Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Técnicas de Factorización No-negativa de Matrices en Sistemas de Recomendación

Autor: Miguel Ángel Pérez García

Tutor: Sergio Antonio Cruces Álvarez

Dep. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Técnicas de Factorización No-negativa de Matrices en Sistemas de Recomendación

Autor:

Miguel Ángel Pérez García

Tutor:

Sergio Antonio Cruces Álvarez

Profesor Titular

Dep. de Teoría de la Señal y Comunicaciones

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado: Técnicas de Factorización No-negativa de Matrices en Sistemas de Recomendación

Autor: Miguel Ángel Pérez García

Tutor: Sergio Antonio Cruces Álvarez

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

Agradecimientos

Esta página está dedicada a las personas que han hecho que este Trabajo Fin de Grado sea posible. Para empezar, quiero agradecer a mis padres la paciencia de tenerme en casa durante estos diez años de maduración, sin desterrarme ni animarme a dejar la Escuela atrás —más bien todo lo contrario—. En segundo lugar, quiero dar gracias a mi hermana pequeña por su ejemplo motivador, mostrándome lo que su incansable tesón es capaz de conseguir. Gracias también a mi pareja por los buenos momentos vividos con ella durante la carrera. Gracias a los compañeros que han hecho que este largo viaje haya parecido más breve de lo que realmente ha sido.

Por último, no puedo cerrar este apartado sin dar las gracias a la E.T.S.I. y a su cuerpo docente por ayudarme a finalizar satisfactoriamente esta etapa universitaria que espero que haya formado los cimientos de una trayectoria profesional de éxito.

Miguel Ángel Pérez García

Sevilla, 2017

Resumen

Cualquier persona que hoy en día utilice internet está, de un modo u otro, alimentando de datos a más de un sistema de recomendación que se encarga de personalizar su experiencia de uso. Ya sea haciendo compras online, visitando páginas de noticias, participando en redes sociales, escuchando música, viendo películas, utilizando aplicaciones en su teléfono móvil, o simplemente exponiendo su navegador web a cookies, el usuario está proporcionando información a terceros que estos consideran de valor, debido a que estos datos pueden utilizarse para hacer recomendaciones individualizadas a potenciales clientes de productos que aún no han consumido, en base a los que ya han consumido tanto ellos mismos como otros clientes con gustos similares. De esta forma, la personalización del servicio se erige como elemento diferenciador frente a otras plataformas que no incorporan un sistema de recomendación en su modelo.

Detrás de un sistema de recomendación siempre hay un algoritmo informático que toma los datos pertinentes y otorga una puntuación a cada uno de los artículos disponibles en base a dichos datos, de forma que los que obtienen mejor puntuación son priorizados frente al resto a la hora de ser mostrados al usuario. La finalidad que tenga el sistema de recomendación, así como la naturaleza de los datos que se manejan, hacen que puedan distinguirse varias implementaciones matemáticas. En este Trabajo Fin de Grado vamos a centrarnos en una de las implementaciones que mejor funcionan, una que hace uso de la factorización no-negativa de matrices, o NMF (Non-negative Matrix Factorization), que no es más que la aproximación de una matriz cuyos elementos son positivos o cero por la multiplicación de otras dos, típicamente de menores dimensiones que la original.

Abstract

Anyone using the internet today is, one way or another, feeding data to a recommender system that enhances the consumer experience. Be it shopping online, browsing news sites, taking part in social networks, listening to music, watching movies, using smartphone apps or just exposing the web browser of choice to cookies, these actions are giving valuable information to third-parties that can be used to provide customized recommendations to potential clients on products yet to be consumed, based on the data generated by those same clients or other clients with similar tastes. Service customization can be a decisive element for the consumer when confronted with several platform choices, making it a vital component for the business model.

The heart of a recommender system is always a computer algorithm that takes user data as input and assigns a score, or rating, to the available products in order to prioritize those with a higher score when the system is in need of presenting the consumer with a selection of products. The mathematical approach to the algorithm can be different depending on the goal of the recommender system, as well as the nature of the retrievable data. In this document, we are going to focus on an implementation based on Non-negative Matrix Factorization, or NMF, which approximates a matrix with elements greater than or equal to zero by the product of other two matrices, usually of a smaller size than the original.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

NMF	Non-negative Matrix Factorization
RS	Recommender System(s)
CF	Collaborative Filtering
D(X, Y)	Distancia o divergencia entre las matrices X e Y
d(x_{ij}, y_{ij})	Distancia o divergencia entre dos elementos individuales de X e Y
X^{T}	Matriz traspuesta de X
$\boldsymbol{x}_{i}$	Vector fila i de la matriz X
$\hat{X}$	Matriz aproximada de X
$\approx$	Aproximadamente igual
RMS(X)	Valor cuadrático medio de X
$\frac{\partial x}{\partial y}$	Derivada parcial de x con respecto a y

1 INTRODUCCIÓN

By far, the greatest danger of artificial intelligence is that people conclude too early that they understand it.

- Eliezer Yudkowsky -

En este capítulo inicial se da a conocer al lector el "por qué" y el "cómo" de este documento para que pueda formarse una visión general del mismo antes de entrar en materia. Por lo tanto, se abordan tanto la motivación como los objetivos que se pretenden alcanzar, hablando por último de la estructura que presenta el documento en sí.

1.1 Motivación

Los sistemas de recomendación están presentes en la vida cotidiana a poco que se utilice algún servicio online en el que se tenga que decidir entre varios objetos, digitales o físicos, a la hora de consumirlos. El cometido de un sistema de recomendación es valerse de los datos proporcionados por los usuarios del mismo, directa o indirectamente, para tratar de averiguar qué producto o productos pueden ser del agrado del consumidor en base a los que anteriormente haya consumido, o en base a las opiniones que haya manifestado, tanto él como otros usuarios que el sistema considere afines. Este concepto puede ser aplicable también a motores de búsqueda. El motor recomienda las páginas web que a su criterio están más relacionadas con lo que se haya introducido en la cadena de búsqueda, por lo que tiene que llevar a cabo un proceso de valoración y recomendación.

Este proceso, desde la disposición de los datos recolectados hasta la recomendación al usuario, está cubierto por un algoritmo informático que se sirve de algún principio matemático para otorgar puntuaciones numéricas a los productos sujetos a recomendación, de modo que los que hayan obtenido una puntuación mayor son mostrados primero al usuario por pensarse que el mismo los valoraría mejor en caso de haberlos conocido y consumido con anterioridad. Por lo tanto, lo que el sistema está llevando a cabo es un ejercicio de *predicción*.

Los algoritmos que se utilizan con este fin hoy en día son potentes, pero susceptibles de ser mejorados en

varios ámbitos, como por ejemplo la carga computacional que suponen, ya sea tiempo de ejecución o memoria requerida, o la propia calidad final de las recomendaciones que generan. Esto último es de vital importancia, como evidenció la compañía Netflix con su famoso concurso que premiaba con un millón de dólares a aquellos que fuesen capaces de diseñar un algoritmo que obtuviera una precisión en las predicciones un 10% mejor que la obtenida por el algoritmo que estaba siendo utilizado en la plataforma en ese momento [1]. Si un sistema de recomendación no es bueno haciendo su trabajo, los usuarios le darán la espalda en favor de otras alternativas.

Aparte de la búsqueda de buenos resultados, la aplicación de técnicas de factorización no-negativa al problema de la recomendación tiene otra razón de peso fundamental, y no es más que la no-negatividad de los elementos que intervienen en el proceso en sí misma. Otras técnicas algorítmicas de búsqueda de componentes principales, como PCA o ICA, pueden dar lugar a combinaciones en las que unas direcciones principales se cancelan con otras, de modo que se pierde el sentido de que hay un todo que está formado por la suma aditiva de sus partes. La no-negatividad refuerza este concepto, dándole un sentido físico que antes no tenía.

Se entiende, por tanto, que la motivación de este documento es realizar una labor instructiva no solo sobre los sistemas de recomendación, sino también sobre los entresijos matemáticos que lo hacen funcionar.

1.2 Alcance y objetivos

Este Trabajo Fin de Grado trata de arrojar luz sobre los sistemas de recomendación a aquellas personas que no posean formación al respecto, por lo que es necesario mantener un carácter didáctico. Se aborda la teoría general para después centrarse en un caso práctico concreto, el de la predicción de valoraciones de usuarios en productos. Por lo tanto, solo se verá en más profundidad este caso concreto, quedando el resto fuera del alcance del documento.

En cuanto al aspecto matemático y algorítmico, aunque existen varias técnicas que posibilitan la recomendación, este documento solo se centra en una: la Factorización No-negativa de Matrices (siglas NMF en inglés). En realidad, como se verá más adelante, la factorización no-negativa de matrices se trata una familia de técnicas algorítmicas, por lo que será necesario fijarse en una implementación en concreto para poder realizar las simulaciones pertinentes. El resto de implementaciones quedarán solo en el ámbito de la teoría.

Además de introducir conceptos de aprendizaje automático, otro objetivo del documento es comparar los resultados obtenidos de la simulación del algoritmo variando sus parámetros, de forma que pueda verse claramente mediante gráficas la evolución de las métricas de evaluación al enfrentar unos resultados con otros.

1.3 Visión general del documento

Tras esta introducción, se dedica un capítulo a la Factorización No-negativa de Matrices, o NMF, exponiendo en qué consiste e introduciendo cada una de las diferentes implementaciones posibles. Se dan alternativas a la medida de similitud, que mide el grado de calidad de la aproximación a cada paso del algoritmo, y se define cómo se deben dar esos pasos. También se prueba inequívocamente que la convergencia a un mínimo está garantizada para la implementación elegida.

Seguidamente, se abordan los Sistemas de Recomendación, o RS, viendo cómo las técnicas de NMF pueden solucionar el problema matemático que plantean. Se discuten varias formas de acercarse al propósito de la predicción de valoraciones de los usuarios y el de la recomendación basándose en esas valoraciones. Se adapta la nomenclatura de las técnicas NMF a los Sistemas de Recomendación, de forma que pueda verse claramente lo directa que es la aplicación y lo natural que resulta.

En el siguiente capítulo, se utiliza MATLAB® para simular la aplicación de una de las técnicas de NMF más básicas a un set de datos de valoraciones a películas, y se analizan los resultados obtenidos mediante salida textual y gráficas. Se explicará todo el proceso de la mano del código, dando detalles de la implementación algorítmica y mostrando capturas de pantalla.

Finalmente, se extraen las conclusiones pertinentes y se exploran futuras líneas de investigación para las que el NMF puede suponer una base sólida, intentando a la vez buscar soluciones a algunas incógnitas que plantea el uso de este tipo de técnicas y que aún no han podido responderse de manera inequívoca.

2 FACTORIZACIÓN NO-NEGATIVA DE MATRICES

We don't have better algorithms, we just have more data.

- Peter Norvig -

En este capítulo se explica en qué consiste el NMF, por y para qué se usa, y las distintas ramas de técnicas algorítmicas que existen actualmente en este ámbito. En particular, se pone el foco en una de las implementaciones básicas para poder entender con más facilidad la matemática que hay detrás de la misma. En resumen, el NMF intenta aproximar una matriz por la multiplicación de otras dos.

Se introducen distintas medidas de similitud entre la matriz aproximada y la original, así como reglas de actualización de los valores en el algoritmo que garantizan que la similitud vaya siempre en aumento.

Otro aspecto importante que se trata aquí es la relación que existe entre el resultado obtenido de aplicar NMF a un conjunto de datos y el estudio de las componentes latentes que se ocultan en el mismo, que ayudan a tener una comprensión más sencilla de la mecánica que hay detrás y que genera dichos datos. Podría afirmarse que esta es la razón principal de usar NMF frente a otro tipo de técnicas algorítmicas.

2.1 Introducción a NMF

La Factorización No-negativa de Matrices, o NMF, consiste en la aproximación de una matriz cuyos valores son positivos o nulos mediante la multiplicación de dos submatrices de elementos también no-negativos que comparten una dimensión previamente determinada. Matemáticamente hablando, lo que se persigue es aproximar una matriz X de dimensiones $I \times J$ por $\hat{X}$, la multiplicación de las matrices A, de dimensiones $I \times K$, y B, de dimensiones $J \times K$, siendo esta última traspuesta previamente para que las dimensiones encajen, y con todos los elementos x_{ij}, a_{ik}, b_{jk} positivos o nulos. Dado el carácter de aproximación, se introduce una matriz E de error de las mismas dimensiones, de modo que pueda hablarse de igualdad.

$$X = AB^T + E$$
$$x_{ij}, a_{ik}, b_{jk} \geq 0 \tag{2-1}$$

Si se quiere hablar de aproximación, se tiene la siguiente expresión.

$$X \approx \hat{X} = AB^T \tag{2-2}$$

La dimensión *K* se corresponde con el número de lo que se conoce como componentes latentes. En NMF, las componentes latentes son el rango de factorización, la dimensión intermedia entre las dos nuevas matrices cuya multiplicación pretende asemejarse a la matriz original. Se las llama componentes porque tratan de recomponer la matriz original a través de unas nuevas bases, y se las llama latentes porque no emergen hasta que el algoritmo de NMF las construye. Normalmente, el número de componentes latentes se escoge para que el número de elementos de las dos matrices de factorización sea menor que el número de elementos en la matriz original.

$$(I + J)K < IJ \tag{2-3}$$

Si esta condición se cumple, aplicar NMF a la matriz *X* puede entenderse como una compresión de datos (con pérdidas, ya que se trata de una aproximación). No obstante, en principio no existe ninguna restricción en el número de componentes. De hecho, elegir una *K* de forma que se supere con creces el número original de elementos puede tener utilidad desde el punto de vista de matrices dispersas [2], en las que casi todos los elementos resultantes son cero y puede establecerse fácilmente cuáles son las componentes más importantes para la labor de reconstrucción de la matriz original. En este documento se va a respetar la ecuación (2-3) para que las simulaciones del algoritmo NMF no sean excesivamente pesadas.

Una posible forma de acercarse al significado real del NMF es fijarse en un elemento cualquiera de la matriz *X* y ver con qué se corresponde al otro lado de la ecuación:

$$x_{ij} = \boldsymbol{a}_i \boldsymbol{b}_j^T + e_{ij} \tag{2-4}$$

Como se puede comprobar, cada elemento de *X* es aproximado por la multiplicación de dos vectores fila de *A* y *B* (este último traspuesto). La fila de los vectores se corresponde, en cada uno de ellos, con uno de los dos subíndices del elemento de *X*. Si se desglosa aún más esta representación, se tiene que:

$$x_{ij} \approx \hat{x}_{ij} = \sum_{k=1}^{K} a_{ik} b_{jk} \tag{2-5}$$

De esta manera se hace más claramente visible que se está sumando en todas las componentes *k* para intentar obtener el total. Si se tiene en cuenta el carácter aditivo de la aproximación junto con el hecho de que los elementos son no-negativos, esto indica que la visión que el NMF aporta es la de un todo compuesto por la suma de sus partes [2], las componentes latentes, que permanecían ocultas en el conjunto de datos y ahora se muestran al ojo observador, reforzando la creencia de que existen unas reglas sencillas para explicar lo que aparentemente es complejo. Cuanto mayor sea el término multiplicativo individual $a_{ik}b_{jk}$ del sumatorio, mayor peso tiene en la suma y, por lo tanto, mayor afinidad tiene el elemento x_{ij} con la componente *k* correspondiente.

Un ejemplo muy ilustrativo de este hecho es la reconstrucción facial expuesta en [3]. La escala de grises está representada por valores numéricos entre 0 y 1. El algoritmo NMF forma dos matrices que se multiplican tras su finalización. Una de ellas acaba conteniendo lo que puede entenderse como partes de la cara (ojos, boca,

sombra de la barbilla, cejas, etc.) y la otra contiene valores que indican visibilidad (cuanto más oscuro, más visible), de modo que al multiplicarse una matriz con otra se tiene cómo de visible es cada parte obtenida, teniéndose una aproximación bastante buena del rostro original.

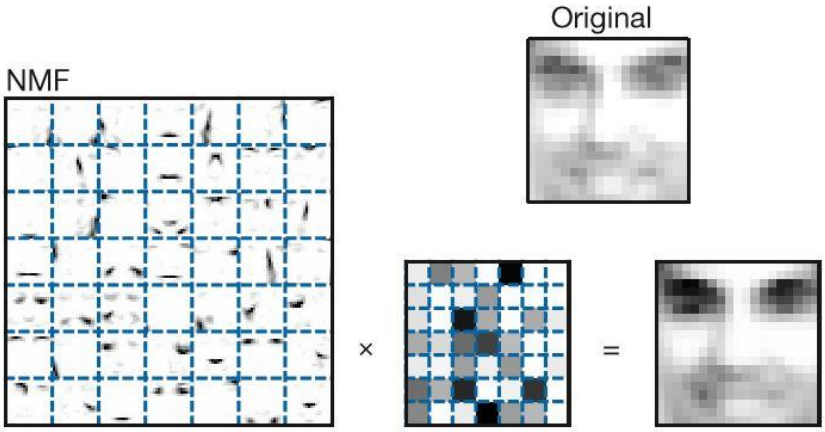

Figura 2-1. Reconstrucción de un rostro mediante NMF.

Otros algoritmos, como PCA o ICA, dan una visión mucho más completa del conjunto, pero en el proceso se pierde este sentido de identificación de las partes de dicho conjunto, ya que no contemplan la restricción de no-negatividad que el NMF tiene de manera natural.

Para factorizar una matriz mediante NMF existen varias vías. Todas ellas están basadas en lo que en definitiva son diferentes algoritmos iterativos, aunque en el fondo todos estén bajo el paraguas de la factorización no-negativa, cada uno con distinta medida de la distancia de la aproximación a la matriz original y reglas de actualización que mantengan dicho carácter positivo [3]. El acercamiento a la solución del problema puede también ser distinto en su concepción, existiendo modelos probabilísticos, bio-inspirados y más [4]. En este Trabajo Fin de Grado se centra la atención en lo que podría considerarse la versión más básica de los algoritmos NMF, que solo aprovecha la no-negatividad de las matrices involucradas, sin apenas imponer restricciones adicionales, de modo que el lector pueda hacerse una idea más clara de los conceptos manejados.

2.2 Medida de similitud

Para saber cómo de parecida es la matriz aproximada a la matriz original, es necesario definir una función que lo exprese mediante un solo número, una distancia que pueda compararse de forma directa (a menor distancia, mejor aproximación) como una función de coste en un proceso de optimización que indique si el algoritmo está más cerca de la solución tras cada iteración o no. Se entiende la distancia como la suma de las distancias individuales entre las matrices original y aproximada, elemento a elemento:

$$D(X, \hat{X}) = \sum_{ij} d(x_{ij}, \hat{x}_{ij}) \tag{2-6}$$

Hay muchas funciones que pueden servir para medir la distancia o divergencia entre los elementos de dos matrices de igual dimensión [5]. En los siguientes subapartados, se exponen algunas de las más utilizadas en técnicas de NMF.

2.2.1 Distancia euclídea

La función de distancia más sencilla y conocida es la euclídea, también llamada norma-2.

$$D_E(X,\hat{X}) = \frac{1}{2}\sum_{ij}(x_{ij} - \hat{x}_{ij})^2 \tag{2-7}$$

Sirve como medida de distancia porque elevar al cuadrado elimina posibles valores negativos. Se divide entre dos el resultado para mayor comodidad a la hora de derivar la función, cuya utilidad veremos en la subsección sobre las reglas de actualización.

Un punto en contra de esta distancia es que elevar al cuadrado acentúa mucho las distancias grandes y disminuye las que sean menores que la unidad. Por esto, es común normalizar el conjunto de datos previamente si los datos son muy dispares, para que los resultados no se desvirtúen demasiado.

Esta distancia es la elegida como medida de similitud en este trabajo, por su facilidad en su manejo durante los desarrollos matemáticos y su buena relación complejidad-resultados.

2.2.2 Distancia Kullback-Leibler

La distancia K-L, llamada así en honor a sus creadores Solomon Kullback y Richard Leibler, aprovecha la no-negatividad de los elementos de las matrices para servirse de propiedades logarítmicas. Si se observa la fórmula detenidamente, se puede comprobar que, si x_{ij} y su aproximación son iguales, entonces el término correspondiente del sumatorio se anula, con lo cual sirve como medida de distancia.

$$D_{KL}(X,\hat{X}) = \sum_{ij}(x_{ij}\ln\frac{x_{ij}}{\hat{x}_{ij}} - x_{ij} + \hat{x}_{ij}) \tag{2-8}$$

Esta distancia suele usarse con distribuciones probabilísticas, y da una idea de la información que se pierde al tomar decisiones basándose en una distribución aproximada en lugar de la verdadera. Junto con la distancia euclídea, es la más usada en documentos académicos de la última década que estudian técnicas de NMF, ya que su derivada también es muy sencilla de implementar en las reglas de actualización.

2.2.3 Distancia Itakura-Saito

La distancia Itakura-Saito, propuesta por los japoneses Fumitada Itakura y Shuzo Saito en los años 60, fue utilizada inicialmente para medir similitud (o falta de ella) entre un espectro original y uno aproximado. Como es natural, encaja perfectamente como medida de similitud aplicable a algoritmos iterativos de aproximación.

$$D_{IS}(X,\hat{X}) = \sum_{ij}(\ln\frac{\hat{x}_{ij}}{x_{ij}} + \frac{x_{ij}}{\hat{x}_{ij}} - 1) \tag{2-9}$$

Esta distancia es muy utilizada para aplicaciones de audio (espectro frecuencial), como separación ciega de fuentes de sonido mezcladas. Cabe destacar que esta distancia no es simétrica, por lo que se tiene que $D_{IS}(X,\hat{X}) \neq D_{IS}(\hat{X},X)$. Si se quiere obtener simetría, una posibilidad es calcular la media de las dos variantes.

2.2.4 **Distancia Alfa-Beta**

La distancia alfa-beta abarca toda una familia de divergencias, ya que está parametrizada por partida doble con α y β. Es la más general de las expuestas en este documento. De hecho, todas las distancias vistas anteriormente pueden deducirse a partir de esta como una particularización de la misma para ciertos valores de los parámetros α y β [5].

$$D_{AB}^{(\alpha,\beta)}(X,\hat{X}) = -\frac{1}{\alpha\beta}\sum_{ij}\left(x_{ij}^{\alpha}\hat{x}_{ij}^{\beta} - \frac{\alpha}{\alpha+\beta}x_{ij}^{\alpha+\beta} - \frac{\beta}{\alpha+\beta}\hat{x}_{ij}^{\alpha+\beta}\right), \quad \alpha,\beta,\alpha+\beta \neq 0 \tag{2-10}$$

Es una medida de similitud robusta ante el ruido presente en los datos, ya que, según el tipo de ruido, pueden ajustarse los parámetros para mejorar el comportamiento del algoritmo.

2.3 Reglas de actualización

Una vez elegida una medida de similitud, el siguiente paso consiste en concebir un algoritmo iterativo que trate de maximizar la similitud entre la matriz aproximada y la original, siguiendo unas reglas para la actualización de las variables implicadas. En este caso, las variables a actualizar son los elementos a_{ik} y b_{jk} de las matrices de la factorización.

Una forma de maximizar la similitud es minimizar la distancia. En nuestro caso, por sencillez en el desarrollo, se elige la distancia euclídea.

$$D = \frac{1}{2}\sum_{ij}\left(x_{ij} - \sum_{k} a_{ik}b_{jk}\right)^2 \tag{2-11}$$

Si se quiere minimizar esta función, una forma válida de hacerlo es mediante un descenso iterativo por el gradiente [3], que toma esta forma:

$$a_{ik} \leftarrow a_{ik} - \alpha_{ik}\frac{\partial D}{\partial a_{ik}}$$
$$b_{jk} \leftarrow b_{jk} - \beta_{jk}\frac{\partial D}{\partial b_{jk}} \tag{2-12}$$

Donde α_{ik} y β_{jk} son números reales positivos que corresponden a las tasas de aprendizaje de cada variable (cuanto mayor sean, más pronunciado será cada paso en el descenso). Resolviendo las derivadas parciales en ambos casos y recordando que $\sum_k a_{ik}b_{jk} = \hat{x}_{ij}$ para simplificar el sumatorio, las reglas quedan:

$$a_{ik} \leftarrow a_{ik} + \alpha_{ik}\sum_{j} b_{jk}(x_{ij} - \hat{x}_{ij})$$
$$b_{jk} \leftarrow b_{jk} + \beta_{jk}\sum_{i} a_{ik}(x_{ij} - \hat{x}_{ij}) \tag{2-13}$$

Resolviendo los paréntesis:

$$a_{ik} \leftarrow a_{ik} + \alpha_{ik} \sum_j b_{jk} x_{ij} - \alpha_{ik} \sum_j b_{jk} \hat{x}_{ij}$$
$$b_{jk} \leftarrow b_{jk} + \beta_{jk} \sum_i a_{ik} x_{ij} - \beta_{jk} \sum_i a_{ik} \hat{x}_{ij} \tag{2-14}$$

Estas reglas de actualización presentan un grave problema: no sirven para aplicar factorización no-negativa, ya que la propiedad innegociable de la no-negatividad puede verse comprometida por el tercer término al llevar un signo negativo. Para solucionar esto, pueden elegirse unas tasas de aprendizaje que permitan mantener la no-negatividad en la actualización:

$$\alpha_{ik} = \frac{a_{ik}}{\sum_j b_{jk} \hat{x}_{ij}}, \qquad \beta_{jk} = \frac{b_{jk}}{\sum_i a_{ik} \hat{x}_{ij}} \tag{2-15}$$

Si se eligen esos valores para las tasas de aprendizaje, se tienen las siguientes reglas de actualización tras operar:

$$a_{ik} \leftarrow a_{ik} \frac{\sum_j b_{jk} x_{ij}}{\sum_j b_{jk} \hat{x}_{ij}}$$
$$b_{jk} \leftarrow b_{jk} \frac{\sum_i a_{ik} x_{ij}}{\sum_i a_{ik} \hat{x}_{ij}} \tag{2-16}$$

El desarrollo ha dado lugar a un paso de carácter multiplicativo que utiliza como la propia variable que se está actualizando como base. Dado que el otro factor de la multiplicación es positivo, pudiendo interpretarse como una ponderación entre los valores originales de la matriz y los aproximados, la no-negatividad entre una iteración del algoritmo y la siguiente queda totalmente garantizada.

En este tipo de algoritmos, una buena práctica es introducir lo que se conoce como término de regularización, una penalización adicional en la función de distancia a las variables para evitar que crezcan hasta el infinito sin control. La función de distancia euclídea regularizada queda de la siguiente manera:

$$D = \frac{1}{2} \sum_{ij} \left(x_{ij} - \sum_k a_{ik} b_{jk} \right)^2 + \frac{\lambda}{2} \sum_{ik} a_{ik}^2 + \frac{\lambda}{2} \sum_{jk} b_{jk}^2 \tag{2-17}$$

El término λ es el parámetro de regularización, ajustable para el buen funcionamiento del algoritmo. Las penalizaciones se han elegido cuadráticas para hacerlas más severas y mantener la consonancia con la norma-2 de la distancia original. Como puede deducirse fácilmente, los desarrollos anteriores cambian levemente al introducir este nuevo término, y las reglas de actualización regularizadas son:

$$a_{ik} \leftarrow a_{ik} \frac{\sum_j b_{jk} x_{ij}}{\sum_j b_{jk} \hat{x}_{ij} + \lambda a_{ik}}$$
$$b_{jk} \leftarrow b_{jk} \frac{\sum_i a_{ik} x_{ij}}{\sum_i a_{ik} \hat{x}_{ij} + \lambda b_{jk}} \quad (2\text{-}18)$$

Hay que destacar que la ejecución de estas dos actualizaciones debe realizarse de manera alternativa, y actualizando los elementos de $\hat{X}$ tras cada una de ellas.

Tras este último paso, podría hablarse de robustez en el algoritmo si no fuese por un detalle: aún no se ha demostrado que estas reglas de actualización garanticen el descenso por el gradiente en cada iteración hasta un mínimo (local en general) de la función de distancia.

2.4 Convergencia del algoritmo

Un algoritmo no podría considerarse válido si no puede demostrarse que siempre termina alcanzando una solución. O, lo que es lo mismo, que en cada iteración del algoritmo el valor de la función de distancia desciende hasta llegar a un mínimo. Para demostrar dicho descenso monótono, se puede utilizar una función auxiliar que mayorice localmente a la función de distancia en cada iteración.

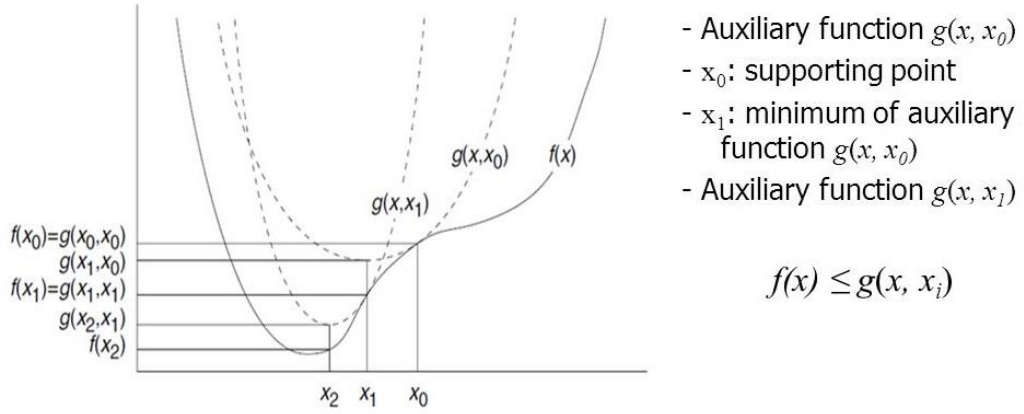

Figura 2-2. Función auxiliar para demostrar el descenso monótono.

En la Figura 2-2 puede verse una representación de dicha función auxiliar $g(x,x_i)$ para el caso de una sola variable. Para llegar al mínimo de la función $f(x)$, puede utilizarse una función cuadrática simple que vaya cambiando en cada paso y la mayorice en todos los puntos salvo uno: el de tangencia (para el que son iguales). Si el algoritmo comienza en el punto x_0 y uno es capaz de encontrar una función auxiliar que cumpla con las condiciones anteriores, minimizando $g(x,x_0)$ puede llegarse al punto x_1, para el que puede hallarse una nueva tangencia en $f(x_1)$ con otra función cuadrática auxiliar $g(x,x_1)$, repitiendo así el proceso hasta alcanzar un mínimo.

Volviendo al caso de NMF, la matriz de aproximación puede verse como una suma de matrices de rango unidad para cada una de las componentes latentes. Tomando (t) como el número de la iteración actual, se tiene:

$$\hat{x}_{ij}^{(t)} = \sum_k a_{ik}^{(t)} b_{jk}^{(t)} = \sum_k \gamma_{ij}^{(k)}(\hat{X}^{(t)})\hat{x}_{ij}^{(t)} \tag{2-19}$$

El parámetro γ vendría a representar el peso que tiene cada uno de los términos $a_{ik}b_{jk}$ del sumatorio para todo k a modo de tanto por uno, como puede deducirse si se recuerda la ecuación (2-4), por lo que se tiene que $\sum_{k=1}^{F} \gamma_{ij}^{(k)}(\hat{X}^{(t)}) = 1$, como es de esperar. La siguiente iteración del algoritmo podría expresarse como:

$$\hat{x}_{ij}^{(t+1)} = \sum_k \gamma_{ij}^{(k)}(\hat{X}^{(t)})\hat{\hat{x}}_{ij}^{(t)} \tag{2-20}$$

Es idéntica a la ecuación anterior salvo por un detalle: $\hat{\hat{x}}_{ij}^{(t)} = \frac{a_{ik}^{(t+1)} b_{jk}^{(t+1)}}{\gamma_{ij}^{(k)}(\hat{X}^{(t)})}$, es decir, $\hat{\hat{x}}_{ij}^{(t)}$ puede entenderse como una predicción de $\hat{x}_{ij}^{(t)}$ obtenida de la característica *k*. En este momento es cuando entra en juego la función auxiliar de la que se hablaba al comienzo de la sección. Debe ser una función que dependa de la iteración en la que se encuentra el algoritmo y de la distancia que haya entre la matrix original y la predicción que acaba de introducirse. Por tanto, podría tenerse la siguiente función auxiliar *G*:

$$G(\hat{X}^{(t+1)}, \hat{X}^{(t)}; X) = \sum_{i,j}\sum_k \gamma_{ij}^{(k)}(\hat{X}^{(t)}) d_E(x_{ij}, \hat{\hat{x}}_{ij}^{(t)}) \tag{2-21}$$

Si se usa la desigualdad de Jensen en el sumatorio agrupado anterior, algo que puede hacerse dada la convexidad fácilmente demostrable de la distancia euclídea para el segundo argumento, se tiene:

$$\sum_{i,j}\sum_k \gamma_{ij}^{(k)}(\hat{X}^{(t)}) d_E(x_{ij}, \hat{\hat{x}}_{ij}^{(t)}) \geq \sum_{i,j} d_E\left(x_{ij}, \sum_k \gamma_{ij}^{(k)}(\hat{X}^{(t)})\hat{\hat{x}}_{ij}^{(t)}\right) = D_E(X, \hat{X}^{(t+1)}) \tag{2-22}$$

Con esto, queda demostrado que la función auxiliar G mayoriza siempre a la distancia en la siguiente iteración del algoritmo, por lo que se puede alcanzar una solución minimizando la función auxiliar en lugar de la original. O, lo que es lo mismo, que el descenso monótono en la función de distancia está garantizado y, dado que la distancia utilizada es una función convexa (norma-2), también está garantizada la convergencia del algoritmo a una solución.

3 SISTEMAS DE RECOMENDACIÓN

Prediction is very difficult, especially about the future.

- Niels Bohr -

En esta sección se explica en qué consisten los sistemas de recomendación, y cómo las técnicas algorítmicas de NMF encajan en la resolución del problema que plantean. Hay que decir que existen otras formas posibles de resolver el problema de la recomendación sin utilizar NMF, pero este documento solo abarca este tipo de técnicas. Se introducen los conceptos fundamentales, así como los problemas que suponen la tarea de la predicción y la recomendación. También se introduce una nueva notación para las matrices de NMF, de forma que el lector comprenda de una forma más directa el papel que juega cada elemento cuando se aplica a los sistemas de recomendación.

Los sistemas de recomendación se pueden aplicar a cualquier entorno en el que existan usuarios que interactúan con un conjunto limitado de objetos en base a sus preferencias, pero durante el desarrollo de este Trabajo Fin de Grado se utiliza como ejemplo recurrente el del cine. Es decir, usuarios que puntúan películas en base a lo buena que haya sido su experiencia al visionarlas, de modo que el sistema intenta recomendar películas que aún no han visto a sus usuarios en base a los gustos expresados sobre las que sí han visto. Se continuará con este ejemplo hasta el final del documento.

3.1 Introducción a los Sistemas de Recomendación

Un sistema de recomendación, o RS (*Recommender System*, en inglés) es un mecanismo informático que intenta predecir la afinidad entre dos conjuntos de datos para los que aún no se tiene toda la información, normalmente un conjunto de usuarios y un conjunto de objetos o *items*. El sistema recaba datos sobre las interacciones de los usuarios con los objetos, los utiliza para predecir futuras interacciones que no se han producido, y finalmente recomienda al usuario una serie de objetos que considera que son del agrado del usuario.

Se distinguen, por tanto, dos tareas fundamentales que un RS debe solventar para su correcto funcionamiento: la tarea de predicción y la tarea de recomendación [8]. La forma de recomendación más extendida es la que se

conoce como filtrado colaborativo (*collaborative filtering* en inglés, o CF), aunque existen muchas otras [6]. Una de ellas podría considerarse la gran alternativa al CF, y es el filtrado basado en contenido (o CBF), que se fundamenta en otorgar a los objetos una serie de atributos compartidos en base a los cuales se realizan las recomendaciones. El concepto de fondo en el CF es que todos los usuarios del sistema contribuyen a la clasificación de los contenidos mediante sus valoraciones.

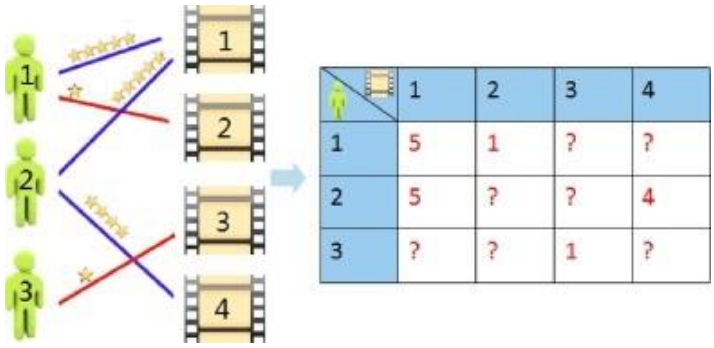

Figura 3-1. Formación de la matriz de valoraciones.

Como puede apreciarse en la figura anterior, se crea una matriz dispersa de valoraciones con tantas filas como usuarios y tantas columnas como objetos (en el ejemplo del cine, películas). La matriz comienza vacía, y los usuarios van rellenando los huecos al emitir valoraciones sobre los objetos. Las valoraciones que son desconocidas son las que están sujetas a predicción. El problema de la predicción es, por tanto, un problema de matriz incompleta.

Tras la tarea de predicción, tiene lugar la tarea de recomendación. Dado un usuario *i*, el RS debe producir una lista de *n* películas con mejor puntuación en el sistema para ese usuario. Esta lista de películas no tiene por qué coincidir obligatoriamente con las *n* películas con la valoración predicha más alta para el usuario en la fase anterior, ya que el RS puede tener en cuenta otros criterios a la hora de generarla. Por ejemplo, puede darse el problema de que a un usuario siempre se le recomienden cosas del mismo tipo y no se le dé la oportunidad de poner a prueba sus gustos en otros aspectos, por lo que probablemente sea buena idea tenerlo en cuenta a la hora de elaborar la lista de recomendaciones.

Uno de los algoritmos que puede usarse para la tarea de predicción es el NMF, con la condición obvia de que las valoraciones que maneje el RS sean mayores que cero para que pueda aplicarse.

3.2 Filtrado Colaborativo

El CF es un método de recomendación que se basa en los comportamientos de los usuarios del sistema a la hora de emitir valoraciones. De manera intuitiva, puede pensarse que, si dos usuarios están en sintonía en sus valoraciones a algunos objetos, pueden estarlo también en otros objetos que uno de ellos haya valorado pero el otro no. Hay dos acercamientos posibles a este planteamiento: uno desde el punto de vista de los usuarios y otro desde el punto de vista de los objetos.

3.2.1 Filtrado colaborativo usuario-usuario

En el filtrado colaborativo usuario-usuario, la premisa es una interpretación bastante directa de los fundamentos del CF. Dado el usuario activo, se buscan usuarios con un patrón de valoraciones similar al suyo, de forma que el sistema se fija en las valoraciones de dichos usuarios sobre objetos que el usuario activo aún no ha valorado para generar las predicciones, y posteriormente las recomendaciones.

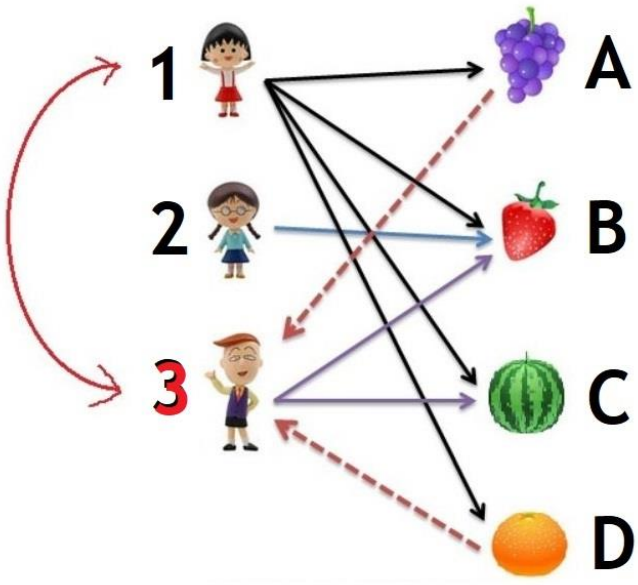

Figura 3-2. Filtrado colaborativo usuario-usuario.

En la Figura 3-2, las flechas rectas en línea continua apuntan a los productos que los usuarios desde los que parten han valorado positivamente. El usuario número 3 es el activo: el que está haciendo uso del RS. Basándose en las valoraciones existentes, el sistema determina qué usuario es más parecido al activo. Dado que el usuario 3 ha valorado los productos B y C, se buscan usuarios que también los hayan valorado. El usuario 2 solo ha valorado B, mientras que el usuario 1 ha valorado ambos. Por lo tanto, se determina que el usuario más parecido al usuario 3 es el número 1. Las flechas en línea discontinua indican productos que el sistema recomienda al usuario activo. Como puede comprobarse, se recomiendan productos que el usuario similar ha valorado positivamente, pero el usuario activo aún no ha valorado.

En un caso práctico, se requiere de una función que califique cómo de parecidos son dos usuarios en su patrón de valoraciones, de forma que se introduce una métrica adicional en el proceso. Teniendo la medida de similitud entre usuarios, puede combinarse con las propias valoraciones para generar predicciones.

3.2.2 Filtrado colaborativo objeto-objeto

Es un método de CF que puede considerarse complementario al anterior, el usuario-usuario. En el filtrado objeto-objeto, el foco de la similitud se sitúa sobre los objetos en lugar de los usuarios. Es decir, el método se basa en establecer similitudes entre objetos. Aunque puede pensarse que este acercamiento es el mismo que el del filtrado basado en contenidos, aquí no se utilizan atributos o metadatos en los objetos para determinar el grado de similitud entre los mismos, sino que se utilizan los patrones de valoración de los usuarios. Si dos objetos tienden a ser valorados de la misma forma por los mismos usuarios, se consideran similares.

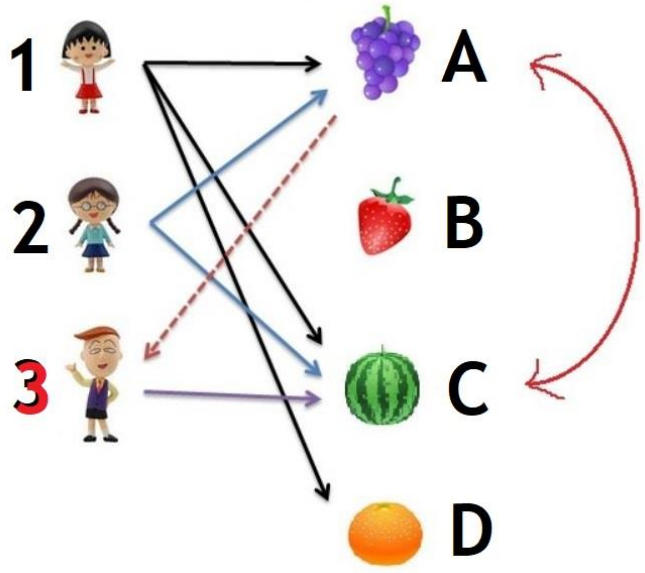

Figura 3-3. Filtrado colaborativo objeto-objeto.

El proceso de recomendación se ilustra en la Figura 3-3. Igual que en el apartado anterior, el usuario activo del RS es el número 3. El usuario 3 solo ha valorado el producto C. Para generar una recomendación por el método objeto-objeto, por lo tanto, es necesario encontrar el producto más similar a C basándose en la forma de valorar que tienen los usuarios. El producto A ha sido valorado positivamente por los usuarios 1 y 2, el producto B no ha sido valorado y el producto D solo ha sido valorado positivamente por el usuario 1. Por lo tanto, se determina que el producto más parecido al C, que ha sido valorado positivamente por 1, 2 y 3, es el producto A. De este modo, la recomendación en flecha discontinua que se genera para el usuario activo es el producto A.

Históricamente, este tipo de filtrado basado en objetos surgió para cubrir la necesidad de los sistemas de recomendación de ser escalables conforme la base de usuarios crece. Normalmente, el ritmo al que crece el número de usuarios es mayor que el de los objetos, de modo que las soluciones basadas en usuarios tienden a escalar mal, y rápidamente empiezan a exigir tiempos de computación altos para generar predicciones [7]. Cuando en un RS el número de usuarios es suficientemente mayor que el número de objetos, si un usuario cambia su valoración sobre un objeto, la similitud objeto-objeto del mismo se verá mínimamente afectada sobre el global. Teniendo esto en cuenta, es razonable que se generen recomendaciones basándose en las valoraciones antiguas, ya que puede suponerse que seguirán siendo lo bastante buenas. Cuando el RS entre en un estado de baja carga computacional, puede aprovecharse el momento para calcular las nuevas similitudes objeto-objeto y actualizar el sistema.

3.3 Problemas típicos de los Sistemas de Recomendación

Los sistemas de recomendación tienen dos grandes obstáculos que deben ser superados si quieren obtenerse buenos resultados durante un funcionamiento normal. Ambos suponen una amenaza a la estabilidad del RS, pero se aportan posibles soluciones que no son extendidas en este documento.

3.3.1.1 Arranque en frío

Cuando un usuario o película nueva entra al sistema, se produce el problema que se conoce como *cold start* [6], o arranque en frío. El sistema de recomendación no cuenta con datos de partida relativos a esa nueva entrada, y se ve en la disyuntiva de tomar una decisión al respecto, ya que si las recomendaciones no son lo suficientemente buenas el usuario podría no ver incentivo al uso del sistema y se marcharía.

En el caso de un usuario nuevo, del que no se cuenta con ninguna valoración sobre películas en el sistema, es necesario tomar una de estas dos vías: hacer una recomendación basada en popularidad (mayor número de visitas de usuarios ya existentes) hasta que el usuario aporte suficientes valoraciones al sistema como para poder llevar a cabo la tarea de recomendación, o pedirle al usuario durante su registro que introduzca dichas valoraciones antes de poder tomar parte en el sistema.

En el caso de una película nueva, lo que se hace normalmente es promocionarla internamente en el sistema (sección de "novedades" que se le presenta al usuario cuando accede, por ejemplo), de modo que los usuarios la vean y aporten valoraciones.

Existen también algoritmos que tienen en cuenta este efecto negativo del arranque en frío [7], pero utilizan modelos probabilísticos o bio-inspirados complejos que quedan fuera del alcance de este trabajo.

3.3.1.2 Problema de la escalabilidad

Otro problema que debe afrontar un sistema de recomendación es el constante crecimiento en número de los datos que se manejan. Tanto usuarios como películas van en aumento.

Ante esta situación, es necesario aplicar técnicas algorítmicas especiales que puedan funcionar objeto a objeto [7], y no sobre el conjunto completo como la que se plantea en este trabajo, de forma que el sistema de recomendación no se haga cada vez más lento y pesado. En la subsección 3.2.2 se habla también sobre esto.

Hay que decir que el algoritmo planteado en este documento no resuelve ninguno de estos problemas fundamentales, y sería necesario introducir otro tipo de técnicas adicionales para garantizar el buen funcionamiento del RS de manera dinámica.

3.4 Adaptación de NMF al problema de la recomendación

Las fórmulas que se vieron en el capítulo 2 siguen siendo válidas, pero son necesarios algunos cambios si se quieren aplicar a una matriz incompleta. Antes de esto, van a introducirse una serie de cambios en la nomenclatura sobre lo anteriormente descrito, ya que el sentido de muchas de las cosas que se llevan a cabo se perdería en caso de no introducirlos.

En el marco del filtrado colaborativo, el NMF no encaja exactamente en ninguno de los dos enfoques distintos planteados, ya que las dos matrices de factorización están asociadas al conjunto de los usuarios y al de los productos simultáneamente.

3.4.1 Cambios en la nomenclatura

Los cambios que se van a introducir en este punto en la nomenclatura de las matrices están destinados a facilitar la comprensión al lector sobre el nuevo significado que adquieren las matrices descritas en el capítulo 2 de este documento.

La matriz *X*, la original, pasa a ser la matriz *R*. Lo mismo sucede con la matriz aproximada. Elegir la letra *R* viene del inglés *rating*, o valoración. Esta matriz contiene todas las valoraciones con las que cuenta el RS, organizadas por filas de usuarios y columnas de películas. Por lo tanto, *I* pasa a ser el número de usuarios del RS, y *J* el número de productos. Una diferencia fundamental es que la matriz *X* se suponía completa, mientras que en la matriz *R* habrá elementos desconocidos, es decir, si no se ha producido valoración de un usuario determinado *i* a un producto *j*, el elemento r_{ij} estará vacío.

La matriz de factorización *A* pasa a ser la matriz *U*. Esta va a ser la matriz de factorización para los usuarios. Desde el punto de vista del enfoque usuario-usuario del CF, dos usuarios serán similares si sus filas correspondientes en *U* contienen valores parecidos.

La matriz de factorización *B* pasa a ser la matriz *P*. Esta matriz estará asociada a los productos. Utilizando ahora el enfoque objeto-objeto, dos productos pueden considerarse similares si sus filas correspondientes en la matriz *P* son parecidas.

Introducidos estos cambios, las ecuaciones (2-2) y (2-5), por ejemplo, quedarían de la siguiente manera.

$$R \approx \hat{R} = UP^T \tag{3-1}$$

$$r_{ij} \approx \hat{r}_{ij} = \sum_{k=1}^{K} u_{ik} p_{jk} \tag{3-2}$$

Las componentes latentes recorridas por el índice *k* pueden entenderse ahora como diferentes características de la relación existente entre usuarios y productos. Por ejemplo, en una interpretación libre aplicada al caso de la recomendación de películas, las características podrían entenderse como distintos géneros de cine, como comedia, terror, romanticismo, acción, etc. Si la matriz *U* contiene un valor alto en la columna *k*-ésima para un usuario determinado, quiere decir que existe una alta afinidad de ese usuario con la característica *k*. Lo mismo podría decirse de un producto cuya fila de la matriz *P* posea un valor alto en la misma columna: se le está otorgando a ese producto una alta afinidad con esa característica. La Figura 3-4 ilustra un ejemplo de factorización sencillo con tan solo 4 usuarios, 4 películas y 2 componentes de factorización.

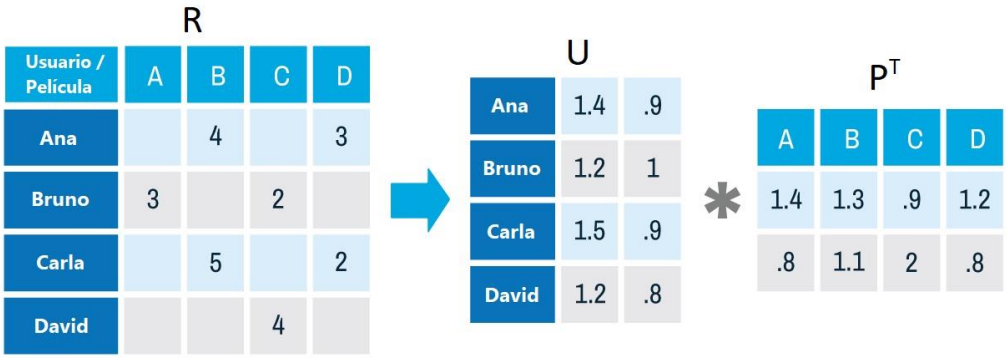

Figura 3-4. Ejemplo de factorización por NMF.

Observando la figura, por ejemplo, puede verse que la película C es la que tiene mayor afinidad con la segunda característica, con un valor de 2, o que la usuaria Carla es la más afín a la primera característica, con 1.5.

3.4.2 Cambios en la formulación

Los cambios en la formulación son todos relativos al hecho de que la matriz ahora es incompleta. O lo que es lo mismo, existen valores r_{ij} desconocidos. Es necesario, por tanto, hacer referencia al conjunto de usuarios que han valorado un producto j, para lo que se utilizará U_j. Para el conjunto de productos que han sido valorados por un usuario i se usará P_i.

Las reglas de actualización de (2-18) quedarían de la siguiente forma tras estas consideraciones.

$$u_{ik} \leftarrow u_{ik} \frac{\sum_{j \in P_i} r_{ij} p_{jk}}{\sum_{j \in P_i} \hat{r}_{ij} p_{jk} + \lambda u_{ik}}$$
$$p_{jk} \leftarrow p_{jk} \frac{\sum_{i \in U_j} r_{ij} u_{ik}}{\sum_{i \in U_j} \hat{r}_{ij} u_{ik} + \lambda p_{jk}} \quad (3\text{-}3)$$

No deben introducirse en los sumatorios términos que conlleven una valoración r_{ij} que no exista, de ahí el uso de los conjuntos anteriormente definidos para acotar el alcance de dichos sumatorios a las valoraciones existentes.

4 SIMULACIÓN DEL ALGORITMO

An algorithm must be seen to be believed.

- Donald Knuth -

En este capítulo se expondrá el proceso que se ha seguido para simular el algoritmo en un ordenador y analizar sus resultados. Primero se darán detalles del equipo informático y el software que se ha usado para la simulación, así como de la base de datos de valoraciones elegida. Seguidamente se expondrá la batería de pruebas realizadas, adjuntando datos obtenidos y gráficas generadas para una mejor comprensión de los resultados. El último apartado del capítulo corresponde al análisis de dichos resultados.

4.1 Entorno de simulación y base de datos

Para simular el algoritmo, se ha utilizado el software MATLAB® en su versión 2012b, corriendo en el S.O. Windows® 10 de 64 bits. El equipo en el que se ha instalado es un portátil con un procesador Intel® Core™ i7-6700HQ @ 2.60 Ghz con 16 GB de memoria RAM.

En cuanto al set de datos utilizado durante las pruebas, se ha extraído uno procedente de la base de datos de valoraciones de películas de MovieLens [5], referido como "*100k*", libre de uso para fines académicos, y que contiene 100000 valoraciones de usuarios, en un rango del 1 al 5, para 943 usuarios anónimos únicos sobre 1682 películas distintas. El formato de los archivos es de valores separados por tabulación, algo a tener en cuenta durante el proceso de extracción de las valoraciones. Una particularidad del set de datos escogido es que contiene cinco versiones particionadas del set completo, todas mutuamente excluyentes, de forma que pueden usarse directamente para tareas de validación cruzada del algoritmo.

4.2 Variables utilizadas

Las pruebas consistieron en la ejecución de varios scripts en MATLAB® que implementan el algoritmo de NMF elegido variando una serie de parámetros. Al finalizar, muestran una serie de resultados por pantalla junto con algunas gráficas relevantes. Los scripts, a su vez, llaman a otras funciones también codificadas para la ocasión, ya que su uso es repetido. En los siguientes subapartados se detallan los pasos seguidos, ilustrando

con muestras del código fuente.

Antes de comenzar con la explicación, es pertinente aportar una tabla con todas las variables recurrentes implicadas en los scripts, de modo que pueda comprenderse mejor su función cuando aparezcan más adelante. A continuación, se listan en orden de aparición en el código.

Tabla 4–1 Variables usadas en la simulación

Nombre	Descripción breve
R	Matriz de valoraciones
nu	Número de usuarios
np	Número de productos
r	Matriz de existencia de valoración
r_max	Valoración máxima
r_min	Valoración mínima (>0)
r_step	Paso entre valoraciones
ratings	Vector de posibles valoraciones
rmsscore_best	Mejor valor RMS obtenido
R_best	Mejor matriz aproximada
R_tr	Matriz de entrenamiento
R_cv	Matriz de valoración cruzada
r_tr	Existencia de valoración (entren.)
r_cv	Existencia de valoración (val. cr.)
U	Matriz de usuarios
P	Matriz de productos
R_approx	Matriz aproximada
n_its	Número máximo de iteraciones
tol	Tolerancia de paso
epsilon	Constante para no dividir por 0
lambda	Parámetro de regularización
J	Vector de costes sucesivos
var_n	Tamaño del vector de varianzas
rmsscore	Puntuación en valor RMS
R_predict	Matriz de predicción
accuracy	Acierto (%)
accuracy_relaxed	Acierto + adyacencias (%)

4.3 Extracción de datos

Lo primero que se hizo fue extraer los datos del archivo de MovieLens. El archivo contiene en realidad varios sets de datos, pero solo nos fijamos en algunos de ellos.

Todos los sets de datos que se usaron están escritos en valores separados por tabulación, con la primera columna conteniendo el ID del usuario, la segunda el ID de la película, la tercera la valoración de ese usuario para esa película y la cuarta lo que se conoce como *timestamp*, un valor numérico relacionado con el momento en el que se produjo la valoración. En el caso que nos ocupa se desechó este dato temporal, ya que no resulta de utilidad para el algoritmo NMF que quiere probar.

Además del set de datos que contiene la totalidad de las valoraciones, también se han usado unos sets de datos parciales que dividen el total en cinco subconjuntos mutuamente excluyentes, de modo que puedan utilizarse durante una misma simulación hasta cinco sets de validación cruzada para afianzar mejor los resultados.

La función que extrae el conjunto de datos completo del archivo es la siguiente:

```
1   function R = readDataset(file_name)
2   % Esta funcion extrae el conjunto de datos del archivo que se indique.
3   D = dlmread(file_name,'\t'); % Lee la matriz de datos del archivo
4                                % asumiendo que los valores estan
5                                % separados por tabulaciones.
6   users = D(:,1); % La primera columna es el ID de usuario.
7   user_n = max(users); % Usuario con ID más alto.
8   prods = D(:,2); % La segunda columna es el ID de producto.
9   prod_n = max(prods); % Producto con ID más alto.
10  ratings = D(:,3); % La tercera columna es la valoracion.
11
12  % Declaramos la matriz en forma "sparse" por comodidad.
13  R = sparse(users,prods,ratings,user_n,prod_n);
14  % Pasamos la matriz a full eliminando posibles filas y columnas nulas.
15  R = full(R(any(R,2),any(R,1)));
16  end
```

Figura 4-1. Función de MATLAB® para la extracción del conjunto de datos.

Se ha utilizado la declaración *sparse* para la matriz inicialmente debido a la sencillez de la misma a la hora de generar la matriz de valoraciones a partir de los IDs de usuario/película en una sola sentencia de código.

4.4 Emparejamientos de entrenamiento y validación cruzada

El siguiente paso es montar un bucle iterativo dentro del cual se pueda repetir todo el procedimiento algorítmico variando el subconjunto de entrenamiento y el de validación cruzada. La estructura iterativa *for* es la más indicada, ya que puede usarse el índice para apuntar a la pareja de ficheros correspondiente.

La función de lectura de los subconjuntos parciales es muy parecida a la ilustrada en el apartado anterior, solo cambia la necesidad de pasar como argumento el número que corresponda a cada iteración para poder extraer los datos correctos de cada pareja de archivos. Hay que recordar que los subconjuntos han sido preparados previamente por el suministrador de los datos de forma que sean mutuamente excluyentes, por lo que no es necesario ningún procesamiento adicional en el código para garantizar la integridad de los mismos.

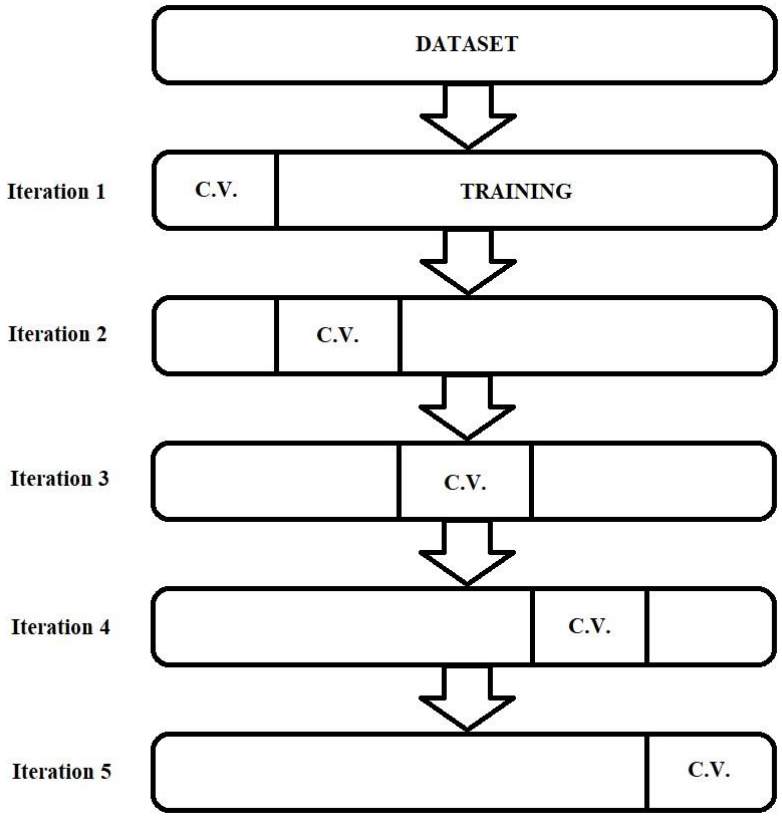

Figura 4-2. Ilustración de subconjuntos de entrenamiento y validación cruzada.

4.5 Algoritmo NMF multiplicativo

Una vez se tienen los datos de los subconjuntos para una iteración, es momento de inicializar las variables pertinentes y comenzar con la ejecución del algoritmo de NMF. Se tomaron las siguientes decisiones:

- La dimensión intermedia *K* se fijó en 25. Como se vio en la sección 1 de este documento, para poder hablar de simpleza en las componentes latentes que conforman los datos, se debe cumplir la ecuación (2-3). En este caso, $K < 943 \cdot 1682/(943 + 1682) \cong 604$, por lo que el valor escogido es suficiente como para mantener sobradamente la sencillez en la simulación, a costa de exactitud en la aproximación. Sumando los elementos que hay en *U* y *P* se tienen 62625, que es también menor que 100000 (el tamaño del conjunto de datos de partida).
- Se asignan valores aleatorios entre 0 y 1 a las matrices *U* y *P* (nunca 0). Son buenos valores para comenzar, ya que el resultado de multiplicar $\boldsymbol{u}_i \boldsymbol{p}_j^T$ para cualquier pareja *ij* es como mucho 5, la valoración máxima posible en el set de datos que se está utilizando.
- El parámetro de regulación *lambda* se fija en 0.1. Es un valor intermedio, ni demasiado grande como para penalizar en exceso el crecimiento de las variables en la función de distancia, ni demasiado pequeño como para ser totalmente irrelevante.
- Se introduce el concepto de la matriz de existencia de valoraciones, *r*, cuyas dimensiones son las

mismas que las de *R*. Toma el valor 1 si existe valoración en esa posición de la matriz, o 0 en caso contrario. Multiplicando por *r* también pueden simularse los conjuntos U_j y P_i.

- El criterio de convergencia se basa en la varianza de los últimos 10 valores de la función de distancia. Si llega un momento en el que es menor que una tolerancia fijada de 0.1, se considera que el algoritmo ha alcanzado un mínimo. Si *t* es la iteración actual, el criterio de convergencia puede escribirse con la siguiente expresión:

$$\sigma(J_{t-9}, J_{t-8}, \dots, J_{t-1}, J_t) \leq 0.1 \tag{4-1}$$

- Como máximo se realizan 5000 iteraciones del algoritmo multiplicativo, aunque aún no se haya cumplido el criterio de convergencia elegido. El script informa si sale del bucle por alcanzar la convergencia o por llegar al máximo de iteraciones permitido.

Terminada la inicialización, se llega al núcleo del código: el algoritmo multiplicativo NMF que va convergiendo a una aproximación de la matriz original llenando de predicciones los huecos que antes estaban vacíos. Es una implementación de las ecuaciones de (2-18), con una salvedad. Las primeras *var_n* veces que se actualizan las matrices *U* y *P* no se atiende a criterios de convergencia, para tener ese número de valores almacenados en el vector de distancias sucesivas *J*.

```
50      %Una vez hay suficientes valores de J como para calcular la varianza
51      %entre ellos, se atiende a criterios de convergencia.
52 -    while (var(J(j-var_n+1:j)) > tol) && (n_its > j)
53 -        U = U.*(R_tr*P./((R_approx.*r_tr)*P+lambda*U+epsilon));
54 -        R_approx = U*P';
55 -        P = P.*(R_tr'*U./((R_approx.*r_tr)'*U+lambda*P+epsilon));
56 -        R_approx = U*P';
57
58 -        j = j+1;
59 -        J(j) = computeCost(R_tr,r_tr,U,P,lambda);
60 -    end
```

Figura 4-3. Paso multiplicativo del algoritmo NMF en MATLAB®.

En cuanto a la función que calcula la distancia (o coste) para cada nueva iteración, es la siguiente:

```
1     function J = computeCost(R, r, U, P,lambda)
2     % Distancia euclidea:
3 -   J = 0.5*(sum(sum(((((U*P').*r)-R).^2))))...
4             + lambda/2*(sum(sum(U.^2))+sum(sum(P.^2)));
5 -   end
```

Figura 4-4. Función de distancia euclídea en MATLAB®.

No es más que una implementación directa de la función de distancia euclídea regularizada de (2-17).

4.6 Métrica de evaluación

Para ver la calidad de la matriz aproximada en cada iteración, se calcula el RMS del error entre la matriz original y la aproximada, punto por punto, para el subconjunto de validación cruzada correspondiente a la iteración en la que se esté. Si se obtiene un RMS más bajo, el script considera la nueva aproximación como mejor y guarda la matriz.

```
73 -        R_approx(R_approx<r_min) = r_min;
74 -        R_approx(R_approx>r_max) = r_max;
75
76          %Metrica de evaluacion, nos quedamos con la mejor aproximacion.
77 -        rmsscore = rms(R(r_cv)-R_approx(r_cv));
78 -        if rmsscore < rmsscore_best
79 -            rmsscore_best = rmsscore;
80 -            R_best = R_approx;
81 -            r_cv_best = r_cv;
82 -        end
```

Figura 4-5. Evaluación mediante RMS en MATLAB®.

Como puede observarse en este trozo de código, antes de calcular el valor RMS del error se elimina la posibilidad de que el algoritmo haya sobrepasado los límites admitidos para las valoraciones, reduciéndose al extremo inferior o superior según sea el caso.

Si se recuerda que el número de usuarios es *I* y el número de películas es *J*, el valor RMS del error es:

$$RMS(R - \hat{R}) = \sqrt{\frac{\sum_{ij}(r_{ij} - \hat{r}_{ij})^2}{IJ}} \tag{4-2}$$

Hay que indicar que el RMS del error no es ni mucho menos el único modo de medir la calidad de la aproximación, existen multitud de métricas que podrían haberse usado en su lugar, como el valor-F1 [6] o el valor-G. Se ha elegido el RMS porque es representativo de lo que una métrica de evaluación pretende conseguir sin recurrir a conceptos complejos, de forma que puede mantenerse el carácter didáctico de este documento.

4.7 Ejecución de algoritmo

Tras explicar en los apartados anteriores los entresijos de la implementación algorítmica, ahora es momento de ponerla en marcha. Utilizando todas las condiciones descritas en el apartado 4.5, se ejecuta el script de MATLAB®.

Para visualizar cómo la distancia entre la aproximación y la matriz original va disminuyendo con cada iteración del algoritmo, se adjunta la Figura 4-6. El descenso es muy pronunciado al principio, y conforme se va avanzando se desciende de manera cada vez más lenta. Finalmente, cuando aún no se han alcanzado las 3000 iteraciones, el criterio de convergencia se cumple al apenas variar los últimos valores de la función de distancia, y el algoritmo se para.

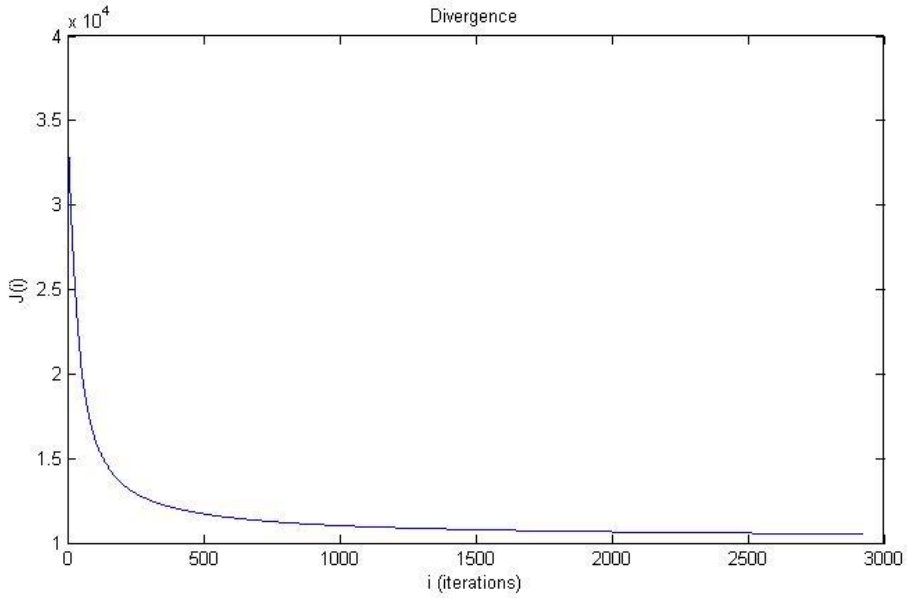

Figura 4-6. Gráfica de la distancia frente al número de iteraciones.

Se adjunta también la salida textual, que informa sobre si se alcanzó la convergencia o no para cada emparejamiento entre subconjunto de entrenamiento y de validación cruzada, además de dar datos sobre el tiempo de ejecución y la calidad de la mejor predicción alcanzada. Más concretamente, se muestra por pantalla el porcentaje de aciertos sobre el set de validación cruzada, el porcentaje de aciertos "permisivo" (admite valores adyacentes al verdadero como válidos) y el valor RMS del error.

```
Command Window
  TR-CV dataset 1: reached convergence
  TR-CV dataset 2: reached convergence
  TR-CV dataset 3: reached convergence
  TR-CV dataset 4: reached convergence
  TR-CV dataset 5: reached convergence
  Accuracy of the best prediction over the CV set: 36.69 %
  Relaxed accuracy of the best prediction over the CV set: 82.43 %
  RMS error of the best prediction over the CV set: 1.147
  Elapsed time is 682.877606 seconds.
fx >>
```

Figura 4-7. Salida textual del algoritmo en MATLAB®.

4.8 Comportamiento frente a variaciones en los parámetros

Además de la ejecución previa con objeto de observar el buen funcionamiento del algoritmo, es pertinente realizar otras pruebas en las que se compruebe cómo se comporta el algoritmo frente a variaciones en algunos de sus parámetros. En concreto, se realizaron baterías de pruebas variando K (número de componentes), λ (parámetro de regularización) y el porcentaje de la matriz original que contiene valoraciones.

4.8.1 **Variación en el número de componentes *K***

Para la prueba de variación en el número de componentes latentes *K*, se eligieron los valores 20, 40, 60, 80 y 100. Se recuerda que la prueba principal se realizó con un valor de 25. Las gráficas generadas fueron estas:

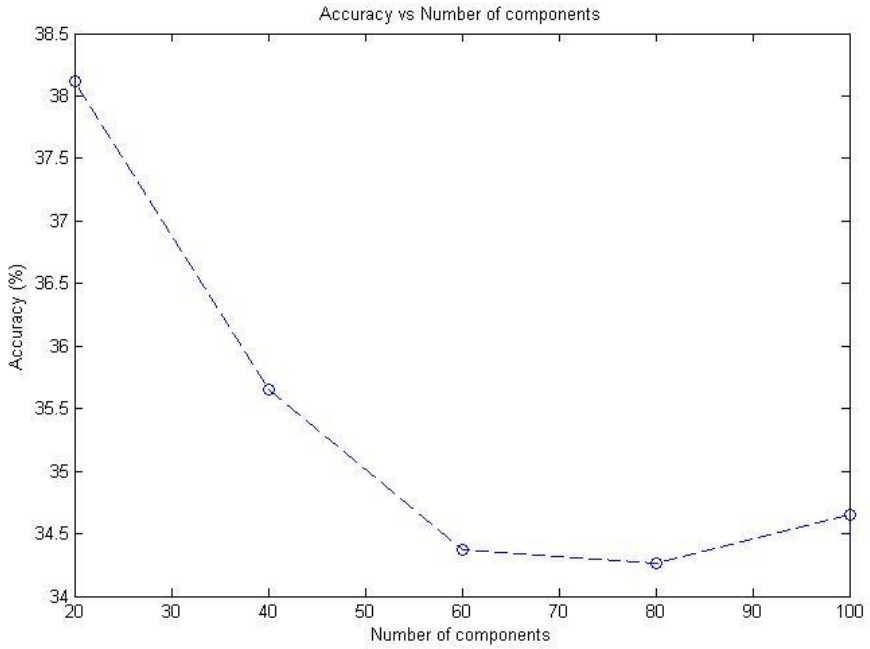

Figura 4-8. Gráfica del porcentaje de aciertos frente a *K*.

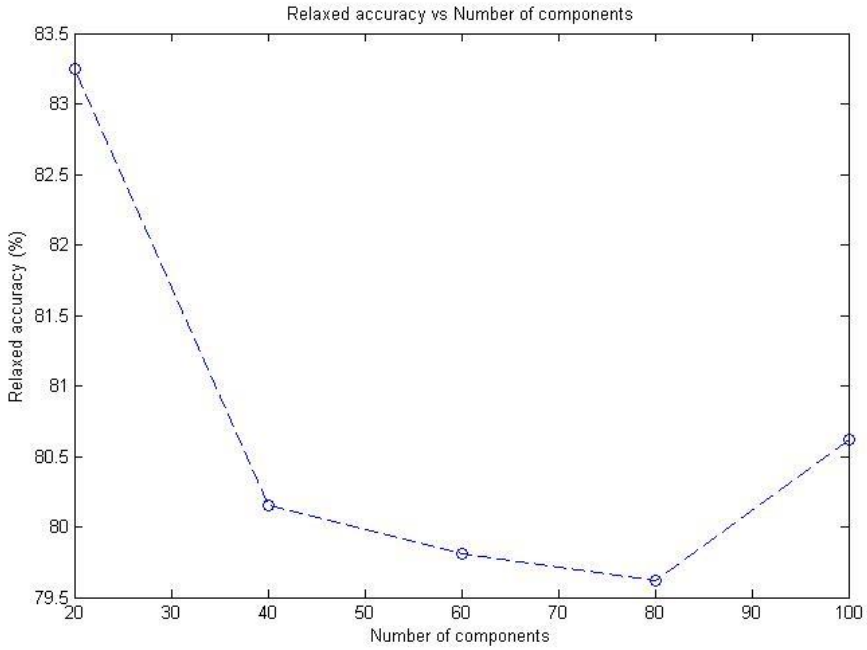

Figura 4-9. Gráfica del porcentaje de aciertos y adyacencias frente a *K*.

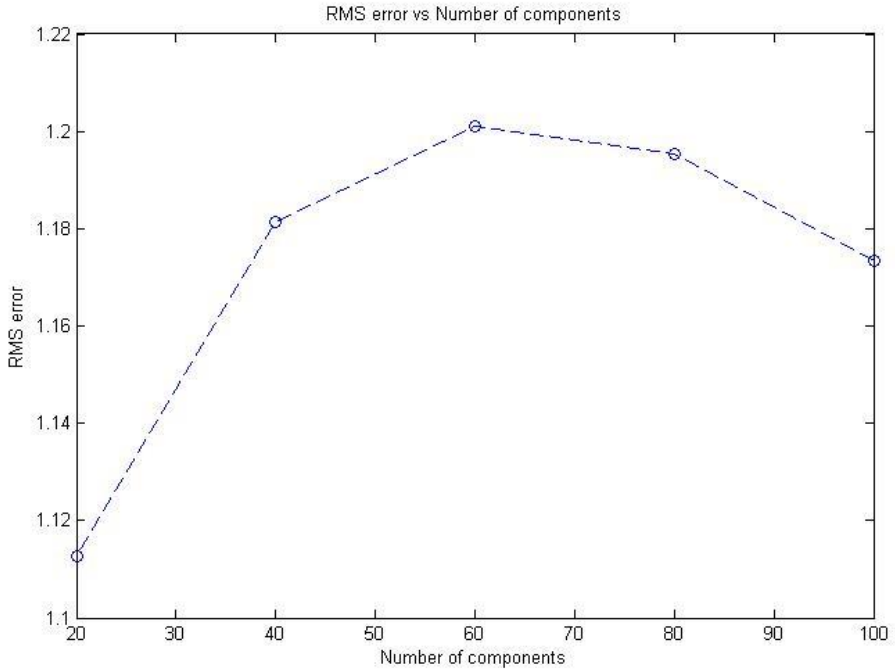

Figura 4-10. Gráfica del valor RMS del error frente a *K*.

Fruto de la observación de las gráficas, se puede decir que cuando el número de componentes se mantiene en unos márgenes bajos el sistema se ve beneficiado, ya que tanto los porcentajes de acierto como el valor RMS del error para K=20 son los mejores de la prueba. Los peores resultados se obtienen con valores de K intermedios, y en el último valor se aprecia algo de mejoría.

Este comportamiento puede deberse al hecho de que si al algoritmo se le da demasiada flexibilidad (K alta) aprende el ruido existente en los datos, y no solo las componentes latentes, por lo que el error en las predicciones aumenta.

4.8.2 Variación en el parámetro de regularización λ

La segunda variación paramétrica programada es en el parámetro de regularización, λ. La presencia de regularización en el algoritmo, en principio, es beneficiosa, ya que ayuda a que las variables no escapen al infinito durante la ejecución, y evita que la aproximación se ajuste demasiado a los datos de entrenamiento, dando margen para que las predicciones se adapten mejor a posibles dinámicas internas.

Para esta prueba, se le dio a λ los valores 0.01, 0.1, 1, 10 y 100. Las gráficas que se generaron fueron las siguientes. Es necesario mencionar que en las gráficas el eje X se corresponde con en logaritmo en base 10 de λ y no con λ en sí.

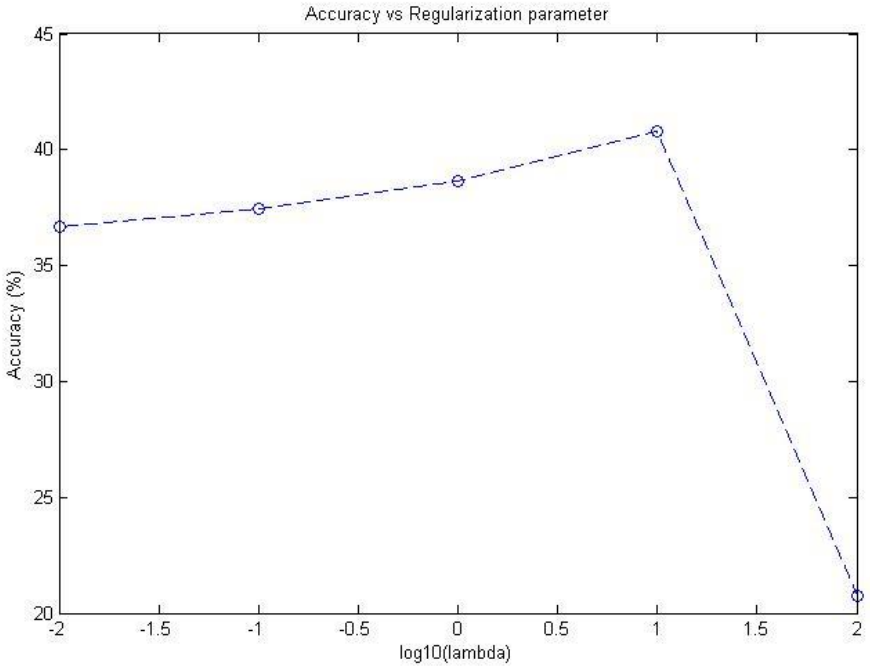

Figura 4-11. Gráfica del porcentaje de aciertos frente a λ.

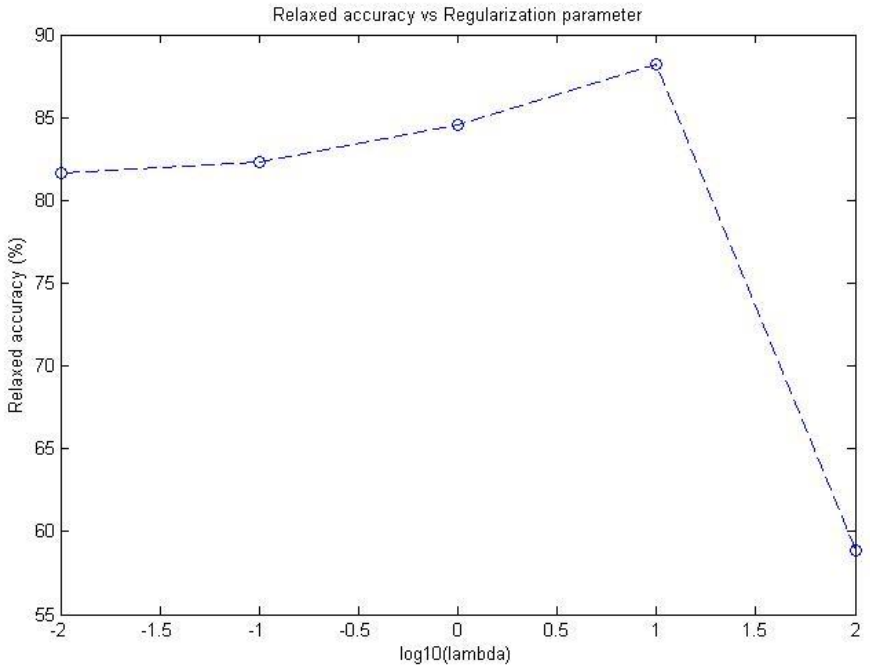

Figura 4-12. Gráfica del porcentaje de aciertos y adyacencias frente a λ.

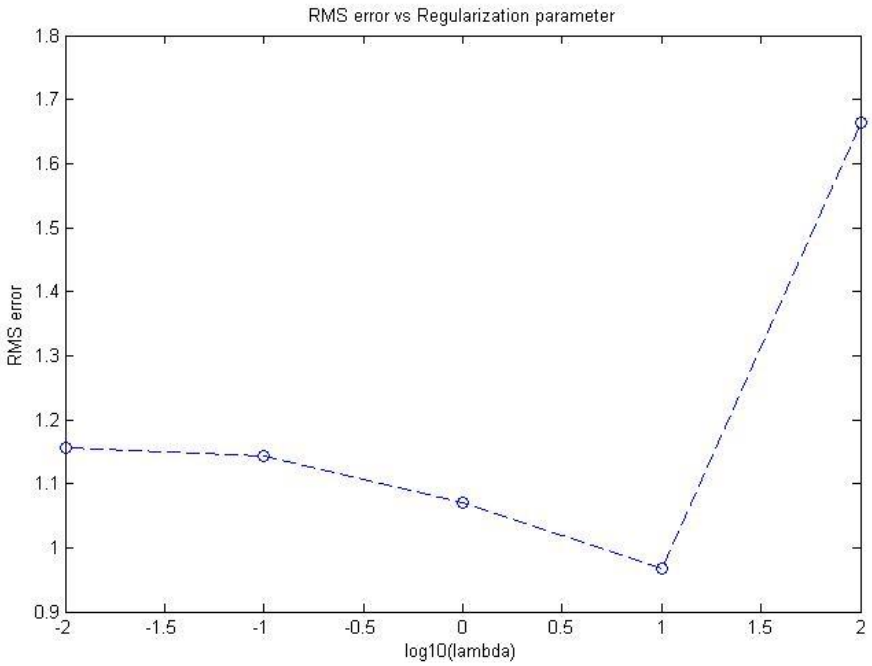

Figura 4-13. Gráfica del valor RMS del error frente a λ.

Como puede observarse en las gráficas, el algoritmo mejora con el aumento del parámetro de regularización, pero solo hasta cierto punto. Una vez se llega a un valor de λ de 100, el algoritmo deja de funcionar bien y el error se dispara. El valor óptimo para la regularización puede que se encuentre por encima de 10, donde se alcanza el óptimo en las gráficas, pero no muy por encima como ya se ha evidenciado.

4.8.3 Variación en porcentaje de matriz completa

La última prueba de variación en los parámetros corresponde al porcentaje de la matriz que contiene valoraciones. Para esta prueba fueron necesarias algunas modificaciones en el código original del algoritmo, ya que había que reordenar las filas y columnas de la matriz en orden de densidad, para luego hacer que el algoritmo se quede con una subdivisión de la matriz en la que pueda afirmarse que el porcentaje relleno ha aumentado o disminuido frente a la iteración anterior. A continuación, se muestra una gráfica indicativa de la densidad de la matriz original del set de datos.

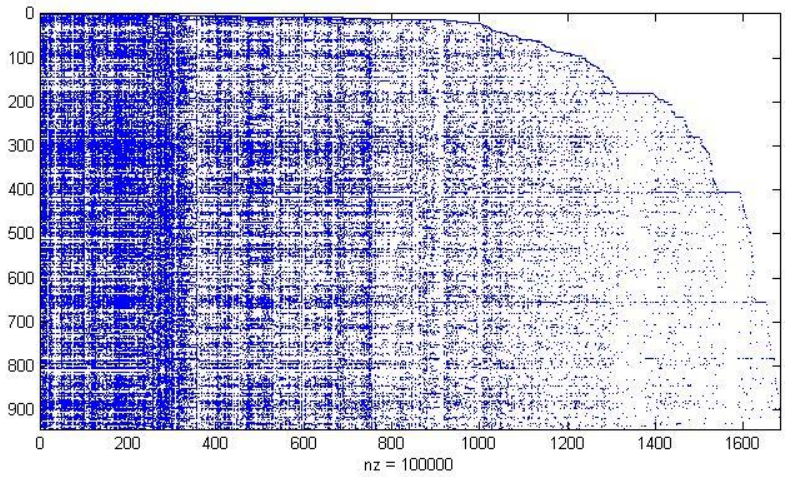

Figura 4-14. Gráfica de la densidad de la matriz de valoraciones.

Tras realizar las permutaciones en filas y columnas en base a buscar el aumento de densidad de la matriz al aumentar el número de la fila y columna, se tiene la misma matriz distribuida de esta otra manera:

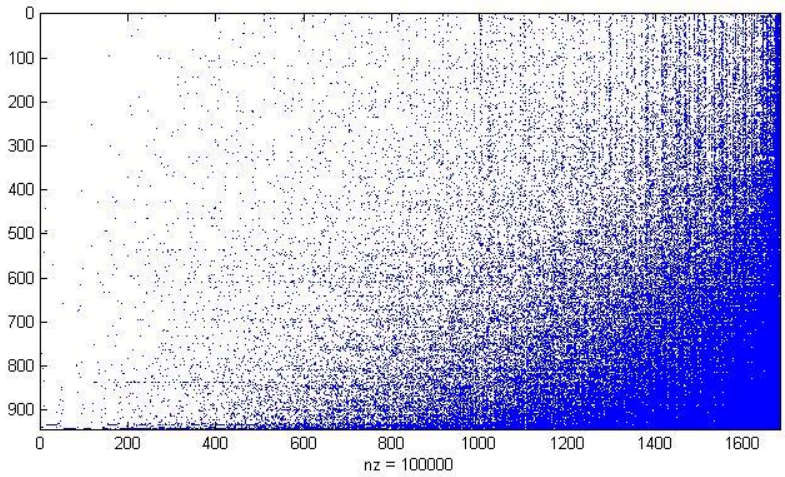

Figura 4-15. Gráfica de la densidad de la matriz de valoraciones tras las permutaciones.

A partir de esa matriz permutada, el algoritmo toma las últimas 100 filas y 200 columnas y ejecuta el algoritmo. En la siguiente iteración, añade las siguientes 100 filas y 200 columnas a las que ya cogió anteriormente, de forma que la densidad disminuye. Cuando se ejecuta el algoritmo teniendo 500 filas y 1000 columnas, se detiene y muestra los resultados.

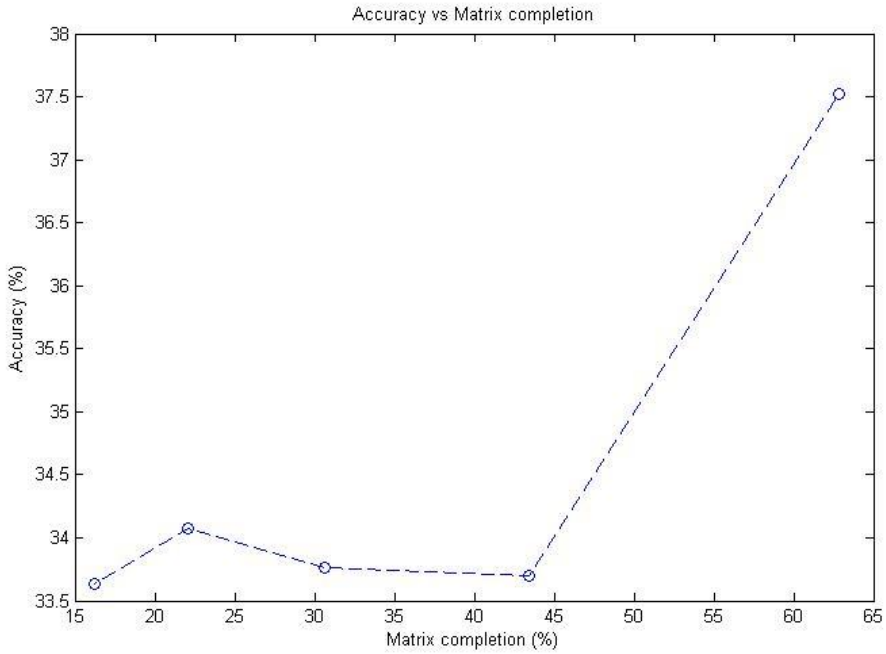

Figura 4-16. Gráfica del porcentaje de aciertos frente al porcentaje de la matriz completo.

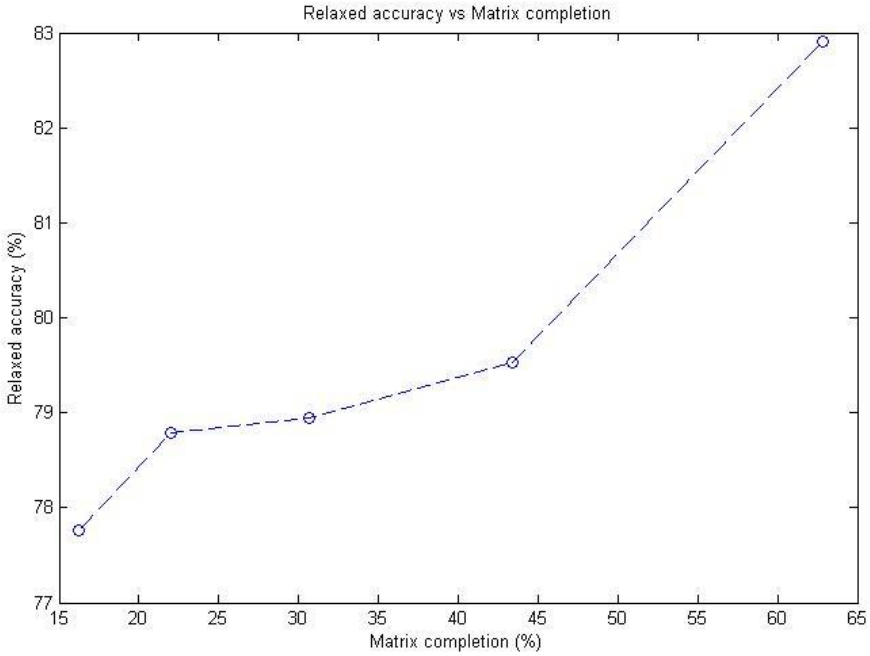

Figura 4-17. Gráfica del porcentaje de aciertos y adyacencias frente al porcentaje de la matriz completo.

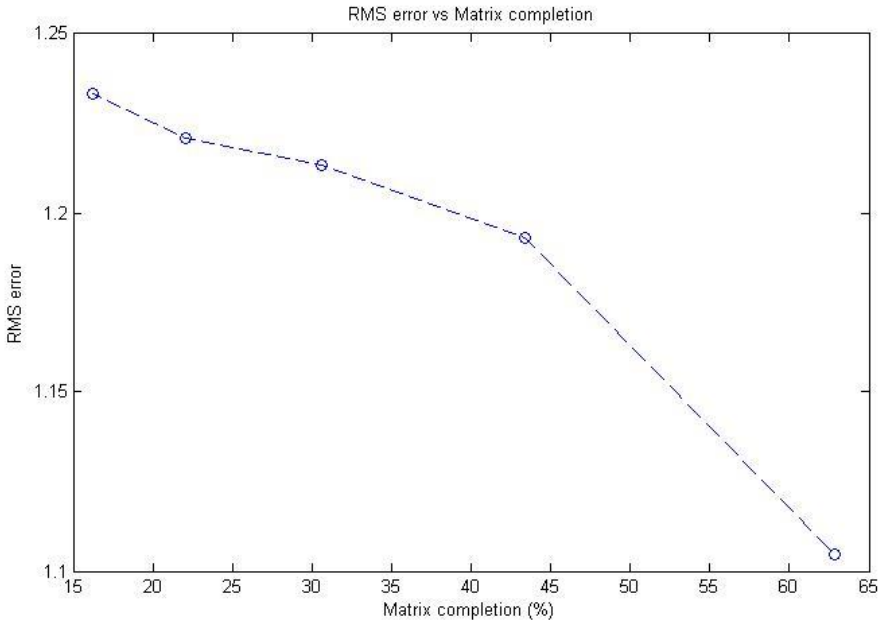

Figura 4-18. Gráfica del valor RMS del error frente al porcentaje de la matriz completo.

Se observa claramente que cuanto mayor es el porcentaje de la matriz que contiene valoraciones, mejor se comporta el algoritmo. El error en la predicción está directamente relacionado con la falta de datos de partida, algo que parece lógico e intuitivo.

4.9 Resumen de resultados

Tras la ejecución de los distintos scripts y el estudio de los resultados, se llega a la conclusión de que la precisión exacta del algoritmo está en torno al 37%, y la precisión relajada a valores adyacentes en torno al 82%. Son resultados que pueden considerarse satisfactorios dado lo básico de la versión del algoritmo de NMF que se ha implementado en MATLAB® para el desarrollo de este trabajo. En trabajos académicos de alto nivel como [6] la comunidad se afana en encontrar métricas y pasos multiplicativos más robustos y, aún así, los resultados mejoran lo que ya existe solo marginalmente.

Un valor RMS del error entre los valores originales y los aproximados de poco más de 1 es también un buen resultado. Hay que recordar que, en el conjunto de datos utilizado, las valoraciones posibles son 1, 2, 3, 4 y 5. Un RMS en torno a 1 indica que las aproximaciones se desvían típicamente a uno de los posibles valores adyacentes, y muy pocas se desvían más allá, algo que las gráficas corroboran.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

En este trabajo se han introducido conceptos básicos de factorización de matrices, aprendizaje supervisado, algoritmos iterativos y sistemas de recomendación. Se ha mantenido un carácter didáctico, divulgativo, de forma que pueda servir como introducción a estas materias a aquellas personas con un poco de base formativa en matemáticas que estén interesadas en estos temas.

Se ha hablado en detalle de los entresijos de la Factorización No-negativa de Matrices, exponiendo alternativas a la medida de similitud, explicando todo el proceso de obtención de las reglas de actualización del algoritmo a partir del descenso por el gradiente, y demostrando la fiabilidad de las mismas a la hora de converger a una solución válida.

En cuanto a los Sistemas de Recomendación, se han introducido distintos acercamientos al problema, entrando en detalle en una familia de los mismos, la del Filtrado Colaborativo. Se han expuesto ejemplos que ilustran de manera sencilla los conceptos de fondo que se están manejando, junto a los problemas que acarrean este tipo de sistemas, que deben ser atajados por la implementación algorítmica si quiere mantenerse un funcionamiento exitoso del sistema de recomendación mantenido en el tiempo.

Se adapta la nomenclatura y la formulación de las técnicas de NMF a otras con más sentido a la hora de aplicarlas a los problemas de predicción de valoraciones y recomendación a usuarios, de forma que el lector entienda rápidamente lo directo de la aplicación y lo natural que resulta.

Asimismo, se han realizado varias simulaciones de un algoritmo de NMF básico para ponerlo a prueba en distintos escenarios, variando una serie de parámetros que tienen influencia sobre el resultado final. Se ha explicado todo el proceso seguido a nivel de código para implementar satisfactoriamente el algoritmo NMF, desde la extracción de la matriz de valoraciones del set de datos escogido hasta la salida por pantalla de los resultados relevantes, pasando por el bucle iterativo donde se actualizan las variables.

Los resultados obtenidos tras dichas simulaciones han sido positivos, y dan margen a la mejora de los mismos si se introducen conceptos más avanzados que no están previstos en el alcance de este documento.

En general, puede decirse que las técnicas de NMF tienen una aplicación clara y directa en los sistemas de recomendación, si bien puede que por sí mismas no puedan conseguir resultados tan buenos como los obtenidos por combinaciones de otras técnicas basadas en conceptos estadísticos o en aprendizaje supervisado mediante etiquetado de los datos de partida.

5.2 Líneas futuras de investigación

Hay varias líneas de investigación en las que este trabajo puede ampliarse en el futuro. Las técnicas de NMF encuentran cada vez más aplicaciones, y ya llevan algún tiempo siendo populares en su aplicación a sistemas de recomendación, sobre todo en combinación con otras técnicas algorítmicas, como se puso en evidencia en el Netflix Grand Prize [1]. Los sistemas de recomendación están presentes no solo en el ámbito del cine y las series, sino también en plataformas musicales, comercio electrónico, y otras muchas aplicaciones. Las técnicas de NMF se postulan como grandes candidatas a aportar soluciones en los sistemas de recomendación.

La inicialización de las dos matrices que conforman la factorización sigue siendo un problema difícil de atacar, ya que no es trivial encontrar unos valores que garanticen una convergencia segura a un mínimo local que no quede demasiado lejos del global, que sería el ideal. Se recuerda que en este trabajo se optó por inicializar de manera aleatoria, lo cual, aunque válido, no es lo suficientemente fiable. Existen modelos probabilísticos que intentan solucionar esta problemática con mayor o menor éxito, y que podrían probarse a nivel de simulación para comprobar hasta qué punto son mejores que una simple inicialización aleatoria.

Otro problema que requiere solución es la selección ciega de un número de componentes latentes que garantice una reducción de la dimensionalidad y unos buenos resultados en la predicción. Actualmente, la forma de elegir el número de componentes es una batería de pruebas que analice el comportamiento del algoritmo frente a distinta K. Sería conveniente encontrar un método para elegir este número sin pasar por las pruebas, ya que es una solución que no escala conforme las bases de datos se van haciendo más y más grandes.

En cuanto a la distancia o divergencia que sirve como base de todo el algoritmo, la búsqueda de funciones que resulten en un paso iterativo computacionalmente sencillo y una mejor convergencia a un mínimo lo más cerca posible del mínimo global es fundamental. Existe mucho trabajo hecho en la comunidad académica en este ámbito, y parece que sigue en expansión. Por ello, sería conveniente ir evaluando las mejoras que aportan las distintas medidas de similitud entre las aproximaciones que se obtienen y los datos originales de partida.

La visión que aporta la factorización no-negativa de matrices de que un todo está formado por la suma de sus partes es tan fácilmente comprensible al ser humano que, ciertamente, sirve como garantía de que las técnicas de NMF van a seguir siendo desarrolladas por la comunidad académica en el futuro, ya que, como dicta el principio de la navaja de Ockham, la explicación más sencilla suele ser la más probable.

REFERENCIAS

[1] Y. Koren, «The BellKor solution to the Netflix Grand Prize,» 2009.

[2] P. O. Hoyer, «Non-negative matrix factorization with sparseness constraints,» *Journal of Machine Learning Research,* 2004.

[3] D. D. Lee y H. S. Seung, «Algorithms for Non-negative Matrix Factorization,» 2000.

[4] D. D. Lee y H. S. Seung, «Learning the parts of objects by non-negative matrix factorization,» *Letters to nature,* 1999.

[5] Y.-X. Wang y Y.-J. Zhang, «Nonnegative Matrix Factorization: A Comprehensive review,» 2013.

[6] A. Cichocki, S. Cruces y S. Amari, «Generalized Alpha-Beta Divergences and Their Application to Robust Nonnegative Matrix Factorization,» *Entropy,* 2011.

[7] M. D. Ekstrand, J. T. Riedl y J. A. Konstan, «Collaborative Filtering Recommender Systems,» *Foundations and trends un human-computer interaction,* 2010.

[8] J. Bobadilla, F. Ortega, A. Hernando y A. Gutiérrez, «Recommender systems survey,» *Knowledge-Based systems,* 2013.

[9] J. Bobadilla, F. Ortega, A. Hernando y J. Bernal, «A collaborative filtering approach to mitigate the new user cold start problem,» *Knowledge-based systems,* 2012.

[10] B. Sarwar, G. Karypis, J. A. Konstan y J. Riedl, «Item-based collaborative filtering recommendation algorithms,» de *10th International Conference on World Wide Web*, 2001.

[11] F. M. Harper y J. A. Konstan, «The MovieLens Datasets: History and Context,» *ACM Transactions on Interactive Intelligent Systems (TiiS),* 2015.

[12] F. Zhang, Y. Lu, J. Chen, S. Liu y Z. Ling, «Robust collaborative filtering based on non-negative matrix factorization and R1-norm,» *Knowledge-based systems,* 2017.

[13] I. Borg y P. J. Groenen, «Modern Multidimensional Scaling: Theory and Applications,» *Springer,* 2005.

ANEXO A – CÓDIGO DE MATLAB®

Script principal

```
clear; clc;
tic

R = readDataset('u.data'); % Obtiene matriz de datos.
nu = size(R,1); % Numero de usuarios.
np = size(R,2); % Numero de productos.
r = R>0; % Matriz de existencia (o no) de valoraciones.
r_max = max(max(R)); % Valoracion más alta encontrada.
r_min = min(R(R>0)); % Valoracion más baja encontrada (no puede ser 0).
r_step = r_max - max(R(R<r_max)); % Calcula el paso entre valoraciones.
ratings = r_min:r_step:r_max; % Vector con las posibles valoraciones.

rmsscore_best = 1e20; % Numero alto para mejorarlo en la primera iteracion.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n = 25; %NUMERO DE COMPONENTES LATENTES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tol = 1e-1; %Tolerancia para convergencia.
epsilon = 1e-15; %Para evitar divisiones por 0.
lambda = 0.1; %Parametro de regularizacion.
var_n = 10; %Varianza de los últimos var_n valores de J para convergencia.

for i = 1:5
    R_tr = readPartialDataset(strcat('u',num2str(i),'.base'),nu,np);
    R_cv = readPartialDataset(strcat('u',num2str(i),'.test'),nu,np);
    r_tr = R_tr>0;
    r_cv = R_cv>0;

    U = rand(nu,n);
    P = rand(np,n);
    R_approx = U*P';

    n_its = 5000; %Numero maximo de iteraciones para convergencia.

    J = zeros(1,n_its); %Vector de costes para tener datos de su evolucion.

    %Algoritmo NMF para distancia euclidea.
```

```
    %Primero se realizan las iteraciones suficientes como para rellenar
    %el vector de costes tantas veces como indique var_n.
    for j=1:var_n
        U = U.*(R_tr*P./((R_approx.*r_tr)*P+lambda*U+epsilon));
        R_approx = U*P';
        P = P.*(R_tr'*U./((R_approx.*r_tr)'*U+lambda*P+epsilon));
        R_approx = U*P';
        %Se actualiza la funcion de distancia.
        J(j) = computeCost(R_tr,r_tr,U,P,lambda);
    end
    %Una vez hay suficientes valores de J como para calcular la varianza
    %entre ellos, se atiende a criterios de convergencia.
    while (var(J(j-var_n+1:j)) > tol) && (n_its > j)
        U = U.*(R_tr*P./((R_approx.*r_tr)*P+lambda*U+epsilon));
        R_approx = U*P';
        P = P.*(R_tr'*U./((R_approx.*r_tr)'*U+lambda*P+epsilon));
        R_approx = U*P';
        %Se actualizan el indice y la funcion de distancia.
        j = j+1;
        J(j) = computeCost(R_tr,r_tr,U,P,lambda);
    end

    fprintf('TR-CV dataset %i: ', i);
    if J(end) > 0
        fprintf('max allowed iterations\n');
    else
        fprintf('reached convergence\n');
    end

    R_approx(R_approx<r_min) = r_min;
    R_approx(R_approx>r_max) = r_max;

    %Metrica de evaluacion, nos quedamos con la mejor aproximacion.
    rmsscore = rms(R(r_cv)-R_approx(r_cv));
    if rmsscore < rmsscore_best
        rmsscore_best = rmsscore;
        R_best = R_approx;
        r_cv_best = r_cv;
        J_best = J;
    end
end

R_predict = r_step*round(R_best/r_step); %Se redondea para predecir.
R_predict(R_predict<r_min) = r_min; %Se eliminan valores fuera de rango.
R_predict(R_predict>r_max) = r_max;
```

```
%Porcentaje de aciertos en la prediccion del conjunto de CV.
accuracy = sum(R_predict(r_cv_best)==R(r_cv_best))/nnz(r_cv_best)*100;
%Porcentaje de precision "relajada" (se aceptan valores adyacentes).
accuracy_relaxed = sum(abs(R_predict(r_cv_best)-
R(r_cv_best))<=r_step)/nnz(r_cv_best)*100;

last_it = find(J_best==0, 1, 'first')-1; %Calcula la ultima iteracion.

figure %Dibuja una grafica con la evolucion de la funcion de distancia.
plot(1:last_it,J_best(1:last_it))
title('Divergence')
xlabel('i (iterations)')
ylabel('J(i)')

fprintf('Accuracy of the best prediction over the CV set: %.2f
%%\n',accuracy);
fprintf('Relaxed accuracy of the best prediction over the CV set: %.2f
%%\n',accuracy_relaxed);
fprintf('RMS error of the best prediction over the CV set:
%.3f\n',rmsscore_best);
toc
```

Función "readDataset"

```
function R = readDataset(file_name)
% Esta funcion extrae el conjunto de datos del archivo que se indique.
D = dlmread(file_name,'\t'); % Lee la matriz de datos del archivo
                             % asumiendo que los valores estan
                             % separados por tabulaciones.
users = D(:,1); % La primera columna es el ID de usuario.
user_n = max(users); % Usuario con ID más alto.
prods = D(:,2); % La segunda columna es el ID de producto.
prod_n = max(prods); % Producto con ID más alto.
ratings = D(:,3); % La tercera columna es la valoracion.

% Declaramos la matriz en forma "sparse" por comodidad.
R = sparse(users,prods,ratings,user_n,prod_n);
% Pasamos la matriz a full eliminando posibles filas y columnas nulas.
R = full(R(any(R,2),any(R,1)));
end
```

Función "readPartialDataset"

```
function R = readPartialDataset(file_name,nu,np)

D = dlmread(file_name,'\t'); % Lee la matriz de datos del archivo.
R = sparse(D(:,1),D(:,2),D(:,3),nu,np); % Crea matriz sparse.
R = full(R); % Devuelve la matriz en formato normal.

end
```

Función "computeCost"

```
function J = computeCost(R, r, U, P,lambda)
% Distancia euclidea:
J = 0.5*(sum(sum(((((U*P').*r)-R).^2)))...
        + lambda/2*(sum(sum(U.^2))+sum(sum(P.^2)));
end
```

Script con variación en K

```
clear; clc;
tic

R = readDataset('u.data'); % Obtiene matriz de datos.
nu = size(R,1); % Numero de usuarios.
np = size(R,2); % Numero de productos.
r = R>0; % Matriz de existencia (o no) de valoraciones.
r_max = max(max(R)); % Valoracion más alta encontrada.
r_min = min(R(R>0)); % Valoracion más baja encontrada (no puede ser 0).
r_step = r_max - max(R(R<r_max)); % Calcula el paso entre valoraciones.
ratings = r_min:r_step:r_max; % Vector con las posibles valoraciones.

rmsscore_best = 1e50*ones(1,5); %Grande para mejorarlo a la primera.
accuracy = zeros(1,5); %Vector de coincidencias.
accuracy_relaxed = zeros(1,5); %Vector de coincidencias + adyacencias.

tol = 1e-1; %Tolerancia para convergencia.
epsilon = 1e-15; %Para evitar divisiones por 0.
lambda = 0.1; %Parametro de regularizacion.
var_n = 10; %Varianza de los últimos var_n valores de J para convergencia.

for K = 20:20:100 %Se va aumentando el numero de componentes latentes.

    for i = 1:5 %Se van cambiando las parejas de training - cv
```

```
        R_tr = readPartialDataset(strcat('u',num2str(i),'.base'),nu,np);
        R_cv = readPartialDataset(strcat('u',num2str(i),'.test'),nu,np);
        r_tr = R_tr>0;
        r_cv = R_cv>0;

        U = rand(nu,K);
        P = rand(np,K);
        R_approx = U*P';

        n_its = 4000; %Numero maximo de iteraciones para convergencia.

        J = zeros(1,n_its); %Vector de costes para datos de su evolucion.

        %Algoritmo NMF para distancia euclidea.

        %Primero se realizan las iteraciones suficientes como para
        %rellenar el vector de costes tantas veces como indique var_n.
        for j=1:var_n
            U = U.*(R_tr*P./((R_approx.*r_tr)*P+lambda*U+epsilon));
            R_approx = U*P';
            P = P.*(R_tr'*U./((R_approx.*r_tr)'*U+lambda*P+epsilon));
            R_approx = U*P';
            %Se actualiza la funcion de distancia.
            J(j) = computeCost(R_tr,r_tr,U,P,lambda);
        end
        %Una vez hay suficientes valores de J como para calcular la
        %varianza entre ellos, se atiende a criterios de convergencia.
        while (var(J(j-var_n+1:j)) > tol) && (n_its > j)
            U = U.*(R_tr*P./((R_approx.*r_tr)*P+lambda*U+epsilon));
            R_approx = U*P';
            P = P.*(R_tr'*U./((R_approx.*r_tr)'*U+lambda*P+epsilon));
            R_approx = U*P';
            %Se actualizan el indice y la funcion de distancia.
            j = j+1;
            J(j) = computeCost(R_tr,r_tr,U,P,lambda);
        end

        fprintf('TR-CV dataset %i: ', i);
        if J(end) > 0
            fprintf('max allowed iterations\n');
        else
            fprintf('reached convergence\n');
        end

        R_approx(R_approx<r_min) = r_min;
        R_approx(R_approx>r_max) = r_max;

        %Metrica de evaluacion, nos quedamos con la mejor aproximacion.
        rmsscore = rms(R(r_cv)-R_approx(r_cv));
        if rmsscore < rmsscore_best(K/20)
            rmsscore_best(K/20) = rmsscore;
            R_best = R_approx;
            r_cv_best = r_cv;
        end

    end

    R_predict = r_step*round(R_best/r_step); %Se redondea para predecir.
    R_predict(R_predict<r_min) = r_min; %Se eliminan valores fuera de rango.
    R_predict(R_predict>r_max) = r_max;
```

```
    %Porcentaje de aciertos en la prediccion del conjunto de CV.
    accuracy(K/20) =
sum(R_predict(r_cv_best)==R(r_cv_best))/nnz(r_cv_best)*100;
    %Porcentaje de precision "relajada" (se aceptan valores adyacentes).
    accuracy_relaxed(K/20) = sum(abs(R_predict(r_cv_best)-
R(r_cv_best))<=r_step)/nnz(r_cv_best)*100;

end

figure %Dibuja una grafica con la evolucion del error RMS
plot(20:20:100,rmsscore_best,'b--o')
title('RMS error vs Number of components')
xlabel('Number of components')
ylabel('RMS error')

figure %Dibuja una grafica con la evolucion de las coincidencias
plot(20:20:100,accuracy,'b--o')
title('Accuracy vs Number of components')
xlabel('Number of components')
ylabel('Accuracy (%)')

figure %Dibuja una grafica con la evolucion de las adyacencias
plot(20:20:100,accuracy_relaxed,'b--o')
title('Relaxed accuracy vs Number of components')
xlabel('Number of components')
ylabel('Relaxed accuracy (%)')

toc
```

Script con variación en lambda

```
clear; clc;
tic

R = readDataset('u.data'); % Obtiene matriz de datos.
nu = size(R,1); % Numero de usuarios.
np = size(R,2); % Numero de productos.
r = R>0; % Matriz de existencia (o no) de valoraciones.
r_max = max(max(R)); % Valoracion más alta encontrada.
r_min = min(R(R>0)); % Valoracion más baja encontrada (no puede ser 0).
r_step = r_max - max(R(R<r_max)); % Calcula el paso entre valoraciones.
ratings = r_min:r_step:r_max; % Vector con las posibles valoraciones.

rmsscore_best = 1e50*ones(1,5); %Grande para mejorarlo a la primera.
accuracy = zeros(1,5); %Vector de coincidencias.
accuracy_relaxed = zeros(1,5); %Vector de coincidencias + adyacencias.

K = 25; %Numero de componentes principales
tol = 1e-1; %Tolerancia para convergencia.
epsilon = 1e-15; %Para evitar divisiones por 0.
var_n = 10; %Varianza de los últimos var_n valores de J para convergencia.
```

```
for lambdalog = (-2):1:2 %Se va aumentando la regularizacion.

    lambda = 10^lambdalog;

    for i = 1:5 %Se van cambiando las parejas de training - cv
        R_tr = readPartialDataset(strcat('u',num2str(i),'.base'),nu,np);
        R_cv = readPartialDataset(strcat('u',num2str(i),'.test'),nu,np);
        r_tr = R_tr>0;
        r_cv = R_cv>0;

        U = rand(nu,K);
        P = rand(np,K);
        R_approx = U*P';

        n_its = 4000; %Numero maximo de iteraciones para convergencia.

        J = zeros(1,n_its); %Vector de costes para datos de su evolucion.

        %Algoritmo NMF para distancia euclidea.

        %Primero se realizan las iteraciones suficientes como para
        %rellenar el vector de costes tantas veces como indique var_n.
        for j=1:var_n
            U = U.*(R_tr*P./((R_approx.*r_tr)*P+lambda*U+epsilon));
            R_approx = U*P';
            P = P.*(R_tr'*U./((R_approx.*r_tr)'*U+lambda*P+epsilon));
            R_approx = U*P';
            %Se actualiza la funcion de distancia.
            J(j) = computeCost(R_tr,r_tr,U,P,lambda);
       end
        %Una vez hay suficientes valores de J como para calcular la
        %varianza entre ellos, se atiende a criterios de convergencia.
        while (var(J(j-var_n+1:j)) > tol) && (n_its > j)
            U = U.*(R_tr*P./((R_approx.*r_tr)*P+lambda*U+epsilon));
            R_approx = U*P';
            P = P.*(R_tr'*U./((R_approx.*r_tr)'*U+lambda*P+epsilon));
            R_approx = U*P';
            %Se actualizan el indice y la funcion de distancia.
            j = j+1;
            J(j) = computeCost(R_tr,r_tr,U,P,lambda);
        end

        fprintf('TR-CV dataset %i: ', i);
        if J(end) > 0
            fprintf('max allowed iterations\n');
        else
            fprintf('reached convergence\n');
        end

        R_approx(R_approx<r_min) = r_min;
        R_approx(R_approx>r_max) = r_max;

        %Metrica de evaluacion, nos quedamos con la mejor aproximacion.
        rmsscore = rms(R(r_cv)-R_approx(r_cv));
        if rmsscore < rmsscore_best(lambdalog+3)
            rmsscore_best(lambdalog+3) = rmsscore;
            R_best = R_approx;
            r_cv_best = r_cv;
        end
```

```
    end

    R_predict = r_step*round(R_best/r_step); %Se redondea para predecir.
    R_predict(R_predict<r_min) = r_min; %Se eliminan valores fuera de rango.
    R_predict(R_predict>r_max) = r_max;

    %Porcentaje de aciertos en la prediccion del conjunto de CV.
    accuracy(lambdalog+3) =
sum(R_predict(r_cv_best)==R(r_cv_best))/nnz(r_cv_best)*100;
    %Porcentaje de precision "relajada" (se aceptan valores adyacentes).
    accuracy_relaxed(lambdalog+3) = sum(abs(R_predict(r_cv_best)-
R(r_cv_best))<=r_step)/nnz(r_cv_best)*100;

end

figure %Dibuja una grafica con la evolucion del error RMS
plot((-2):1:2,rmsscore_best,'b--o')
title('RMS error vs Regularization parameter')
xlabel('log10(lambda)')
ylabel('RMS error')

figure %Dibuja una grafica con la evolucion de las coincidencias
plot((-2):1:2,accuracy,'b--o')
title('Accuracy vs Regularization parameter')
xlabel('log10(lambda)')
ylabel('Accuracy (%)')

figure %Dibuja una grafica con la evolucion de las adyacencias
plot((-2):1:2,accuracy_relaxed,'b--o')
title('Relaxed accuracy vs Regularization parameter')
xlabel('log10(lambda)')
ylabel('Relaxed accuracy (%)')

toc
```

Script con variación en el porcentaje de matriz completo

```
clear; clc;
tic

R = readDataset('u.data'); % Obtiene matriz de datos.
r = R>0; % Matriz de existencia (o no) de valoraciones.
r_max = max(max(R)); % Valoracion más alta encontrada.
r_min = min(R(R>0)); % Valoracion más baja encontrada (no puede ser 0).
r_step = r_max - max(R(R<r_max)); % Calcula el paso entre valoraciones.
ratings = r_min:r_step:r_max; % Vector con las posibles valoraciones.

rmsscore_best = 1e50*ones(1,5); %Grande para mejorarlo a la primera.
accuracy = zeros(1,5); %Vector de coincidencias.
accuracy_relaxed = zeros(1,5); %Vector de coincidencias + adyacencias.
```

```
K = 50; % Numero de componentes latentes.
tol = 1e-1; %Tolerancia para convergencia.
epsilon = 1e-15; %Para evitar divisiones por 0.
lambda = 0.1; %Parametro de regularizacion.
var_n = 10; %Varianza de los últimos var_n valores de J para convergencia.
completion = zeros(1,5); %Porcentaje de matriz rellena.

for tam = 1:5 %Se va aumentando el multiplicador de tamaño.

    n_its = 4000; %Numero maximo de iteraciones para convergencia.
    nu = tam*100;
    np = tam*200;

    u_ord = colperm(R'); u_ord = u_ord((end-nu+1):end);
    p_ord = colperm(R); p_ord = p_ord((end-np+1):end);
    R_ord = R(u_ord,p_ord); %Crea matriz de trabajo.
    R_ord = R_ord(randperm(nu),randperm(np)); %Matriz preparada
    r_ord = R_ord>0;

    completion(tam) = nnz(R_ord)/(nu*np)*100; %Calcula el porcentaje relleno.

    %Se escoge aleatoriamente un subconjunto de datos de training y CV.
    [fils, cols, vals] = find(R_ord);
    previa = randperm(length(fils));
    lista_tr = previa(1:floor(0.6*length(fils))); %60% training
    lista_cv = previa(floor(0.6*length(fils)+1):end); %40% cross-validation
    R_tr = sparse(fils(lista_tr),cols(lista_tr),vals(lista_tr),nu,np);
    R_tr = full(R_tr);
    R_cv = sparse(fils(lista_cv),cols(lista_cv),vals(lista_cv),nu,np);
    R_cv = full(R_cv);
    r_tr = R_tr>0;
    r_cv = R_cv>0;

    U = rand(nu,K);
    P = rand(np,K);
    R_approx = U*P';

    J = zeros(1,n_its); %Vector de coste.

    %Algoritmo NMF para distancia euclidea.

    %Primero se realizan las iteraciones suficientes como para
    %rellenar el vector de costes tantas veces como indique var_n.
    for j=1:var_n
        U = U.*(R_tr*P./((R_approx.*r_tr)*P+lambda*U+epsilon));
        R_approx = U*P';
        P = P.*(R_tr'*U./((R_approx.*r_tr)'*U+lambda*P+epsilon));
        R_approx = U*P';
        %Se actualiza la funcion de distancia.
        J(j) = computeCost(R_tr,r_tr,U,P,lambda);
    end
    %Una vez hay suficientes valores de J como para calcular la
    %varianza entre ellos, se atiende a criterios de convergencia.
    while (var(J(j-var_n+1:j)) > tol) && (n_its > j)
        U = U.*(R_tr*P./((R_approx.*r_tr)*P+lambda*U+epsilon));
        R_approx = U*P';
        P = P.*(R_tr'*U./((R_approx.*r_tr)'*U+lambda*P+epsilon));
        R_approx = U*P';
        %Se actualizan el indice y la funcion de distancia.
        j = j+1;
```

```
        J(j) = computeCost(R_tr,r_tr,U,P,lambda);
    end

    fprintf('TR-CV dataset %i: ', tam);
    if J(end) > 0
        fprintf('max allowed iterations\n');
    else
        fprintf('reached convergence\n');
    end

    R_approx(R_approx<r_min) = r_min;
    R_approx(R_approx>r_max) = r_max;
    %Metrica de evaluacion sobre el subconjunto de validacion cruzada.
    rmsscore_best(tam) = rms(R_ord(r_cv)-R_approx(r_cv));

    R_predict = r_step*round(R_approx/r_step); %Se redondea para predecir.
    R_predict(R_predict<r_min) = r_min; %Se eliminan valores fuera de rango.
    R_predict(R_predict>r_max) = r_max;

    %Porcentaje de aciertos en la prediccion del conjunto de CV.
    accuracy(tam) = sum(R_predict(r_cv)==R_ord(r_cv))/nnz(r_cv)*100;
    %Porcentaje de precision "relajada" (se aceptan valores adyacentes).
    accuracy_relaxed(tam) = sum(abs(R_predict(r_cv)-
R_ord(r_cv))<=r_step)/nnz(r_cv)*100;

end

figure %Dibuja una grafica con la evolucion del error RMS
plot(completion,rmsscore_best,'b--o')
title('RMS error vs Matrix completion')
xlabel('Matrix completion (%)')
ylabel('RMS error')

figure %Dibuja una grafica con la evolucion de las coincidencias
plot(completion,accuracy,'b--o')
title('Accuracy vs Matrix completion')
xlabel('Matrix completion (%)')
ylabel('Accuracy (%)')

figure %Dibuja una grafica con la evolucion de las adyacencias
plot(completion,accuracy_relaxed,'b--o')
title('Relaxed accuracy vs Matrix completion')
xlabel('Matrix completion (%)')
ylabel('Relaxed accuracy (%)')

toc
```