



Universidad
Internacional
Menéndez Pelayo

Máster Universitario en Investigación en Inteligencia Artificial

Curso 2020-2021

Sistemas de Recomendación

Recomendación para grupos en Python

5 de mayo de 2021

Laura Rodríguez Navas

DNI: 43630508Z

e-mail: rodrigueznava@posgrado.uimp.es

Índice

1. Introducción	3
1.1. Filtrado Colaborativo	3
1.2. Ventajas y Desventajas del Filtrado Colaborativo	4
2. Conjunto de datos	4
3. Sistema de Recomendación	6
3.1. Coeficiente de Correlación de Pearson	8
3.2. Predicción y Resultado	11
4. Conclusiones	14
Bibliografía	15

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 mecanismos de filtrado de información eficientes. Uno de los sistemas utilizados para hacer frente a este problema son los sistemas de recomendación.

En este documento se describe el desarrollo práctico en Python de un sistema de recomendación. El sistema de recomendación elegido se denomina de Filtrado Colaborativo (Usuario-Usuario). La implementación se puede encontrar en [1], la cual está formada por el desarrollo de un programa que implementa el sistema de recomendación de [Filtrado Colaborativo](#). El documento se divide en diferentes secciones donde se va describiendo paso a paso el trabajo realizado.

En la primera parte del documento se empieza describiendo la técnica de Filtrado Colaborativo, con sus ventajas y desventajas (ver secciones 1.1 y 1.2). El documento sigue con la descripción del conjunto de datos que usa 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. En esta sección también se describe el [coeficiente de Correlación de Pearson](#) y porqué se ha elegido como medida para encontrar las similitudes entre los usuarios representados en el conjunto de datos y un nuevo usuario que he creado manualmente (ver sección 3.1). Además, también se aportan los resultados y su evaluación.

Finalmente, en la parte final del documento se añaden las conclusiones y la bibliografía.

1.1. Filtrado Colaborativo

El Filtrado Colaborativo (FC) es una técnica utilizada por algunos sistemas de recomendación para predecir el grado en que, a un usuario U le gustaría un producto X . Esta técnica nos permite crear recomendaciones personalizadas, ayudar a los usuarios a encontrar productos de su interés, etc. analizando datos de muchos usuarios; asumiendo que los usuarios con gustos parecidos tienden a valorar los productos de forma similar. Es decir, si un usuario A tiene la misma opinión que un usuario B sobre un tema, el usuario A es más probable que tenga la misma opinión que el usuario B en otro tema diferente que la opinión que tendría un usuario elegido al azar. En resumen, el Filtrado Colaborativo es un método para hacer predicciones automáticas (filtrado) sobre los intereses de un usuario mediante la recopilación de las preferencias o gustos de información de muchos usuarios (colaboradores).

Los sistemas de filtrado colaborativos basados en los usuarios, siguen una metodología que puede reducirse en los dos pasos siguientes:

- En 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).
- En utilizar las valoraciones por parte de aquellos usuarios afines que se encuentran en el paso 1 para calcular una predicción para el usuario activo.

En la sección 3 podremos ver detalladamente la metodología usada por el sistema de recomendación que se ha implementado en esta práctica.

1.2. Ventajas y Desventajas del Filtrado Colaborativo

Algunas ventajas de un sistema de recomendación de Filtrado Colaborativo son:

- Tiene en cuenta las valoraciones de otros usuarios.
- No necesita estudiar o extraer la información de los elementos recomendados.
- Se adapta a los intereses del usuario si estos cambian con el tiempo.

Algunas desventajas de un sistema de recomendación de Filtrado Colaborativo son:

- Difícil de aplicar con grandes cantidades de usuarios.
- Difícil para encontrar suficientes vecinos para recomendar.
- Predice peor cuando existen pocas valoraciones.
- Difícil de aplicar cuando tenemos pocos datos de un nuevo producto (no ha sido valorado o ha sido poco valorado por los usuarios); o de un nuevo usuario (desconocemos sus valoraciones). Este problema se denomina el problema del [arranque frío](#).

Para aliviar las desventajas 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. Los sistemas de recomendación basados en contenido son aquellos que tomando en cuenta algunos datos del historial del usuario intenta predecir que busca el usuario y que sugerencias similares puede mostrar. Este tipo de sistemas es uno de los que tiene mayor presencia en la actualidad.

En esta práctica no me ha parecido necesario el uso de un sistema de recomendación híbrido.

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/abhikjha/movielens-100k/download>. Este conjunto de datos llamado *ml-latest-small*, describe las valoraciones (entre 1 y 5 estrellas) y la actividad del etiquetado de [MovieLens](#) [2], un servicio de recomendación de películas. 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*, y estos se encuentran en la carpeta *dataset* de [1]. Para mejorar las recomendaciones del sistema de recomendación, primero realizaremos un seguido de transformaciones en el conjunto de datos. Inicialmente empezamos cargando los archivos *movies.csv* y *ratings.csv* dentro de un dataframe (ver Definición 1) con el uso de la librería *pandas* [3].

```
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 está 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 del dataframe *movies_df* inicialmente.

Cada película tiene un único identificador (*movieid*), un título con su año de estreno y diferentes géneros. Como los años contienen caracteres *unicode* y para que no haya problemas más adelante, los sacaremos de la columna de los títulos y los ubicaremos en su propia columna que nombraremos *year*. Para ello, primero creamos una expresión regular para encontrar los años guardados entre paréntesis, y con la función *extract* de la librería *pandas* los extraemos de la columna de los títulos para guardarlos en su propia columna. Después borramos los años de la columna de los títulos y con la función *strip* nos aseguramos de sacar los espacios finales que pudiera haber. También eliminamos la columna de los géneros, ya que no la tendremos en cuenta 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 final:

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* final.

Definición 1. Un *DataFrame* es una estructura de datos etiquetada bidimensional 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, miraremos cómo está organizado el dataframe *ratings_df*. En este caso también eliminamos una columna, la columna *timestamp*, porque que tampoco se tendrá en cuenta en el sistema de recomendación.

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

Así es cómo queda el dataframe *ratings_df* definitivo:

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* final.

Cada fila del dataframe *ratings_df* tiene un identificador de usuario asociado con al menos una película y una valoración de esta. Por ejemplo, en la Tabla 3 vemos como que el usuario 1 ha visto y valorado diferentes películas.

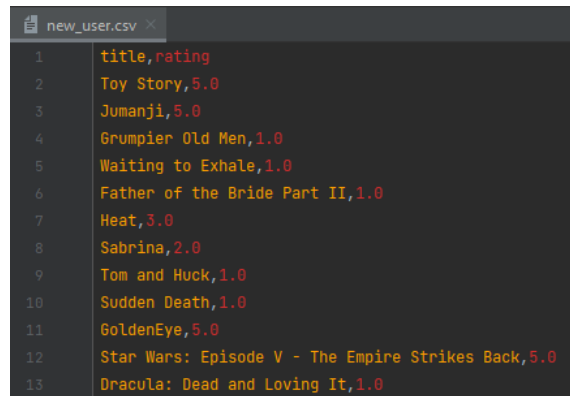
3. Sistema de Recomendación

En esta sección describimos el sistema de recomendación, que como se ha comentado en la sección 1.1, utiliza la técnica de Filtrado Colaborativo o también conocida como técnica de Filtrado de Usuario a Usuario. Con esta técnica el sistema de recomendación va a predecir recomendaciones de películas a un nuevo usuario acordes a sus gustos después de añadir nuevos datos al sistema. Para predecir las recomendaciones, el sistema de recomendación buscará las similitudes entre los datos introducidos por el nuevo usuario y los datos de los 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 al nuevo usuario películas acordes a sus valoraciones. Existen varios métodos para encontrar las similitudes entre los diferentes usuarios del sistema, y en el caso de este sistema, el método elegido se basa en el [coeficiente de Correlación de Pearson](#) (ver sección 3.1).

El flujo de trabajo de este sistema de recomendación sigue estos pasos:

1. Crea un nuevo usuario con las películas del conjunto de datos que el usuario a mirado.
2. Basándose en el índice de selección de películas del nuevo usuario, encuentra a sus vecinos (usuarios similares a él).
3. Obtiene los identificadores de las películas (movieId) donde el nuevo usuario y los vecinos coinciden, es decir, obtiene los identificadores de las películas que tanto el nuevo usuario como sus vecinos hayan visto.
4. Calcula las similitudes entre el nuevo usuario y sus vecinos.
5. Recomienda películas al nuevo usuario según las similitudes con sus vecinos.

Así, siguiendo el flujo de trabajo 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. En este caso, he valorado las películas según mi criterio. Este archivo 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. A continuación, visualizamos como se organiza parte del contenido del archivo *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

Figura 1: Nuevo usuario del sistema.

Para finalizar con la creación del nuevo usuario, el archivo *new_user.csv* se carga dentro de un nuevo dataframe que llamaremos *user_df*:

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

Con los datos del nuevo usuario en el sistema de recomendación, extraemos los títulos de las películas que el usuario haya visto, los guardamos en la variable *titles* y los unimos a los datos del usuario almacenados en el dataframe *user_df*. En este punto para ahorrar un poco de espacio de memoria, aprovechamos a eliminar la columna *year* del dataframe del nuevo usuario, por qué 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 dataframe *user_df*:

movielid	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*.

Ahora que hemos añadido los identificadores de las películas con 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).

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

Observemos a uno de los usuarios, por ejemplo, el usuario 525.

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: Valoraciones del usuario 525.

El usuario 525 en total ha valorado 14 películas, 12 de las cuales no han sido valoradas por el nuevo usuario del sistema (ver Tabla 4). Este hecho podría entorpecer la predicción de las recomendaciones, así que, para una mejor recomendación, donde no será necesario pasar por todos los usuarios, ordenaremos el conjunto de datos de tal forma que los usuarios que compartan la mayor cantidad de películas tengan prioridad (usuarios comunes).

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

3.1. Coeficiente de Correlación de Pearson

Una vez encontrados los usuarios comunes o similares, 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, viendo cómo se relacionan entre sí. Es decir, el sistema buscará las similitudes entre los usuarios y el nuevo usuario para encontrar aquellos que se parecen más a este. Para ello usaremos el [coeficiente de Correlación de Pearson](#). La fórmula para calcular este coeficiente sobre un estadístico muestral (ver Definición 2) se puede ver 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 gustos.
- Si $0 < r < 1$, entonces existe una correlación positiva. Los usuarios tendrán gustos parecidos.
- Si $-1 < r < 0$, entonces existe una correlación negativa. Los usuarios tendrán pocos gustos parecidos.
- Si $r = -1$ (correlación negativa perfecta), entonces las valoraciones estarán inversamente correlacionadas. Los usuarios no tendrán los mismos gustos.

Se ha decidido usar el coeficiente de Correlación de Pearson 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 gustos similares) contaríamos con valoraciones muy parecidas en escalas variadas y esto crearía grandes problemas en la predicción de las recomendaciones. El coeficiente de Correlación de Pearson es la medida más utilizada cuando hay variaciones entre magnitudes y escala de las valoraciones, aunque podríamos 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 los coeficientes de Correlación Pearson. Los coeficientes los almacenaremos en el diccionario `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 hacer las iteraciones, no he querido añadir demasiado sobrecoste computacional pasando por cada usuario.

```
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, 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*). Esto lo hacemos porque siempre es mucho más fácil recorrer una lista que un dataframe. A continuación, calculamos el coeficiente de correlación de Pearson de cada usuario instanciado con el nuevo usuario con la fórmula (1), comprobando que el denominador sea diferente a cero, sino el valor del coeficiente será 0.

Vemos el resultado, el contenido del diccionario *pearsonCorrelationDict*:

```
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)])
```

Guardamos el resultado para una mejor visualización en un nuevo dataframe (*pearson_df*), donde la columna *similarityIndex* contendrá los coeficientes de cada usuario representado en la columna *userId*.

```
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))
```

Así queda organizado el resultado en el nuevo dataframe *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: Resultado de la Correlación de Pearson.

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 del vecindario. La selección del vecindario o la selección de los K vecinos más cercanos, es la aproximación más común para seleccionar los K usuarios más relevantes o mejores. K es el número de vecinos que vamos a seleccionar, en teoría cuantos más vecinos seleccionamos para nuestro vecindario, mejores correlaciones entre ellos. Porqué aplicando una selección del vecindario conseguiremos penalizar aquellas situaciones donde el cálculo de la similitud de los usuarios haya obtenido pocas valoraciones comunes. Además, también ahorraremos coste computacional al reducir el número de usuarios, ya que no tendremos que desperdiciar mucho tiempo pasando por cada usuario. En este caso, seleccionamos $K=50$ (50 usuarios).

Vemos el resultado:

similarityIndex	userId
0.171034	414
0.401235	599
0.203293	6
0.210447	474
0.353077	274

Tabla 7: Contenido del dataframe *pearson_df*.

Ahora obtenemos los 50 primeros usuarios más parecidos:

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

Recomendemos películas al nuevo usuario puntuando a los usuarios elegidos para todas las películas. Haremos esto tomando el peso promedio de los ratings de las películas utilizando la Correlación Pearson. Pero para hacer esto, primero necesitamos que los usuarios vean las películas en nuestro *pearson_df* a partir del dataframe de valoraciones y luego guardar su correlación en una nueva columna llamada similarityIndex. Estos se logra juntando estas dos tablas de debajo.

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

similarityIndex	userId
1.0	551
1.0	513
1.0	409
1.0	257
1.0	335

Tabla 8: Los 50 primeros usuarios más parecidos.

similarityIndex	userId	movieId	rating
1.0	551	47	4.5
1.0	551	110	3.5
1.0	551	260	3.5
1.0	551	293	4.5
1.0	551	296	3.5

Tabla 9: Los 50 primeros usuarios más parecidos.

Ahora todo lo que se necesita hacer es multiplicar el puntaje de la película por su peso (El índice de similitud), luego se suman los nuevos puntajes y dividen por la suma de los pesos. Esto se logra sencillamente multiplicando dos columnas, luego agrupando el dataframe por la columna movieId y luego dividiendo dos columnas: Aquí se muestra la idea de todos los usuarios similares respecto de las películas candidatas para el usuario ingresado:

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

sum_similarityIndex	sum_weightedRating
9.903247	40.868223
2.453957	9.295957
1.654654	4.463961
2.415235	4.745706
1.866025	7.464102

Se crea un dataframe vacío Ahora se toma el promedio ponderado

```
recommendation_df = pd.DataFrame()
recommendation_df['weighted average recommendation score'] = tempTopUsersRating['sum_weightedRating'] / \
tempTopUsersRating['sum_similarityIndex']
recommendation_df['movieId'] = tempTopUsersRating.index
```

weighted average recommendation score	movied
4.126750	1
3.788150	2
2.697822	3
1.964904	5
4.000000	6

Luego, ordenemos y veamos las primeras 20 películas que el algoritmo recomendó:

Después de obtener las recomendaciones vamos a valorarlas.

4. Conclusiones

Automatizar la creación del archivo *dataset/user_ratings.csv*.

El enlace al vídeo debéis incluirlo en la memoria. Trabajo práctico: la idea es que vayáis explicando el porqué de las decisiones que habéis tomado (qué me llevó a utilizar unos determinados hiperparámetros, por qué he utilizado esta manera de medir la calidad de la solución,...) y hagáis un razonamiento sobre los resultados obtenidos.

Ahora, calculemos el coeficiente de correlación de Pearson entre el nuevo usuario y el resto de los usuarios del conjunto de datos. El resultado lo vamos a almacenar en un diccionario, donde la clave es el identificador del usuario y el valor es el coeficiente.

Este estudio presenta una aproximación a la recomendación que muestra noticias ordenadas según el interés de los usuarios, expresados mediante visitas previas y los términos contenidos en dichos artículos, así como en sus categorías. Se han diseñado dos modelos probabilísticos basados en el Aspect Model. Los resultados muestran que cuando se considera la clasificación de las noticias al general el modelo, ayuda mejor al usuario a acceder a las mismas. Esta propuesta es interesante y una alternativa competitiva en el contexto de los modelos de recomendación de noticias. El modelo Triadic es mejor que el Diadic cuando se combina con la cuenta de números de accesos, términos y categorías.

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.