UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA UNIVERSITARIA DE INFORMÁTICA

PROYECTO FIN DE MASTER

SVD APLICADO A SISTEMAS DE RECOMENDACIÓN BASADOS EN FILTRADO COLABORATIVO

MASTER EN CIENCIAS Y TECNOLOGÍAS DE LA COMPUTACIÓN

Curso 2012-2013

Autor: Ricardo Moya García
Tutor: Antonio Hernando Esteban

Fecha: Junio del 2013

"Está bien celebrar el éxito, pero es más importante prestar atención a las lecciones del fracaso"

Bill Gates

A mis padres por el apoyo y el esfuerzo que realizan y han realizado por mis estudios y formación; a mi hermana por su apoyo; a mis compañeros de master que han hecho de él una experiencia muy gratificante y amena; a mis profesores y en especial a aquellos que me han hecho ver lo duro y fascinante que es el mundo de la investigación y de la innovación; a Jesús y Fernando (Bobi y Fer) por haberme abierto las puertas al mundo de la investigación y haber podido aprender muchísimo de ellos; y por último y muy especialmente a Antonio, por dirigirme este proyecto, por todas las magníficas clases impartidas, por todos los conocimientos trasmitidos y sobre todo por ser mi guía en el mundo de la investigación.

Resumen

En este proyecto fin de master se va a presentar el método del SVD (Singular Value Descomposition) aplicado a los sistemas de recomendación basados en filtrado colaborativo. Este método, es un método basado en modelos que permite realizar unas recomendaciones muy buenas (en comparación con las técnicas clásicas de los k-vecinos) a los usuarios del sistema de recomendación. La prueba de que el método del SVD es uno de los mejores métodos para recomendar, esta en que la aplicación de esta técnica logró ganar el premio Netflix (valorado en uno millón de dólares) de sistemas de recomendación que consistía en mejorar las recomendaciones realizadas con los métodos existentes hasta el momento.

En este proyecto fin de master se va ha hacer un repaso del estado del arte sobre los sistemas de recomendación basados en filtrado colaborativo y se explicará la técnica matemática del SVD. Posteriormente se explicará como se aplica el SVD a los sistemas de recomendación, viendo sus propiedades, ventajas, inconvenientes, algoritmos, etc. y posteriormente se mostrarán los experimentos realizados con la base de datos de Movielens 1M y se compararan con las métricas tradicionales de los k-vecinos y con dos de las mejores métricas propuestas en los artículos de investigación (JCR).

Abstract

In this master proyect it is going to be presented SVD method (Singular Value Descomposition) applied to the recommender systems based on collaborative filtering. This method is based on models that allows to do very good recommendations (in comparison with classic technics of k-neighbor) for recommender systems users. The proof that the SVD method is one of the best methods to recommend, is in the application of this technique won the Netflix Prize (valued at one million dollars) recommendation system which consisted on improving the recommendations made to existing methods so far.

In this master proyect it is going to be done a review of the state of art recommender systems based on collaborative filtering and it will be explained the mathematical technique of SVD. Afterwards, it will be explained how to apply the SVD recommender systems, looking at its qualities, advantages, disadvantages, algorithms, etc. finally, it will be shown the experiments made with database Movielens 1M and they will be compared with the traditional metrics of k-neighbor and with two of the best metrics proposed in research papers (JCR).

Índice

Índice de figuras

Índice de tablas

1. Introducción

Hoy en día, Internet es una importante herramienta de comunicación y de obtención de información que es utilizada a diario por mucha gente para trabajar, entretenerse, mantenerse informado sobre la actualidad, realizar transacciones, compras, etc. Es sin duda Internet, una de las herramientas mas importantes y versátiles creadas por el ser humano, pero es justo esa versatilidad el punto mas débil que tiene Internet, ya que la mayoría de los usuarios se sienten impotentes ante la gran cantidad de información que hay en la red. A este fenómeno se le conoce con el nombre de "*el problema de la sobrecarga de información*".

Con el objetivo de reducir este problema de la sobrecarga de información, se han desarrollado herramientas conocidas como "Buscadores", que tienen como objetivo mostrar al usuario la información que hay en Internet, filtrando esa información por palabras clave. El resultado que ofrecen esos buscadores al usuario, es una lista de paginas web (tras realizar un análisis del contenido que hay en Internet) que contienen las palabras claves buscadas y que les permitirá encontrar los elementos que buscan en Internet. Uno de los buscadores más conocidos y utilizados por los usuarios es Google, que ordena los resultados de las búsquedas con el algoritmo conocido como el *PageRank*, que es un algoritmo que otorga un factor de importancia a las paginas web y que las lista en función de la búsqueda realizada por el usuario y de la importancia de la web.

Con el paso del tiempo Internet ha ido evolucionando hacia lo que se conoce como la Web 2.0 o Web social, que ha permitido que los propios usuarios de Internet puedan generar información (a través de Blogs, Wikis, etc.) y agravar de alguna forma el problema de la sobrecarga de información. Ni mucho menos estos se ha convertido en un problema ya que gracias a los usuarios, Internet tiene cada vez mas contenido y permite a los usuarios obtener mayor información sobre determinados temas, pero sigue siendo necesario que esa información se filtre de alguna forma. Otro éxito de la Web social, han sido las redes

sociales, como Facebook, Twitter, Linkedin que también permiten a los usuarios generar información de carácter más personal.

Pese al aplastante éxito de la Web social, para un usuario de Internet inexperto o sin ganas de preocuparse por elegir el tipo de información que desea recibir, este tipo de paradigma de Internet es insuficiente. Seria mucho mas cómodo para los usuarios que un sistema informático monitorizase su actividad en la red y que descubriera lo que realmente les gusta y les consiguiese aquella información que fuese verdaderamente interesante para el usuario. Estas herramientas existe y han sido denominadas como *Sistemas de Recomendación.*

Un sistema de recomendación, es un sistema inteligente, capaz de aprender las preferencias o gustos de un usuario y poder ofrecerle aquella información que pueda ser de utilidad para dicho usuario. Por tanto, podemos entender el sistema de recomendación como un filtro que deja pasar aquella información que le va a resultar de interés al usuarios y va a desechar aquella información que le pueda resultar al usuario indiferente.

La gran ventaja que tienen los sistemas de recomendación frente a los buscadores o a la Web social, es precisamente el filtrado automático de la información. Otro punto favorable de los sistemas de recomendación, es que son capaces de cubrir cualquier campo en el que se requieran realizar recomendaciones como por ejemplo recomendar libros, peliculas, paginas web, restaurantes, viajes, etc. En definitiva, cualquier objeto que pueda ser clasificado de forma implícita o explícita podrá ser recomendado por un sistema de recomendación.

La figura 1 trata de demostrar la eficiencia de los sistemas de recomendación. En ella puede observarse el porcentaje de recomendaciones exitosas realizadas en la página web de FilmAffinity [1], respecto a un total de 10, utilizando como predictores de voto: el sistema de recomendación, la recomendación media de todos los usuarios y la votación media de los amigos de cada usuario. Como se observa, los mejores resultados se obtienen

con el sistema de recomendación, seguido de los amigos y de la media. Con esta figura se demuestra que es posible realizar recomendaciones personalizadas a los usuarios con mayor eficiencia que las técnicas mas tradicionales.

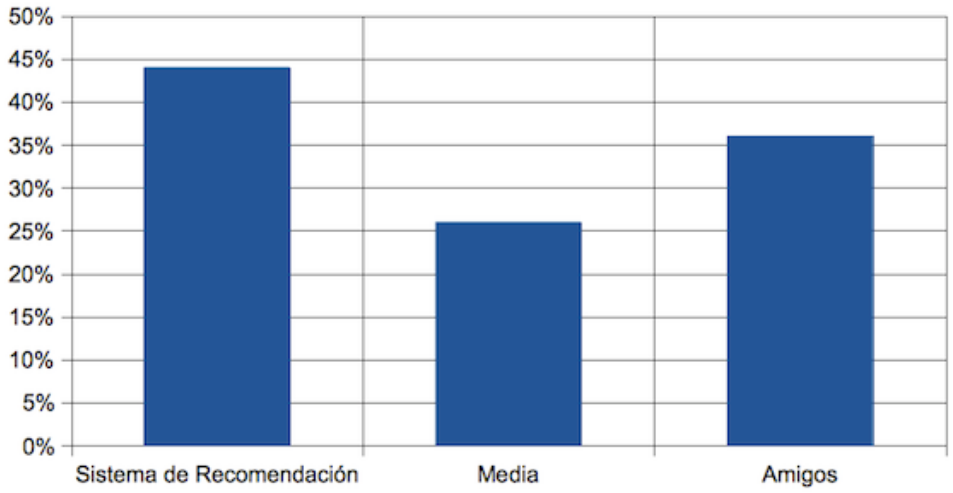

Figura 1: Porcentaje de recomendaciones exitosas logradas en FilmAffinity utilizando: el sistema de recomendación, la votación media de los usuarios y la votación media de los amigos como predictores.

En la actualidad, numerosas empresas tienen en funcionamiento sistemas de recomendación con la finalidad de ayudar a los usuarios a encontrar la información que buscan. Por ejemplo Amazon, recomienda productos a los usuarios en función de las compras que han realizado, FilmAffinity realiza recomendaciones de películas basándose en las votaciones de los usuarios, Last.fm analiza la música que escucha un usuario para aprender lo que le gusta y poder recomendar nuevas canciones o grupos, etc. Como puede verse es un mercado emergente y bien valorado por las compañías de Internet, por lo que, su estudio resulta muy interesante y prometedor.

2. Sistemas de Recomendación

2.1. Introducción

Un sistema de recomendación es un sistema inteligente que proporciona a los usuarios una serie de sugerencias personalizadas (recomendaciones) sobre un determinado tipo de elementos (items). Los sistemas de recomendación estudian las características de cada usuario y mediante un procesamiento de los datos, encuentra un subconjunto de items que pueden resultar de interés para el usuario.

La forma en la que se realizarán las recomendaciones puede variar mucho en función de que es lo que se quiera recomendar en un sistema de recomendación. No obstante, todos los sistemas de recomendación deben de tener en común las siguientes características[9] [17]:

1. ***Existe un conjunto de usuarios a los que realizar recomendaciones***. Sobre estos usuarios se puede tener la información que el sistemas de recomendación necesite tener, como nombre, edad, sexo, etc.

2. ***Se dispone de un conjunto de items que se quiere recomendar.*** Los items a recomendar pueden diferir mucho entre cada sistema (libros, música, películas, viajes, etc).

3. ***Se tienen registrados las valoraciones que realizan los usuarios sobre los items.*** La forma en la que los usuarios valoran los items puede ser implícita (el usuario es monitorizado y se evalúan los items en función de su comportamiento) o explícita (el usuario decide que items valorar).

De forma adicional y dependiendo del sistema de recomendación, el sistema puede tener registrada más información sobre los usuarios y los items para llevar a cabo el proceso de recomendación.

2.2. Clasificación

El funcionamiento de los sistemas de recomendación puede variar en función del tipo de items que se quiera recomendar y de la forma en la que tanto los items como las preferencias de los usuarios son almacenadas. Por tanto, los sistemas de recomendación emplean diferentes mecanismos para recomendar los items a los usuarios. Es habitual que al sistema de recomendación se le denomine "filtro"debido a que actúa como tal y actualmente se clasifican los sistemas de recomendación en cuatro tipos que son los siguientes [9][8][19]:

1. ***Filtrado basado en contenido*** [24]: las recomendaciones que se realizan al usuario se basa en el conocimiento que se tiene de los items que el usuario ha valorado en el pasado. Un ejemplo de este tipo de filtrado podría ser la recomendación de un determinado libro de ciencia ficción (que el usuario no haya leído), debido a que el usuario ha votado de forma positiva un libro de ese genero que previamente ha leído.

2. ***Filtrado demográfico*** [16]: las recomendaciones que se realizan al usuario activo, se realizan comparando las valoraciones positivas de otros usuarios que comparten edad, sexo, situación geográfica, profesión, etc. con el usuario activo. En este tipo de filtrado se presupone que usuarios con características sociales similares comparten las preferencias sobre los items a recomendar.

3. ***Filtrado colaborativo*** [9][8][11]: los datos de los usuarios y los items es almacenado en una base de datos que contienen las votaciones de un gran número de usuarios sobre un gran numero de items (peliculas, libros, viajes, etc.). El filtrado colaborativo consiste en ver que usuarios son similares al usuario activo y a continuación, recomendar aquellos items que no han sido votados por el usuario activo y que han resultado bien valorados por los usuarios similares.

4. ***Métodos de filtrado híbridos*** [21][27][19][18]: los métodos híbridos mezclan alguno de los tres filtrados mencionados anteriormente. Por lo general se suele mezclar

el filtrado basado en contenidos o el filtrado demográfico con el filtrado colaborativo. Con los métodos híbridos podemos realizar recomendaciones en base a un conjunto más amplio de información y además manejar las situaciones de *cold-star* [4][14] en las que es difícil realizar las recomendaciones con la técnica del filtrado colaborativo debido a que la base de datos de las votaciones es pequeña.

Generalmente el filtrado colaborativo obtiene mejores resultados que el resto de sistemas de filtrado descritos [19] por lo que su uso es el más extendido tanto en el mundo de la investigación como en los sistemas comerciales. El problema que tiene el filtrado colaborativo es que necesita tener una base de datos relativamente grande para poder buscar usuarios similares y realizar las recomendaciones [11]. Generalmente un sistema de recomendación tendrá almacenado en la base de datos decenas de miles de usuarios y miles de items los cuales tendrán millones de votaciones realizado por los usuarios sobre los items. Por otro lado se ha de tener en cuenta que un usuario únicamente vota un pequeño subconjunto de los items que hay registrados en la base de datos, por lo que las matrices de votaciones entre usuarios e items tendrán un elevado grado de dispersión [7][22], lo cual supone un problema para el filtrado colaborativo.

Dado que el filtrado colaborativo es la técnica de filtrado que mejores resultados genera y la que se utilizara para la realización de este trabajo, se he establecido una clasificación de los sistemas de filtrado colaborativo que los clasifican en función del método usado para determinar a los usuarios que son similares al usuario activo. Sobre esta clasificación se encuentran los siguientes tres tipos [9][8][19]:

1. ***Métodos basados en memoria***: emplean métricas de similaridad para determinar el parecido entre una pareja de usuarios. Para ello calculan los items que han sido votados por ambos usuarios y comparan dichos votos para calcular la similaridad.

2. ***Métodos basados en modelos***: utilizan la matriz de votaciones para crear un modelo a través del cual establecer el conjunto de usuarios similares al usuario ac-

tivo. Algunos ejemplos de estos modelos son los clasificadores bayesianos, las redes neuronales, algoritmos genéticos, sistemas borrosos y el SVD que trataremos en este trabajo con más profundidad.

3. ***Métodos híbridos***: son métodos que se encargan de combinar los dos métodos descritos anteriormente.

Este trabajo se centrará en el estudio de los sistemas de recomendación basados en filtrado colaborativo, al ser este un método que genera unos resultados bastante buenos. Por un lado se realizará un introducción al filtrado colaborativo basado en memoria, explicando las principales características y los resultados que se obtienen con este tipo de métodos. Por otro lado, este trabajo se centrará en explicar el modelo denominado SVD (Singular Value Descomposition), que es un método basado en modelos, y se comparará con los métodos basados en memoria para ver cuales son las ventajas y los inconvenientes de utilizar un modelo u otro.

2.3. Filtrado Colaborativo basado en memoria

En este punto del trabajo se van a mostrar las principales características del filtrado colaborativo basado en memoria, para posteriormente poder compararlo con la técnica del SVD y ver cuales son las ventajas e inconvenientes de utilizar uno u otro. Por otro lado también se muestran las características del filtrado colaborativo basado en memoria porque se puede aplicar también a los factores que nos da la técnica del SVD.

2.3.1. Esquema general del filtrado colaborativo

Como ya se ha comentado, el filtrado colaborativo (al igual que el resto de filtrados), tiene como objetivo presentar al usuario una serie de items que le puedan resultar de interés. El proceso por el cual se consigue mostrar al usuario estos items, se consigue mediante la búsqueda de usuarios similares al activo y el análisis de sus votaciones. En

lineas generales, la forma en la que se realizan las recomendaciones al usuario activo puede ser divido en las siguientes cuatro etapas [9][8]:

1. ***Cálculo de la similaridad entre usuarios***: En primer lugar se ha de elegir una métrica para determinar la similaridad[17][10][28][15] entre una pareja de usuarios. Algunas de las métricas más utilizadas son: la correlación de Pearson, el coseno, la correlación con restricciones, el coeficiente de correlación de Spearman, la diferencia cuadrática media (MSD) y el indice de Jaccard (JMSD).
2. ***Calcular los K-Vecinos***: Haciendo uso de la métrica de similaridad seleccionada, se obtienen los k usuarios más similares al usuario activo. A estos usuarios se les denomina como los k-vecinos del usuario activo.
3. ***Calcular las predicciones de los items***: A partir de los k-vecinos del usuario activo, se determinan las posibles valoraciones que el usuario activo haría sobre los items que no ha votado, es decir, se predice como el usuario valoraría esos items.
4. ***Realizar las recomendaciones***: Tras el calculo de las predicciones, se eligen los N items más adecuados para ser recomendados al usuario, es decir, las predicciones más altas, mas novedosas, mas votadas, etc. De forma opcional, puede incluirse un umbral para evitar que los items con una predicción inferior a dicho umbral sean recomendados.

En los siguientes puntos de este trabajo, se mostrará mas en detalle los pasos descrito anteriormente y se mostrará la formalización matemática de cada uno de ellos. Para clarificar estos pasos se mostrará un pequeño ejemplo del funcionamiento de cada una de las técnicas expuestas.

2.3.2. Elementos fundamentales del filtrado colaborativo

Para que un sistema de recomendación basado en un filtrado colaborativo funcione correctamente, se debe disponer de una base de datos en la que n usuarios han votado

m items dentro de un rango de votaciones *[min,...,max]*, siendo este rango discreto o continuo (en nuestro caso discreto), en la que la ausencia de voto se representa mediante el simbolo $\bullet$.

Esencialmente vamos a disponer de los siguientes elementos para completar el proceso de recomendación a los usuarios:

$$U = \{u \in |1 \leq u \leq n\}\text{, conjunto de usuarios.} \tag{1}$$

$$I = \{i \in |1 \leq i \leq m\}\text{, conjunto de items.} \tag{2}$$

$$W = \{w \in |min \leq w \leq max\} \cup \{\bullet\}\text{, conjunto de posibles votaciones;} \tag{3}$$

$$R_u = \{(i, v)\,|i \in I, v \in W\}\text{, votos del usuario u} \tag{4}$$

$$\text{Denotamos el voto } v \text{ del usuario } u \text{ al item } i \text{ como } r_{u,i} = v. \tag{5}$$

Definimos la cardinalidad de un conjunto C como el número de elementos válidos de dicho conjunto:

$$\#C = \#\{x \in C | x \neq \bullet\}\ . \tag{6}$$

$$\text{Definimos } I_u = \{i \in I | r_{u,i} \neq \bullet\} \text{ como el conjunto de votos válidos de } u. \tag{7}$$

$$\text{Denotamos la votación média de un usuario } u \text{ como } \overline{r_u} = \frac{1}{\#I_u} \cdot \sum_{i \in I_u} r_{u,i}\ . \tag{8}$$

A través de estos elementos y de algunos otros derivados de ellos, seremos capaces de realizar el proceso de filtrado colaborativo completo, o dicho de otra forma, seremos capaces de realizar las recomendaciones de los items a los usuarios del sistema.

Ejemplo de filtrado colaborativo tradicional

En este punto se empezará a mostrar el ejemplo a través del cual va a explicarse el funcionamiento del filtrado colaborativo tradicional. En apartados posteriores se irá completando el ejemplo hasta indicar los items que serian recomendados a cada usuario y los errores que se comenten al recomendar esos items. Los siguientes parametros que van a ser utilizados durante la ejecución de todo el ejemplo se encuentran representados en la siguiente tabla 1.

Parámetro	Valor
Número de usuarios	6
Número de items	12
Valor mínimo de la votación	1
Valor máximo de la votación	5
Ausencia de voto	•
Número de vecinos (k)	2
Número de recomendaciones (N)	2
Umbral de relevancia de un item (θ)	4

Cuadro 1: Parámetros utilizados en el ejemplo de filtrado colaborativo.

La tabla 2 contiene las votaciones que los 6 usuarios del sistema han realizado sobre los 12 items disponibles.

	I1	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I9**	**I10**	**I11**	**I12**
U1	1	2	•	•	2	•	3	4	•	4	1	•
U2	•	•	1	5	•	5	3	1	•	5	2	1
U3	1	•	•	2	•	1	•	3	4	•	•	•
U4	•	1	4	4	•	•	3	•	5	4	•	1
U5	2	•	5	•	1	•	1	•	•	•	2	1
U6	•	•	5	2	1	•	•	4	•	1	•	2

Cuadro 2: Votaciones de los usuarios en el ejemplo de filtrado colaborativo

2.3.3. Busqueda de los K-vecinos

El proceso de búsqueda de los k-vecinos, es decir, los k usuarios más similares al usuario activo, puede resumirse en los siguientes dos pasos:

1. Calcular la similaridad del usuario activo con el resto de usuarios del sistema.
2. Encontrar los k usuarios con la similaridad más alta respecto al usuario activo.

El primer punto es uno de los elementos fundamentales del filtrado colaborativo, ya que sino se calcula correctamente el parecido entre una pareja de usuarios, no seremos capaces de encontrar aquellos items que realmente van a resultar interesantes para el usuario activo. Uno de los problemas que suele tener el filtrado colaborativo, es que la

base de datos de las votaciones con la que se trabaja tiene un grado de dispersión muy elevado, es decir; que los usuarios votan un pequeño porcentaje de los items que hay en el sistema, por lo que la similaridad entre un par de usuarios debe ser calculada únicamente con aquellos items que hayan sido comúnmente valorados por ambos usuarios.

Definimos el conjunto de items que han sido valorados tanto por el usuario x como por el usuario y como:

$$B_{x,y} = \{i \in I | r_{x,i} \neq \bullet \wedge r_{y,i} \neq \bullet\} \ . \tag{9}$$

A partir de este conjunto de items somos capaces de calcular la similaridad entre dos usuarios con alguna de las métricas de similaridad existentes. A lo largo de los años se han propuesto varias métricas de similaridad para el filtrado colaborativo tradicional. Algunas de las más utilizadas son [9][8][17][15]: la diferencia cuadrática media (MSD) (10), la correlación de Pearson (11) y el coseno (12).

$$sim(x,y) = 1 - \frac{1}{\#B_{x,y}} \sum_{i \in I_u} \left(\frac{r_{x,i} - r_{y,i}}{max - min} \right)^2 \in [0,1] \text{ cuando } B_{x,y} \neq \emptyset \tag{10}$$

$$sim(x,y) = \frac{\sum_{i \in B_{x,y}} (r_{x,i} - \bar{r_x}) \cdot (r_{y,i} - \bar{r_y})}{\sqrt{\sum_{i \in B_{x,y}} (r_{x,i} - \bar{r_x})^2 \cdot \sum_{i \in B_{x,y}} (r_{y,i} - \bar{r_y})^2}} \in [-1,1] \text{ cuando } B_{x,y} \neq \emptyset \tag{11}$$

$$sim(x,y) = \frac{\sum_{i \in B_{x,y}} r_{x,i} \cdot r_{y,i}}{\sqrt{\sum_{i \in B_{x,y}} r_{x,i}^2} \cdot \sqrt{\sum_{i \in B_{x,y}} r_{y,i}^2}} \in [0,1] \text{ cuando } B_{x,y} \neq \emptyset \tag{12}$$

En el caso de que no existan items valorados por ambos usuarios, es decir, si $B_{x,y} = \emptyset$, se puede decir que:

$$sim(x,y) = \bullet \text{ cuando } B_{x,y} = \emptyset. \tag{13}$$

Igualmente definimos que la similaridad de un usuario con el mismo no puede ser calculada:

$$sim(x,y) = \bullet \text{ cuando } x = y. \tag{14}$$

Además, todas las métricas anteriores verifican la propiedad conmutativa, por lo que, dados el usuario x y el usuario y se cumple:

$$sim(x,y) = sim(y,x). \tag{15}$$

Por último, todas las medidas de similaridad mostradas verifican que, a mayor similaridad real entre parejas de usuarios, mayor será el valor de la métrica. En otras palabras, si un usuario x se parece lo mismo o más a un usuario y que a un usuario z se tiene que, ante la misma métrica de similaridad usada:

$$sim(x, y) \geq sim(x, z). \tag{16}$$

El segundo punto es una tarea trivial, en el que se trata de encontrar a los k usuarios que posean mayor similaridad con el usuario activo. Por tanto, determinamos el conjunto de k-vecinos del usuario activo u (al que representamos como K_u) verificando lo siguiente:

$$K_u \subset U \wedge \#K_u = k \wedge u \notin K_u. \tag{17}$$

$$\forall x \in K_u, \forall y \in (U - K_u) : sim(u, x) \geq sim(u, y) \tag{18}$$

Ejemplo de búsqueda de los k-vecinos

Para poder determinar el conjunto de k-vecinos de cada usuario, primero debemos determinar la similaridad existente entre cada pareja de usuarios. Para realizar este ejemplo vamos a utilizar como medida de similaridad, la medida más simple y sencilla de entender, que es la diferencia cuadrática media 10.

En la tabla 3 contiene todas las medidas de similaridad entre cada pareja de usuarios. Hay que recordar que la similaridad entre cada pareja de usuarios cumple la propiedad conmutativa (15) por lo que solo se ha rellenado la tabla en la parte triangular superior, es decir, si la tabla fuese una matriz, seria simétrica. Para ver como se ha calculado la

Sim (x,y)	**U1**	**U2**	**U3**	**U4**	**U5**	**U6**
U1	•	0.828	0.968	0.979	0.891	0.792
U2		•	0.396	0.863	0.688	0.363
U3			•	0.844	0.938	0.969
U4				•	0.896	0.766
U5					•	0.979
U6						•

Cuadro 3: Similaridad entre cada pareja de usuarios en el ejemplo de filtrado colaborativo.

similaridad entre usuarios, se pasa a mostrar el calculo de la similaridad entre los usuarios 1 y 2 utilizando la métrica de la diferencia cuadrática media (10) (MSD):

$$B_{U_1,U_2} = \{i \in I | r_{x,i} \neq \bullet \wedge r_{y,i} \neq \bullet\} = \{I_7, I_8, I_{10}, I_{11}\} \tag{19}$$

$$sim(U_1, U_2) = 1 - \frac{1}{4} \cdot \left(\left(\frac{3-3}{5-1}\right)^2 + \left(\frac{4-1}{5-1}\right)^2 + \left(\frac{4-5}{5-1}\right)^2 + \left(\frac{1-2}{5-1}\right)^2 \right) = 0{,}828 \tag{20}$$

Para determinar el conjunto de k-vecinos de cada uno de los usuarios del sistema, debemos encontrar los k usuarios que tengan una similaridad más alta con el usuario activo, tal y como se definió en (17) y (18). En la siguiente tabla 3 se muestran los k-vecinos de cada usuario para un valor de $k = 2$.

K=2	**U1**	**U2**	**U3**	**U4**	**U5**	**U6**
K_u	$\{U_4, U_3\}$	$\{U_4, U_1\}$	$\{U_6, U_5\}$	$\{U_1, U_5\}$	$\{U_6, U_3\}$	$\{U_5, U_3\}$

Cuadro 4: Conjunto de k-vecinos de cada usuario del ejemplo del filtrado colaborativo para k=2.

2.3.4. Obtención de las recomendaciones

Una vez que tenemos el conjunto de usuarios similares al usuario activo, tenemos que combinar las votaciones realizadas por los k-vecinos hacia aquellos items que no han sido votados por el usuario activo. De este modo podremos predecir la votación que el usuario activo realizará sobre aquellos items que no ha valorado y obtener una serie de recomendaciones para el mismo, que por lo general, serán aquellos items con la predicción más alta.

En este punto nos volvemos a encontrar con el problema de la dispersión que tienen las matrices de votaciones. Cuando queremos realizar la predicción de un item para el usuario activo, suele suceder que no todos los k-vecinos han valorado ese item, por lo que la predicción se realizará únicamente con aquellos vecinos que si han valorado el item. Puede ocurrir en el peor de los casos que ninguno de los k-vecinos haya votado el item, en cuyo caso no podemos obtener la predicción sobre el item deseado. Por tanto, definimos

el conjunto de vecinos del usuario u que han votado el item i:

$$G_{u,i} = \{n \in K_u | r_{n,i} \neq \bullet\} \tag{21}$$

Las formas más habituales de combinar las votaciones de los k-vecinos para obtener la predicción de un item i con respecto al usuario activo u, $p_{u,i}$ son [9][8][17]: la media (22), la media ponderada (23) y la media ponderada de la desviación con respecto a la media (24) (esta última es la que generalmente mejores resultados proporciona [17]). En la media, todos los usuarios aportan el mismo peso a la predicción, mientras que en la media ponderada y en la media ponderada de la desviación con respecto a la media, el peso de cada vecino se realiza en función de la similaridad con el usuario activo.

$$p_{u,i} = \frac{1}{G_{u,i}} \cdot \sum_{n \in G_{u,i}} r_{n,i} \text{ cuando } G_{u,i} \neq \emptyset \tag{22}$$

$$p_{u,i} = \frac{\sum_{n \in G_{u,i}} sim(u,n) \cdot r_{n,i}}{\sum_{n \in G_{u,i}} sim(u,n)} \text{ cuando } G_{u,i} \neq \emptyset \tag{23}$$

$$p_{u,i} = \overline{r_u} + \frac{\sum_{n \in G_{u,i}} sim(u,n) \cdot (r_{n,i} - \overline{r})}{\sum_{n \in G_{u,i}} sim(u,n)} \text{ cuando } G_{u,i} \neq \emptyset \tag{24}$$

En el caso extremo de que ninguno de los k-vecinos haya valorado el item que se intenta predecir, es decir, si $G_{u,i} = \emptyset$, la predicción no podrá ser calculada:

$$p_{u,i} = \bullet \text{ cuando } G_{u,i} \neq \emptyset \tag{25}$$

Para obtener las N mejores recomendaciones para cada usuario debemos predecir el voto del usuario sobre los item que no haya votado y encontrar los N que dispongan de una predicción más alta (generalmente es así, aunque puede haber determinados sistemas en los que se quieran recomendar otro tipo de items, como por ejemplo los más novedosos o los más valorados). Formalmente definimos X_u como el conjunto de posibles recomendaciones a realizar al usuario u y Z_u como el conjunto de las N recomendaciones realizadas al usuario u. Las siguientes expresiones deben verificarse asumiendo que cada usuario es capaz de recibir N recomendaciones:

$$X_u \subset I \wedge \forall i \in X_u, r_{u,i} = \bullet, p_{u,i} \neq \bullet \tag{26}$$

$$Z_u \subseteq X_u, \#Z_u = N, \forall i \in Z_u, \forall j \in (X_u - Z_u) : P_{u,i} \geq p_{u,j} \tag{27}$$

Ejemplo de obtención de las recomendaciones

Utilizados los k-vecinos calculados en la tabla 4 vamos a combinar las votaciones de los mismos para obtener la predicción de voto del usuario activo. Con el fin de simplificar los cálculos, se empleará la media aritmética 22 como mecanismo de combinación de las votaciones de los k-vecinos.

La tabla 6 contiene todas la predicciones de voto realizadas sobre todos los items del sistema para todos los usuarios del mismo. Las celdas de la tabla que están de color azul indican que el usuario no puntuó el item y las celdas con el símbolo • indica la imposibilidad de calcular la predicción de un item para un usuario, al no ser ese item votados por ninguno de sus vecinos.

$p_{u,i}$	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I9**	**I10**	**I11**	**I12**
U1	1	1	**4**	**3**	2	**1**	3	3	**4,5**	4	•	**1**
U2	**1**	**1**	4	3,5	**2**	1	3	3	**5**	4	•	1
U3	2	•	**5**	2	**1**	•	**1**	4	•	**1**	**2**	**1,5**
U4	**1,5**	1	4,5	3	**1,5**	**1**	2	**3**	•	4	**2**	1
U5	2	•	5	•	1	•	•	4	•	**1**	2	2
U6	**2**	•	5	2	1	•	•	4	•	•	**2**	2

Cuadro 5: Predicciones de voto realizadas mediante la votación de los k-vecinos en un ejemplo de filtrado colaborativo. Los items de color azul, son los items que no han sido votados por los usuarios.

A continuación detallamos cómo calcular la predicción de voto del *usuario* 4 con el *item* 1:

$$p_{U_4,I_1} = \frac{1}{2} \cdot (r_{U_1,I_1} + r_{U_5,I_1}) = \frac{1}{2} \cdot (1 + 2) = 1,5 \tag{28}$$

Ahora, a cada usuario le serán recomendados aquellos items que hayan obtenido una predicción más alta de entre aquellos que no hayan votado.

N=2	U1	U2	U3	U4	U5	U6
X_u	$\{I_3,I_4,I_6,I_9,I_{12}\}$	$\{I_1,I_2,I_5,I_9\}$	$\{I_3,I_5,I_7,I_{10},I_{11},I_{12}\}$	$\{I_1,I_5,I_6,I_8,I_{11}\}$	$\{I_8,I_{10}\}$	$\{I_1,I_{11}\}$
Z_u	$\{I_9,I_3\}$	$\{I_9,I_5\}$	$\{I_3,I_{11}\}$	$\{I_8,I_{11}\}$	$\{I_8,I_{10}\}$	$\{I_1,I_{11}\}$

Cuadro 6: Items susceptibles de ser recomendados (X_u) e items recomendados (Z_u) a los usuarios del filtrado colaborativo.

2.4. Evaluación de los sistemas de recomendación

2.4.1. Objetivos de la evaluación

Una vez que se han calculado las recomendaciones a los usuarios (o al usuario activo), debemos de ser capaces de cotejar la calidad de los resultados obtenidos. Con este propósito se han desarrollado una serie de medidas de calidad que nos permiten comprobar si los mecanismos que estamos empleando para realizar recomendaciones están funcionando de forma correcta.

Asimismo, las medidas de calidad tienen un peso muy importante en el área de la investigación de los sistemas de recomendación pues estas permiten comprobar si las variaciones incluidas en el sistema de recomendación, mejoran los resultados que otros investigadores hayan logrado. Como medidas de calidad más empleadas encontramos [11][15]:

1. Medidas de calidad de las predicciones:

   - *Accuracy*: indica el nivel medio de la calidad de las predicciones realizadas en el sistema de recomendación. Las medidas de *accuracy* más empleadas son el error medio absoluto y algunas de sus variantes, como el error cuadrático medio.
   - *Coverage*: indica el porcentaje de predicciones realizadas de entre todas las posibles. Potencialmente se puede realizar una predicción a todos los items que no haya votado el usuario activo; sin embargo, realmente sólo es posible realizar predicciones sobre un subconjunto de estos items. En filtrado colaborativo, por ejemplo, sólo se pueden predecir aquellos items valorados por al menos uno de los k-vecinos.

- *Porcentaje de predicciones perfectas, porcentaje de predicciones malas, etc.*

2. Medidas de calidad de las recomendaciones:

   - *Precision*: indica la proporción de items relevantes recomendados respecto del total recomendados (N).
   - *Recall*: indica la proporción de items relevantes recomendados respecto de todos los items relevantes disponibles.
   - *Especificidad, novedad, credibilidad, etc.*

En el ejemplo que se esta realizando en este trabajo, se emplearan como medidas de calidad el error medio absoluto para medir la precisión con la que estamos realizando las predicciones, el *coverage*, para medir la capacidad de recomendación del sistema y la *precision* y el *recall*, para obtener la calidad de las recomendaciones.

2.4.2. Error medio absoluto (MAE)

El error medio absoluto, también denominado **MAE** (*Mean Absolute Error*), calcula la distancia absoluta existente entre las predicciones realizadas y la votación real del usuario.

Definimos C_u como el conjunto de items valorados por el usuario u para los que ha sido posible obtener la predicción:

$$C_u = \{i \in I | r_{u,i} \neq \bullet \wedge p_{u,i} \neq \bullet\} \tag{29}$$

Calculamos el MAE de un usuario, MAE_u como el valor medio de la diferencia absoluta entre las predicciones realizadas y las votaciones reales del usuario:

$$MAE_u = \frac{1}{\#C_u} \cdot \sum_{i \in C_u} |r_{u,i} - p_{u,i}| \tag{30}$$

Calculamos el MAE como el promedio del error medio absoluto cometido en cada usuario:

$$MAE_u = \frac{1}{\#U} \cdot \sum_{u \in U} MAE_u \tag{31}$$

Ejemplo de cálculo de MAE

A partir de las predicciones calculadas en la tabla 6 y de las votaciones de los usuarios mostradas en la tabla 2, vamos a obtener el *MAE* del sistema. Como ya se menciono anteriormente, el *MAE* debe calcularse a partir de las predicciones realizadas sobre aquellos items que ya ha valorado el usuario.

Como ejemplo, vamos a detallar como calcular el *MAE* cometido por el *usuario 1*:

$$C_{u_1} = \{I_1, I_2, I_5, I_7, I_8, I_{10}\} \tag{32}$$

$$MAE_{U_1} = \frac{|1-1| + |2-1| + |2-2| + |3-3| + |4-3| + |4-4|}{6} = 0,33 \tag{33}$$

La tabla 7 muestra el MAE cometido al predecir las votaciones de todos los usuarios del sistema. En la tabla se muestra (para aquellos items que se les ha podido recomendar al usuario) el error absoluto que se ha cometido al predecir la votación del item correspondiente, $|r_{u,i} - p_{u,i}|$. La última columna de la tabla refleja el *MAE* cometido en cada usuario del sistemas MAE_u:

$\lvert r_{u,i} - p_{u,i}\rvert$	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I9**	**I10**	**I11**	**I12**	MAE_u
U1	0	1			0		0	1		0			**0,33**
U2			3	1,5		4	0	2		1		0	**1,64**
U3	1			0				1					**0,66**
U4		0	0,5	1			1			0		0	**0,42**
U5	0		0		0						0	1	**0,2**
U6			0	0	0			0				0	**0**

Cuadro 7: Error medio absoluto obtenido en el ejemplo de filtrado colaborativo.

Sabiendo el MAE_u obtenemos el *MAE* global del sistema:

$$MAE_u = \frac{1}{\#U} \cdot \sum_{u \in U} MAE_u = \frac{0,33 + 1,64 + 0,66 + 0,42 + 0,2 + 0}{6} = 0,54 \tag{34}$$

2.4.3. Capacidad de recomendación (coverage)

El *coverage* tiene como finalidad medir la capacidad de recomendación que tienen los k-vecinos de cada usuario, o dicho con otras palabras, indica el porcentaje de items que han podido ser predichos respecto del total de los que podían haberlo sido. El *coverage* de un usuario, $coverage_u$, se calcula tal y como se indica en la siguiente expresión:

$$coverage_u = 100 \cdot \frac{\#\{i \in I | r_{u,i} = \bullet \wedge p_{u,i} \neq \bullet\}}{\#\{i \in I | r_{u,i} = \bullet\}} \tag{35}$$

Por su parte, el *coverage* del sistema es computado como el valor promedio del *coverage* de cada usuario:

$$coverage = \frac{1}{\#U} \cdot \sum_{u \in U} coverage_u \tag{36}$$

Ejemplo de cálculo del *coverage*

Partiendo de las predicciones de la tabla 6 podemos obtener el *coverage* del sistema calculando la proporción de items para los que se ha podido realizar una predicción respecto del total de items no valorados por el usuario. Es importante resaltar que ahora únicamente nos fijamos en las predicciones de los items no votados por el usuario porque no necesitamos la votación real del usuario para calcular la medida de calidad.

Como ejemplo detallado vamos a mostrar cómo se puede calcular el *coverage* del *usuario* 5:

$$coverage_u = 100 \cdot \frac{\#\{I_8, I_{10}\}}{\#\{I_2, I_4, I_6, I_8, I_9, I_{10}\}} = 100 \cdot \frac{2}{6} = 33\,\% \tag{37}$$

La tabla 8 contiene el *coverage* obtenido a partir de las predicciones realizadas en la tabla 6. Las celdas que no contienen ningún dato corresponden a los items votados por el usuario, mientras que las celdas no sombreadas contienen las predicciones de voto de los items no valorados. La columna de la derecha contiene el *coverage* para cada usuario, $coverage_u$:

$p_{u,i}$	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I9**	**I10**	**I11**	**I12**	$Coverage_u$
U1			4	3		1			4,5			1	**100 %**
U2	1	1			2				5				**100 %**
U3		•	5		1		1			1	2	1,5	**85,71 %**
U4	1,5				1,5	1		3			2		**100 %**
U5		•		•		•			•	1			**33 %**
U6	2	•				•	•		•		2		**33 %**

Cuadro 8: *Coverage* obtenido en el ejemplo de filtrado colaborativo.

Sabiendo el $Coverage_u$ obtenemos el *Coverage* del sistema:

$$coverage = \frac{1}{\#U} \cdot \sum_{u \in U} coverage_u = \frac{100 + 100 + 85,71 + 100 + 33 + 33}{6} = 75,28\,\% \quad (38)$$

2.4.4. Calidad de las recomendaciones (precision y recall)

La calidad de las recomendaciones realizadas puede medirse mediante dos medidas que, habitualmente, aparecen unidas que son la *precision* y el *recall*. La *precision* busca comprobar cuantos de los items recomendados al usuario activo eran realmente relevantes para él. El *recall* se encarga de comprobar qué porcentaje de los items relevantes para un usuario le fueron recomendados. Para poder definir cuando un item es o no relevante se emplea un parámetro Θ que especifica la nota mínima que debe tener un item para ser considerado como relevante.

El cálculo de estas medidas requiere redefinir los criterios con los cuales se recomienda un item descrito en (17) y (18). El problema que tienen X_u y Z_u es que no permiten evaluar si un item es o no relevante para el usuario u porque no se conoce la votación real del usuario. Vamos a redefinirlos en (39) y (40), asumiendo que todos los usuarios pueden recibir hasta un total de N recomendaciones:

$$X_u^{'} \subset I \wedge \forall i \in X_u^{'}, r_{u,i} \neq \bullet, p_{u,i} \neq \bullet \quad (39)$$

$$Z_u^{'} \subseteq X_u^{'}, \#Z_u^{'} = N, \forall i \in Z_u^{'}, \forall j \in (X_u^{'} - Z_u^{'}) : p_{u,i} \geq p_{u,j} \quad (40)$$

Ahora es posible comprobar que items son relevantes y cuáles no para el usuario activo. Definimos los siguientes conjuntos para el usuario u:

$$Y_u = \left\{ i \in Z_u^{'} | r_{u,i} \geq \Theta \right\}, \text{ items recomendados y relevantes.} \tag{41}$$

$$N_u = \left\{ i \in Z_u^{'} | r_{u,i} < \Theta \right\}, \text{ items recomendados y no relevantes.} \tag{42}$$

$$O_u = \left\{ i \in (I_u - Z_u^{'}) | r_{u,i} \geq \Theta \right\}, \text{ items no recomendados y relevantes.} \tag{43}$$

Definimos la *precision* de un usuario u, y la denotamos como $precision_u$, como el porcentaje de items relevantes entre los recomendados:

$$precision_u = \frac{\#Y_u}{\#Y_u + \#N_u} \tag{44}$$

La *precision* del sistema puede calcularse como el valor promedio de la *precision* de cada usuario:

$$precision = \frac{1}{\#U} \cdot \sum_{u \in U} precision_u \tag{45}$$

Definimos el *recall* de un usuario u, y la denotamos por $recall_u$, como el porcentaje de items relevantes recomendados respecto del total de relevantes:

$$recall_u = \frac{\#Y_u}{\#Y_u + \#O_u} \tag{46}$$

El *recall* del sistema puede calcularse como el valor promedio del *recall* de cada usuario:

$$recall = \frac{1}{\#U} \cdot \sum_{u \in U} recall_u \tag{47}$$

Ejemplo de cálculo de la *precision* y el *recall*

Para poder calcular las medidas del precision y el recall debemos obtener los items que recomendaríamos de los que ya han sido votados. Para ello, haciendo uso de las predicciones mostradas en la tabla 6, la tabla 9 muestra los items que han sido valorados por los usuarios y para los que ha sido posible realizar una predicción, $X_u^{'}$, y los items que serian recomendados a cada usuario entre los citados anteriormente, $Z_u^{'}$.

N=2	U1	U2	U3
$X_u^{'}$	$\{I_1, I_2, I_5, I_7, I_8, I_{10}, I_{11}\}$	$\{I_3, I_4, I_6, I_7, I_8, I_{10}, I_{11}\}$	$\{I_1, I_4, I_6, I_8, I_9\}$
$Z_u^{'}$	$\{I_{10}, I_7 \vee I_8\}$	$\{I_3, I_{10}\}$	$\{I_8, I_1 \vee I_4\}$

N=2	U4	U5	U6
$X_u^{'}$	$\{I_2, I_3, I_4, I_7, I_9, I_{10}, I_{12}\}$	$\{I_1, I_3, I_5, I_7, I_{11}, I_{12}\}$	$\{I_3, I_4, I_5, I_8, I_{10}, I_{12}\}$
$Z_u^{'}$	$\{I_3, I_{10}\}$	$\{I_3, I_1 \vee I_{11} \vee I_{12}\}$	$\{I_3, I_8\}$

Cuadro 9: Items susceptibles de ser recomendados ($X_u^{'}$) e items recomendados ($Z_u^{'}$) entre aquellos valorados por el usuario en el ejemplo de filtrado colaborativo.

Para poder calcular las dos últimas medidas de calidad necesitamos definir el umbral a partir del cual consideramos un item como relevante. En este caso definiremos el umbral de relevancia como $\Theta = 4$. Aquellos items que hayan sido votados por el usuario con una nota igual o superior a ese umbral serán considerados relevantes, mientras que los que tengan una valoración estrictamente inferior a la indicada por el umbral serán considerados como no relevantes.

Vamos a ver en detalle como calcular la precision y el recall del usuario 1 a partir de sus votos (tabla 2) y de las predicciones obtenidas (tabla 6):

$$precision_{U_1} = \frac{\#\{I_{10}\}}{\#\{I_{10}\} + \#\{I_7\}} = \frac{1}{1+1} = 0,5 \tag{48}$$

$$recall_{U_1} = \frac{\#\{I_{10}\}}{\#\{I_{10}\} + \#\{I_8\}} = \frac{1}{1+1} = 0,5 \tag{49}$$

La tabla 10 permite calcular la *precision* del sistema de filtrado colaborativo. Las celdas de la tabla contienen los votos de los usuarios para los items del sistema. Aquellas celdas que aparecen con los votos de color azul son las de los items que han sido recomendados y han resultado ser relevantes, mientras que aquellas celdas que están en color rojo, son las de los items que han sido recomendados y que nos han resultado relevantes. La columna de la derecha muestra la *precision* de cada usuario, $precision_u$:

Sabiendo el $precision_u$, obtenemos la *precision* del sistema:

$$precision = \frac{1}{\#U} \cdot \sum_{u \in U} precision_u = \frac{0,5 + 0,5 + 0 + 1 + 0,5 + 1}{6} = 0,58 \tag{50}$$

$r_{u,i}$	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I9**	**I10**	**I11**	**I12**	$Precision_u$
U1	1	2	•	•	2	•	**3**	4	•	**4**	1	•	$\frac{1}{1+1} = 0,5$
U2	•	•	**1**	5	•	5	3	1	•	**5**	2	1	$\frac{1}{1+1} = 0,5$
U3	**1**	•	•	2	•	1	•	**3**	4	•	•	•	$\frac{0}{0+2} = 0$
U4	•	1	**4**	4	•	•	3	•	5	**4**	•	1	$\frac{2}{2+0} = 1$
U5	**2**	•	**5**	•	1	•	1	•	•	•	2	1	$\frac{1}{1+1} = 0,5$
U6	•	•	**5**	2	1	•	•	**4**	•	1	•	2	$\frac{2}{2+0} = 1$

Cuadro 10: Precision obtenida en el ejemplo de filtrado colaborativo

La tabla 11 permite calcular el *recall* del sistema de filtrado colaborativo. Las celdas de la tabla contienen los votos de los usuarios para los items del sistema. Aquellas celdas que aparecen de color azul son la de los items que han sido recomendados y han resultado ser relevantes, mientras que aquellas que aparecen de color rojo son los items que no han sido recomendados y que han resultado relevantes para el usuario. La columna de la derecha muestra el *recall* de cada usuario ($recall_u$):

$r_{u,i}$	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I9**	**I10**	**I11**	**I12**	$Recall_u$
U1	1	2	•	•	2	•	3	**4**	•	**4**	1	•	$\frac{1}{1+1} = 0,5$
U2	•	•	1	**5**	•	**5**	3	1	•	**5**	2	1	$\frac{1}{1+2} = 0,33$
U3	1	•	•	2	•	1	•	3	**4**	•	•	•	$\frac{0}{0+1} = 0$
U4	•	1	**4**	**4**	•	•	3	•	**5**	**4**	•	1	$\frac{2}{2+2} = 0,5$
U5	2	•	**5**	•	1	•	1	•	•	•	2	1	$\frac{1}{1+0} = 1$
U6	•	•	**5**	2	1	•	•	**4**	•	1	•	2	$\frac{2}{2+0} = 1$

Cuadro 11: Recall obtenido en el ejemplo de filtrado colaborativo

Sabiendo el $recall_u$, obtenemos el *recall* del sistema:

$$recall = \frac{1}{\#U} \cdot \sum_{u \in U} recall_u = \frac{0,5 + 0,33 + 0 + 0,5 + 1 + 1}{6} = 0,56 \tag{51}$$

3. Descomposición en Valores Singulares (SVD)

En esta sección se explicara y se pondrá un ejemplo de la Descomposición en valores singulares (en adelante SVD) explicando también todos los conceptos y propiedades matemáticas necesarias para su calculo.

3.1. Definición del SVD

La Descomposición en Valores Singulares (*en inglés Singular Value Descomposition, SVD*) [30] es una técnica de factorización de matrices que permite descomponer una matriz A en otras tres matrices U, S, y V de la siguiente manera:

$$SVD(A) = U \times S \times V^t \tag{52}$$

Es decir, que el producto matricial de la matriz U por S por V^t da como resultado la matriz inicial A.

Respecto a las dimensiones de las matrices tenemos que la matriz A va a tener unas dimensiones de $(n \times m)$ es decir n filas y m columnas. La matriz U va a tener dimensión $(n \times n)$, la matriz S tendrá dimensión $(n \times m)$ y por último la matriz V tendrá dimensión $(m \times m)$. Al ser V una matriz cuadrada las dimensiones de la matriz traspuesta serán las mismas que la matriz original. A continuación mostramos (53) como serian las dimensiones de las matrices para esta descomposición:

$$SVD(A)_{n\times m} = U_{n\times n} \times S_{n\times m} \times V^t_{m\times m} \tag{53}$$

Las matrices U y V han de cumplir las siguientes propiedades (54) (55) respectivamente:

$$U^t \times U = U \times U^t = I_{n\times n} \tag{54}$$

$$V^t \times V = V \times V^t = I_{m\times m} \tag{55}$$

Por otro lado la matriz S (56) ha de ser una matriz diagonal, la cual tendrá en su diagonal lo que se denominan *Valores Singulares*, y estos han de estar puestos en orden decreciente (y tendrán siempre valores mayores o iguales a cero), es decir que la matriz S tendrá la siguiente forma:

$$S = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_n \end{bmatrix}, \text{ donde } \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \tag{56}$$

Antes de pasar a explicar el calculo de las tres matrices (U, S y V), es necesario conocer la definición de lo que son los *autovalores*, *autovectores* y que es el *polinomio característico de una matriz* ya que son conceptos que hemos de manejar para la obtención de estas matrices. Las definiciones son la siguientes [3]:

Definición 1 *Sea $T : S \to V$ una transformación de S en V. Un escalar λ se denomina autovalor de T si existe un elemento no nulo x en S tal que $T(x) = \lambda x$. El elemento x se le llama autovector de T perteneciente a λ. El escalar λ se llama autovalor correspondiente a x.*

Definición 2 *Si A es una matriz cuadrada $n \times n$, el determinante $f(\lambda) = det(\lambda I - A)$ se denomina polinomio característico de A.*

Antes de pasar a explicar detalladamente como obtener estas 3 matrices, adelantamos que las matrices U y V van a estar compuestas por lo autovectores (puestos en las filas) obtenidos de las matrices cuadradas obtenidas tras multiplicar $(A \times A^t)$ y $(A^t \times A)$ respectivamente y la matriz S va a estar compuesta (su diagonal) por la raiz cuadrada de los autovalores asociados a los autovectores de la matrices U y V. Estos valores que tienen como resultado la raiz cuadrada de los autovalores y lo que se denomina como *Valores Singulares*.

Cálculo de la Matriz U

El primer paso que tenemos que dar para obtener la matriz U es multiplicar la matriz original A por su traspuesta para que nos de como resultado una matriz cuadrada y poder obtener los autovalores y autovectores. Hay que recordar que solo se pueden obtener autovalores y autovectores a partir de una matriz cuadrada. A continuación mostramos como obtenemos esta matriz cuadrada que llamaremos C y cuales son las dimensiones de cada matriz:

$$A_{n\times m} \times A^t_{m\times n} = C_{n\times n} \tag{57}$$

A partir de esta matriz cuadrada C obtengo sus autovalores (los λ_n) igualando su polinomio característico a cero y despejando λ:

$$det(\lambda I - C) = 0 \tag{58}$$

Es decir que si el determinante fuese de dimensión 2 tendríamos que obtener los autovales (λ_1 y λ_2) resolviendo la siguiente ecuación que seria de segundo grado en este ejemplo:

$$|C| = \begin{vmatrix} a-\lambda & b \\ c & d-\lambda \end{vmatrix} = (a-\lambda)\cdot(d-\lambda) - b\cdot c \tag{59}$$

Una vez que se tienen los autovalores, se puede pasar a calcular los autovectores asociados a cada autovalor. Para el cálculo estos autovalores ($\vec{v}$) se ha de cumplir la siguiente ecuación (60):

$$C\vec{v} = \lambda\vec{v} \tag{60}$$

Por tanto con cada autovalor (λ_n) obtendremos su autovector correspondiente ($\vec{v}$) resolviendo el sistema de ecuaciones que se de para cada autovalor; es decir, tal y como se muestra en el siguiente ejemplo para una matriz cuadrada C de dimensiones 2×2:

$$C\vec{v}_1 = \lambda\vec{v}_1; \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda_1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{61}$$

Es decir que el autovector $\vec{v}_1$ se obtendría resolviendo el siguiente sistema de ecuaciones:

$$\begin{aligned} c_{1,1}x_1 + c_{1,2}x_2 &= \lambda_1 x_1 \\ c_{2,1}x_1 + c_{2,2}x_2 &= \lambda_1 x_2 \end{aligned} \tag{62}$$

Llegados a este punto en el que tenemos calculados los autovales y autovectores de la matric cuadrada C, ya podemos generar la matriz U que pretendíamos calcular ya que esta esta compuesta en cada fila por los autovectores calculados, es decir que en la final 1 se tendrá el autovector $\vec{v}_1$ en la fila 2 el autovector $\vec{v}_2$ y así hasta completar la matriz con el autovector $\vec{v}_n$:

$$U = \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vdots \\ \vec{v}_n \end{bmatrix} = \begin{bmatrix} v_{1,1} & \cdots\cdots & v_{1,n} \\ v_{2,1} & \cdots\cdots & v_{2,n} \\ \vdots & \ddots & \vdots \\ v_{n,1} & \cdots\cdots & v_{n,n} \end{bmatrix} \tag{63}$$

Para la creación de esta matriz U no tendremos en cuenta los autovales, ya que estos los utilizaremos para generar la matriz de *Valores Singulares* S.

Cálculo de la Matriz V

Para obtener la matriz V hay que realizar los mismos pasos que para obtener la matriz U con la diferencia de que en este caso obtendremos los autovalores y autovectores con otra matriz cuadrada de dimensiones $(m \times m)$ en vez de dimensiones $(n \times n)$, lo cual quiere decir que es posible que el numero de autovalores y autovectores con los que trabajemos para el calculo de esta matriz V sera distinto que para el calculo de la matriz U. Esto sera así para el caso de que la matriz original A no sea cuadrada. La matriz cuadrada de la que obtendremos los autovalores y autovectores para calcular V la calculamos de la siguiente manera:

$$A^t_{m\times n} \times A_{n\times m} = C_{m\times m} \tag{64}$$

Una vez calculada la matriz V tendremos m autovalores, los cuales tendrán el mismo

valor que los autovalores obtenidos en el calculo de la matriz U. En el caso de que $n \neq m$ los autovales que no sean igual por la diferencia de dimensión tendrán valor cero, es decir que si $n = 2$ y $m = 3$, tendremos al menos $3 - 2 = 1$ autovalores con valor 0, siendo los otros 2 autovalores iguales.

Cálculo de la Matriz S

Una vez que se tienen los autovalores obtenidos en el cálculo de la matriz U y V, el calculo de la matriz S es casi inmediato, ya que en su diagonal va a estar formada por la raiz cuadrada de los autovalores.

La matriz S va a tener la misma dimensión que la matriz original A es decir dimensión $n \times m$ y como se ha dicho va a ser una matriz de ceros, salvo su diagonal que estará formada por la raiz cuadrada de los autovalores obtenidos en el calculo de U y V que además cada par de autovalores tendrán el mismo valor no nulo. A continuación ponemos un ejemplo aclaratorio:

Sea A la matriz original de dimensión 2×3 obtendremos como autovalores tras el cálculo de la matriz U λ_1 y λ_2 y como autovalores tras el cálculo de la matriz V λ_1, $\lambda_2 y \lambda_3$ siendo $\lambda_3 = 0$ necesariamente y teniendo el mismo valor λ_1 y λ_2, quedando la matriz S de la siguiente forma:

$$S = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \end{bmatrix} \tag{65}$$

Y cumpliendo además la propiedad antes descrita de que $\lambda_1 \geq \lambda_2$ y evidentemente la propiedad de $\lambda_1 \geq \lambda_2 \geq \lambda_3$ ya que $\lambda_3 = 0$.

Hasta ahora ya hemos visto de que forma calcular las tres matrices U, S y V pero hay que decir varias cosas al respecto sobre este calculo:

1. Para comprobar que el calculo de las matrices U y V es correcto, se puede demostrar que haciendo el producto escalar de una fila consigo misma da valor '1'.

2. Otra comprobación para comprobar que el calculo de las matrices U y V es correcto, se demuestra que haciendo el producto escalar de una fila cualquiera con otra y dando como resultado 0 al ser los autovectores ortonormales.

Por último hay que destacar una propiedad importantísima que tiene el SVD y que la hace especialmente interesante no solo para los sistemas de recomendación sino también para otras muchas áreas y es que reduciendo el numero de valores singulares de la matriz S a los primeros k valores, se obtiene una aproximación de la matriz original A, que permite reconstruirla a partir de las versiones reducidas de las matrices en las que se descompone cometiendo un cierto error, pero haciendo que las dimensiones de las matrices U, S y V sean menores es decir que se cumpliría lo siguiente si cogiésemos los k primeros valores de la diagonal de la matriz S:

$$A_{n\times m} \simeq U_{n\times k} \times S_{k\times k} \times V^t_{k\times m} \tag{66}$$

La explicación a esta propiedad nos la da el ***teorema de Eckart-Young*** que nos asegura que la mejor aproximación a la matriz original A, la obtenemos poniendo a '0'los n valores singulares más pequeños, por lo que poniendo esos valores a '0', podemos reducir las matrices U e I al número de valores singulares no nulos que tenga la matriz S.

Este método resulta en una transformación de los datos originales de alta dimensionalidad en una representación reducida de los mismo de dimensión K. Esta propiedad es muy importante ya que si se quiere hacer el SVD partiendo de una matriz original A de dimensiones (100,000 $\times$ 500,000) se tendrían que calcular (y en el caso de este trabajo fin de master guardar en alguna memoria del ordenador) tres matrices de dimensiones (100,000$\times$100,000), (100,000$\times$500,000) y (500,000$\times$500,000) que son las dimensiones que tendrían las matrices U, S y V respectivamente, mientras que si redujésemos las matrices U, S y V (por ejemplo) a las dimensiones (100,000 $\times$ 100), (100 $\times$ 100) y (100 $\times$ 500,000), tendríamos una aproximación muy buena a la matriz original A y reduciríamos muy considerablemente el tiempo de computo para el cálculo de las tres matrices a la par de que

se necesitaría menos memoria para guardar la información de estas tres matrices.

Sobre como calcular las matrices U, S y V y adaptarlas a los sistemas de recomendación se hablara en el siguiente capítulo al igual que de las propiedades de la dimensionalidad de las matrices que se acaba de explicar.

3.2. Ejemplo matemático

En este punto se va a poner un ejemplo práctico de como obtener las matrices U, S y V, es decir, como hacer el SVD a partir de la siguiente matriz A de dimensiones 2×3:

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 3 & 0 \end{bmatrix} \tag{67}$$

Cálculo de la Matriz U

El primer paso que debemos de dar es calcular la matriz U. Es indiferente empezar calculando la matriz U o la V, pero comenzaremos por la U con el fin de mantener el orden que se ha seguido a la hora de explicar este método. Lo primero que tenemos que hacer es calcular la matriz cuadrada (que llamaremos C_1) de la que obtendremos los autovalores y autovectores. Para ello hacemos el siguiente calculo (ver ecuación 57):

$$C_1 = A \times A^t = A = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 3 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 0 & 3 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 2 & 13 \end{bmatrix} \tag{68}$$

Una vez que tenemos la matriz cuadrada C_1, pasamos a calcular los autovalores igualando a cero el polinomio característico de la matriz C_1 (ver ecuación 58 y 59):

$$|C_1| = \begin{vmatrix} 2-\lambda & 2 \\ 2 & 13-\lambda \end{vmatrix} = (2-\lambda) \cdot (13-\lambda) - 2 \cdot 2 = \lambda^2 - 15\lambda + 22 \tag{69}$$

El siguiente paso es despejar λ para que nos de los autovalores. En este caso tenemos

que resolver una ecuación de segundo grado:

$$\lambda = \frac{15 \pm \sqrt{15^2 - 4 \cdot 1 \cdot 22}}{2 \cdot 1} \Rightarrow \lambda_1 = 13{,}35 \text{ y } \lambda_2 = 1,65 \tag{70}$$

Con este cálculo ya hemos obtenido los autovalores de la matriz C_1 que son $\lambda_1 = 13{,}35$ y $\lambda_2 = 1,65$.

El siguiente paso que tenemos que dar es el de calcular los autovectores asociados a cada uno de los autovalores obtenidos. Esto lo hacemos de la siguiente forma (ver ecuación 60):

$$C_1\vec{v}_1 = \lambda_1\vec{v}_1; \begin{bmatrix} 2 & 2 \\ 2 & 13 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 13{,}35 \begin{bmatrix} x \\ y \end{bmatrix} \tag{71}$$

$$C_1\vec{v}_2 = \lambda_2\vec{v}_2; \begin{bmatrix} 2 & 2 \\ 2 & 13 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 1{,}65 \begin{bmatrix} x \\ y \end{bmatrix} \tag{72}$$

El sistema de ecuaciones resultante de la ecuación (71) es el siguiente:

$$\begin{aligned} 2x + 2y &= 13{,}35x \\ 2 + 13y &= 13{,}35y \end{aligned} \tag{73}$$

Este sistema de ecuaciones lo resolvemos de la siguiente manera:

$$\begin{aligned} -11{,}35x + 2y &= 0; \\ \text{si } x &= 1 \\ y &= \tfrac{11{,}35}{2} = 5{,}675 \end{aligned} \tag{74}$$

Por tanto tenemos que $x = 1$ e $y = 5{,}675$ pero estos valores no forman todavía el autovector ya que el autovector tiene que tener norma '1', por lo tanto hay que normalizarlo:

$$|\vec{v}_1| = \sqrt{1^2 + 5{,}675^2} = 5{,}848 \tag{75}$$

Por tanto tengo que el primer autovector va a ser el siguiente:

$$\vec{v}_1 = \begin{bmatrix} \frac{1}{5{,}848} & \frac{5{,}675}{5{,}848} \end{bmatrix} = \begin{bmatrix} 0{,}17 & 0{,}98 \end{bmatrix} \tag{76}$$

Para obtener el segundo autovector tengo que realizar los mismo pasos pero en este caso tengo que resolver el siguiente sistema de ecuaciones:

$$\begin{aligned} 2x + 2y &= 1{,}65x \\ 2 + 13y &= 1{,}65y \end{aligned} \tag{77}$$

Resolviendo este sistema de ecuaciones obtengo el segundo autovector:

$$\vec{v}_2 = \begin{bmatrix} 0{,}98 & -0{,}17 \end{bmatrix} \tag{78}$$

Una vez que tengo estos todos los autovectores (en este caso 2) ya puedo construir la matriz U que estará compuesta por los dos autovectores de la siguiente forma:

$$U = \begin{bmatrix} 0{,}17 & 0{,}98 \\ 0{,}98 & -0{,}17 \end{bmatrix} \tag{79}$$

Para comprobar que hemos calculado bien esta matriz podemos comprobar que cumple las propiedades descritas anteriormente, que decían se si un autovector lo multiplicamos por si mismo (producto escalar) da como resultado 1 (80) y si multiplicamos un autovector con otro da como resultado 0 (81):

$$\vec{v}_1 \cdot \vec{v}_1 = \begin{pmatrix} 0{,}17 & 0{,}98 \end{pmatrix} \cdot \begin{pmatrix} 0{,}17 \\ 0{,}98 \end{pmatrix} = 0{,}17 \cdot 0{,}17 + 0{,}98 \cdot 0{,}98 = 1 \tag{80}$$

$$\vec{v}_1 \cdot \vec{v}_2 = \begin{pmatrix} 0{,}17 & 0{,}98 \end{pmatrix} \cdot \begin{pmatrix} 0{,}98 \\ -0{,}17 \end{pmatrix} = 0{,}17 \cdot 0{,}98 + 0{,}98 \cdot (-0{,}17) = 0 \tag{81}$$

Cálculo de la Matriz V

Para el cálculo de la matriz V realizaremos los mismos pasos que con la matriz U, pero en este caso obtendremos los autovalores y autovectores de la matriz cuadrada resultante

de multiplicar $A^t \times A$ (ver ecuación 64):

$$C_2 = A^t \times A = \begin{bmatrix} 1 & 2 \\ 0 & 3 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \\ 2 & 3 & 0 \end{bmatrix} = \begin{bmatrix} 5 & 6 & 1 \\ 6 & 9 & 0 \\ 1 & 0 & 1 \end{bmatrix} \tag{82}$$

EL siguiente paso es el de calcular los autovalores igualando a cero en polinomio característico de la matriz C_2:

$$|C_2| = \begin{vmatrix} 5-\lambda & 6 & 1 \\ 6 & 9-\lambda & 0 \\ 1 & 0 & 1-\lambda \end{vmatrix} = -\lambda^3 + 15\lambda^2 - 22\lambda \tag{83}$$

Para obtener los autovalores tenemos que despejar λ en esta ecuación de tercer grado. Como ya se comento los autovalores deben ser los mismos que los obtenidos en el calculo de la matriz U y la diferencia de autovalores tendrán valor 0, así que es fácil comprobar que para los valores de $\lambda_1 = 13{,}35$, $\lambda_2 = 1{,}65$ y $\lambda_3 = 0$ se cumple la siguiente igualdad y por lo tanto son estos los autovalores de C_2:

$$-\lambda^3 + 15\lambda^2 - 22\lambda = 0 \tag{84}$$

Para el cálculo de los autovectores, tendríamos que resolver los siguientes sistemas de ecuaciones (85), (86) y (87):

$$C_2\vec{v}_1 = \lambda_1\vec{v}_1; \begin{bmatrix} 5 & 6 & 1 \\ 6 & 9 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 13{,}35 \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{85}$$

$$C_2\vec{v}_2 = \lambda_2\vec{v}_2; \begin{bmatrix} 5 & 6 & 1 \\ 6 & 9 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 1{,}65 \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{86}$$

$$C_2\vec{v}_3 = \lambda_3\vec{v}_3; \begin{bmatrix} 5 & 6 & 1 \\ 6 & 9 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 0 \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{87}$$

Resolviendo estos sistemas de ecuaciones obtenemos los siguientes autovectores (88), (89) y (91):

$$\vec{v}_1 = \begin{bmatrix} 0{,}58 & 0{,}49 & -0{,}64 \end{bmatrix} \tag{88}$$

$$\vec{v}_2 = \begin{bmatrix} 0{,}81 & -0{,}41 & -0{,}43 \end{bmatrix} \tag{89}$$

$$\vec{v}_3 = \begin{bmatrix} 0{,}05 & 0{,}77 & 0{,}64 \end{bmatrix} \tag{90}$$

Con el cálculo de estos autovectores ya podemos obtener la matriz V que mostramos a continuación:

$$V = \begin{bmatrix} 0{,}58 & 0{,}49 & -0{,}64 \\ 0{,}81 & -0{,}41 & -0{,}43 \\ 0{,}05 & 0{,}77 & 0{,}64 \end{bmatrix} \tag{91}$$

Al igual que en el cálculo de la matriz U, podemos comprobar que se ha calculado correctamente haciendo el producto escalara de cada una de las filas consigo misma y el producto escalar de una fila con otra y nos deberá dar como resultados 1 y 0 respectivamente.

Cálculo de la Matriz S

El último paso es calcular la matriz S y este es el cálculo más sencillo ya que prácticamente lo tenemos hecho. Lo primero que hay que ver es la dimensión que tiene la matriz original A que es de (2×3) para construir la matriz S con la misma dimensión, por tanto la matriz S también tendrá dimensión (2×3).

El siguiente paso es poner en su diagonal, la raíz cuadrada de los autovalores calculados en el cálculo de la matriz U y V que serán los mismos salvo lo autovalores que han dado valor cero, es decir que la matriz S tendrá la siguiente forma:

$$S = \begin{bmatrix} \sqrt{\lambda_1} & 0 & 0 \\ 0 & \sqrt{\lambda_2} & 0 \end{bmatrix} = \begin{bmatrix} \sqrt{13{,}35} & 0 & 0 \\ 0 & \sqrt{1{,}65} & 0 \end{bmatrix} \tag{92}$$

Por tanto concluimos que la matriz S tendrá el siguiente valor (93):

$$S = \begin{bmatrix} 3{,}65 & 0 & 0 \\ 0 & 1{,}28 & 0 \end{bmatrix} \tag{93}$$

Una vez que se tiene hecho el cálculo de las tres matrices U, S y V es muy fácil comprobar que el producto matricial de estas tres matrices (haciendo el producto matricial con la matriz V traspuesta) es igual a la matriz A original:

$$\begin{bmatrix} 1 & 0 & 1 \\ 2 & 3 & 0 \end{bmatrix} = \begin{bmatrix} 0{,}17 & 0{,}98 \\ 0{,}98 & -0{,}17 \end{bmatrix} \times \begin{bmatrix} 3{,}65 & 0 & 0 \\ 0 & 1{,}28 & 0 \end{bmatrix} \times \begin{bmatrix} 0{,}58 & 0{,}81 & 0{,}05 \\ 0{,}49 & -0{,}41 & 0{,}77 \\ -0{,}64 & 0{,}43 & 0{,}64 \end{bmatrix} \tag{94}$$

4. Sistemas de Recomendación con SVD

El objetivo principal de los sistemas de recomendación es sugerir nuevos *items* a los *usuarios* y la técnica del SVD aplicada a los sistemas de recomendación basados en filtrado colaborativo realiza muy buenas recomendaciones, pero tiene el problema de que el tiempo que tarda en realizar las recomendaciones es muy elevado en comparación con las métricas utilizadas para el filtrado colaborativo basado en memoria, lo cual supone un problema aplicar esta técnica a nivel comercial ya que los dos principales objetivos de los sistemas de recomendación son recomendar bien y hacerlo rápido.

Como ya se vio en el capítulo 3 la técnica del SVD descompone una matriz A en tres matrices U, S y V cuyo producto matricial ($U \cdot S \cdot V^t$) es la propia matriz A. Estas tres matrices que obtenemos van a tener un significado, el cual nos permitirá realizar predicciones sobre *items* que el *usuario* (dentro del filtrado colaborativo) no ha valorado. En este capítulo 4 vamos a ver y a explicar el significado de estas tres matrices, las ventajas y problemas que tiene el SVD en el filtrado colaborativo y por último los algoritmos y propuestas para el SVD en el filtrado colaborativo.

4.1. Relación entre SVD y Sistemas de Recomendación basados en Filtrado Colaborativo

Dentro de los Sistemas de Recomendación basados en Filtrado Colaborativo se tiene una matriz de votos, que contiene las votaciones que realizan los *usuarios* sobre los *items*. Si consideramos la matriz de votos como la matriz A (*usuarios* x *items*) sobre la que realizamos el SVD, obtenemos las 3 matrices U, S y V con sus dimensiones correspondientes. Si reducimos las dimensiones de la matriz S a k dimensiones, tendremos que la matriz U tendrá dimensiones de (*usuarios* x k) y la matriz V tendrá dimensiones de (*items* x k). En este punto se tiene que la matriz U va a tener $k - factores$ que representarán a cada unos de los *usuarios* y la matriz V va a tener los mismos $k - factores$ que representaran a cada uno de los *items*.

En resumen, lo que se pretende al realizar el SVD sobre la matriz de votos, es obtener una serie de $factores$ que caractericen a cada uno de los *usuarios* y de los *items* del sistema, por tanto, y a nivel conceptual para el caso del filtrado colaborativo, podemos simplificar el proceso del SVD, obteniendo solamente las matrices de factores de los *usuarios* e *items*. Esto es posible ya que si la matriz S la descomponemos en dos matrices iguales (haciendo la raíz cuadrada de cada valor singular) y multiplicamos cada una de ellas a las matrices U y V obtendremos dos matrices $Ufac$ y $Ifac$ con los factores característicos de los *usuarios* e *items*. Esta simplificación la mostramos a continuación:

Descompongo la matriz de valores singulares en dos matrices iguales:

$$S_{kxk} = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_k \end{bmatrix} = \begin{bmatrix} \sqrt{\lambda_1} & & \\ & \ddots & \\ & & \sqrt{\lambda_k} \end{bmatrix} \cdot \begin{bmatrix} \sqrt{\lambda_1} & & \\ & \ddots & \\ & & \sqrt{\lambda_k} \end{bmatrix} \tag{95}$$

Veo que el producto de estas cuatro matrices me sigue dando la mejor aproximación a la matriz original, ya que es lo mismo que tener las tres matrices U, S y V:

$$\begin{bmatrix} \\ Votos_{nxm} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ U_{nxk} \\ \\ \end{bmatrix} \cdot \begin{bmatrix} \sqrt{\lambda_1} & & \\ & \ddots & \\ & & \sqrt{\lambda_k} \end{bmatrix} \cdot \begin{bmatrix} \sqrt{\lambda_1} & & \\ & \ddots & \\ & & \sqrt{\lambda_k} \end{bmatrix} \cdot \begin{bmatrix} \\ V_{kxm} \\ \\ \end{bmatrix} \tag{96}$$

Obtengo la nueva matriz de factores de los *usuarios*:

$$\begin{bmatrix} \\ Ufac \\ \\ \end{bmatrix} = \begin{bmatrix} \\ U_{nxk} \\ \\ \end{bmatrix} \cdot \begin{bmatrix} \sqrt{\lambda_1} & & \\ & \ddots & \\ & & \sqrt{\lambda_k} \end{bmatrix} \tag{97}$$

Obtengo la nueva matriz de factores de los *items*:

$$\begin{bmatrix} \\ Ifac \\ \\ \end{bmatrix}^t = \begin{bmatrix} \sqrt{\lambda_1} & & \\ & \ddots & \\ & & \sqrt{\lambda_k} \end{bmatrix} \cdot \begin{bmatrix} \\ V_{kxm} \\ \\ \end{bmatrix} \tag{98}$$

Se puede demostrar que este producto matricial es el mismo que el de ($U_{nxk} \cdot S_{kxk} \cdot V^t_{kxm}$):

$$\begin{bmatrix} \\ Votos_{nxm} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ Ufac \\ \\ \end{bmatrix} \cdot \begin{bmatrix} \\ Ifac \\ \\ \end{bmatrix}^t \tag{99}$$

4.1.1. Significado oculto de factores

Obtenidas las matrices de factores de *usuarios* e *items*, bien sean las matrices U y V del SVD original o las matrices que hemos denominado $Ufac$ e $Ifac$ (tras realizar la abstracción del SVD al filtrado colaborativo) es lógico preguntarse cual es el significado de los factores que hemos obtenido. Pues bien, no existe ninguna explicación sobre el significado de estos factores que nos da el método SVD, solo podemos explicar que este método caracteriza de alguna forma a los *usuarios* e *items* y esta caracterización nos permite predecir de forma muy buena los *items* que no han sido votados por los *usuarios*. Es por este motivo por el que se dice del método del SVD, que obtiene una serie de ***factores latentes*** [33] característicos para los *usuarios* e *items*.

Si bien la abstracción de estos *factores latentes* es cierta, se ha de decir que dependiendo del sistema de recomendación que tengamos y observando las características que tienen los *usuarios* e *items* del sistema, se puede observar en algunos casos y viendo las cosas con mucho detalle, que un cierto factor puede representar alguna característica real de los *usuarios* e *items*, como por ejemplo la edad, el sexo, un tag, etc. aunque el propio método SVD no te dice que característica esta asociado a cada factor.

4.1.2. Significado de los factores asociados a los Usuarios e Items

Como ya se ha dicho en el apartado anterior, el método SVD no te dice lo que representa cada factor, pero una vez que se han obtenido estos factores, podemos ver y asociar el significado de estos factores.

Este proceso no es obvio y en la mayoría de los casos es imposible hacerlo, pero vamos a ver un ejemplo para mostrar cual es el significado y la lógica de los factores latentes que se obtienen con el SVD.

Supongamos un Sistema de Recomendación de películas en la que tenemos de 5 *películas* (*items*) y 4 *usuarios*, tal y como se muestra en la siguiente figura 2. Tenemos la matriz de votos de los 4 usuarios sobre las películas de Terminator, Resacon en las Vegas, Titanic, Toy Story y El Padrino, es decir que tenemos películas de géneros muy diferentes (acción, comedia, romance, etc.). Con la matriz de votos, realizamos el SVD y obtenemos los siguientes tres factores para los *usuarios* e *items* que se muestran en la siguiente figura:

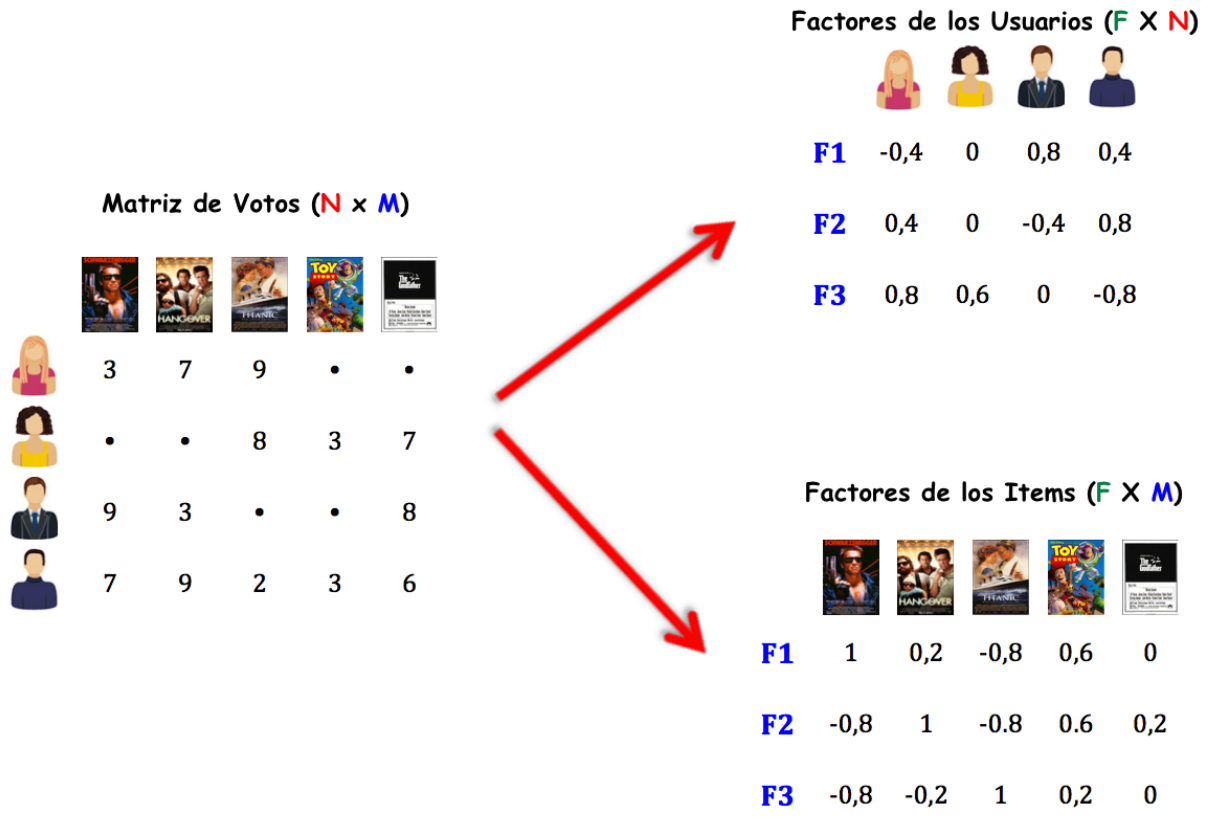

Figura 2: Ejemplo de factores de los *usuarios* e *items* obtenidos con el método SVD

Si observamos bien las votaciones que han realizado los *usuarios* y los valores de los factores de los *usuarios* y de los *items*, podemos sacar en conclusión que el factor $F1$ esta asociado al género cinematográfico de la *Acción*, el factor $F2$ esta asociado a la *Comedia* y el factor $F3$ esta asociado al género *Romántico*. Si vemos los valores de estos factores podemos ver el gusto de cada usuario sobre ese género y la pertenencia de cada

película a ese género. Por ejemplo podemos ver en la matriz de factores de las películas como a la película *Terminator* se le asocia el máximo valor para el factor de la *Acción* ya que es la película de más *Acción* de todas y lo mismo ocurre con *Resacón en la Vegas* y *Titanic* en los géneros de *Comedia* y *Romántico* respectivamente. Sobre los usuarios también podemos ver ese comportamiento y ver como por ejemplo el *tercer usuario* que ha votado la película de *Terminator* con valor 9 tiene asociado un valor muy alto sobre el factor que determina lo que le gusta a un usuario las películas de *Acción*.

Con los datos que nos dan estos factores podemos por ejemplo clasificar a los usuarios e items por su grado de pertenencia a un determinado género o a un determinado factor, como por ejemplo al del género de la acción tal y como mostramos en la siguiente figura:

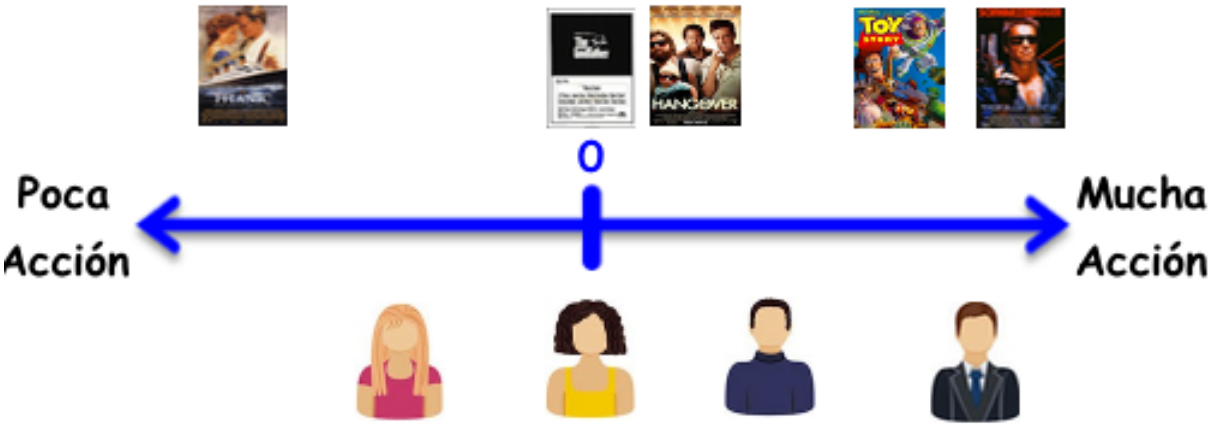

Figura 3: Ejemplo de clasificación de *usuarios* e *items* en función del valor de su factor.

Este es un posible ejemplo del significado que pueden tener los factores de los *usuarios* e *items* para un caso muy concreto de un sistema de recomendación de películas basado en filtrado colaborativo. Este ejemplo que se ha mostrado no es un ejemplo que matemáticamente sea correcto ya que si se multiplican las matrices de factores de *usuarios* y *películas* no va a dar como resultado la matriz de votos, pero se ha realizado así con el fin de poder mostrar que los factores obtenidos con el SVD pueden tener un significado si se conocen las características de los *usuarios* y de los *items*, en este caso las características de las *películas*.

Con la obtención de estos factores, es posible también ver la similaridad que hay entre

usuarios e *items* calculando las distancias que hay entre ellos a partir se sus factores y poder aplicar alguna métrica de similaridad de las del filtrado colaborativo basado en memoria, para obtener los k-vecinos de un *usuario* dado. Esto es posible ya que si dos *usuarios* o *items* son muy parecidos tendrán también factores muy similares.

4.1.3. Significado de los Valores Singulares

Como ya se vio en el tema 3, la matriz S es una matriz diagonal, que esta compuesta por la raíz cuadrada de los autovalores (ordenados de mayor a menor) asociados a cada uno de los autovectores que forman las filas de las matrices U y V de los factores de los *usuarios* e *items*.

El sentido que tienen estos valores singulares no es ni más ni menos que el de asignarle una determinada importancia a cada uno de los factores de los *usuarios* e *items* y más concretamente esa importancia va a ser decreciente al igual que los valores singulares, es decir; que el factor 1 va a ser más importante o igual de importante que el factor 2, el 2 más importante o igual que el factor 3 y así sucesivamente. Esto es debido a la posición y el valor que tienen cada uno de los valores singulares en la matriz S y se puede comprobar fácilmente como si descomponemos las matriz S en dos matrices iguales y las multiplicamos cada una de ellas por las matrices U y V tal y como se muestra en (96), se podrá ver que los valores de los factores de las matrices $Ufac$ e $Ifac$ van teniendo valores más pequeños según vayamos aumentando el número de factores.

4.2. Problema del enfoque analítico del SVD en los Sistemas de Recomendación

Los orígenes de aplicar la técnica del SVD a los sistemas de recomendación basados en filtrado colaborativo, viene a partir de las técnica del LSI (Latent Sematic Indexing) que se aplicaba en los sistemas de recomendación basados en contenido (ver capítulo 2) que consistía en aplicar la técnica matemática del SVD a una matriz que contenía el numero

de apariciones de ciertas *palabras* en unos determinados *items*. Está técnica se aplicaba a matrices (la matriz original A) que estaban en su mayoría compuesta por ceros y esos valores con valor 0 tenían el significado de que una determinada palabra no aparecía en ese *item*; por tanto, aplicar la técnica del SVD (o LSI) al filtrado basado en contenido funcionaba muy bien.

Dentro del filtrado colaborativo se presentan dos problemas a la hora de aplicar el SVD. El primero es de que forma se tratan aquellos *items* que el *usuario* no ha valorado, porque si a ese item se le valora con valor 0 la predicción sera de un valor muy próximo a 0 y ese no es el resultado que se espera, ya que lo que se pretende que nos diga la técnica del SVD es la *nota* que le va a predecir al *usuario* sobre el *item*. El segundo problema que se plantea es la dispersión que tienen las matrices de votos; es decir, que el usuario solo vota un porcentaje muy pequeño de los items que hay en el sistema, por tanto hay que trabajar con matrices de votos muy dispersas.

4.2.1. Matriz no Rellena

Planteemos el siguiente problema. Tenemos la siguiente matriz de votos en la que es muy intuitivo saber cual va a ser la predicción de los *items I2, I3* y *I5* sobre el *usuario U3*:

	I1	**I2**	**I3**	**I4**	**I5**
U1	1	3	5	3	1
U2	1	4	4	3	2
U3	1	•	•	3	•

Cuadro 12: Ejemplo de votaciones para plantear el problema de las matrices no rellenas

Si suponemos que los *usuarios* $U1$ y $U2$ son vecinos de $U3$ es muy fácil predecir que al *usuario* $U3$ votará los *items I2, I3* y *I5* con las notas *3.5, 4.5* y *1.5* respectivamente.

El problema que se nos plantea es como predecir los *items I2, I3* y *I5* al *usuario U3*. Si hacemos el SVD poniendo esos items a valor cero y reducimos las matrices a n factores,

tendremos como predicción de esos items a un valor muy cercano a cero (ejemplo con 2 factores):

$$\begin{bmatrix} 1 & 3 & 5 & 3 & 1 \\ 1 & 4 & 4 & 3 & 2 \\ 1 & 0 & 0 & 3 & 0 \end{bmatrix} \approx \begin{bmatrix} -0{,}69 & 0{,}11 \\ -0{,}70 & 0{,}13 \\ -0{,}17 & -0{,}98 \end{bmatrix} \cdot \begin{bmatrix} 9{,}58 & 0 \\ 0 & 2{,}75 \end{bmatrix} \cdot \begin{bmatrix} -0{,}16 & -0{,}51 & -0{,}65 & -0{,}49 & -0{,}22 \\ -0{,}27 & 0{,}31 & 0{,}38 & -0{,}81 & 0{,}13 \end{bmatrix} \tag{100}$$

Si se hace el producto matricial de estas tres matrices se obversa que efectivamente los items que hemos puesto con valor 0 nos lo predice con un 0 y queríamos que nos lo predijese con otros valores:

$$U_{3x2} \cdot S_{2x2} \cdot V_{2x5} = \begin{bmatrix} 1 & 3{,}47 & 4{,}46 & 3 & 1{,}49 \\ 0{,}99 & 3{,}53 & 4{,}53 & 2{,}99 & 1{,}52 \\ 0{,}99 & -0{,}01 & 0{,}01 & 2{,}99 & -0{,}01 \end{bmatrix} \tag{101}$$

Otra de las propuestas que había, era pasar esos valores con el valor de la media del *item* para que pudiese predecir algo mejor. En este ejemplo funcionaria a la perfección ya que justo coincide que la predicción es la media de los *items*, pero si hubiese muchísimos mas *usuarios* e *items* no funcionaria bien.

Este problema de las matrices no rellenas tiene solución (propuesta por Simon Funck [30]), la cual se explicará en el capítulo 4.3.1.

4.2.2. Soluciones al problema de la dispersión

El segundo problema que tiene el SVD aplicado al filtrado colaborativo es la gran dispersión (*items* no votados) que tienen las matrices de votos. Esto se debe a que en un sistema de recomendación real (o comercial) es muy difícil que todos los *usuarios* voten la gran mayoría de los *items*. Esto suele ser mas bien al revés, ya que los *usuarios* votan un pequeño subconjunto de *items*.

Prueba de esta dispersión, están en la bases de datos MovieLens y Netflix (que son las bases de datos más utilizadas para la investigación en Sistemas de Recomendación), en las cuales los usuarios votan entorno al 10-20 % de todos los *items* posibles, por lo tanto se ha de predecir el 80-90 % de los *items* restantes.

Este problema de la dispersión también es tenido en cuenta en los algoritmos propuestos por Simon Funck (capítulo 4.3.1) para los sistemas de recomendación basados en filtrado colaborativo, tratando solamente los votos realizados por los *usuarios* y no teniendo en cuenta los *items* no votados.

4.3. Propuestas

Como ya se vio en puntos anteriores los principales problemas que tenemos a la hora de aplicar el SVD al filtrado colaborativo son la dispersión de las matrices de votos y el como tratar los *items* no votados. Por otro lado hemos visto la abstracción del SVD aplicado al filtrado colaborativo, sacando en conclusión que lo que nos interesa es obtener una serie de factores que caractericen a los *usuarios* y a los *items* ($Ufac$ e $Ifac$ visto en el capítulo 4.1).

Propuesta esta abstracción para el SVD, lo que tenemos que buscar son los factores de los *usuarios* e *items* a partir de la matriz de votos, planteándose entonces un problema de regresión. El problema seria encontrar todos los $u_{n,k}$ e $i_{m,k}$ (el valor de los factores) que trate de satisfacer lo máximo posible la siguiente igualdad (siendo los $r_{n,m}$ valores conocidos):

$$\begin{bmatrix} r_{1,1} & \cdots & r_{1,m} \\ \vdots & \ddots & \vdots \\ r_{1,n} & \cdots & r_{n,m} \end{bmatrix} = \begin{bmatrix} u_{1,1} & \cdots & u_{1,k} \\ \vdots & \ddots & \vdots \\ u_{n,1} & \cdots & u_{n,k} \end{bmatrix} \cdot \begin{bmatrix} i_{1,1} & \cdots & i_{1,m} \\ \vdots & \ddots & \vdots \\ i_{k,1} & \cdots & i_{k,m} \end{bmatrix} \tag{102}$$

Es decir que para encontrar estos factores habría que resolver un sistema de ecuaciones con tantas incógnitas como numero de factores haya (los $u_{n,k}$ e $i_{m,k}$), es decir habría que

resolver un sistema de ecuaciones como el siguiente:

$$\begin{aligned} r_{1,1} &= u_{1,1} \cdot i_{1,1} + \cdots + u_{1,k} \cdot i_{k,1} \\ &\vdots \\ r_{n,m} &= u_{n,1} \cdot i_{1,m} + \cdots + u_{n,k} \cdot i_{k,m} \end{aligned} \tag{103}$$

Una vez vista la abstracción y viendo cual es el problema que tenemos que resolver (encontrar los $u_{n,k}$ e $i_{m,k}$), no tenemos todavía la solución a los problemas de la dispersión y de las matrices de votos no rellenas. La solución a estos problemas se verán en el siguiente punto (4.3.1).

4.3.1. Simon Funck

La solución a todos los problemas planteados hasta el momentos (matrices no rellenas y dispersión), la dio Simon Funck [30], que propuso no tener en cuenta en la matriz de votos, aquellos votos que no se han realizados para calcular los factores de los *usuarios* y de los *items*. Analíticamente se obtiene que un voto dado ha de ser el producto escalar del *vector de factores del usuario* que realiza el voto sobre el *item* y de *los factores del item* sobre el que se realiza el voto. Si consideramos q_i al vector que representa los factores de un *item* y p_u al vector que representa los factores del *usuario* se tendría que cumplir lo siguiente:

$$r_{u,i} = q_i^t \cdot p_u \tag{104}$$

Esto es lo mismo que se plantea en el sistema de ecuaciones propuesto en (103), pero Simon Funck, plantea para el SVD que solo se ajusten los factores para aquellos *usuarios* e *items* que hayan emitido un voto, es decir que las ecuaciones del sistema de ecuaciones se reducen considerablemente y de esta forma se soluciona el problema de las matrices no rellenas y de la dispersión. Para ver este concepto vamos a poner un ejemplo de la diferencia que hay entre el LSI y el SVD.

Supongamos que tenemos la siguiente matriz de votos A (considerando el valor cero

como un voto no emitido) y las matrices de factores de *usuarios* e *items*, de las que queremos obtener 2 factores:

$$A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} u_{1,1} & u_{1,2} \\ u_{2,1} & u_{2,2} \end{bmatrix} \cdot \begin{bmatrix} i_{1,1} & i_{1,2} \\ i_{2,1} & i_{2,2} \end{bmatrix} \tag{105}$$

Para el caso del LSI tendríamos que resolver un sistema de 4 ecuaciones que sería el siguiente para obtener los factores:

$$\begin{aligned} 3 &= u_{1,1} \cdot i_{1,1} + u_{1,2} \cdot i_{2,1} \\ 0 &= u_{1,1} \cdot i_{1,2} + u_{1,2} \cdot i_{2,2} \\ 0 &= u_{2,1} \cdot i_{1,1} + u_{2,2} \cdot i_{2,1} \\ 2 &= u_{2,1} \cdot i_{1,2} + u_{2,2} \cdot i_{2,2} \end{aligned} \tag{106}$$

Para el caso del SVD, no se tendrían en cuenta las ecuaciones que se igualan a 0 y por tanto en este caso solo habría que resolver el siguiente sistema con menos ecuaciones:

$$\begin{aligned} 3 &= u_{1,1} \cdot i_{1,1} + u_{1,2} \cdot i_{2,1} \\ 2 &= u_{2,1} \cdot i_{1,2} + u_{2,2} \cdot i_{2,2} \end{aligned} \tag{107}$$

Si observamos bien el ejemplo (107) vemos que resolviendo el sistema de 2 ecuaciones con 8 incógnitas vamos a obtener los 8 valores de los factores y ninguno de ellos quedara sin calcular. Una vez que hemos calculado las 8 incógnitas y procedemos ha hacer el producto matricial de las matrices de factores podremos ver claramente la diferencia entre el LSI y el SVD ya que el LSI ajustará los factores para que las posiciones de (1,2) y (2,1) de la matriz A den como resultado un valor muy próximo a cero, mientras que el SVD dará como resultado un valor entre 2 y 3 que será la predicción del voto que realizaría el *usuario* sobre el *item*.

Haciendo el LSI sobre la matriz A obtenemos las siguientes matrices de factores de *usuarios* e *items* respectivamente, cuyo producto matricial es la matriz A:

$$\begin{bmatrix} \sqrt{3} & 0 \\ 0 & \sqrt{2} \end{bmatrix} \cdot \begin{bmatrix} \sqrt{3} & 0 \\ 0 & \sqrt{2} \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} = A \tag{108}$$

Mientras que si resolvemos el sistema de ecuaciones (107), obtenemos lo siguiente:

$$\begin{bmatrix} 1{,}731 & 0{,}006 \\ 1{,}414 & 0{,}012 \end{bmatrix} \cdot \begin{bmatrix} 1{,}732 & 1{,}414 \\ 0{,}017 & 0{,}024 \end{bmatrix} = \begin{bmatrix} 2{,}99 & 2{,}45 \\ 2{,}45 & 1{,}99 \end{bmatrix} \neq A \tag{109}$$

Tras realizar el sistema de ecuaciones observamos dos cosas. La primera que los valores de las posiciones (1,2) y (2,1) de la matriz resultante son unos valores dentro de lo esperado ya que son una predicción de dos items que el usuario no ha votado y entra dentro de la lógica que el valor que nos ha dado haya sido un valor que este entre 2 y 3.

Lo segundo que observamos es que se ha cometido un pequeño error a la hora de obtener los valores de las posiciones (1,1) y (2,2) de la matriz resultante. Esto es algo normal al aplicar el SVD al filtrado colaborativo y gracias a ello podemos obtener las predicciones de los *items* que el *usuario* no ha votado. Es por ello que definimos el error que se comente a la hora de ajustar los factores de la siguiente forma:

$$error = \left| r_{u,i} - q_i^t \cdot p_u \right| \tag{110}$$

Por tanto lo que plantea Simon Funck es encontrar el mínimo de la siguiente expresión (111):

$$\min_{q^*,p^*} \sum_{(u,i)\in k} (r_{u,i} - q_i^t \cdot p_u)^2 \tag{111}$$

Es decir aquella que minimice todos los errores que se pueden cometer al calcular los factores de los *usuarios* y de los *items*.

Para encontrar el mínimo de esta expresión (111) existen diferentes técnicas como la técnica del *gradiente descendente* o la de *alternar los mínimos cuadrados*, que en definitiva las dos buscan el ir ajustando las matrices de factores poco a poco. En este proyecto fin de master se utilizará la técnica del *gradiente descendente* que se verá en el capítulo 4.4.

4.3.2. Overfitting o Regularización

Una vez que se ha mostrado el objetivo del SVD aplicado al filtrado colaborativo que es encontrar el mínimo de las expresión (111), existe un problema asociado no solo a este caso sino a todos los algoritmos de aprendizaje que es la *sobre especialización* u *Overfitting* debido a que el problema de regresión que queremos resolver se ajusta a los datos de entrenamiento; es decir, en nuestro caso se ajustará muy bien a la hora de predecir los votos que tenemos en la matriz de votos, pero no tanto a la hora de predecir los votos que el usuario no ha valorado. Con la expresión (111) tendríamos el problema del *Overfitting* ajustándose el sistema a los puntos dados. Un ejemplo de *Overfitting* en dos dimensiones lo mostramos en la siguiente imagen:

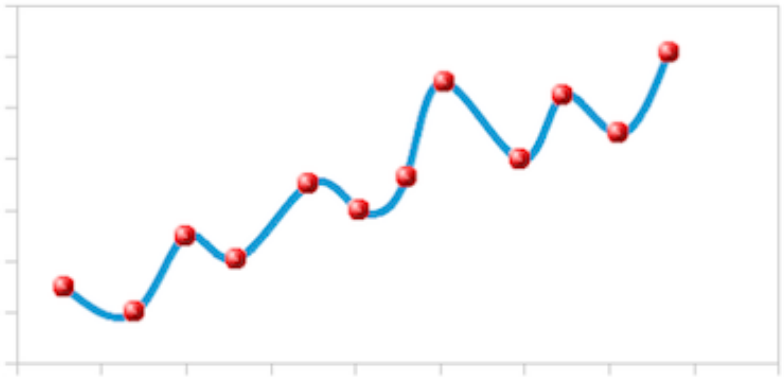

Figura 4: Ejemplo de *sobre especialización* u *Overfitting*.

Es decir, vemos como la función se ajusta a cada uno de los puntos conocidos y por tanto no generaliza. En este caso se ha mostrado para 2 dimensiones, pero para el caso del SVD pasaria para n dimensiones.

Para aliviar el problema del *Overfitting* se le añade a la expresión (111) que hay que minimizar una constante de control o de regularización que llamaremos λ, que nos permitirá que la regresión que realicemos no se ajuste tanto a los datos de entrenamiento que tenemos; es decir, a los votos conocidos, y nos permita obtener unos factores más generalizados de los *usuarios* e *items* para poder hacer mejores recomendaciones. Dado

el parámetro de regularización λ, debemos ahora encontrar el mínimo de la siguiente expresión (112):

$$\min_{q^*,p^*} \sum_{(u,i)\in k} (r_{u,i} - q_i^t \cdot p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \tag{112}$$

De esta forma conseguimos que no se adapte la función a los datos de entrenamiento y de esta forma conseguiremos que generalice más. De forma un poco exagerada y en 2 dimensiones tendríamos una regresión lineal (tal y como se muestra en la siguiente imagen (figura 5)) y no una función que se ajusta a los datos de entrenamiento (figura 4):

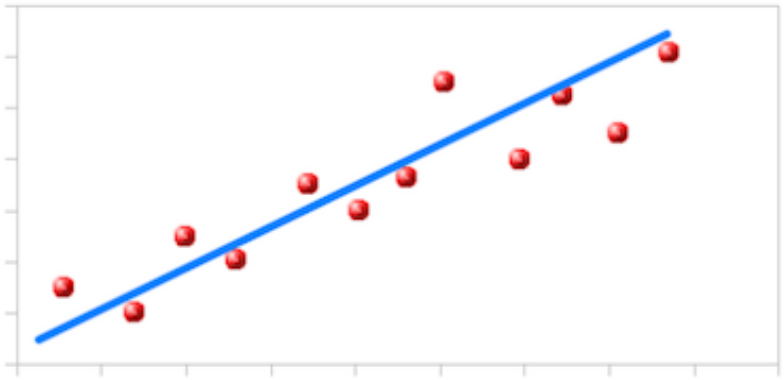

Figura 5: Ejemplo de regresión lineal.

4.4. Implementación del SVD

Visto cual es el objetivo del SVD aplicado a sistemas de recomendación basados en filtrado colaborativo que es encontrar el mínimo de la función (112), debemos de ser capaces de encontrar ese mínimo (o acercarnos a el) con algún algoritmo. Para resolver este problema, se ha elegido en este caso la técnica del *gradiente descendente* que consiste en ir moviéndose poco a poco por la función (n-dimensional) hacia el mínimo. Esta técnica del *gradiente descendente* es una técnica muy utilizada en los algoritmos de aprendizaje, aunque tiene ciertos problemas asociados, como por ejemplo, caer en mínimos locales y no encontrar la solución esperada, o caer en mesetas en las que se tarde mucho tiempo en encontrar la solución.

Para encontrar la solución con la técnica del gradiente descendente, debemos de evaluar la función en un punto dado y moverse hacia otro punto una cierta distancia, y ese punto hacia el que nos debemos de mover será hacia el lado opuesto que nos indique la derivada de la función que queremos minimizar. Un ejemplo de la técnica del gradiente descendente para 2 dimensiones lo vemos en la siguiente figura 6:

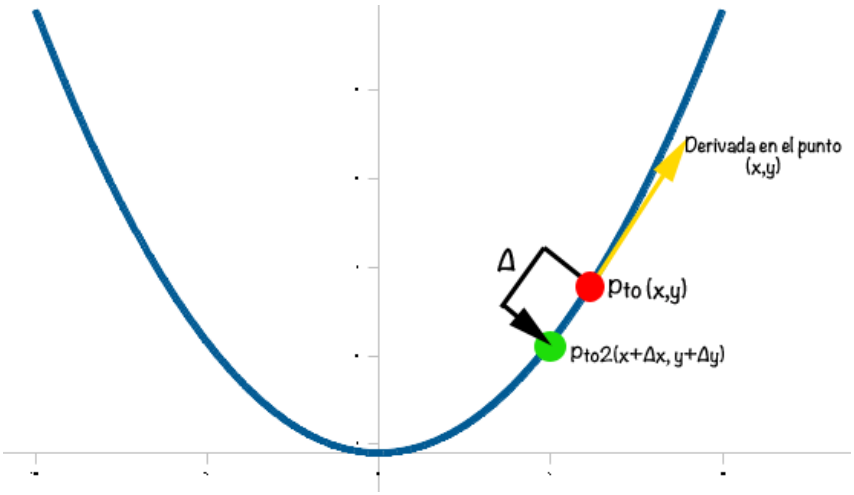

Figura 6: Ejemplo de como funciona el gradiente descendente en 2 dimensiones

Como se observa en la figura 6, la función se va moviendo una cierta cantidad Δ hacia el mínimo de la función, que es hacia el sentido contrario de la derivada en ese punto. Los dos problemas que hay que tener en cuenta para utilizar esta técnica son el número de veces que realizamos el desplazamiento del punto y la distancia a la que se desplazará el nuevo punto.

Uno de los problemas que tenemos al minimizar la función (112) con la técnica del *gradiente descendente*, es que no vamos a saber si el mínimo que nos da el algoritmo que proponemos es verdaderamente el mínimo de la función o un punto muy cercano al mínimo. Esto en realidad no es un problema muy importante para el caso de los sistemas de recomendación (salvo que nos dé como resultado un mínimo local) ya que lo que pretendemos es encontrar un valor muy cercano al mínimo de la función, aunque ese resultado no sea el mínimo. En resumen, queremos encontrar un buen resultado. Para ello es muy importante definir el número de desplazamientos (o salto) que el algoritmo ha de

dar. Para ello nos tenemos que definir una constante que llamaremos ***Epochs*** cuyo valor sera el número de desplazamientos o evaluaciones de puntos que realicemos.

El segundo problema que se nos plantea, es ver las distancia que se mueve el punto para evaluar la función. Como es lógico en un estado inicial se evaluará la función en un punto aleatorio y después empezaremos a desplazarnos por la función. Para ver la distancia que nos desplazamos en cada momento utilizaremos la técnica del *recocido simulado* que es una técnica de inteligencia artificial que consiste en simular el comportamiento del moldeado de aceros y cerámicas, siendo más fácil modelar estos elementos a temperaturas muy altas y mas difícil según disminuya la temperatura. Esto quiere decir que los primeros movimientos (o salto) serán más pequeños según vayamos realizando los movimientos, o en nuestro caso según vayamos realizando *Epochs*. Esta técnica del recocido simulado viene definida por la siguiente función:

$$f(i) = \frac{\gamma}{(\frac{1+i}{K})} \text{ siendo } \gamma \text{ la temperatura y k una constante del material.} \tag{113}$$

Es fácil observar en la función que según aumente el valor de i (que para nuestro algoritmo sera el *Epoch* de la ejecución) el valor de la función será más pequeño y por tanto el salto que dé sera más pequeño.

4.4.1. Pseudocódigo

A continuación pasamos a ver el *Pseudocódigo* del algoritmo que se ha implementado para utilizar la técnica del SVD a los sistemas de recomendación basados en filtrado colaborativo.

En primer lugar vamos a mostrar las variables y constantes utilizadas para el algoritmo que son las siguientes:

```
1 /**
  * Matriz que contine los votos. Si no hay voto la matriz tendra valor cero
  */
int [][] matrizDeVotos = new int [numeroUsuarios][numeroItems];

/**
  * Numero de factores a calcular
```

```
 */
int numFactores;

/**
 * Matriz de factores de los usuarios
 */
double [][] U = new double [numeroUsuarios][numFactores];

/**
 * Matriz de factores de los items
 */
double [][] I = new double [numeroItems][numFactores];

/**
 * Numero maximo de ajustes de los factores
 */
int maxEpochs ;

/**
 * Constante de regularizacion
 */
double landa;

/**
 * Array que guardara la variancion del punto de la funcion
 * que se realizara en funcion del epoch que se este ejecutando
 */
double [] recocidoSimulado = new double [maxEpochs];

/**
 * Constante que representa la temperatura en el recocido simulados
 */
double gamma;

/**
 * Constante que representa a la constante k del recocido simulado
 */
int annealingRate;

/**
 * Maximo valor inicial para inicializar las matrices de factores
 * de los usuarios y de los items.
 */
double initFactors;
```

Antes de ver el *Pseudocódigo* del algoritmo, se han de calcular y rellenar las matrices de factores de los usuarios e items con números aleatorios, cuyos valores serán números comprendidos entre $\pm initFactors$ y también se ha de calcular el array del recocido simulado a partir de las dos constantes *gamma* y *annealingRate*.

El *Pseudocódigo* que se muestra a continuación, es el *Pseudocódigo* del calculo de la matrices de factores de *usuarios* y de *items*, que es la finalidad del algoritmo. Como se puede ver, el algoritmo va ajustando las matrices de factores de *usuarios* e *items*, factor a factor; es decir, que la primera ejecución que haga será modificar el primer factor del primer usuario que haya votado el primer item, luego ajustara el primer factor del usuario

y el primer factor del segundo item que haya votado y así sucesivamente. En resumen lo que hace es ir ajustan dos factores por *epoch* (uno del usuario y otro del item) sobre aquellos *usuarios* que han emitido un voto sobre un *item*. Como se observa también el ajuste variará en función del *epoch* que nos encontremos por el *recocido simulado* que estamos aplicando, haciéndose ajustes más pequeños según se vayan incrementando los *epoch*.

```
for k=0 to numFactores{
  for epoch=0 to maxEpochs {
    for i=0 to numUsers{
      for j=0 to numItems{
        if (matrizDeVotos[i][j] != 0) { // Para no tratar los items no votados

          // Calculamos el error que cometemos con los factores actuales
          double error = matrizDeVotos[i][j] - (productoEscalar(U[i],I[j]));

          // Valor actual de los factores que estamos ajustando
          double uTemp = U[i][k]; double iTemp = I[j][k];

          // Ajuste de las matrices de factores de usuarios e items
          U[i][k] += recocidoSimulado[epoch] * ((error * iTemp) - (landa * uTemp));
          I[j][k] += recocidoSimulado[epoch] * ((error * uTemp) - (landa * iTemp));
        }
      }
    }
  }
}
```

Como puede verse en el Pseudocódigo, este algoritmo tiene una complejidad muy alta. Tal y como está mostrado es un algoritmo muy lento y se ha planteado alguna mejora para que sea más rápido y que veremos en el siguiente punto 4.2.2.

Otra de las mejoras que propuso Simon Funck para que este algoritmos hiciese mejores recomendaciones y se ajustase más a las predicciones esperadas, fue el de hacer el *SVD* pasandole en vez de la matriz de votos, una matriz que contenía el valor de los votos menos la media que tiene el *item* en la base de datos y restándole también lo que denominó el OffSet del usuario, que es lo que se desvían las votaciones del usuario con respecto a la valoración media que tiene el item en la base de datos. En definitiva lo que propuso fue sustituir el voto de la matriz de votos por el siguiente valor:

$$r'_{u,i} = r_{u,i} - \overline{r_i} - OffSet_u \tag{114}$$

Siendo el Offset para cada *usuario u*:

$$OffSet(u) = \frac{\sum_{i \in I} |r_{u,i} - \overline{r_i}|}{N} \tag{115}$$

El algoritmo utilizado es exactamente el mismo, lo único que en este caso se le pasa una matriz de votos distinta para calcular los factores de los *usuarios* y de los *items*.

Una vez que se hayan calculado los factores de los *usuarios* y los *items* a partir de la matriz de votos modificada, se tendrá que hacer el proceso contrario una vez calculadas las predicciones (multiplicando las matrices de los factores); es decir, a las predicciones realizadas se les tendrá que sumar la media del voto del *item* de la base de datos y el offset del *usuario*. Esta regularización propuesta por Simon Funck (y que más adelante la denotaremos como SVD_SF) consigue mejores resultados en las predicciones ya que en realidad lo que realiza es el obtener los factores a partir de como el usuario vota con respecto a la media de como votan los demás usuarios.

4.4.2. Características

Este algoritmo propuesto es un algoritmo muy costoso por su elevada complejidad ciclomática y por las estructuras de datos tan grandes que tienen que soportar ya que es algo muy normal en los sistemas de recomendación comerciales que tengan registrados millones de votos y cientos de miles de *usuarios* e *items* por tanto el algoritmo tiene que realizar muchas iteracciones y tardaría mucho tiempo.

En este proyecto se han propuesto algunas mejoras al algoritmo sobre todo para reducir los tiempos de ejecución del algoritmo.

La primera mejora que se ha planteado ha sido la de crearse una ***"memoria cache"*** que irá guardando el error acumulado una vez que se ha calculado un factor para todos los *usuarios* e *items*. De esta forma para calcular la variable *error* que se ha declarado en el *Pseudocódigo*, necesitaríamos hacer el producto de dos números en vez de hacer un

producto escalar con dos vectores, por tanto el tiempo de ejecución del algoritmo se reduce en torno a un 30-35 %, en experimentos realizados con las bases de datos de MovieLens. La diferencia que hay al trabajar con esta *"memoria cache"*, es que ahora se trabajará sobre una *matriz de errores* (en la cual se irá guardando el error cometido para cada predicción), en vez de sobre una matriz de votos en la que obtenemos el voto para ver el error que cometemos al predecir con los valores de los factores que llevamos calculados. De esta forma cada vez que se calcula un factor se actualiza la *matriz de errores*, y para calcular el *error* de cada iteracción solo tenemos que hacer la multiplicación de dos números y una resta. El Pseudocódigo del algoritmo con esta mejora se muestra a continuación:

```
double [][] errores = matrizDeVotos;

for k=0 to numFactores{
  for epoch=0 to maxEpochs {
    for i=0 to numUsers{
      for j=0 to numItems{
        if (errores[i][j] != 0) { // Para no tratar los items no votados

          // Valor actual de los factores que estamos ajustando
          double uTemp = U[i][k];
          double iTemp = I[j][k];

          // Calculamos el error que cometemos con los factores actuales
          double error = errores[i][j] - (uTemp * iTemp);

          // Ajuste de las matrices de factores de usuarios e items
          U[i][k] += recocidoSimulado[epoch] * ((error * iTemp) - (landa * uTemp));
          I[j][k] += recocidoSimulado[epoch] * ((error * uTemp) - (landa * iTemp));
        }
      }
    }
  }

  // Cuando he terminado de calcular un factor, guardo el error en la matriz de errores.
    Es decir, actualizo la memoria cache.
  for i=0 to errores.NumeroFilas {
    for j=0 to errores.NumeroColumnas {
      errores[i][j] -= U[i][k] * I[j][k];
    }
  }
}
```

La segunda mejora que se ha incluido ha sido la de parar el algoritmo cuando el *valor singular* obtenido tras el cálculo de un factor (salvo el primero) sea un cierto porcentaje menor que el primer *valor singular* calculado. Esto es debido a que los *valores singulares* (ver capítulo 4.1.3) nos dicen la importancia que tiene cada factor para los *usuarios* y los *items* y por tanto podemos definir una condición de parada cuando consideremos que no merece la pena seguir calculando factores que puedan tener muy poco peso en

la predicción o que incluso puedan ser insignificantes. Esta mejora reduce muchísimo el tiempo de ejecución del algoritmo y los resultados obtenidos son prácticamente los mismos que si se calculasen muchos más factores sin poner la condición de parada, siempre y cuando se elija bien ese porcentaje de parada. Por tanto lo que se hace en este caso, es que cada vez que se ha terminado de calcular un factor, se obtiene el *valor singular* a partir de los factores calculados y se compara con el primer *valor singular*. Para obtener el *valor singular* de cada factor se hace el producto de los módulos de los factores de lo *usuarios* y de los *items*, es decir:

$$ValorSingular_i = \left|\overrightarrow{Ufac_i}\right| \cdot \left|\overrightarrow{Ifac_i}\right| \tag{116}$$

La tercera y última mejora que se propone es poner otra condición de parada que vendrá dada por la modificación de los factores de los usuarios e items; es decir, lo que se modifican los factores para cada epoch. Como se puede ver en el siguiente fragmento del *Pseudocódigo* cuando el factor del *usuario* e *item* que se esta modificando en esa iteracción es menor que una cierta cantidad prefijada (*double minimaMejora*), dejo de calcular mas *epochs* para ese factor, ya que las siguientes iteracciones, (siguiendo la lógica del *recocido simulados*) tendrán mejoras mas pequeñas que la actual al ser el valor del *recocidoSimulado[epoch]* un valor más pequeño que el actual.

```
double minimaMejora

// Valor actual de los factores que estamos ajustando
double uTemp = U[i][k];
double iTemp = I[j][k];

// Calculamos el error que cometemos con los factores actuales
double error = errores[i][j] - (uTemp * iTemp);

// Ajuste de las matrices de factores de usuarios e items
U[i][k] += recocidoSimulado[epoch] * ((error * iTemp) - (landa * uTemp));
I[j][k] += recocidoSimulado[epoch] * ((error * uTemp) - (landa * iTemp));

if (|uTemp-U[i][k]|< minimaMejora AND |iTemp-I[j][k]|< minimaMejora)
  Break //PARO DE CALCULAR MAS EPOCHS PARA ESTE FACTOR 'K'
```

En resumen, con estas mejoras planteadas en este proyecto, se mejora considerablemente los tiempos de ejecución. En primer lugar se mejora entorno a un 30-35 % de tiempo en el calculo de los factores utilizando la memoria cache y con el uso de las dos condiciones

de parada se consigue disminuir mucho más el tiempo de ejecución aunque los resultados que se obtiene no serán tan precisos como los que se obtendrían sino pusiésemos estas condiciones de parada, aunque eligiendo correctamente los valores para esas condiciones de parada, los resultados que se obtienen son prácticamente iguales que sino se pusiesen, por tanto es una muy buena opción para aplicar este algoritmo a un sistema de recomendación comercial.

4.5. Folding-In

El *Folding-In* es una técnica que nos permite obtener los factores de un *usuario* nuevo, una vez que se tienen calculadas las matrices de factores de los *usuarios* y los *items*. Al ser el *SVD* una técnica computacionalmente muy costosa, seria muy malo para los sistemas de recomendación que una vez que ingrese un nuevo *usuario*, se tuviese que volver a calcular otra vez todos los factores para poder realizar las predicciones al nuevo *usuario*. Es por ello que esta técnica permite obtener los factores del nuevo *usuario* a partir de los *votos* ($Votos_{NU}$) que ha realizado sin la necesidad del calculo de todo el modelo. La obtención de estos factores del nuevo usuario ($Factores_{NU}$) se calcularía de la siguiente forma:

$$Factores_{NU} = Votos_{NU} \cdot Ifac \cdot S^{-1} \tag{117}$$

Esta técnica del *Folding-In* tiene un problema muy importante y es que al consistir en dos productos matriciales, no podemos dejar de tratar aquellos *items* que el *usuario* no ha votado, teniendo por tanto el problema que se planteaba a la hora de ajustar el SVD al filtrado colaborativo (ver capítulo 4.2) que es el de tener una matriz de votos (la matriz $Votos_{NU}$ de (117)) no rellena.

Al ser normalmente esta técnica del *Folding-In* una técnica que se aplica a *usuarios* nuevos que por lo general entran a un sistema de recomendación votando un porcentaje de *items* muy pequeño, seria una buena aproximación que se hiciera este cálculo rellenando los votos no realizados con la media del *item* no votado.

En resumen esta técnica del *Folding-In*, aporta una solución relativamente buena para poder recomendar a un *usuario* nuevo sin la necesidad de volver a calcular de nuevo las matrices de factores. Evidentemente las recomendaciones no serán igual de buenas que si se hiciera el *SVD* de nuevo, pero los tiempos a la hora de recomendar serían prácticamente insignificantes ya que el cálculo de las recomendaciones se reduce a dos productos matriciales para obtener los factores del nuevo usuario y un tercer producto matricial para obtener las recomendaciones ($Factores_{NU} \cdot Ifac$).

4.6. Ejemplo

En este punto vamos a ver un sencillo ejemplo de como funcionaría el algoritmo y como haría las predicciones. Para ello vamos a utilizar la siguiente matriz de votos, representando • a un voto no realizado el cual se ha de predecir.

	I1	**I2**	**I3**	**I4**	**I5**
U1	3	•	5	2	•
U2	4	2	•	3	4
U3	•	3	5	3	•

Cuadro 13: Matriz de votos usada para mostrar un ejemplo de la ejecución del SVD

Para realizar el SVD con esta matriz de votos vamos a utilizar los siguientes parámetros para ejecutar el algoritmo:

Número de Factores	2
Número máximo de Epochs	5.000
Número mínimo de Epochs	100
Landa	0,2
Gamma	0,005
Annealing Rate	1.000
Inicialización de Factores	0,01

Cuadro 14: Matriz de votos usada para mostrar un ejemplo de la ejecución del SVD

Los primeros pasos que hay que dar (y daría igual el orden), son los de calcular el array del *recocido simulado* e inicializar las matrices de factores de *usuarios*, *items* y la matriz de *errores*. El array para el recocido simulada y la matriz de errores, van a tener valores fijos, mientras que las matrices de factores van a tener números aleatorios,

por tanto vamos a mostrar los valores de estas matrices para este ejemplo que serán de dimensión (3 x 2) para la matriz de *usuarios* y (5 x 2) para la de *items*:

	Factor 1	Factor 2
U1	0.00490625	-0.02132719
U2	-0.00284106	-0.00118192
U3	0.0096044	0.00948969

	Factor 1	Factor 2
I1	0.00642210	-0.001026460
I2	-0.00726764	0.00681780
I3	-0.0132577	-7.43789E-4
I4	-0.00844146	-0.01224746
I5	0.00959260	0.00782492

Cuadro 15: Matrices de factores de *usuarios* e *items*.

A continuación vamos a mostrar algunas trazas de la ejecución del algoritmo.

En primer lugar se va a coger de la matriz de votos el primer voto (en de la posición (1,1)) y se ajustará el primer factor del *usuario* 1 (U1) y del *item* 1 (I1):

```
... PROCESANDO FACTOR 1 ...

... PROCESANDO EPOCH 1 ...

Procesamos posicion 1,1  con valor = 3.0
uTemp: 0.00490625
iTemp: 0.00642210
Error: 3.0 - (0.00490625*0.00642210) = 2.99996

U[1][1] +=0.0050 * ((2.99996 * 0.00642210) - (0.2 * 0.00490625)) = 0.004997678
I[1][1] +=0.0050 * ((2.99996 * 0.00490625) - (0.2 * 0.00642210)) = 0.006489275

Matriz de Errores
2.99996  0.0  5.0  2.0  0.0
4.0      2.0  0.0  3.0  4.0
0.0      3.0  5.0  3.0  0.0
```

Tras realizar este primer *epoch* vemos como hemos ajustado el primer factor del *usuario U1* y del *item I1*, quedando ahora las matrices de factores de la siguiente manera (siendo los factores en color rojo los que se han modificado):

	Factor 1	Factor 2
U1	**0.004997678**	-0.02132719
U2	-0.00284106	-0.00118192
U3	0.0096044	0.00948969

	Factor 1	Factor 2
I1	**0.006489275**	-0.001026460
I2	-0.00726764	0.00681780
I3	-0.0132577	-7.43789E-4
I4	-0.00844146	-0.01224746
I5	0.00959260	0.00782492

Cuadro 16: Matrices de factores de *usuarios* e *items* tras la primera ejecución del algoritmo.

La siguiente iteracción sera para ajustar los factores del *usuario U1* y del *item I3* ya que el ajuste del *usuario U1* y del *item I2* no se realizará al ser un item no votado. La

ejecución de este ajuste seria la siguiente:

```
... PROCESANDO FACTOR 1 ...

... PROCESANDO EPOCH 1 ...

Procesamos posicion 1,3  con valor = 5.0
uTemp: 0.004997678
iTemp: -0.013257724
Error: 5.0 - (0.004997678*-0.013257724) = 5.00006

U[1][1] +=0.0050 * ((5.00006 * -0.013257724) - (0.2 * 0.004997678)) = 0.00466123
I[3][1] +=0.0050 * ((5.00006 * 0.004997678) - (0.2 * -0.013257724)) = -0.01311952

Matriz de Errores
2.99996  0.0  5.00006  2.0  0.0
4.0      2.0  0.0      3.0  4.0
0.0      3.0  5.0      3.0  0.0
```

Tras esta ejecución cambiaría de nuevo el factor del *usuarios U1* y del *item I3*, quedando las matrices de factores con los siguientes valores (en color azul se muestran las modificaciones):

	Factor 1	Factor 2
U1	**0.004997678**	-0.02132719
U2	-0.00284106	-0.00118192
U3	0.0096044	0.00948969

	Factor 1	Factor 2
I1	0.006489275	-0.001026460
I2	-0.00726764	0.00681780
I3	**-0.013257724**	-7.43789E-4
I4	-0.00844146	-0.01224746
I5	0.00959260	0.00782492

Cuadro 17: Matrices de factores de *usuarios* e *items* tras la segunda ejecución del algoritmo.

De esta forma es como va ajustando los factores de los *usuarios* e *items*; es decir, que en cada epoch ajusta dos factores, uno del *usuario* y otro del *item*. Tras realizar los *5.000 epochs* por factor y terminando la ejecución del algoritmo, obtenemos las siguientes matrices de factores de *usuarios* e *items*:

	Factor 1	Factor 2
U1	1.672127	-0.091302
U2	1.874025	0.048448
U3	1.924238	0.033178

	Factor 1	Factor 2
I1	1.865566	0.050233
I2	1.250532	0.007761
I3	2.606298	-0.084768
I4	1.389312	0.067018
I5	2.019446	0.020709

Cuadro 18: Matrices de factores de *usuarios* e *items* tras la ejecución de todo algoritmo.

Una vez que tenemos los factores ya podemos calcular las predicciones de los items que los usuarios no han votado (y de los que ha votado también aunque no nos interese), haciendo el producto matricial de la matriz de factores de usuarios y la traspuesta de la

matriz de factores de los items:

$$\begin{bmatrix} 1{,}672 & -0{,}091 \\ 1{,}874 & 0{,}048 \\ 1{,}924 & 0{,}033 \end{bmatrix} \cdot \begin{bmatrix} 1{,}865 & 1{,}251 & 2{,}606 & 1{,}389 & 2{,}019 \\ 0{,}050 & 0{,}008 & -0{,}084 & 0{,}067 & 0{,}021 \end{bmatrix} = \begin{bmatrix} 3{,}11 & 2{,}09 & 4{,}36 & 2{,}32 & 3{,}75 \\ 3{,}49 & 2{,}35 & 4{,}88 & 2{,}61 & 3{,}78 \\ 3{,}59 & 2{,}41 & 5 & 2{,}68 & 3{,}88 \end{bmatrix} \tag{118}$$

Por tanto las predicciones de los *items* son las siguientes (mostrando en color azul los items no votados):

	I1	**I2**	**I3**	**I4**	**I5**
U1	3.11	**2.09**	4.36	2.32	**3.75**
U2	3.49	2.35	**4.88**	2.61	3.78
U3	**3.59**	2.41	5	2.68	**3.88**

Cuadro 19: Predicciones de los votos tras la ejecución del SVD

Como puede observarse y si se hace un pequeño estudio de la matriz de votos original (tabla 15) los resultados de las predicciones obtenidas son muy coherentes ya que (para los pocos datos que hay) dan unas predicciones que son aproximadamente la media de las votaciones de los *items* votados.

No se ha puesto un ejemplo de como se ejecutaría el algoritmo con las condiciones de parada explicadas ya que la matriz de votos es muy pequeña al contener solo 10 votos. El ejemplo con las condiciones de parada tendría mas sentido si la matriz de votos tuviese registrado millones de votos. Los ejemplo de tiempos de ejecución se verán en el siguiente capítulo.

5. Experimentos realizados con SVD

En este capítulo se van a mostrar una serie de experimentos que estarán enfocados a evaluar la técnica del SVD y a compararlo con algunas de las métricas de similaridad utilizadas para la técnica de los k-vecinos. Los experimentos y resultados que se muestran en este capítulo serán en parte los expuestos en el capítulo 2.4. Además se mostraran algunos resultados sobre los tiempos de ejecución del algoritmo y de la precisión del cálculo de predicciones en función del número de factores que se seleccione.

Para hacer estas pruebas se ha seleccionado la base de datos de *MovieLens 1M* que es junto con la base de datos de *Netflix* la base de datos más utilizada para evaluar los modelos o métricas propuestas en los artículos de investigación del área de los sistemas de recomendación. A continuación mostramos en una tabla las características de la base de datos que vamos a utilizar y el porcentaje de *usuarios* e *items* de test que se van a seleccionar para realizar todos los experimentos:

Número de votos	1.000.000
Número de Usuarios	6.040
Número de Items	3.706
Porcentaje de Usuarios de test	20 %
Porcentaje de Items de Test	20 %

Cuadro 20: Características de la base de datos utilizada para los experimentos.

Para comparar el SVD con otras técnicas de predicción de votos, se van a utilizar las métricas tradicionales de la *Correlación*, el *Coseno* y el *MSD* (Minimum Square Distance). Por otro lado se comparará también con otras dos métricas más modernas y que dan mejores resultados que las antes mencionadas, que son la métrica del *JMSD* [8] (Jaccard MSD) y la métrica de las Singularidades [6].

5.1. Error Medio Absoluto (MAE)

El primer experimento que se ha realizado ha sido calcular el *MAE* (Mean Absolute Error) que nos dice la distancia absoluta que hay entre la predicción realizada y el valor

real del voto. Los resultados que se muestran se comparan con las métrica de similaridad de la técnica de los k-vecinos por tanto se muestra para estas métricas la evolución del *MAE* en función del número de vecinos que se cojan para las predicciones. Para el caso del SVD, se observa que el *MAE* es constante, ya que es independiente del número de vecinos, al ser una técnica basada en modelos y no en memoria.

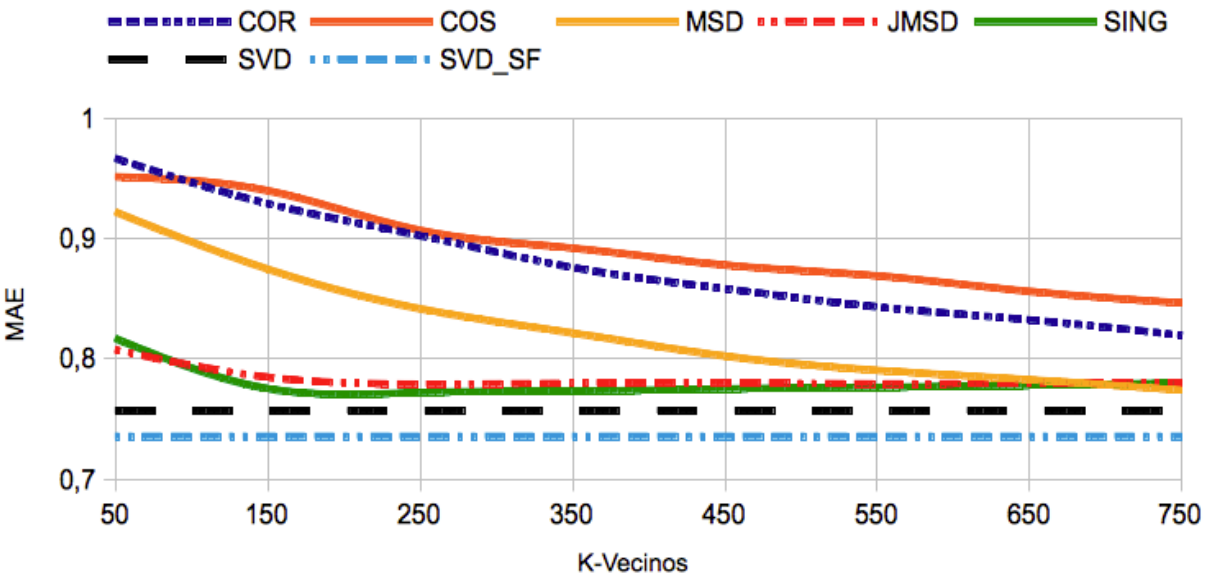

Figura 7: MAE (con MovieLens 1M) para las métricas COS, CORR, MSD, JMSD y SING; y con los modelos del SVD y del SVD con la regularización propuesta por Simon Funck

Con los resultados obtenidos, podemos ver que el *SVD* es el método que mejores predicciones realiza, ya sea con el *SVD* normal o con el *SVD_SF* que es el *SVD* con la regularización de votos propuesta por Simon Funck. Vemos como las métricas tradicionales se alejan mucho de la precisión del *SVD*, mientras que las métricas del *JMSD* y las *Singularidades* tienen un *MAE* muy parecido al del *SVD* para sus vecinos óptimos, aunque el *SVD* es mejor. Otra ventaja que tiene el *SVD* frente a la técnica de los k-vecinos, es que este no necesita calcular el número de vecinos óptimos, lo cual esto no supone un problema para el *SVD*.

Otro de los experimentos realizado en relación con el *MAE*, es el de ver como evoluciona el *MAE* en función del número de factores que se utilicen. Como se observa en la siguiente figura 8 se puede ver perfectamente como llegado a un determinado factor la mejora del *MAE* es muy pequeña e incluso se podía decir que peor si se cogen más fac-

tores. Como se observa para el *SVD*, se alcanza el mejor *MAE* para 4 factores, mientras que para el *SVD_SF* se tiene prácticamente el mismo *MAE* para cualquier número de factores. Esta estabilización del *MAE* es debido a la poca importancia que van teniendo los factores de los *usuarios* e *items* según los vamos teniendo en cuenta; es decir, que esa importancia viene determinado por el valor de los valores singulares.

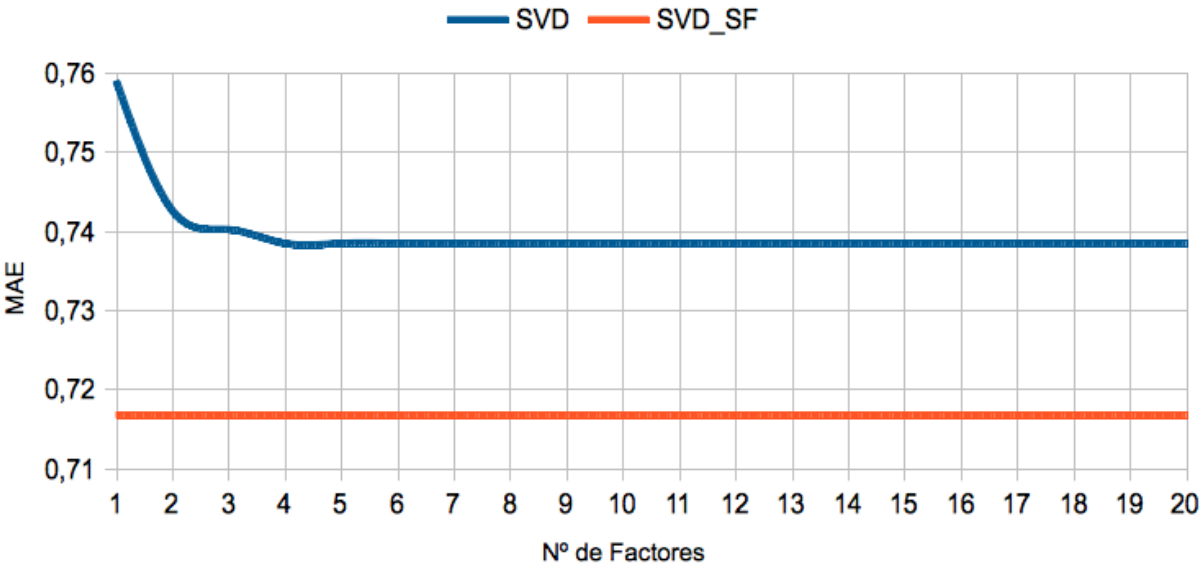

Figura 8: MAE (con MovieLens 1M) obtenido cogiendo n factores.

Esta gráfica es muy importante y nos aporta mucha información, ya que es muy importante valorar el número de factores que hay que calcular para obtener los mejores resultados en las recomendaciones, no tanto por la precisión de los resultados, sino por el tiempo de cómputo del algoritmo. Como se observa en la gráfica igual es más importante calcular el modelo con solo 4 factores, habiendo calculado el modelo en un tiempo menor que si hubiésemos calculado el modelo con mas factores en un tiempo considerablemente superior.

5.2. Capacidad de Recomendación (Coverage)

Respecto a la capacidad de recomendación que tiene el *SVD*, se ha de decir que es del 100 % ya que una vez que obtiene las matrices de factores de los *usuarios* e *items*, es capaz de predecir todos los votos de los *usuarios* sobre los *items*. Esta es una gran ventaja sobre

las técnicas que utilizan los *k-vecinos* ya que estos necesitan un número considerable de *vecinos* para poder calcular muchos *items* y aun asi no se acercan al 100 % como el *SVD*. A continuación mostramos la gráfica de *Coverage* para el *SVD* y las métricas usadas para los *k-vecinos*.

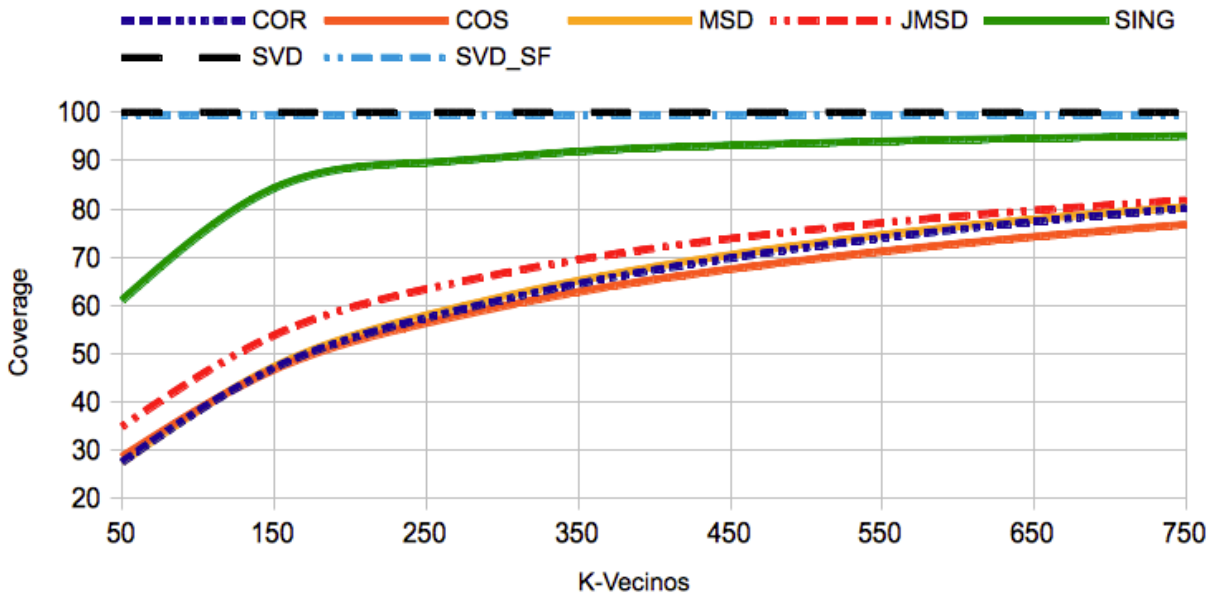

Figura 9: Coverage (con MovieLens 1M) para las métricas COS, CORR, MSD, JMSD y SING; y con los modelos del SVD y del SVD con la regularización propuesta por Simon Funck

5.3. Calidad de las Recomendaciones (Precision y Recall)

Sobre el *Precision* y el *Recall*, el SVD no aporta ninguna mejora respecto a las métricas del *JMSD* y las *Singularidades* aunque si son un poco mejores en comparación con las métricas tradicionales de la *Correlación* el *Coseno* y el *MSD*.

En lo referente a la *Precisión*; que es el porcentaje de *items* relevantes entre los *items* recomendados (ver capítulo 2.4.4), se observa que el SVD recomienda de una forma parecida que el *JMSD* y las *Singularidades*, siendo algo mejor que para las métricas tradicionales. La gráfica que se muestra a continuación se ha realizado obteniendo los datos para un umbral de relevancia de 5:

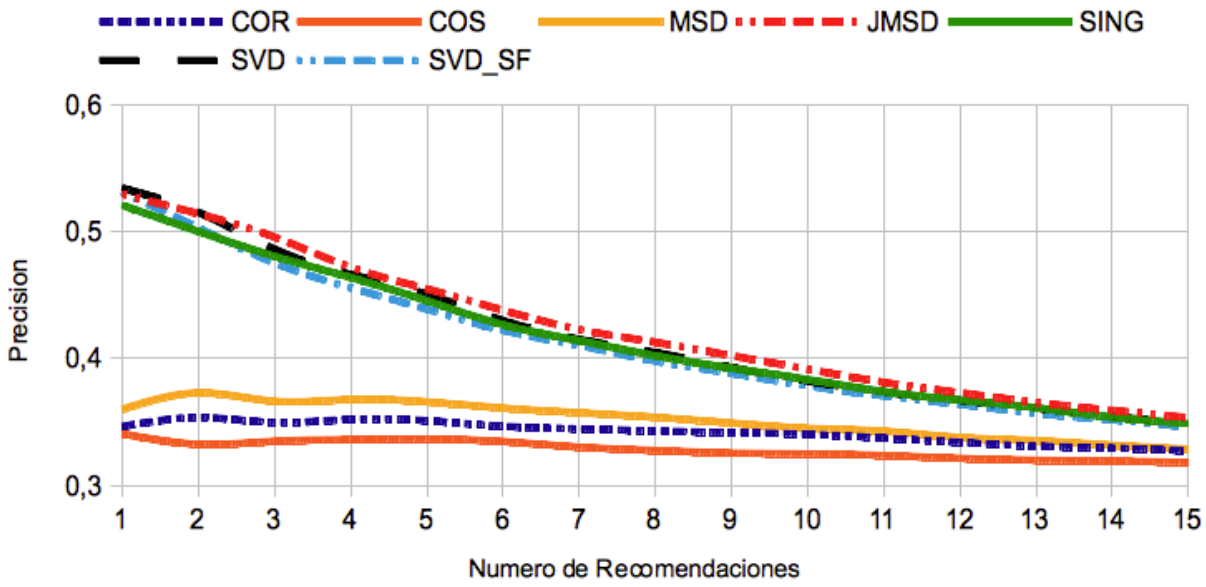

Figura 10: Precision (con MovieLens 1M) para las métricas COS, CORR, MSD, JMSD y SING; y con los modelos del SVD y del SVD con la regularización propuesta por Simon Funck

En lo referente al *Recall*; que es el porcentaje de *items* relevantes recomendados respecto del total de los relevantes, se observa el mismo comportamiento que el *Precision*; es decir, que el *SVD* se tiene unos resultados parecidos al *JMSD* y a las *Singularidades* (umbral de relevancia 5):

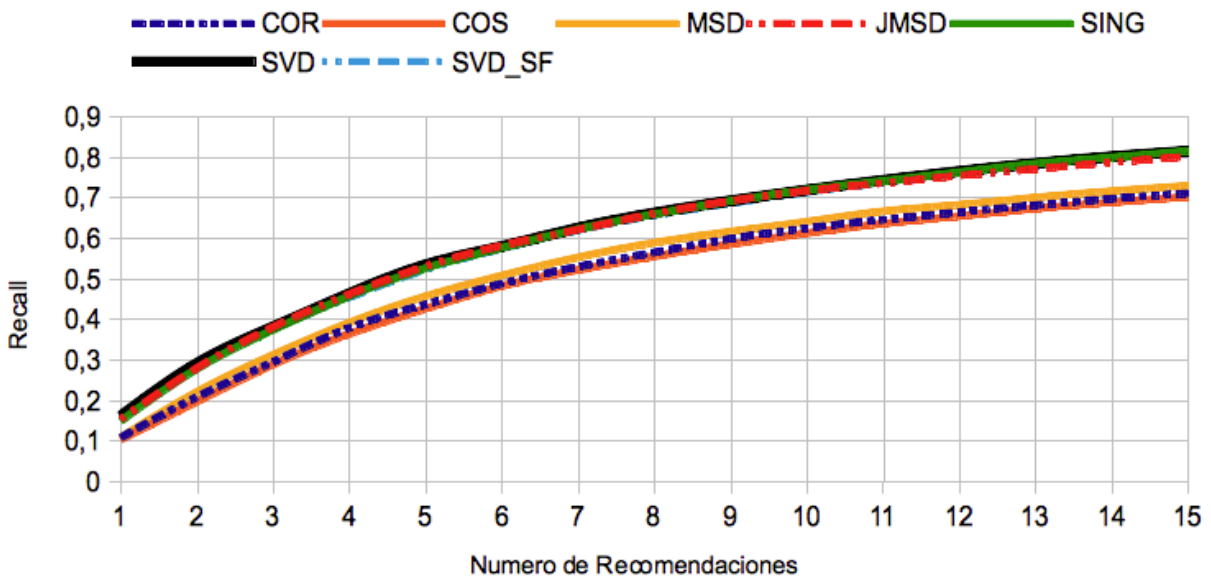

Figura 11: Recall (con MovieLens 1M) para las métricas COS, CORR, MSD, JMSD y SING; y con los modelos del SVD y del SVD con la regularización propuesta por Simon Funck

Por último en la gráfica del *Precision/Recall*, que es la gráfica que verdaderamente interesa para ver la calidad de las recomendaciones, se observa el mismo comportamiento que en las gráficas por separados; es decir, que los dos métodos del *SVD* y las métricas del *JMSD* y las *Singularidades* realizan unas recomendaciones de una calidad similar, mientras que si se observa que la calidad de las recomendaciones es mejor respecto de las métricas tradicionales.

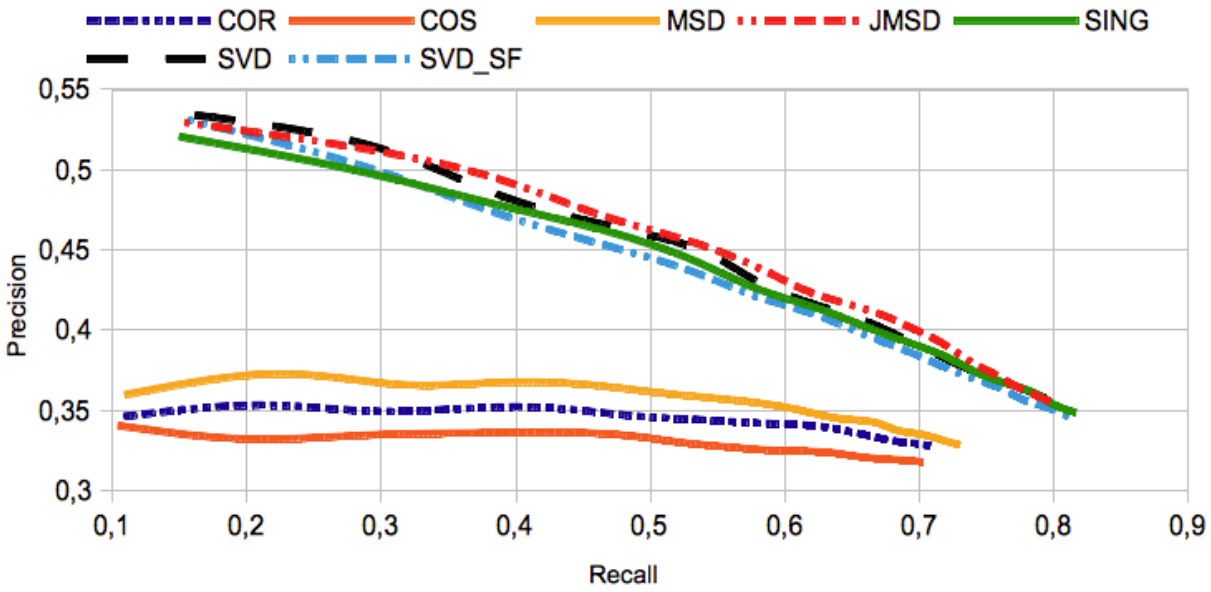

Figura 12: Precision/Recall (con MovieLens 1M) para las métricas COS, CORR, MSD, JMSD y SING; y con los modelos del SVD y del SVD con la regularización propuesta por Simon Funck

Viendo estos resultados, sacamos en conclusión que utilizando la técnica del SVD obtenemos una precisión de predicción de voto mejor que utilizando la métrica de los k-vecinos (con cualquiera de las métricas de similaridad), pero por el contrario vemos que la calidad de las recomendaciones son parecidas a las que obtenemos con algunas de las métricas de la técnica de los k-vecinos.

5.4. Tiempos de Ejecución

El último experimento realizado ha sido el de comprobar los tiempos que tardaba el algoritmo en calcular las predicciones. Esta ejecuciones se han realizado en un ordenador personal con un procesados *Core i7 de intel* de *3,1GHz* de frecuencia y con 4 núcleos

(aunque el algoritmo no es paralelizable y solo se usa un solo núcleo).

Este experimento de tiempo se ha realizado con el *SVD* y el *SVD_SF* propuesto por Simon Funck, utilizando por separado las 2 condiciones de parada explicadas en el capítulo 4.4.2 y los resultados obtenidos con la ejecución de los tiempos que se muestran han dado unos resultados del *MAE* prácticamente iguales.

En la gráfica de la figura 13 se puede observar como al ejecutar los algoritmos sin poner ninguna condición de parada (SVD y SVD_SF), tardan muchísimo más tiempo que los demás, tardando el cálculo de cada factor el mismo tiempo.

Por otro lado podemos ver (en SVD % y SVD_SF %) que si ponemos como condición de parada que el porcentaje del valor singular calculado sea inferior a un cierto porcentaje del primer valor singular (en este caso fue de un 0,1 %) el algoritmo deja de calcular más factores y realiza las predicciones con los factores que lleva calculados, en este caso 4. Si contrastamos el número de factores (que son 4) con la gráfica del *MAE* por factor (figura 8), se observa que una vez calculado el cuarto factor, el MAE se mantiene prácticamente constante por muchos más factores que se calculen, por tanto se observa que la condición de parada es muy buena.

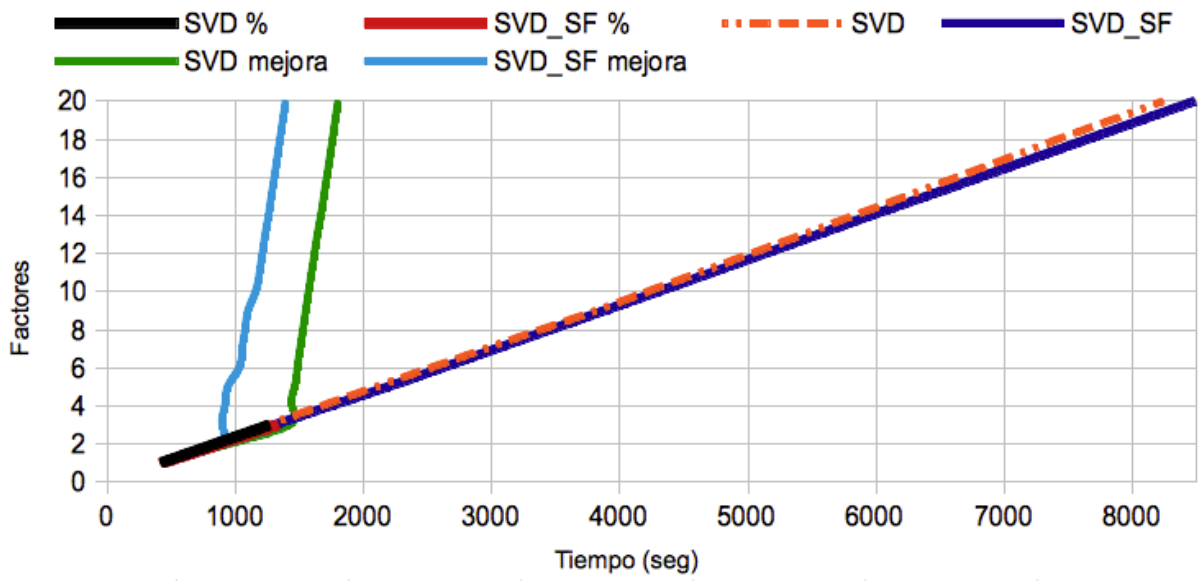

Figura 13: Tiempos de Ejecución del SVD

Por último vemos (en SVDmejora y SVD_SFmejora) como aplicando la segunda condición de parada; es decir, dejando de calcular un factor cuando este varia por debajo de un umbral, se obtienen unos resultados de tiempo mucho mejores que sino se pone esa condición de parada y ademas a partir del tercer o cuarto factor el tiempo de cálculo de cada factor va siendo cada vez menor al tener estos valores muy pequeños y llegar enseguida a tener mejoras que están por debajo de ese umbral.

En resumen, se puede ver que las mejoras en el tiempo de ejecución propuestas en este proyecto fin de master, son considerablemente buenas, consiguiendo ademas unos resultados en la precisión de las predicciones similares. Con conclusión se obtiene que el mejor algoritmo a ejecutar para obtener las mejores recomendaciones en el menor tiempo posible es el algoritmo de *SVD_SF* con las condiciones de parada del número de factores y combinándolo con el umbral de variación de factores por *epoch*.

6. Conclusiones

En este proyecto fin de master se ha realizado una revisión del estado actual de los sistemas de recomendación basados en filtrado colaborativo, y se ha estudiado e implementado la técnica del SVD (técnica basada en modelos) aplicado a los sistemas de recomendación basados en filtrado colaborativo. Esta técnica es hasta la fecha, la técnica que mejores resultados obtiene ha la hora de hacer predicciones dentro de un sistema de recomendación. El problema que tiene son los tiempos tan elevados (en comparación con la técnica de los k-vecinos) que tienen a la hora de calcular las predicciones; por tanto, hace que esta técnica no sea muy atractiva para ser aplicado en un sistema de recomendación comercial.

La principal aportación de este proyecto ha sido la de mejorar los algoritmos (tras el estudio de los mismos) para que estos obtengan predicciones en un tiempo menor. Este estudio no se ha enfocado en conseguir paralelizar los algoritmos existentes, sino en ver sus características y poder establecer unos puntos de parada cuando el modelo conseguido por el algoritmo cumpla unos umbrales mínimos de calidad en las predicciones, que como se ha visto con las condiciones de parada propuestas en este proyecto, los resultados en la calidad de las predicciones son igual de buenos e incluso mejores (si conseguimos saber el número de factores óptimos) que si dejásemos los algoritmos ejecutándose sin condiciones de parada.

Como se ha mostrado; bien sea con el SVD normal o con el SVD con la regularización propuesta por Simon Funck, los resultados del *MAE* y del *Coverage* son resultados bastante mejores que los que se consigue con la técnica de los k-vecinos, pero sorprendentemente los resultados obtenidos en relación con la calidad de las recomendaciones (*Precision-Recall*) no son mejores (aunque tampoco peores) que los que se consiguen con algunas de las métricas de similaridad aplicadas a las técnicas de los k-vecinos como son las métricas del *JMSD* y las *Singularidades*.

En resumen; y con las mejoras propuestas en este proyecto fin de master, ha sido posible mejorar esta técnica del SVD en lo que a tiempos de ejecución de los algoritmos se refiere, y se deja una puerta abierta a que esta técnica sea aplicada a sistemas de recomendación comerciales, ya que la relación *tiempo/calidad de recomendación*, ha mejorado notablemente con las mejoras propuestas en este proyecto.

7. Trabajos Futuros

La idea principal de este proyecto ha sido presentar un sistema de recomendación (dentro del filtrado colaborativo), basado en modelos cuyas predicciones son las mejores predicciones que hasta la fecha hace ninguna otra técnica que no sea la de obtener una serie de factores asociados a *usuarios* e *items*, como pueden ser las técnicas del *PMF* (Probabilistic Matrix Factorization) [29] o *CTR* (Collaborative Topic Regresion) que son técnicas que hacen lo mismo que el SVD, pero obteniendo estos factores de forma diferente.

Dentro del campo de los sistemas de recomendación; su principal objetivo, es realizar recomendaciones buenas en un tiempo relativamente pequeño. El primer punto el SVD lo cumple con creces ya que realiza unas recomendaciones muy buenas y los posibles trabajos futuros enfocados en la mejora de las predicciones, seria mejorar la técnica del SVD para que las predicciones fuesen mejores o encontrar una nueva técnica, basada en factores que mejorase los resultados.

En relación con la mejora de tiempos, es sin duda la peor característica de la técnica del SVD y es por ello que la hace muy poco atractiva para que esta sea aplicada a los sistemas de recomendación comerciales. Es sin duda el objetivo pendiente y el trabajo futuro mas inmediato, el de mejorar los tiempos de obtención de predicciones y disminuir la complejidad ciclomática de estos algoritmos.

Por último destacar que esta técnica a nivel de investigación, tendrá en un futuro mejoras importantes y podrá ser aplicado a la obtención de ciertas características de los sistemas de recomendación, ya sea el *reliability*, el *novelty*, el *discovery*, etc. Con esta técnica es posible que se puedan conseguir mejoras en los sistemas de recomendación (no tanto de precisión) con un estudio más profundo del significado de los factores y de las propiedades que esta técnica tiene, por tanto esta técnica puede tener un cierto margen de recorrido dentro de la investigación si se sigue trabajando en ella.

Referencias

[1] *FilmAffinity*. http://www.filmaffinity.com/, Marzo 2013.

[2] J. Burguillo M. Rey-López F. Mikic-Fonte A. Peleteiro A. Barragáns-Martínez, E. Costa-Montenegro. A hybrid content-based and item-based collaborative filtering approach to recommend tv programs enhanced with singular value decomposition. *Information Sciences*, 180(22):4290–4311, 2010.

[3] Tom M. Apostol. *Calculus*. Ed. Reverte, 1973.

[4] F. Chung C. Leung, S. Chan. An empirical study of a cross-level association rule mining approach to cold-start recommendations. *Knowledge-Based Systems*, 21(7):515–529, 2008.

[5] Ortega. F. *Filtrado Colaborativo Basado en Transcendencia*. Tesis de Máter, Facultad de Informática (UPM), Junio 2011.

[6] J. Bobadilla F. Ortega, A. Hernando. A collaborative filtering similarity measure based on singularities. *Information Processing and Management*, page 204–217, 2012.

[7] J. Bobadilla F. Serradilla. The incidence of sparsity on collaborative filtering metrics. *Australian Database Conference*, page 9–18, 2009.

[8] J. Bobadilla F. Serradilla, J. Bernal. A new collaborative filtering metric that improves the behavior of recommender systems. *Knowledge-Based Systems*, 23(6):520–528, 2010.

[9] A. Tuzhilin G. Adomavicius. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[10] J. Herlocker. An empirical analysis of design choices in neighborhood-based collaborative. *Information Retrieval*, 5:287–310, 2002.

[11] L Terveen J Riedl J. Herlocker, J Konstan. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.

[12] G. Karypis J. Riedl. Learning preferences of new users in recommender systems : An information theoretic approach preferences george of new users in recommender systems : An theoretic. *SIGKDD Explorations*, 10(2):90–100.

[13] N. Antonopoulos J. Salter. Recommender agent : Collaborative and content-based filtering. *IEEE Computer Society*, pages 35–41, 2006.

[14] J. Herlocker S. Sen J. Schafer, D. Frankowski. Collaborative filtering recommender systems. *The Adaptive Web*, page 291–324, 2007.

[15] E. Martínez J. Bobadilla J. Sánchez, F. Serradilla. Choice of metrics used in collaborative filtering and their impact on recommender systems. *IEEE Internacional Conference on Digital Ecosystems and Technologies*, page 432–436, 2008.

[16] B. Krulwich. Using large-scale demographic data. *American Association for Artificial Intelligence*, 18(2):37–46, 1997.

[17] M. Boull L. Candillier, F. Meyer. Comparing state-of-the-art collaborative filtering systems. *LNAI,*, 4571:548–562, 2007.

[18] C. Li L. Gao. Hybrid personalized recommended model based on genetic algorithm. *4th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4, 2008.

[19] J.F. Huete M.A. Rueda-Morales L.M. De Campos, J.M. Fernández-Luna. Combining content-based and collaborative recommendations: A hybrid approach based on bayesian networks. *International Journal of Approximate Reasoning*, 51(7):785–799, 2010.

[20] Chih-Chao. M. *Large-scale Collaborative Filtering Algorithms*. Master Thesis, College of Electrical Engineering & Computer Science National Taiwan University, Junio 2008.

[21] K. Bharadwaj M. Al-Shamri. Fuzzy-genetic approach to recommender systems based on a novel hybrid user model. *Expert Systems with Applications*, 35(3):1386–1399, 2008.

[22] T. Kutsuras M. Papagelis, D. Plexousakis. Alleviating the sparsity problem of collaborative. page 224–239, 2005.

[23] K.Margaritis M. Vozalis. Using svd and demographic data for the enhancement of generalized collaborative filtering. *Information Sciences*, 177(15):3017–3037, 2007.

[24] J. Salter N. Antonopoulus. Cinema screen recommender agent: combining collaborative and content-based filtering. *IEEE Intelligent Systems*, pages 35–41, 2006.

[25] Nuri. O. *A Singular Value Descomposition Approach for Recommendation Systems.* A Thesis Tubmitted to the Graduate School of Natural and Applied Sciences, Julio 2010.

[26] R. Mahony & R. Sepulchre P.-A. Absil. *Optimization Algorithms on Matrix Manifolds.* Princeton University Press, 2008.

[27] Burke. R. Hybrid recommender systems : Survey and experiments. *User Modeling and User-Adapted Interaction*, 19(4):331–370, 2002.

[28] Y. Xu R. Nayak, L. T. Weng. An improvement to collaborative filtering for recommender systems. *IEEE International Conference on Computational Intelligence for Modeling, Control and Automatitation*, pages 792–795, 2005.

[29] A. Mnih R. Salakhutdinov. Probabilistic matrix factorization. 2008.

[30] Funk. S. *Netflix Update: Try This at Home.* http://sifter.org/ simon/journal/20061211.html, Diciembre 2006.

[31] X. Ge S. Ge. An svd-based collaborative filtering approach to alleviate cold-start problems. *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*, pages 1474–1477, 2012.

[32] Y.Dai S. Gong, H. Ye. Combining singular value decomposition and item-based recommender in collaborative filtering. *2009 Second International Workshop on Knowledge Discovery and Data Mining*, pages 769–772, 2009.

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