciones donde el cálculo de la similitud entre los usuarios haya obtenido pocas valoraciones en común, como parece ser que hemos observado en la sección anterior. En este caso, seleccionamos el valor de  $K$  igual a 50. Los 50 usuarios seleccionados son ordenados por sus coeficientes de forma ascendente, así en las primeras posiciones tendremos los usuarios más parecidos al nuevo usuario.

```
topUsers = pearson_df.sort_values(by='similarityIndex', ascending=False)[0:50]
```

Veamos los primeros 10 usuarios más parecidos al nuevo usuario, que en este caso como yo he valorado las películas según mi criterio, son los primeros 10 usuarios más parecidos a mí.

similarityIndex	userId
0.577426	94
0.573225	584
0.554931	524
0.529648	385
0.486203	219
0.477746	294
0.450127	380
0.431985	387
0.418447	411
0.401235	599

Tabla 7: Los 50 usuarios más similares al nuevo usuario.

Parece ser que mis intereses no son muy similares a los de los otros usuarios que forman el conjunto de datos. Los coeficientes de correlación Pearson que podemos observar en la Tabla 7 no son muy altos. Por ejemplo, el valor de similitud con el usuario que más se parece a mí, el usuario 94, es igual a 0.577426. Aun así, vamos a calcular las recomendaciones. Aún así, tomaremos la media ponderada de las valoraciones de las películas utilizando los coeficientes de la correlación Pearson. Se ha elegido la media ponderada como medida de predicción porque se considera fácil de calcular y suele funcionar bastante bien, aunque no tiene en cuenta el sesgo en las valoraciones de los usuarios si tienden a valorar alto o bajo.

La media ponderada se calcula aplicando la siguiente fórmula:

$$\hat{r}_{u,i} = \frac{1}{\sum_{a \in K} sim(u, a)} \sum_{a \in K} sim(u, a) r_{a,i} \quad (2)$$

A fin de calcular la media ponderada, primero debemos unir las valoraciones de todas las películas del dataframe *ratings\_df* con los usuarios más similares al nuevo usuario (*topUsers*). Guardamos el resultado de la unión en un nuevo dataframe que llamamos *topUsersRating*. A la hora de realizar esta unión usamos la función *merge* de la librería pandas, que nos permite realizar la unión de los dos dataframes (*ratings\_df* y *topUsersRating*) sobre las columnas *userId*, considerando solo las filas con etiquetas comunes (*inner*).

```
topUsersRating = topUsers.merge(ratings_df, left_on='userId', right_on='userId', how='inner')
```

Vemos como queda organizado el contenido de *topUsersRating*:

similarityIndex	userId	movieId	rating
0.577426	94	2	4.0
0.577426	94	10	3.0
0.577426	94	11	3.0
0.577426	94	17	1.0
0.577426	94	19	2.0
0.577426	94	21	3.0
0.577426	94	32	5.0
0.577426	94	34	4.0
0.577426	94	39	1.0
0.577426	94	44	1.0

Tabla 8: Unión de los dataframes *ratings\_df* y *topUsersRating*.

A continuación, calculamos la media ponderada aplicando la fórmula (2). El cálculo se realiza sencillamente multiplicando las columnas *similarityIndex* y *rating* del dataframe *topUsersRating*; y guardando el resultado en una nueva columna (*weightedRating*).

```
topUsersRating['weightedRating'] = topUsersRating['similarityIndex'] * topUsersRating['rating']
```

Después se suman los valores de las columnas *similarityIndex* y *weightedRating*, y se agrupan por la columna *movieId*. Por último, separamos el resultado en dos columnas: la columna *sum\_similarityIndex* que contiene las sumas de los coeficientes de correlación Pearson, y la columna *sum\_weightedRating* que contiene las sumas de las valoraciones de los usuarios por los coeficientes. Ver la Tabla 10 de la página siguiente.

```
tempTopUsersRating = topUsersRating.groupby('movieId').sum()[['similarityIndex', 'weightedRating']]
tempTopUsersRating.columns = ['sum_similarityIndex', 'sum_weightedRating']
```

Vemos el contenido de la columna *weightedRating*:

similarityIndex	userId	movieId	rating	weightedRating
0.577426	94	2	4.0	2.309703
0.577426	94	10	3.0	1.732277
0.577426	94	11	3.0	1.732277
0.577426	94	17	1.0	0.577426
0.577426	94	19	2.0	1.154852
0.577426	94	21	3.0	1.732277
0.577426	94	32	5.0	2.887129
0.577426	94	34	4.0	2.309703
0.577426	94	39	1.0	0.577426
0.577426	94	44	1.0	0.577426

Tabla 9: Multiplicación de las columnas *similarityIndex* y *rating*.

Vemos el contenido de las columnas `sum_similarityIndex` y `sum_weightedRating`:

<code>sum_similarityIndex</code>	<code>sum_weightedRating</code>
11.302189	43.880774
9.326976	30.408888
4.882914	12.700033
0.621741	1.446775
2.796576	8.189528
8.893940	34.706954
3.739738	10.699062
0.727404	2.182212
1.353227	4.100684
11.269351	40.952455

Tabla 10: Las columnas `sum_weightedRating` y `sum_weightedRating`.

Por último, se divide el contenido de las columnas `sum_weightedRating` y `sum_weightedRating`. Como resultado de ello obtenemos los valores de la media ponderada que se almacenan en la columna `weighted_average_score` de un nuevo dataframe que llamamos `recommendation_df`. Junto con los valores de la media ponderada también se almacenan los identificadores de las películas (`movied`).

```
recommendation_df = pd.DataFrame()
recommendation_df['weighted_average_score'] = tempTopUsersRating['sum_weightedRating'] /
    tempTopUsersRating['sum_similarityIndex']
recommendation_df['movied'] = tempTopUsersRating.index
```

Vemos como queda organizado el contenido de `recommendation_df`:

<code>weighted_average_score</code>	<code>movied</code>
3.882502	1
3.260316	2
2.600913	3
2.326975	4
2.928412	5
3.902315	6
2.860912	7

Tabla 11: Media ponderada por película.

Una vez calculados los valores de la media ponderada, se ordenan de forma ascendente para mostrar los primeros 10 valores más altos, o que es lo mismo, las primeras 10 películas que recomienda el sistema de recomendación al nuevo usuario. Podemos observar estas recomendaciones en la Tabla 12 de la página siguiente.

```
recommendation_df = recommendation_df.sort_values(by='weighted_average_score', ascending=False)
recommendation_df = movies_df.loc[movies_df['movied'].isin(recommendation_df.head(10)['movied']).
    tolist()]
```

movielfid	title	year
2295	Impostors, The	1998
7121	Adam's Rib	1949
74754	Room, The	2003
82744	Faster	2010
95149	Superman/Batman: Public Enemies	2009
102084	Justice League: Doom	2012
108795	Wonder Woman	2009
147376	Doctor Who: A Christmas Carol	2010
172875	A Detective Story	2003
173145	War for the Planet of the Apes	2017

Tabla 12: Las 10 mejores recomendaciones.

A pesar de una estimación inicial que parecía poco realista y que podríamos llegar a intuir que el sistema podría no dar una buena recomendación a causa de que mis intereses no se parecían mucho a los de los otros usuarios, el sistema de recomendación des de mi punto de vista me ha dado una buena recomendación. Me gustan mucho las películas de superhéroes i de ciencia ficción. También me gustan bastante las películas de temática policiaca. Aunque el sistema de recomendación ha patinado un poco al recomendarme la película "Room, The", no le vería nunca.

## 4. Conclusiones

En este documento se ha descrito una sencilla implementación en Python de un sistema de recomendación de filtrado colaborativo basado en usuarios, con el fin de recomendar películas a un nuevo usuario según sus intereses. Paso a paso se ha ido describiendo el desarrollo del sistema de recomendación, hasta obtener las recomendaciones que han sido buenas. Llegando a la conclusión que, aunque seguramente el sistema de recomendación se podría mejorar adoptando un enfoque híbrido y así aprovechar las ventajas de los sistemas colaborativos y los no colaborativos, contar con sistemas de recomendación que ayuden a los usuarios a buscar información y tomar decisiones en este entorno es un tema muy útil y de crucial importancia.

Quiero destacar como valoración personal, que he elegido hacer un ejercicio práctico para llevar a cabo la teoría aprendida durante el aprendizaje teórico. Siempre me parece muy interesante llevar a la práctica conocimiento nuevo para ver su utilidad. He escogido un sistema de recomendación de filtrado colaborativo basado en usuarios, porque me ha parecido el adecuado y más sencillo de llevar a la práctica.

Sé que existen muchísimos sistemas de recomendación ya implementados, pero me pareció muy interesante desarrollar un sistema de recomendación propio, muy básico y simple de entender. Además, nunca había intentado implementar un sistema de recomendación en ningún tipo de lenguaje de programación. Decidí que fuera en Python, ya que es el lenguaje que uso más a diario y es uno de los lenguajes de programación más utilizados en el mundo, además uno de los mayores aliados para la Ciencia de Datos.

Se podrían considerar muchas mejoras en el sistema de recomendación desarrollado en esta práctica, seguramente las primeras que realizaría en un futuro con más tiempo, serían la automatización de la creación de un nuevo usuario y añadir la modularización.

## Bibliografía

- [1] Laura Rodríguez-Navas. Recomendación para grupos. <https://github.com/lrodrin/masterAI/tree/master/A13>, 2021.
- [2] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015.
- [3] Jeff Reback, Wes McKinney, jbrockmendel, Joris Van den Bossche, Tom Augspurger, Phillip Cloud, gyoung, Sinhrks, Simon Hawkins, Matthew Roeschke, Adam Klein, Terji Petersen, Jeff Tratner, Chang She, William Ayd, Shahar Naveh, Marc Garcia, Jeremy Schendel, Andy Hayden, Daniel Saxton, Vytas Jancauskas, Ali McMaster, Pietro Battiston, Skipper Seabold, patrick, Kaiqi Dong, chris b1, h vetinari, Stephan Hoyer, and Marco Gorelli. pandas-dev/pandas: Pandas 1.1.5, December 2020.