Universidad de Sonora

Reconocimiento de Patrones

Sistemas de recomendacion: prediccion de preferencias

Autor:
Rafael Castillo

Profesor:
Dr. Ramón Soto de la Crúz

December 7, 2017

Contents

1 Resumen

Los sistemas de recomendación son métodos cuya importancia cada vez es mayor en los sistemas que interactúan con usuarios. En este proyecto de implementaron cuatro método de recomendación diferentes aplicados a la misma base de datos con el fin de analizar las ventajas y desventajas de cada uno de ellos. En este reporte se introduce el problema de recomendación, los factores que influyen en el diseño de un sistema de recomendación, así como el tipo de soluciones que existen para estos problemas. Se hace una exploración de diversos métodos de predicción de calificaciones, incluyendo recomendación basada en contenido, filtrado colaborativo basado en memoria y filtrado colaborativo basado en modelos. Finalmente se propone una solución al problema del arranque en frió. Se concluye que el metodo de factorización SVD tiene muchas ventajas sobre los otros metodos pero que cada uno tiene un nicho que llenar.

2 Introducción

Un sistema de recomendación es un sistema que ayuda a un usuario encontrar algo cuando el proceso de búsqueda es desafiante por la cantidad de opciones existentes [1]. La recomendación ha existido en la naturaleza y en la humanidad por milenios. Cuando las hormigas buscan comida, primero todas exploran el espacio. Si una hormiga encuentra comida, deja una feromona en esa ubicación, lo que lleva a otras hormigas a ir a ese lugar. Estas hormigas están basándose en la experiencia de otra para tomar decisiones, una recomendación.

Existen ejemplos de sistemas de recomendación desde antes de que los humanos desarrollaran la escritura o el habla. Si un hombre prehistórico veía una planta desconocida, tenia dos opciones: Si era un hombre que no conocía los sistemas de recomendación, se arriesgaba y comía la planta; si era un hombre que sí sabía sobre sistemas de recomendaciones, esperaba a que un cavernícola menos dotado comiera la planta, si el cavernícola parecía disfrutarla, el también la comía, pero si el otro caía al suelo dolorido, entonces su vida fue salvada gracias a la recomendación del otro.

Asi, podemos ver que el concepto de la recomendación ha existido por mucho tiempo. A través de toda nuestra historia hemos encontrado formas de poder evaluar articulos sin tener que consumirlos nosotros mismos. Tradicionalmente aprovechabamos a otros humanos que estaban dispuestos a consumir los articulos que no queriamos (individuos tradicionalmente llamados "criticos"), pero en un mundo donde la automatización esta por doquier, hemos desarrollado tecnicas que nos permiten generar recomendaciones a partir de datos recolectados de usuarios, aprovechando la llamada "inteligencia collectiva" de muchos para aproximar los gustos de un individuo

Los sistemas de recomendación tienen un alto valor. Las recomendaciones pueden mostrarse en varias interfaces, amplificando su eficacia. Por ejemplo, las recomendaciones pueden tener interfaces obvias como listas de sugerencia, pero también pueden existir interfaces mas sutiles, como filtros de correo electrónico o sistemas de evaluación de candidatos de empleo.

Por la naturaleza de muchos de estos metodos, como muchos otros metodos de aprendizaje, funcionan mucho mejor cuando la cantidad de datos es mucho mayor. Por esto, muchas empresas muestan interes en aprovechar la gran cantidad de datos que recolectan para crear sistemas de recomendación, ya que son un recurso que tiene un gran impacto enriquecedor en las experiencias

de usuario.

2.1 Familias de métodos de recomendación

Hay muchas formas de lograr hacer recomendaciones automáticas, algunas mas sofisticadas que otras. Existen los métodos sencillos no personalizados, que se basan simplemente en métricas como popularidad. Otro método es usar la llamada asociación de productos, es decir recomendar a un usuario artículos similares a uno que este viendo. Estos métodos no no son de mucho interés, pues las recomendaciones no personalizadas no son de tanto valor. Por esta razón nos centraremos en los otros dos métodos.

Un categoría de métodos capaces de personalizar las recomendaciones a un usuario la de los métodos basados en contenido. Aquí aprovechamos conocimiento previo sobre los artículos (características de estos) para aproximar los gustos de un usuario a partir de las calificaciones que ya ha impartido. Este método también es conocido como filtrado basado en contenido y esta estrechamente relacionado al problema mas general de filtrado de información.

La ultima familia de métodos a mencionar son los del filtrado colaborativo. La idea del filtrado colaborativo es aprovechar las calificaciones de todos los usuarios del sistema para poder predecir preferencias y dar recomendaciones. El filtrado colaborativo además se separa en dos categorías: el filtrado que utiliza todas las calificaciones en el sistema, conocido como filtrado basado en memoria; y el filtrado que utiliza aprendizaje para poder predecir calificaciones, conocido como filtrado basado en modelos.

Una ventaja del filtrado colaborativo es que tiene mas potencial de encontrar correlaciones sorprendentes entre ciertos articulos muy distintos, algo que dificilmente sucederia en un metodo basado en contenido. Esto ocurre porque los metodos de filtrado colaborativo existen solamente basados en las preferencias de los usuarios, no en preconcepciones sobre el contenido de los articulos, llevando a recomendaciones basadas más en opinion colectiva que en prejuicios sobre los articulos.

2.2 Calificaciones

La base para el filtrado colaborativo y los sistemas de recomendación en general, una calificación es un indicador de la preferencia que tiene un usuario hacia un articulo. Hablaremos de calificaciones en dos categorías, implícitas y explicitas.

Las calificaciones explicitas son en las que se vienen a la cabeza cuando pensamos en calificación. Una calificación se considera explicita cuando el usuario explícitamente expresa su opinión sobre un articulo. Esto puede ser desde una escala numérica como las muy comunes de 1 a 5 estrellas, hasta un sistema de votos como el de Facebook o Reddit. Las escalas mas granulares tienen la ventaja de ser mas expresivas, mientras que un voto binario es mas fácil para el usuario, lo que lo hace útil para contenido mas efímero, como una noticia o un tweet.

Una calificación implícita se extrae de comportamiento del usuario que no tiene relación directa con su preferencia hacia un articulo. Hacer clic sobre un enlace puede considerarse una calificación positiva hacia este. Ver un vídeo hasta el final puede considerarse una calificación positiva, mientras de que dejarlo de ver a la mitad es uno negativo. Estas calificaciones implícitas pueden significar mucho y ser muy abundantes, dando buenas recomendaciones.

La variedad de fuentes de donde se pueden minar calificaciones es una de los elementos que hacen que los sistemas de recomendación sean útiles en aplicaciones donde anteriormente no se considerarían. Esto combinado con la recolección masiva de datos le da a las grandes corporaciones nuevas y emocionantes formas de llevar a cabo practicas horripilantes.

3 Datos para el desarrollo de sistemas de recomendación

Existen varios conjuntos de datos públicamente disponibles que se utilizan para el desarrollo de sistemas de recomendación. Estos son los que se consideraron para el desarrollo de los métodos.

MovieLens [2] es una base de datos de calificaciones de película. Se encuentra disponible en varios tamaños, pero aquí solo se tomara en cuenta la versión pequeña (100,000 calificaciones sobre 9,000 películas por 700 usuarios).

Book Crossings [3] es un conjunto de datos de calificaciones de libros. Tiene 1 millón de calificaciones para 90,000 usuarios en 270,000 libros, volviéndola la base de datos con calificaciones mas dispersas.

Jester [4] es un conjunto de datos que incluye 5 millones de calificaciones por 100,000 usuarios en 150 chistes. Esta es la base de datos mas densa.

Estos son conjuntos de datos explicitamente generados para desarrollo de sistemas recomendadores, como se mencionó antes, las calificaciones necesarias para esta clase de sistemas pueden surgir de lugares inesperados. Por ejemplo, se puede obtener una calificación a partir del tiempo tomado para leer un articulo, o por analisis de sentimientos aplicado a un comentario dejado en un video.

Para la mayoría de los métodos de filtrado colaborativo, necesitamos que los datos tengan la forma de una matriz usuario-articulo (user-item matrix), que tiene como filas cada articulo y como columnas a los usuarios. El único dataset que viene ya en este formato es Jester, los demás se encuentran en un formato mas parecido al de una base de datos relacional. En el apéndice A.1, se muestra un ejemplo para accesar a la base de datos *movielens*

Para la prueba de los métodos que se presentaran, se utilizo la base de datos de MovieLens, ya que viene en varios tamaños y tiene una densidad adecuada para las pruebas que realizaremos.

4 Métodos para predecir preferencias

El componente mas crucial para el desarrollo de un sistema de recomendaciones es el que predice las preferencias de un usuario por un articulo. A continuación se presenta la descripción e implementación de diversos métodos para predecir preferencias utilizados para el desarrollo de sistemas de recomendaciones.

4.1 Recomendaciones basadas en contenido

Los primeros métodos a revisar son los que se basan en características ya bien conocidas sobre los artículos calificados [5]. Estos métodos son también conocidos como métodos de filtrado de contenido ya que utilizan las características del

La idea es que dado un vector con N características $x^{(i)}$ y conociendo un vector $\theta^(j)$, también de tamaño N donde $\theta_k^{(j)}$ sea la preferencia del usuario j por la característica k, podemos predecir la calificación del articulo i esta dada por

$$(\theta^{(j)})^T x^{(i)}$$

La ventaja de este método es que puramente se basa en contenido. No se necesita tomar en cuenta las calificaciones de usuarios además de la del usuario activo. Por esta razón este método tiene un bajo uso de recursos, lo cual lo hace deseable para muchas aplicaciones.

La desventaja es la obvia: es necesario tener caracteristicas concretas de los articulos previo a la aplicación del metodo. Muchas veces esto hace que el metodo sea indeseable para aplicaciones donde hay una cantidad alta de articulos, o en situaciones donde es dificil cuantificar calidades de los articulos, como en un sistema de recomendación de arte. Pocas veces escuchas "me gustan las pinturas azules" en lugar de "me gustan las pinturas que me hacen sentir melancolico".

4.1.1 Características de los artículos

La primera parte que define un método basado en contenido es la fuente de donde se obtienen las características de los artículos. La opción mas obvia es irse por la ruta del conocimiento experto y asignar las características manualmente para cada articulo. Este método puede sonar demasiado ingenuo pero puede ser útil en muchos casos, por ejemplo para asignar características técnicas o fácilmente cuantificables a un conjunto de artículos similares (por ejemplo, en una tienda donde solo vendan un tipo de articulo).

También es factible crear interfaces para que los usuarios mismos determinen las características de los artículos. Este método puede llevar a la generación de una cantidad enorme de características, pero no es necesariamente confiable. Muchos servicios utilizan esta opción en forma de etiquetas que un usuario puede asignar a un articulo y que un ingeniero puede aprovechar después para determinar características de este.

Una ultima opción a mencionar es la extracción automática de características de un articulo. Por ejemplo, las palabras de un nota periodística pueden ser característicos de este, como lo pueden ser las frecuencias mas importantes de una canción, o la longitud de un vídeo.

4.1.2 Gustos de los usuarios

La pregunta ahora se vuelve como determinar los gustos del usuario j, dados por $\theta^{(j)}$. Una forma posible es preguntarle estos datos al usuario. Este enfoque se ha usado históricamente, por ejemplo en aplicaciones que le piden a un usuario que especifique sus gustos o que presentan cuestionarios acerca del contenido que quisiera ver.

Un enfoque tal vez mas interesante es usar las calificaciones previas del usuario para aproximar $\theta^{(j)}$. Una opción simple es simplemente hacer una suma de las características de los artículos X_j que ya consumió el usuario:

$$\theta^{(j)} = \sum_{i \in X_j} r_{i,j} x^{(i)}$$

También podemos modelar los gustos como un problema de regresión lineal donde nuestra función de error esta dada por

$$\frac{1}{2}\sum_{i:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

Si encontramos el $\theta^{(j)}$ que minimice este error, tendremos una aproximación de los gustos del usuario, que podremos usar para predecir sus calificaciones a artículos que no conoce aun.

En el anexo A.2 se presentan una implementación del método de vecinos mas cercanos para la base de datos de *MovieLens*

4.2 Vecinos mas cercanos

El siguiente método que veremos es un método de filtrado colaborativo, es decir, que no solo toma en cuenta las calificaciones del usuario para el cual estamos prediciendo calificaciones, sino que tomamos en cuenta las preferencias de nuestros usuarios.

El método utiliza una búsqueda de los k vecinos mas cercanos para encontrar una vecindad del usuario para el cual queremos predecir preferencias. Una vez que encontremos a los k vecinos mas cercanos U_a y sus respectivas distancias, podemos predecir la calificación del usuario a en el articulo i como

$$p_{a,i} = \frac{\sum_{i\in U_a} w_{u,i} r_{ui}}{\sum_{i\in U_a} w_{u,i}}$$

La idea aqui es encontrar una vecindad de usuarios que compartan gustos con el usuario, gustos que usaremos despues para predecir sus calificaciones futuras. De esta forma, aprovechamos la opinion colectiva de usuarios similares para obtener una predicción decente de las preferencias de un individuo.

La predicción de preferencias usando este método produce resultados decentes y el tiempo de computo para una calificación predicha es mínima. La desventaja obvia del método es que requiere que toda la base de datos se mantenga en memoria, por lo que este método puede volverse prohibitivo para conjuntos de datos muy grandes si no se realizan trucos de optimización.

El método clásico para filtrado colaborativo usando vecinos cercanos surgió en el grupo de investigación GroupLens, y esta descrito en detalle por Sarwar et al. en [6].

En el anexo A.3 se presenta un ejemplo del metodo de vecinos mas cercanos para el filtrado colaborativo en la base de datos de MovieLens

4.3 Filtrado colaborativo usando regresión

En la sección de de recomendación basada en contenido, se hablo de un método de aproximar $\theta^{(j)}$ usando regresión. También es claro ver que se puede hacer un proceso similar para aproximar las características $x^{(i)}$ de un articulo conociendo previamente los gustos $\theta^{(j)}$ de un usuario. Estos dos procesos de regresión pueden extenderse para calcular las características $x^{(i)}$ para todas las i resultando en una matriz X, así como una análoga matriz Θ. Aun mas allá, podemos aproximar estos dos dentro de la misma regresión lineal. Llevando a cabo este proceso, efectivamente

obtenemos una descomposición de la matriz R que nos permite aproximar todas las calificaciones de nuestros usuarios simplemente calculando $\Theta^T X$.

Este método aprovecha todos los valores de la matriz de calificaciones para predecir calificaciones faltantes, convirtiéndolo en un método de filtrado colaborativo. Además, utiliza un modelo de aprendizaje en lugar de mantener todas las calificaciones en memoria, volviéndolo un método de filtrado colaborativo basado en modelo[7].

Al utilizar este método, tenemos la ventaja de que una vez que se completa el entrenamiento, el uso de memoria se reduce. Además de esto, obtenemos como producto secundario una matriz X aproximada. Podemos considerar esta matriz X una matriz de características arbitraria (aunque esta contenga mas que nada características difíciles de interpretar para un humano), que podemos usar para realizar operaciones que requieran de una matriz como esta, incluyendo uno que veremos en una seccion futura.

Este y otros métodos basados en modelos que resultan en dos matrices de rango menor, predeciblemente son conocidos como factorizacion de matrices de rank menor (low rank matrix factorization).

En el anexo A.4 se incluye una implementación de la descomposición basada en regresión y como se utilizaría para obtener recomendaciones.

4.3.1 Normalización de medias

Al aplicar el método de regresión sobre nuestros datos, podemos darnos cuenta de un detalle peculiar sobre su naturaleza: para un usuario nuevo, todas sus calificaciones se aproximan como 0. Esta observación es parte del problema de arranque en frió, que veremos una sección futura.

Una forma de remediar este problema es restar a cada calificación existente la media de calificaciones para ese articulo en particular. Haciendo esto, una calificación 0, en vez de tener un valor arbitrario, se vuelve la calificación media de ese articulo. Esto ocasiona que para un usuario nuevo, se prediga una calificación promedio para cada articulo, un resultado mas razonable.

4.3.2 Descomposición por valores singulares

SVD es una técnica de factorización matricial que busca reducir las características de una matriz a un espacio vectorial $K < N$. SVD nos deja con tres matrices tales que $R = U \times S \times V^T$ [8]. Lo interesante de esto es que la factorización SVD es equivalente al proceso de regresión descrito arriba. La diferencia radica en que existen algoritmos ya optimizados para calcular la SVD de una matriz incluidas en paquetes de álgebra lineal populares, como LAPACK.

El método SVD es de los mas importantes y mas usados para el desarrollo de sistemas de recomendación. Del 2006 al 2009, Netflix llevo a cabo una competencia para determinar el mejor algoritmo de filtrado colaborativo [8]. En el 2009 finalmente se otorgo el premio a un equipo que había desarrollado un método basado en la descomposición SVD, mostrando que al menos para el caso de uso de Netflix, el SVD es el método mas óptimo.

En el anexo A.5, se incluye una aplicación de las recomendaciones usando una factorización SVD.

5 El problema de arranque en frío

Todos los métodos de predicción que se vieron solo pueden dar buenas recomendaciones si el usuario ya tiene calificaciones en el sistema.El problema del arranque en frió radica en como presentarle recomendaciones a un usuario nuevo.

A continuación proponemos un método como solución para el problema de arranque en frió. El requerimiento para aplicar este método es solamente contar con un conjunto de vectores de características de las películas. Estos pueden surgir directa o indirectamente para los métodos basados en contenido o de descomposición matricial. Incluso es posible obtener resultados aplicando este método directamente a la matriz de calificaciones.

La idea general del método es agrupar los artículos del sistema según sus características. Se debe buscar dividir los artículos de una forma que se logra división representativa, pero que no en demasiados grupos, pues la idea es que el usuario califique un articulo de cada grupo.

Para crear los grupos de artículos, se empleo el método de las K medias. No es necesario el uso de un algoritmo de clustering mas sofisticado, ya que el propósito es crear una separación mas o menos uniforme del espacio, no encontrar agrupamientos lógicos de los artículos.

Una vez que dividimos los artículos, proponemos presentar como recomendaciones el articulo mas relevante de cada cluster. Para este propósito, necesitamos definir una medida de relevancia, inspirados por el TFIDF, proponemos la siguiente métrica para la relevancia t_i para el articulo i

$$t_i = \sum_{j \in U} R_{i,j} log\big(1 + \frac{N}{n_i}\big)$$

Una vez que obtenemos las medidas de relevancia de los artículos, obtenemos el articulo mas relevante de cada cluster, que es el que recomendaremos al usuario inicialmente. Estas recomendaciones son una muestra significativa del espacio gracias al clustering, pero gracias a la medida de relevancia tambien resulta en articulos a los cuales el usuario probablemente ya ha sido expuesto antes, permitiendo que el sistema tenga una idea inicial de su gusto decente.

Una ejemplo de implementación de este metodo puede encontrarse en el anexo A.6.

6 Nota sobre codigo fuente

Todo el codigo relacionado a este proyecto se encuentra en linea en el siguiente repositorio de GitHub: `https://github.com/eltrufas/Sistemas-recomendacion`. Ahi tambien se encuentra el codigo fuente usado para generar este reporte y algunos otros programas adicionales que se desarrollaron en relación a sistemas de recomendación.

7 Conclusión

La variedad de enfoques que se pueden tomar para el desarrollo de sistemas de recomendación hace que sea un área interesante y rica del reconocimiento de patrones. Si se tiene un conocimiento solido del área, los sistemas de recomendación pueden presentar aplicaciones

Metodo	Filtrado de contenido	Vecinos mas cercanos	Regresión/SVD
Facilidad de implementación	⋆	⋆⋆⋆	⋆⋆
Flexibilidad	⋆	⋆⋆⋆	⋆⋆⋆
Calidad de recomendaciones	⋆⋆	⋆⋆	⋆⋆⋆
Uso de memoria	⋆⋆⋆	−⋆	⋆⋆⋆

Comparación de metodos de predicción de preferencias

en muchos dominios. La tabla 1 muestra una comparación cualitativa de los distintos metodos de recomendación revisados.

En general, la conclusión a la que se llego es que para sistemas modernos donde se quieren recomendaciones de alta calidad con muchos usuarios, SVD es el metodo a cual acudir. Esto se ve claramente en el campo de investigación sobre sistemas recomendadores, donde muchos de los nuevos avances son relacionados a la factorización SVD, incluyendo metodos que permiten calcular la descomposición de forma distribuida.

Esto no significa que los otros metodos no tengan su lugar. Los metodos basados en contenido son utiles para campos donde existe mucha información experta o donde se puede minar conocimiento colectivo de otra forma. Los metodos de vecinos mas cercanos son mas faciles de implementar y funcionan bien cuando la proporción del sistema es adecuada.

Los métodos de recomendaciones basadas en contenido y de filtrado colaborativo tienen un lugar importante en el desarrollo de aplicaciones modernas, en parte impulsado por el boom de datos moderno. La recolección de datos masivos hace que sea posible derivar datos de calificación de lugares sorprendentes, abriendo las puertas de creatividad a sistemas de recomendación nuevos e interesantes.

Bibliografía

[1] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[2] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015.

[3] Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32. ACM, 2005.

[4] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[5] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[6] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[7] Slobodan Vucetic and Zoran Obradovic. Collaborative filtering using a regression-based approach. *Knowledge and Information Systems*, 7(1):1–22, 2005.

[8] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug 2009.

A Anexos

A.1 Ejemplo de acceso a la base de datos *Movielens*

```
In [3]: import numpy as np
        import pandas as pd

        df = pd.read_csv('datasets/ml-latest-small/ratings.csv')
        df.columns = ['user_id', 'movie_id', 'rating', 'timestamp']

        df.head()
```

Out[3]:

	user_id	movie_id	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

Podemos usar pandas para preparar los datos a el formato que necesitamos

```
In [4]: ui_df = df.pivot(index='movie_id', columns='user_id', values='rating')

        ui_df.head()
```

Out[4]:

user_id	1	2	3	4	5	6	7	8	9	10	...	662	663
movie_id											...		
1	NaN	NaN	NaN	NaN	NaN	NaN	3.0	NaN	4.0	NaN	...	NaN	4.0
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	5.0	NaN
3	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN

[5 rows x 671 columns]

A.2 Filtrado de contenido para *MovieLens*

MovieLens incluye una base de datos de etiquetas así como la relevancia de cada etiqueta a cada película. Podemos usar estas relevancias para crear nuestra matriz de características para las

películas del conjunto. Hagamos esto y veamos cuales son las etiquetas mas relevantes para Toy Story.

```
In [1]: import pandas as pd
        import numpy as np

        movies = pd.read_csv('datasets/ml-20m/movies.csv', index_col=0)

        genome_tags = pd.read_csv('datasets/ml-20m/genome-tags.csv', index_col=0).tag
        genome_scores = pd.read_csv('datasets/ml-20m/genome-scores.csv')
        genome_scores.columns = ['movie_id', 'tag_id', 'relevance']
        genome_matrix = genome_scores.pivot(columns='tag_id', index='movie_id',
                                            values='relevance')
        genome_matrix.columns = [genome_tags[id] for id in genome_matrix.columns]

        # reducimos las peliculas a las que tienen etiquetas
        movies = movies.loc[genome_matrix.index]

        # normalizamos el genoma
        genome_matrix = genome_matrix * 2 - 1

        genome_matrix.loc[1].sort_values(ascending=False).head(10)
```

```
Out[1]: toys                 0.9985
        computer animation   0.9970
        pixar animation      0.9920
        kids and family      0.9815
        animation            0.9715
        kids                 0.9585
        pixar                0.9335
        children             0.9285
        cartoon              0.9130
        imdb top 250         0.8840
        Name: 1, dtype: float64
```

Ahora vamos a cargar el archivo de calificaciones y seleccionar las calificaciones de un usuario aleatorio.

```
In [2]: ratings = pd.read_csv('datasets/ml-20m/ratings.csv')
        ratings.columns = ['user_id', 'movie_id', 'rating', 'timestamp']
        # reducimos a solo las peliculas con etiquetas
        ratings = ratings.loc[ratings.movie_id.isin(movies.index), :]
```

```
In [3]: user = np.random.choice(ratings.user_id.unique())
```

```
user_ratings = ratings.loc[ratings.user_id == user]
user_ratings = user_ratings.assign(title=[movies.loc[id].title
                                          for id in user_ratings.movie_id])

print(user_ratings.shape)
user_ratings.sort_values(by='rating', ascending=False).head(20)
```

```
(82, 5)
```

Out[3]:

	user_id	movie_id	rating	timestamp	title
13937369	96288	2997	5.0	1159793149	Being John Malkovich (1999)
13937382	96288	4848	5.0	1159792275	Mulholland Drive (2001)
13937362	96288	2329	5.0	1159794208	American History X (1998)
13937367	96288	2858	5.0	1159794335	American Beauty (1999)
13937348	96288	1227	5.0	1159794784	Once Upon a Time in America (1984)
13937347	96288	1222	5.0	1159794563	Full Metal Jacket (1987)
13937345	96288	1193	5.0	1159793167	One Flew Over the Cuckoo's Nest (1975)
13937376	96288	3949	5.0	1159792156	Requiem for a Dream (2000)
13937356	96288	1732	5.0	1159794733	Big Lebowski, The (1998)
13937406	96288	26150	5.0	1159794905	Andrei Rublev (Andrey Rublyov) (1969)
13937396	96288	5954	5.0	1159794332	25th Hour (2002)
13937336	96288	296	5.0	1159792664	
13937334	96288	288	5.0	1159793780	
13937388	96288	4995	4.5	1159792982	
13937403	96288	7371	4.5	1159794656	
13937363	96288	2542	4.5	1159794369	
13937402	96288	7153	4.5	1159792638	
13937365	96288	2692	4.5	1159794799	
13937397	96288	5995	4.5	1159792380	
13937368	96288	2959	4.5	1159793322	

```
13937336                                Pulp Fiction (1994)
13937334                        Natural Born Killers (1994)
13937388                           Beautiful Mind, A (2001)
13937403                                    Dogville (2003)
13937363          Lock, Stock & Two Smoking Barrels (1998)
13937402  Lord of the Rings: The Return of the King, The...
13937365                   Run Lola Run (Lola rennt) (1998)
13937397                                Pianist, The (2002)
13937368                                  Fight Club (1999)
```

Programamos nuestro modelo de regresión y lo corremos para nuestros datos

```
In [4]: from scipy.optimize import fmin_cg

        def fit_users(ratings, mask, features, means=None, lam=20):
            if means is None:
                means = np.zeros(ratings.shape)
            r = mask.astype(int).T
            ratings = ratings.T

            theta_shape = (ratings.shape[1], features.shape[1])

            theta0 = np.random.rand(*theta_shape).flatten()

            def optimization_target(folded):
                theta = np.reshape(folded, theta_shape)
                differences = r * (features @ theta.T - ratings)
                return 0.5 * np.sum(differences**2) + 0.5 * lam * np.sum(theta**2)

            def gradient(folded):
                theta = np.reshape(folded, theta_shape)
                differences = r * (features @ theta.T - ratings)

                return (differences.T @ features + lam * theta).flatten()

            theta = fmin_cg(f=optimization_target, x0=theta0, fprime=gradient)

            return theta

        user_rating_vector = pd.Series(index=movies.index)
        user_rating_vector[user_ratings.movie_id] = user_ratings.rating
        user_ratings_matrix = user_rating_vector.values.reshape(1, -1)
        ratings_mask = np.isfinite(user_ratings_matrix)
        user_ratings_matrix = np.nan_to_num(user_ratings_matrix)
```

```
        theta = fit_users(user_ratings_matrix, ratings_mask, genome_matrix.values)
```

```
Optimization terminated successfully.
         Current function value: 5.082708
         Iterations: 96
         Function evaluations: 210
         Gradient evaluations: 210
```

Ahora podemos tomar el θ que obtuvimos como las preferencias de nuestro usuario

```
In [5]: preferences = pd.Series(theta, index=genome_matrix.columns)

        preferences.sort_values(ascending=False).head(20)
```

```
Out[5]: dark                 0.058778
        masterpiece          0.057257
        cult classic         0.056889
        innocence lost       0.049008
        oscar winner         0.048513
        mentor               0.048007
        mad scientist        0.043156
        imagination          0.041792
        imdb top 250         0.039944
        love                 0.039931
        dreams               0.039512
        gunfight             0.039445
        monster              0.038432
        childhood            0.037762
        prison               0.037117
        social commentary    0.036180
        weird                0.035976
        reflective           0.035807
        great music          0.035077
        alternate universe   0.034496
        dtype: float64
```

Y podemos usar estas preferencias para darle recomendaciones

```
In [6]: predictions = pd.DataFrame(genome_matrix.values @ theta, index=movies.index)

        predictions = predictions.assign(title=movies.title)
        predictions.columns = ['rating', 'title']
        predictions.loc[~ratings_mask.flatten(), :] \
            .sort_values(by=['rating'], ascending=False).head(20)
```

```
Out[6]:             rating                                              title
        movie_id
        778       5.293866                                Trainspotting (1996)
        5147      5.287688         Wild Strawberries (Smultronstället) (1957)
        1237      5.286941    Seventh Seal, The (Sjunde inseglet, Det) (1957)
        111       5.256287                                   Taxi Driver (1976)
        1206      5.253421                          Clockwork Orange, A (1971)
        1251      5.228539                                8 1/2 (8½) (1963)
        99764     5.175642                  It's Such a Beautiful Day (2012)
        7361      5.158738       Eternal Sunshine of the Spotless Mind (2004)
        7068      5.119508  Last Year at Marienbad (L'Année dernière à Mar...
        1228      5.119437                                Raging Bull (1980)
        1199      5.104201                                     Brazil (1985)
        25793     5.091687                                     Vampyr (1932)
        4878      5.089001                               Donnie Darko (2001)
        6440      5.058139                                Barton Fink (1991)
        3476      5.040493                             Jacob's Ladder (1990)
        61240     5.029298  Let the Right One In (Låt den rätte komma in) ...
        4658      5.022652                               Santa Sangre (1989)
        3676      5.021929                                 Eraserhead (1977)
        1213      5.019151                                 Goodfellas (1990)
        55444     5.012329                                    Control (2007)
```

A.3 F.C. con vecinos mas cercanos para *MovieLens*

A continuación se presenta la implementación del metodo de filtrado colaborativo para vecinos mas cercanos para la base de datos de MovieLens. Primero cargamos los datos como lo hemos hecho antes:

```
In [1]: import pandas as pd
        import numpy as np

        pd.read_csv
        ratings_df = pd.read_csv('datasets/ml-latest-small/ratings.csv')
        movies = pd.read_csv('datasets/ml-latest-small/movies.csv', index_col=0)

        ratings_df.columns = ['user_id', 'movie_id', 'rating', 'timestamp']

        user_ratings = ratings_df.pivot(index='movie_id', columns='user_id', values='rating')
```

Escogemos una columna al azar de la matriz, que representa las calificaciones del usuario correspondiente. Imprimimos sus peliculas mejor calificadas.

```
In [2]: user = user_ratings.sample(axis=1).assign(title=movies.title[user_ratings.index])
         user.columns = ['rating', 'title']
```

```
In [2]: user.sort_values(by='rating', ascending=False).head(20)
```

Out[2]:

movie_id	rating	title
1258	5.0	Shining, The (1980)
1255	5.0	Bad Taste (1987)
1349	5.0	Vampire in Venice (Nosferatu a Venezia) (Nosfe...
1348	5.0	Nosferatu (Nosferatu, eine Symphonie des Graue...
1339	5.0	Dracula (Bram Stoker's Dracula) (1992)
1387	5.0	Jaws (1975)
247	5.0	Heavenly Creatures (1994)
1333	4.0	Birds, The (1963)
1261	4.0	Evil Dead II (Dead by Dawn) (1987)
1343	4.0	Cape Fear (1991)
1340	4.0	Bride of Frankenstein, The (Bride of Frankenst...
188	4.0	Prophecy, The (1995)
1321	4.0	American Werewolf in London, An (1981)
1	4.0	Toy Story (1995)
1219	4.0	Psycho (1960)
1215	4.0	Army of Darkness (1993)
25	3.0	Leaving Las Vegas (1995)
296	3.0	Pulp Fiction (1994)
1341	3.0	Burnt Offerings (1976)
593	3.0	Silence of the Lambs, The (1991)

Ahora, inicializamos nuestro algoritmo de aprendizaje.

```
In [3]: from sklearn.neighbors import NearestNeighbors
        ratings_mask = np.isfinite(user_ratings.values).T
        user_matrix = np.nan_to_num(user_ratings.values).T

        nnbrs = NearestNeighbors(n_neighbors=5, algorithm='brute', metric='cosine').fit(user
```

Obtenemos los vecinos de nuestro usuario y aproximamos sus calificaciones tomando en cuenta las calificaciones de sus vecinos.

```
In [4]: distance, neighbors = nnbrs.kneighbors(np.nan_to_num(user.rating.values.reshape(1, -1

        similarity = 1 - distance.flatten()[1:]

        neighbor_mask = ratings_mask[neighbors.flatten()[1:]]
        neighbors = user_matrix[neighbors.flatten()[1:]]

        np.seterr(divide='ignore', invalid='ignore')
```

```
        pred = similarity @ neighbors
        pred = pred / (similarity * neighbor_mask.T).sum(axis=1)

        pred = np.nan_to_num(pred).flatten()
```

Finalmente, para tener una idea de nuestros resultados, imprimimos las películas con mayor calificación predicha.

```
In [5]: user['predicted'] = pred
        user.sort_values(by='predicted', ascending=False).head(20)
```

Out[5]:

movie_id	rating	title	predicted
2650	NaN	Ghost of Frankenstein, The (1942)	5.0
2652	NaN	Curse of Frankenstein, The (1957)	5.0
2634	NaN	Mummy, The (1959)	5.0
2644	NaN	Dracula (1931)	5.0
1721	NaN	Titanic (1997)	5.0
2647	NaN	House of Frankenstein (1944)	5.0
2648	NaN	Frankenstein (1931)	5.0
2649	NaN	Son of Frankenstein (1939)	5.0
2636	NaN	Mummy's Ghost, The (1944)	5.0
2651	NaN	Frankenstein Meets the Wolf Man (1943)	5.0
2635	NaN	Mummy's Curse, The (1944)	5.0
2654	NaN	Wolf Man, The (1941)	5.0
2664	NaN	Invasion of the Body Snatchers (1956)	5.0
2716	NaN	Ghostbusters (a.k.a. Ghost Busters) (1984)	5.0
3018	NaN	Re-Animator (1985)	5.0
1348	5.0	Nosferatu (Nosferatu, eine Symphonie des Graue...	5.0
1354	NaN	Breaking the Waves (1996)	5.0
1219	4.0	Psycho (1960)	5.0
326	NaN	To Live (Huozhe) (1994)	5.0
534	NaN	Shadowlands (1993)	5.0

A.4 Recomendaciones de MovieLens usando filtrado colaborativo con regresion

Comenzamos implementando el algoritmo de regresión para todos nuestro parametros.

```
In [1]: import pandas as pd
        import numpy as np
        from scipy.optimize import fmin_cg


        def simple_fit(ui_matrix, r_matrix, num_features, lam):
```

```
"""
Ajusta el modelo de regresion dada una matriz de calificaciones y
una mascara de calificados
"""
y = ui_matrix
r = r_matrix
num_items, num_users = y.shape

theta0 = np.random.rand(num_users, num_features)
x0 = np.random.rand(num_items, num_features)

def fold_matrices(x_matrix, theta_matrix):
    return np.concatenate([x_matrix.flatten(), theta_matrix.flatten()])

def unfold_vector(x):
    x_matrix = np.reshape(x[:x0.size],
                          x0.shape)
    theta_matrix = np.reshape(x[x0.size:],
                              theta0.shape)
    return x_matrix, theta_matrix

def unfold_parameter(f):
    def wrapper(x):
        return f(*unfold_vector(x))

    return wrapper

@unfold_parameter
def optimization_target(x, theta):
    differences = r * (x @ theta.T - y)
    square_error = (0.5) * np.sum(differences**2)
    regularization = (lam / 2) * (np.sum(x**2) + np.sum(x**2))

    return square_error + regularization

@unfold_parameter
def gradient(x, theta):
    differences = np.multiply((np.dot(x, theta.T) - y), r)
    x_grad = np.dot(differences, theta) + lam * x
    theta_grad = np.dot(x.T, differences).T + lam * theta

    return fold_matrices(x_grad, theta_grad)
```

```
            init_fold = fold_matrices(x0, theta0)
            result = fmin_cg(f=optimization_target, x0=init_fold, fprime=gradient)

            x, theta = unfold_vector(result)

            return x, theta
```

Ahora, envolvemos el algoritmo en una funcion de normalización para obtener los resultados incluyendo normalización de medias.

```
In [2]: def normalized_fit(y, *args):
            means = np.nanmean(y, axis=1)
            y = y - means.reshape(-1, 1)

            r = -(np.isnan(y).astype(int) - 1)
            y = np.nan_to_num(y)

            x, theta = simple_fit(y, r, *args)

            return x, theta, means
```

Como de costumbre, cargamos el conjunto de datos de MovieLens

```
In [3]: ratings_df = pd.read_csv('datasets/ml-latest-small/ratings.csv')
        movies = pd.read_csv('datasets/ml-latest-small/movies.csv', index_col=0)

        ratings_df.columns = ['user_id', 'movie_id', 'rating', 'timestamp']

        user_ratings = ratings_df.pivot(index='movie_id', columns='user_id', values='rating')
```

Ajustamos el modelo de regresión para la matriz usando 200 caracteristicas y un valor de regularización de 0.2.

```
In [4]: x, theta, means = normalized_fit(user_ratings.values, 200, 0.2)

        feature_df = pd.DataFrame(x, index=user_ratings.index)
```

```
Warning: Desired error not necessarily achieved due to precision loss.
         Current function value: 915.220243
         Iterations: 1381
         Function evaluations: 2409
         Gradient evaluations: 2397
```

Seleccionamos un usuario al azar y vemos sus calificaciones mas altas.

```
In [30]: user = user_ratings.sample(axis=1)
         user_id = user.columns[0]
         user = user.assign(title=movies.title[user_ratings.index])
         user.columns = ['rating', 'title']
         user.sort_values(by='rating', ascending=False).head(20)
```

Out[30]:

movie_id	rating	title
47	5.0	Seven (a.k.a. Se7en) (1995)
318	5.0	Shawshank Redemption, The (1994)
1704	5.0	Good Will Hunting (1997)
1196	5.0	Star Wars: Episode V - The Empire Strikes Back...
1387	5.0	Jaws (1975)
1407	5.0	Scream (1996)
1625	5.0	Game, The (1997)
1617	5.0	L.A. Confidential (1997)
1689	4.0	Man Who Knew Too Little, The (1997)
1672	4.0	Rainmaker, The (1997)
1619	4.0	Seven Years in Tibet (1997)
1597	4.0	Conspiracy Theory (1997)
1584	4.0	Contact (1997)
1438	4.0	Dante's Peak (1997)
953	4.0	It's a Wonderful Life (1946)
1183	4.0	English Patient, The (1996)
1198	4.0	Raiders of the Lost Ark (Indiana Jones and the...
1488	3.0	Devil's Own, The (1997)
1687	3.0	Jackal, The (1997)
1686	3.0	Red Corner (1997)

Calculamos las calificaciones aproximadas del usuario como $X^T\theta^{(j)} + \bar{X}$

```
In [31]: theta_df = pd.DataFrame(theta, index=user_ratings.columns)
         user_theta = theta_df.loc[user_id]

         pred = (user_theta.values @ x.T) + means
```

```
In [32]: user['predicted'] = pred
         user.sort_values(by='predicted', ascending=False).head(20)
```

Out[32]:

movie_id	rating	title	predicted
593	NaN	Silence of the Lambs, The (1991)	5.344535
2571	NaN	Matrix, The (1999)	5.034324
4088	NaN	Big Town, The (1987)	5.000031
92494	NaN	Dylan Moran: Monster (2004)	5.000028

```
5960         NaN                                Bad Influence (1990)   5.000025
3216         NaN             Vampyros Lesbos (Vampiras, Las) (1971)   5.000024
4617         NaN                                   Let It Ride (1989)   5.000023
4522         NaN                                    Masquerade (1988)   5.000022
3038         NaN                          Face in the Crowd, A (1957)   5.000014
26151        NaN                           Au Hasard Balthazar (1966)   5.000005
107412       NaN  Kidnapping, Caucasian Style (Kavkazskaya plenn...   5.000005
62115        NaN                                   Six Shooter (2004)   5.000004
118468       NaN                        Mei and the Kittenbus (2002)   5.000004
961          NaN                       Little Lord Fauntleroy (1936)   5.000004
1531         NaN                                  Losing Chase (1996)   5.000003
50703        NaN                                   Secret, The (2006)   5.000003
3281         NaN                    Brandon Teena Story, The (1998)   5.000003
32460        NaN                  Knockin' on Heaven's Door (1997)   5.000003
107559       NaN  Am Ende eiens viel zu kurzen Tages (Death of a...   5.000003
3612         NaN  The Slipper and the Rose: The Story of Cindere...   5.000003
```

A.5 Filtrado colaborativo con descomposicion SVD

Usemos SVD para acelerar nuestro filtrado colaborativo de películas. Cargamos los datos y preparamos nuestra matriz con normalización de medias.

```
In [1]: import pandas as pd
        import numpy as np

        movies = pd.read_csv('datasets/ml-latest-small/movies.csv', index_col=0)
        ratings_df = pd.read_csv('datasets/ml-latest-small/ratings.csv')
        ratings_df.columns = ['user_id', 'movie_id', 'rating', 'timestamp']

        user_ratings = ratings_df.pivot(index='movie_id', columns='user_id',
                                        values='rating')
```

```
In [2]: ui_matrix = np.copy(user_ratings.values)
        popularity = np.isfinite(ui_matrix).astype(int).sum(axis=1)
        means = np.nanmean(ui_matrix, axis=1)
        ui_matrix = ui_matrix - means.reshape(-1, 1)
        ui_matrix = np.nan_to_num(ui_matrix)
```

Hacemos nuestra descomposición matricial especificando 500 caracteristicas.

```
In [3]: from scipy.sparse.linalg import svds

        u, s, vt = svds(ui_matrix, k=500)
```

Seleccionamos un usuario aleatorio.

```
In [4]: user = user_ratings.sample(axis=1)
        user_id = user.columns[0]
        user = user.assign(title=movies.title[user_ratings.index])
        user.columns = ['rating', 'title']
        user.sort_values(by='rating', ascending=False).head(20)
```

Out[4]:

movie_id	rating	title
1	5.0	Toy Story (1995)
592	5.0	Batman (1989)
2858	5.0	American Beauty (1999)
2797	5.0	Big (1988)
3114	5.0	Toy Story 2 (1999)
3176	5.0	Talented Mr. Ripley, The (1999)
1639	5.0	Chasing Amy (1997)
3481	5.0	High Fidelity (2000)
3578	5.0	Gladiator (2000)
1356	5.0	Star Trek: First Contact (1996)
1259	5.0	Stand by Me (1986)
1221	5.0	Godfather: Part II, The (1974)
1214	5.0	Alien (1979)
1198	5.0	Raiders of the Lost Ark (Indiana Jones and the...
1193	5.0	One Flew Over the Cuckoo's Nest (1975)
4306	5.0	Shrek (2001)
2918	5.0	Ferris Bueller's Day Off (1986)
5952	5.0	Lord of the Rings: The Two Towers, The (2002)
4846	5.0	Iron Monkey (Siu nin Wong Fei-hung ji: Tit Ma ...
260	5.0	Star Wars: Episode IV - A New Hope (1977)

Calculamos nuestras calificaciones aproximadas.

```
In [5]: theta_df = pd.DataFrame(vt.T, index=user_ratings.columns)

        user_theta = theta_df.loc[user_id]

        pred = (user_theta.values @ (s * u).T) + means
```

Imprimimos los resultados.

```
In [6]: user['predicted'] = pred
        user['relevance'] = user['predicted'] * popularity
        user[user.rating.isnull()].sort_values(by='relevance', ascending=False).head(20)
```

Out[6]:

movie_id	rating	title	\

296	NaN	Pulp Fiction (1994)
356	NaN	Forrest Gump (1994)
527	NaN	Schindler's List (1993)
1196	NaN	Star Wars: Episode V - The Empire Strikes Back...
608	NaN	Fargo (1996)
589	NaN	Terminator 2: Judgment Day (1991)
1270	NaN	Back to the Future (1985)
110	NaN	Braveheart (1995)
858	NaN	Godfather, The (1972)
1210	NaN	Star Wars: Episode VI - Return of the Jedi (1983)
50	NaN	Usual Suspects, The (1995)
47	NaN	Seven (a.k.a. Se7en) (1995)
588	NaN	Aladdin (1992)
32	NaN	Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
780	NaN	Independence Day (a.k.a. ID4) (1996)
364	NaN	Lion King, The (1994)
2028	NaN	Saving Private Ryan (1998)
590	NaN	Dances with Wolves (1990)
1580	NaN	Men in Black (a.k.a. MIB) (1997)
1197	NaN	Princess Bride, The (1987)

movie_id	predicted	relevance
296	4.248962	1376.663713
356	4.032926	1375.227797
527	4.304056	1050.189678
1196	4.237824	991.650929
608	4.257189	953.610249
589	4.005997	949.421358
1270	4.014198	907.208763
110	3.937018	897.640189
858	4.465236	893.047173
1210	4.057057	880.381442
50	4.344517	873.247952
47	4.026175	809.261176
588	3.662208	787.374662
32	3.928926	770.069566
780	3.487695	760.317444
364	3.794904	758.980718
2028	3.934426	751.475424
590	3.707650	748.945372
1580	3.671700	697.622927
1197	4.206574	685.671613

A.6 Recomendaciones iniciales con clustering en MovieLens

Se presenta una implementacion del algoritmo de recomendaciones iniciales usando clustering descrito en la seccion 5 para una matriz de contenido obtenida por factorización SVD usando la base de datos de *MovieLens*.

```
In [1]: import pandas as pd
    import numpy as np

    movies = pd.read_csv('datasets/ml-latest-small/movies.csv', index_col=0)
    ratings_df = pd.read_csv('datasets/ml-latest-small/ratings.csv')
    ratings_df.columns = ['user_id', 'movie_id', 'rating', 'timestamp']

    user_ratings = ratings_df.pivot(index='movie_id', columns='user_id',
                                    values='rating')
```

```
In [2]: ui_matrix = np.copy(user_ratings.values)
    popularity = np.isfinite(ui_matrix).astype(int).sum(axis=1)
    means = np.nanmean(ui_matrix, axis=1)
    ui_matrix = ui_matrix - means.reshape(-1, 1)
    ui_matrix = np.nan_to_num(ui_matrix)
```

```
In [3]: from scipy.sparse.linalg import svds

    u, s, vt = svds(ui_matrix, k=500)
```

Ahora realizamos clustering K medias para encontrar clusters de películas similares. El numero de clusters es una situación delicada. Necesitamos un numero de clusters que divida los artículos representativamente, pero tampoco puede ser muy bajo, ya que la idea es tener al usuario calificar un articulo de cada cluster.

```
In [4]: from sklearn.cluster import KMeans

    n_clusters = 10
    kmeans = KMeans(n_clusters)
    clusters = kmeans.fit_predict(u)

    cluster_mask = np.asarray([clusters == i for i in range(n_clusters)])

    ratings_mask = np.isfinite(user_ratings.values)
```

Lo siguiente es obtener el articulo mas relevante de cada cluster. Para hacer esto, primero necesitamos una métrica de relevancia para nuestros articulos. Inspirandonos en el TFIDF, definimos la relevancia de un articulo i en funcion sus calificaciones existentes R_i como

$$t_i = \sum_{j \in U} R_{i,j} log(1 + \frac{N}{n_i})$$

Usando esta métrica, calculamos la relevancia de todos los artículos y ordenamos los objetos de nuestros clusters usándola.

```
In [5]: np.sum(ratings_mask, axis=1).size
    relevance = ((np.sum(ui_matrix, axis=1) / user_ratings.shape[1]) *
                 (user_ratings.shape[0] / np.sum(ratings_mask, axis=1)))

    relevance_df = pd.DataFrame(relevance, index=user_ratings.index)
    relevance_df['title'] = [movies.title[id] for id in relevance_df.index]
    relevance_df.columns = ['relevance', 'title']

    relevance_df.sort_values(by='relevance', ascending=False)



    masked_array = np.tile(relevance, (cluster_mask.shape[0], 1))
    masked_array[~cluster_mask] = -np.inf
    sorted_array = np.argsort(masked_array, axis=1)

    relevance_df.iloc[sorted_array[:, -1]]
```

Obtenemos los articulos mas relevantes de cada cluster. Estas seran las que recomendaremos al usuario.

```
Out[5]:              relevance                                              title
    movie_id
    1304      5.920166e-15          Butch Cassidy and the Sundance Kid (1969)
    4447     -2.042610e-15                                 Legally Blonde (2001)
    3354      8.000224e-16                               Mission to Mars (2000)
    344      -4.114401e-16                    Ace Ventura: Pet Detective (1994)
    778      -5.806614e-15                                 Trainspotting (1996)
    3730      5.739291e-15                              Conversation, The (1974)
    780      -5.504741e-16                 Independence Day (a.k.a. ID4) (1996)
    25       -2.613935e-15                               Leaving Las Vegas (1995)
    7153     -1.159123e-15  Lord of the Rings: The Return of the King, The...
    1196     -8.205358e-16  Star Wars: Episode V - The Empire Strikes Back...
```