Universidad de Valladolid
Facultad de Ciencias

TRABAJO FIN DE GRADO

Grado en Estadística

SISTEMAS DE RECOMENDACIÓN

Autor: Álvaro Berrío Galindo

Tutor: Luis Ángel García Escudero

Índice

Resumen

Los sistemas de recomendación son herramientas utilizadas cada vez por más compañías y de cada vez más sectores diferentes para ofrecer a sus clientes un catálogo personalizado, incluyendo los productos que más pueden interesar a cada cliente. Para llevar a cabo este procedimiento se requiere información acerca de estos usuarios, que puede ser implícita (con qué artículos ha interaccionado) o explícita (valoraciones proporcionadas sobre diferentes artículos).

Existen muchas formas de llevar a cabo un sistema de recomendación y, a medida que se siga investigando en este ámbito, se irán conociendo otras nuevas. A día de hoy se pueden encontrar principalmente dos tipos de filtro, los que estudian los gustos de los usuarios y las características de los productos y los que intentan imitar el comportamiento humano a la hora de recomendar diferentes items.

En este trabajo se presentan las principales formas de llevar a cabo un sistema de recomendación haciendo hincapié en aquellas consideradas más importantes. Además, se incluye un ejemplo práctico sobre uno de estos aplicado a un conjunto de datos real.

Abstract

Recommender systems are tools used by different companies belonging to distinct sectors, in order to offer their customers precise suggestions about products or services. The number of companies utilising these systems is increasing in the last years very rapidly. To create these recommender systems it is necessary to have data about the customers, which can be implicit (products the customer has interacted with) or explicit (ratings about different products).

There are many different ways of implementing a recommender system and as long as people are working with them and investigating, soon there will be many more. At present, there are mainly two types of filtering, the ones that study the preferences of the users and features of the items and the ones that try to mimic the human behaviour about recommending products.

This work include the explanation of these different types of recommender systems paying more attention to those considered more interesting. At the end, there is a practical example of a recommender applied to a real dataset.

1 Introducción

La sociedad y el momento en que vivimos han modificado los hábitos de consumo de los usuarios dotando de más importancia a la personalización, ya sea mediante la visualización de series y películas o escuchando música, entre otros. Se entiende personalización como la recomendación, por parte de las diferentes plataformas de contenido, de productos o servicios que puedan ser de interés para cada usuario. Gran parte del auge de este fenómeno se le puede atribuir al desarrollo de las tecnologías móviles como los smartphones o las tablets, que permiten a los usuarios disfrutar de música y vídeos en cualquier parte. En la actualidad, la gran cantidad de posibilidades y productos que un consumidor puede adquirir hace difícil la toma de decisiones. Por

esto, muchas empresas han aprovechado para intentar sacar el máximo beneficio de los usuarios haciendo uso de sistemas de recomendación, dotando de un toque personal a su oferta.

Jussi Karlgren, experto en lingüística computacional, fue el primero en acuñar el término *sistema de recomendación* en un informe publicado en 1990 en la Universidad de Columbia. Unos años más tarde, en 1994, empezaron a aparecer las primeras implementaciones de la mano del propio Jussi y de grupos de investigación del MIT (Massachusetts Institute of Technology).

Los sistemas de recomendación son herramientas que sirven para sugerir contenidos a los usuarios de una aplicación o programa concretos. Utilizan la información sobre los gustos de los usuarios en base a los ratings que han dado a los contenidos consumidos o el número de veces que lo han reproducido o visitado, entre otros parámetros. También se tienen en cuenta las relaciones entre los diferentes usuarios de manera que si se llega a la conclusión de que dos usuarios tienen gustos similares, se le podrá recomendar el contenido que uno de ellos ha consumido al otro, y viceversa.

Los sistemas de recomendación son un subtipo de los sistemas de filtrado de la información. Estos tratan de eliminar la información innecesaria o redundante de un flujo de información. Son, por ejemplo, los que se usan a la hora de filtrar el correo como *spam* o no.

Pero estos sistemas no se aplican solamente en el mundo de los audiovisuales, empresas de venta online como *Amazon* u otras pertenecientes al sector hostelero los usan para sugerir a los usuarios productos que puedan entrar dentro de sus gustos o necesidades. Además, las redes sociales son también un buen ejemplo de uso de estos sistemas, ya que en estas se pueden recomendar artículos, posts o tuits en el caso de *Twitter*.

1.1 The Netflix Price

En el año 2006, la famosa compañía audiovisual *Netflix* lanzó un concurso para mejorar el sistema que utilizaban en ese momento para recomendar contenidos[1], llamado *Cinematch*, el cual era de 0.9525. La empresa prometía un millón de dólares al equipo que consiguiera reducir un 10 % el RECM (raíz cuadrada del error cuadrático medio) del algoritmo que usaban en ese momento. Cada año, hasta que se consiguiera el objetivo, darían 50.000 dólares al equipo que ocupara el primer puesto en ese momento como premio al progreso. Este concurso captó la atención de muchos expertos en este campo y de usuarios sin tanta experiencia.

Para el desarrollo de los algoritmos, los equipos contaban con un conjunto de entrenamiento de 100 millones de valoraciones realizadas por 500.000 usuarios anónimos sobre 17.000 películas diferentes, cada una entre 1 y 5 estrellas. Los participantes debían enviar predicciones para un conjunto de entrenamiento que contaba con 3 millones de ratings. Sería Netflix la encargada de calcular el RECM y completar la tabla

de clasificación con los resultados de los diferentes equipos participantes. Cada rating consistía en una tupla de 4 elementos:

- ID del usuario
- ID de la película
- Fecha de la valoración
- Estrellas (de 1 a 5)

Finalmente, el premio fue a parar al equipo denominado *BellKor's Pragmatic Chaos*. Fue este mismo grupo de gente el que lideró la tabla en el año 2007 y 2008 hasta que en septiembre del año 2009 consiguieron una mejora del 10.06 % que supuso su victoria, alcanzando un RECM de 0.8567. En el más que exitoso concurso se inscribieron 41.305 equipos de 186 países diferentes, de los cuales más de 5.000 enviaron resultados válidos.

Un requerimiento del concurso era publicar de forma detallada el funcionamiento del mismo por lo que todo el que quiera, puede consultarlo. Es de destacar que la propia empresa no usó el algoritmo ganador ya que había otros, presentados en el propio concurso, cuyos resultados eran ligeramente peores pero con menores necesidades computacionales.

2 Algoritmos

No existe una única forma de implementar un sistema de recomendación. Cada uno de los diferentes tipos sirven para distintos propósitos, conjuntos de datos u objetivos. Se pueden encontrar varios algoritmos principales: filtro basado en contenido [2, 3], filtro colaborativo [2, 3, 4], filtro basado en conocimiento [4, 3] y el filtro híbrido [4].

2.1 Filtro basado en contenido

Para llevar a cabo un filtro basado en contenido es necesario tener dos tipos de información: descripción de un item y un perfil de un usuario. La descripción de un item aporta características sobre el mismo. El perfil de un usuario se crea a partir de los gustos y preferencias de este. La recomendación consiste en determinar qué items se ajustan más a las preferencias del usuario.

Por lo tanto, cuando se quiere construir un filtro basado en contenido es necesaria información implícita sobre los items, es decir, un despliegue de las caractertísticas, así como datos que puedan aportar una visión sobre las preferencias de los usuarios. Este tipo de sistema no requiere de mucha información para funcionar, puede llegar a hacerlo con un solo usuario.

Cuando se habla de descripción de un item depende mucho del contexto en el que se esté trabajando. En el ámbito cinematográfico, un item (una película), puede incluir características como el/los género/s en los que se encuadra o una lista de los actores

y actrices que participan en ella. Sin embargo, si se trata de un proyecto musical se pueden tener en cuenta el estilo de la canción, los instrumentos utilizados o el idioma en que está cantada. También existen otro tipo de características de los items que tienen una condición más subjetiva. Estas son las realmente difíciles de obtener y que pueden hacer que el sistema no funcione como se espera. Por eso, se suelen utilizar las características más técnicas. El conjunto de todas estas características conforman el *perfil del item*.

Lo mismo ocurre con los usuarios. Se crea un *perfil* para cada uno de ellos basado en las características que mejor los definan. Siguiendo con el ejemplo de las películas, un usuario puede estar más interesado en el género cómico y las películas protagonizadas por Tom Hanks. Para conseguir crear estos perfiles de los usuarios, se les puede preguntar directamente o se puede basar en valoraciones que han producido sobre una lista de items. A partir de esa lista, se pueden extraer un conjunto de palabras clave presentes en los items que más le han gustado a cada uno. Por ejemplo, si un usuario es aficionado a las novelas policiacas, una de las palabras clave presentes en su perfil puede ser *detective*.

Tanto para los perfiles de los usuarios como de los items, la inclusión de información referida a fechas de producción de los items o localización geográfica de los usuarios, entre muchas otras, puede servir al sistema para crear perfiles más concretos y por lo tanto, realizar recomendaciones más precisas. Evidentemente, cuanta más información tengan los perfiles, mejor funcionará el sistema.

Al tratar de crear conjuntos de palabras clave para los items, este es un tipo de sistema de recomendación más utilizado para productos que contengan texto, como libros, artículos o posts en Internet. Y es que los filtros basados en contenido pueden tener un gran peso en la recomendación en Internet, como nuevos foros, usuarios a los que seguir en redes sociales, noticias, etc.

Una vez creados los perfiles tanto para los usuarios como para los items, un filtro basado en contenido basa su funcionamiento en la búsqueda de similaridades entre los items consumidos y los no consumidos para un usuario concreto. Existen diferentes formas para realizar esta tarea, las cuales serán explicadas en la sección 3.

Otra idea fundamental de los filtros basados en contenido es que trabajan con información estática, es decir, que a priori no tiene por qué cambiar con el tiempo. Esto tiene sentido ya que al usarse especialmente en el filtrado de documentos, estos albergarán siempre las mismas palabras y también se presupone que, teniendo en cuenta cierto margen, los gustos de los usuarios no se suelen modificar.

2.2 Filtro colaborativo

El otro tipo principal de algoritmos usado en los sistemas de recomendación es el filtro colaborativo. A diferencia del primero, este filtrado no necesita de la creación de perfiles para los usuarios y los items ya que se basa en el comportamiento de los usuarios en el pasado. Se fundamenta en el análisis de las relaciones entre los usuarios

y las interdependecias con los productos para encontrar nuevas asociaciones entre ellos. Existen dos métodos principales a la hora de desarrollar un filtro colaborativo.

- **Métodos basados en memoria**: se centran en las relaciones entre usuarios o entre items. Se trata de encontrar los usuarios más similares al de estudio para extrapolar sus predicciones en el enfoque *basado en usuarios*. El enfoque *basado en items* evalúa las preferencias de un usuario por un item en función de las valoraciones proporcionadas por ese mismo usuario a items vecinos del que se quiere estudiar.
- **Métodos basados en modelos**: utilizan algoritmos y técnicas propias del aprendizaje automático con el objetivo de predecir cuáles serán las valoraciones que harían los usuarios de ciertos items para poder así decidir si recomendarlo o no.

Ambos métodos serán enunciados con mayor detalle en la sección 3.

Uno de los grandes problemas de los filtros colaborativos es el conocido como *cold start*. Este problema ocurre cuando se introducen nuevos usuarios o items en el conjunto y no existe información suficiente como para realizar recomendaciones adecuadas. Si el sistema ha sido arrancado hace poco, este problema puede afectar al algoritmo por completo ya que no se podrán realizar las sugerencias óptimas para ningún usuario/item. Otro de los inconvenientes de los filtros colaborativos es la *escalabilidad*, ya que normalmente se requiere de mucha potencia de cálculo debido a la gran cantidad de usuarios e items. A estos problemas se añade el inconveniente de las *'sparse matrix'*, ya que este tipo de filtrado genera matrices que tienen un tamaño muy grande y en las que muchas casillas están vacías.

2.3 Comparación entre filtro basado en contenido y filtros colaborativo

Estos dos son los tipos de filtro más utilizados en la creación de sistemas de recomendación. La diferencia principal es que los primeros crean perfiles para los usuarios y los items de manera que se pueden realizar comparaciones entre ellos. Los filtros colaborativos, en cambio, basan su funcionamiento en el comportamiento que tienen los usuarios en cuanto a compras, valoraciones o las características que se contemplen. Además, pueden trabajar con cualquier tipo de información ya que el sistema no requiere una definición de los items.

Se podría decir que los filtros basados en contenido hacen recomendaciones basándose en las preferencias de los usuarios y las características de los productos mientras que los filtros colaborativos tratan de imitar el comportamiento de las personas que recomiendan productos mediante el *boca a boca* [3]. De una manera concisa, los filtros colaborativos recomiendan items que gustaron a usuarios similares al usuario en cuestión, mientras que los filtros basados en contenido recomiendan items similares a los items que gustaron al usuario en cuestión. Esta es la relación que se puede apreciar en la figura **1**.

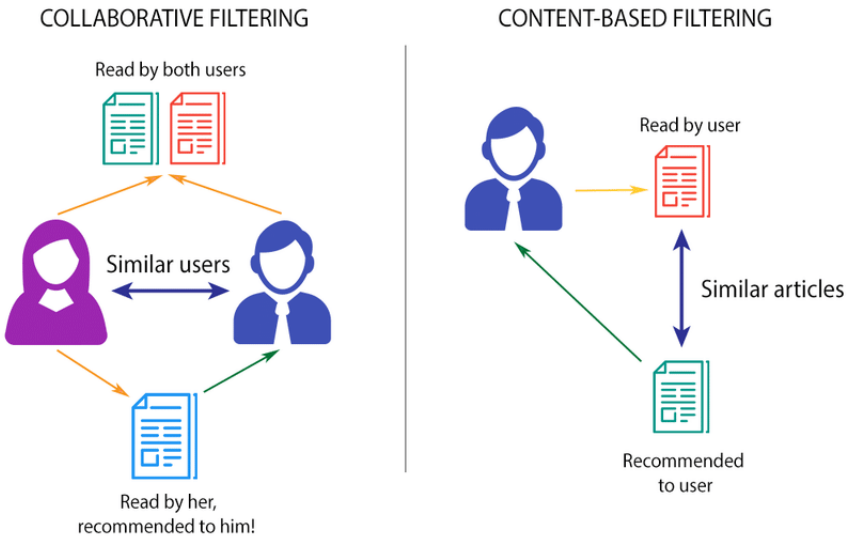


Figura 1: Filtro colaborativo vs. filtro basado en contenido

Los filtros basados en contenido solo recomiendan items similares a los ya consumidos por el usuario. Sin embargo, los filtros colaborativos pueden ofrecer nuevos artículos que no tienen que ver con lo ya consumido por el usuario en estudio, pero usuarios similares han valorado positivamente.

Generalmente, los filtros colaborativos son más precisos y además no se necesita de la creación de perfiles de usuarios e items. Esta afirmación no siempre se cumple ya que también hay que tener en cuenta el contexto del que provienen los datos y los objetivos del proyecto. Por otra parte, los filtros colaborativos sufren del ya mencionado problema del cold start además de introducir 'sparsity' y el inconveniente relacionado con la escalabilidad. Estos no se dan en los filtros basados en contenido porque al crearse perfiles, se pueden proporcionar sugerencias adecuadas aunque la información sea escasa. Entre las desventajas de los filtros basados en contenido se puede mencionar, en primer lugar, la necesidad de que exista una descripción detallada o información intrínseca (como en el caso de los documentos). En segundo lugar, los filtros basados en contenido no generalizan por lo que pueden pasar por alto items que, a pesar de no coincidir con el perfil del usuario, sí podrían interesarle.

2.4 Filtro basado en conocimiento

A pesar de que los filtros colaborativos y los basados en contenido son los más populares, existen otros muchos tipos de recomendadores. El filtro basado en conocimiento es uno de ellos. Este filtro se usa cuando no se tiene información acerca de las preferencias de un usuario ni items basados en textos. En cambio, sirve en los casos en los que se presenta un conocimiento profundo acerca de los items. Existen varios tipos, principalmente el *basado en restricciones* y el *basado en casos*. No sufre del ya mencionado problema del *cold start*, ya que puede funcionar con tan solo un usuario y un item mientras se cumplan las reglas, ni de 'sparsity' porque lo único que se evalúa es

si cumple o no las reglas establecidas.

Es un tipo de recomendador que se suele usar cuando el espacio de items es muy complejo o usado con menor frecuencia, como en el mercado inmobiliario o la venta de vehículos. El usuario establece unas reglas o guías que el item buscado debe cumplir. En el caso de las páginas web dedicadas a la venta de inmuebles, es el usuario el que configura ciertos filtros sobre el precio, número de baños y habitaciones, etc.

El *filtro basado en restricciones* consta de tres elementos principales:

- Variables: conjunto de características que se tienen en cuenta. Pueden ser las *propiedades del usuario*, es decir, los deseos de los posibles clientes como el precio dispuesto a pagar, o *propiedades del producto* entre los que se encuentran características de los items a comprar, como el tipo de combustible de un vehículo.
- Dominio: define los valores que pueden tomar las diferentes variables.
- Restricciones: proporcionan las posibles combinaciones de valores que pueden tomar las variables. Se pueden encontrar las *restricciones de compatibilidad*, que definen los valores permitidos para las propiedades de los usuarios, por ejemplo, si el coche es diésel, el cliente está dispuesto a pagar hasta 10.000 euros. Otras restricciones son las *condiciones de filtro*, que relacionan las propiedades del cliente con las del producto, por ejemplo si un usuario quiere un coche 4x4, todos los vehículos que cumplen esta restricción miden más de 1.75 metros de altura. Por último, existen las *restricciones del producto*, las cuales proporcionan la selección de productos disponible.

Por otra parte, también existen filtros de conocimiento *basados en casos*. En estos, el sistema proporciona un conjunto de items en función de unas medidas de similaridad entre los productos y una serie de restricciones. Se define una función que proporciona esta similaridad entre los objetos disponibles.

$$similaridad(p, REC) = \frac{\sum_{r \epsilon REC} w_r \times sim(p, r)}{\sum_{r \epsilon REC} w_r}$$

En la ecuación, p define un producto, REC es el conjunto de restricciones y r es cada una de ellas. El término w_r define el peso que toma cada restricción y *sim(p,r)* se refiere a la distancia entre un atributo del item y la restricción del usuario. Existen mejoras a esta fórmula ya que existen atributos que se quieren maximizar, como la autonomía de un vehículo y otros que se quieren minimizar, como el precio.

2.5 Filtro híbrido

Los filtros híbridos surgen para intentar cubrir las carencias que presentan los tipos de recomendadores ya vistos. En definitiva, el objetivo de su creación es conseguir recomendadores más robustos y precisos a través del uso de diferentes tipos de recomendadores. Principalmente se pueden encontrar tres enfoques:

- **Ponderado**: se basa en inferir las recomendaciones finales mediante la combinación de las predicciones producidos por otros recomendadores individuales. A

la hora de establecer los pesos para cada sistema, se pueden tomar las métricas utilizadas para conocer la precisión de cada uno de ellos, como el RECM.

- **Mezcla**: en este caso, las recomendaciones finales se hacen siguiendo un orden en el conjunto de sistemas de recomendación. Por ejemplo, si se tiene un filtro colaborativo y otro basado en contenido, la primera recomendación del sistema resultante podría ser la primera sugerencia del filtro colaborativo, la segunda será la primera del filtro basado en contenido. Después se escogería la siguiente recomendación del filtro colaborativo. Así se continúa hasta que se llegue al número deseado de items sugeridos.
- **Cascada**: la idea de este enfoque es ir aplicando diferentes recomendadores en secuencia. Es decir, se podría aplicar un filtro basado en conocimiento a una serie de items de manera que solo algunos de ellos cumplan las reglas establecidas en este filtro. A estos items resultantes se le puede aplicar luego otro tipo de sistema, como puede ser un filtro colaborativo. Así, se puede aumentar la eficiencia de los sistemas.

3 Metodologías

El objetivo en este apartado es mostrar de una manera más teórica cómo funcionan los diferentes sistemas de recomendación que se han expuesto con anterioridad y desglosar todos los tipos de enfoques que se tienen en cuenta en el momento actual.

3.1 Filtros basados en contenido

Anteriormente, se mencionó que los filtros basados en contenido basan su funcionamiento en las similaridades entre los perfiles creados para los items y los usuarios. Por lo tanto, la clave en estos es encontrar aquellos productos similares a los ya consumidos por el usuario en estudio.

Dado que este tipo de filtrado se suele utilizar para el análisis de items cuyo contenido es texto, se hace uso de la lista de palabras clave mencionada en el apartado anterior. Esta se crea a partir de la descripción del item o buscando entre las que más aparecen en el propio item. Evidentemente es más sencillo el primer enfoque ya que en el segundo se plantean problemas como las palabras compuestas o que las más repetidas son artículos y preposiciones. Además hay que contextualizar las palabras dentro de un documento, por ejemplo, la palabra *vegano* tendrá una connotación negativa en un texto en favor de la industria cárnica.

A la hora de conocer el grado de similaridad entre dos documentos se utilizan métricas como el coeficiente de Dice. Siendo B_i y B_j los items a comparar, su coeficiente de Dice es:

$$2 \times \frac{|palabras\_clave(Bi) \cap palabras\_clave(Bj)|}{|palabras\_clave(Bi)| + |palabras\_clave(Bj)|}$$

Según esta métrica, dos documentos serán más similares cuantas más palabras se solapen. Pero este enfoque no es muy acertado ya que dota de la misma importancia a todas las palabras y prioriza aquellos artículos más largos. La solución por la que se suele optar es tomar la métrica de TF-IDF(term frequency/frecuencia de palabra - inverse document frequency/frecuencia inversa del documento). Esta técnica consiste en establecer un espacio euclídeo con tantas dimensiones como términos. La coordenada de cada documento en las diferentes dimensiones será el producto de TF y IDF.

TF se refiere al número de apariciones de cada palabra clave teniendo en cuenta la longitud del documento. Para evitar los problemas previamente mencionados se realiza una normalización de esta cifra. Para ello se divide la frecuencia de cada palabra por la frecuencia máxima en el documento, es decir:

$$TF(i,j) = \frac{frecuencia(i,j)}{maxFrec(j)}$$

Donde *frecuencia(i,j)* es el número de apariciones de la palabra clave i en el documento j y *maxFrec(j)* es la frecuencia máxima de una palabra clave en el documento j.

El objetivo de IDF es reducir el peso que se otorga a las palabras clave que se repiten en exceso. La idea detrás de este término es dar mayor importancia a las palabras que aparecen en un número reducido de documentos.

$$IDF(i) = \log \frac{N}{n(i)}$$

Siendo N el número de documentos que se incluyen y $n(i)$ el número de documentos donde aparece la palabra i.

Así, el peso combinado TF-IDF se establece como:

$$TF\text{-}IDF(i,j) = TF(i,j) \times IDF(i)$$

De esta manera, un documento viene representado por un vector de pesos TF-IDF que indica la importancia de cada palabra clave en el documento y no por una lista booleana que indica solamente si la palabra está o no.

Teniendo esto una de las formas más usadas para hallar la similaridad entre documentos es la *medida de similaridad del coseno*, ya que la matriz de TF-IDF representa el espacio de todas las palabras y los documentos, cada columna es un vector con el peso dado a cada palabra. Para hallar la similaridad del coseno se realiza el producto interior de los dos documentos a evaluar dividido entre la norma de cada uno de ellos.

$$similaridad(A, B) = \frac{A \cdot B}{||A|| \times ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Teniendo las similaridades entre objetos y las valoraciones aportadas por los usuarios, se pueden crear predicciones para cada uno. Haciendo uso de técnicas como el *k-nearest neighbours* se pueden encontrar los objetos más cercanos a uno en concreto. Por ejemplo, si se quieren encontrar las recomendaciones para un usuario en concreto bastaría con recorrer todos los objetos no comsumidos y, para cada uno de ellos, encontrar los k más cercanos de los ya consumidos por el usuario en cuestión.

Una vez se han hallado los k items más cercanos, se analiza cuántos de ellos le han gustado al usuario y en función de la proporción, se recomenderá ese objeto o no. Si las valoraciones no son simplemente *Gustó* o *No Gustó* sino que presentan valores numéricos, se puede aplicar una media ponderando según la similaridad de los objetos al item en cuestión hallando así una predicción.

3.2 Filtros colaborativos

Los filtros colaborativos basan su funcionamiento en las interacciones de los usuarios entre ellos y con los items. A diferencia del filtro basado en contenido, es capaz de tratar aspectos de los datos más escondidos o complejos. Existen dos grandes divisiones dentro de los filtros colaborativos.

3.2.1 Basado en memoria

También conocido como *vecinos más cercanos*. Recomienda contenido a un usuario en función de lo consumido por los *vecinos más cercanos*. Estos son los usuarios con un patrón de búsquedas y valoraciones más afines a los del usuario en cuestión. Al igual que el filtro basado en contenido, el filtro colaborativo basado en memoria necesita de un conjunto de usuarios y de items. La relación principal que se tiene en cuenta entre ambos elementos es el rating/la valoración, que corresponde con una nota que el usuario ha asignado a cada item. Por lo tanto, una vez hallados los vecinos más cercanos al usuario en cuestión, se extrapolan los ratings de estos vecinos. Entre este tipo de filtrado se pueden distinguir dos clases diferentes: basado en usuario o basado en item.

- **Filtrado basado en usuario**: Se identifica a los k-vecinos más cercanos al usuario de estudio y se calcula la predicción para un item determinado. Para hallar la similaridad entre los usuarios y elegir así a los más cercanos al usuario en estudio es común hacer uso del coeficiente de correlación de Pearson.

$$similaridad(U_{\mathrm{a}}, U_{\mathrm{x}}) = \frac{\sum_{l_{\mathrm{i}} \epsilon LU_{\mathrm{c}}} (r_{\mathrm{a,l_i}} - \overline{r_{\mathrm{a}}}) \times (r_{\mathrm{x,l_i}} - \overline{r_{\mathrm{x}}})}{\sqrt{\sum_{l_{\mathrm{i}} \epsilon LU_{\mathrm{c}}} (r_{\mathrm{a,l_i}} - \overline{r_{\mathrm{a}}})^2} \times \sqrt{\sum_{l_{\mathrm{i}} \epsilon LU_{\mathrm{c}}} (r_{\mathrm{x,l_i}} - \overline{r_{\mathrm{x}}})^2}}$$

En la fórmula, U_a es el usuario del que queremos conocer los más cercanos y U_x se corresponde con el usuario con el que se va a comparar. LU_c es el conjunto de items que han sido valorados por ambos usuarios, $r_{j,\mathrm{l_i}}$ es el rating del usuario j al item l_i y $\overline{r_i}$ es la media de ratings del usuario i.

Los sistemas de recomendación se usan en proyectos con una cantidad de datos muy grande, pero este no es el mayor problema. Lo realmente problemático es que la matriz de estos datos contiene muchas celdas vacías ya que existen multitud de items que un usuario no ha consumido ni valorado. Volviendo sobre el ejemplo de Netflix, una persona concreta habrá podido visualizar una pequeña parte del catálogo de películas y series ofrecido por esta plataforma.

Se usa la similaridad entre el usuario de estudio y los vecinos para calcular las predicciones de los ratings que esta persona proporcionaría a un item concreto. El término *NN*, *nearest neighbours*, se refiere al conjunto de vecinos más cercanos al usuario en estudio según las medidas de similaridad explicadas anteriormente. Uno de los apartados a estudiar en caso de implementar un sistema de este tipo es qué valor de vecinos a considerar es el idóneo.

$$prediccion(U_a, item) = \overline{r_a} + \frac{\sum_{U_j \epsilon NN} similaridad(U_a, U_j) \times (r_{\mathrm{j,item}} - \overline{r_j})}{\sum_{U_j \epsilon NN} similaridad(U_a, U_j)}$$

- **Filtrado basado en item**: En este tipo de filtrado, se localizan los items que recibieron una valoración similar por parte del mismo usuario. La fórmula para hacerlo se extrapola de la usada con anterioridad para hallar la similaridad entre dos usuarios ya que se sigue usando el coeficiente de correlación de Pearson.

$$similaridad(l_\mathrm{a}, l_\mathrm{b}) = \frac{\sum_{u \epsilon U} (r_{\mathrm{u,l_a}} - \overline{r_{\mathrm{l_a}}}) \times (r_{\mathrm{u,l_b}} - \overline{r_{\mathrm{l_b}}})}{\sqrt{\sum_{u \epsilon U} (r_{\mathrm{u,l_a}} - \overline{r_{\mathrm{l_a}}})^2} \times \sqrt{\sum_{u \epsilon U} (r_{\mathrm{u,l_b}} - \overline{r_{\mathrm{l_b}}})^2}}$$

En este caso, U se refiere al conjunto de los usuarios que valoraron tanto al item $\mathrm{l_a}$ como a $\mathrm{l_b}$, $\mathrm{r_{u,l_x}}$ es la valoración del usuario u al item $\mathrm{l_x}$ y $\overline{r_{\mathrm{l_x}}}$ es la media de valoraciones del item $\mathrm{l_x}$.

De igual forma, para hallar la predicción del rating que un usuario en concreto proporcionará a cierto item, se utiliza una fórmula similar a la usada en el filtrado basado en usuario.

$$prediccion(U_a, item) = \frac{\sum_{it \epsilon NN} similaridad(item, it) \times r_{\mathrm{a,it}}}{\sum_{it \epsilon NN} similaridad(item, it)}$$

En esta ocasión los vecinos más cercanos (NN) son de items ya evaluados por el usuario en cuestión. Este análisis es similar al filtrado basado en contenido en cuanto a que se halla la similaridad de los items y no de los usuarios. La diferencia radica en la forma en que se hallan las similaridades ya que en este caso están basadas en las valoraciones proporcionadas por los usuarios mientras que en el filtrado basado en contenido, las similaridades se consiguen a través de las características de los items.

A la hora de hallar los vecinos más cercanos (NN), si el número de usuarios y/o items se mantiene constante, entonces se elegirá un valor fijo de vecinos a considerar. Cuando no es así, se establece un umbral de manera que si la similaridad lo supera se considerarán vecinos cercanos. Es más lógico usar el segundo enfoque ya que normalmente hay usuarios que se dan de baja y de alta e items nuevos que ofertar y otros que se retiran. En general, el filtrado basado en usuarios funciona mejor con conjuntos de datos de gran tamaño mientras que el basado en items proporciona mejores resultados en datasets más compactos.

Entre las ventajas del enfoque basado en memoria de los filtros colaborativos se encuentra la facilidad de implementación de estos así que como la claridad en los resultados proporcionados, los cuales son sencillos de explicar. Pero también podemos encontrar inconvenientos respecto a otros enfoques, el más determinante es la reducción de precisión cuando los datos son muy dispersos.

3.2.2 Basado en modelos

Dentro de este grupo entran los algoritmos que hacen uso de técnicas de machine learning para averiguar las valoraciones que los usuarios proporcionarían a items que todavía no han puntuado.

- **Factorización de matrices**
  En estos modelos se establecen una serie de factores latentes que puedan explicar ciertas características de los items [2]. Por ejemplo, si se toman 3 factores haciendo un análisis sobre películas, estos podrían ser: contenido de acción, época en la que está ambientada y cómo son los efectos especiales. En base a las valoraciones aportadas por los diferentes usuarios a los distintos items, el objetivo es encontrar en qué medida están representados esos factores tanto en los usuarios como en los items.

  Los factores latentes, por lo tanto, representan categorías que se dan en los datos. Así, se podría clasificar a los items y usuarios a partir de estos factores. A medida que aumente el número de factores latentes en el sistema, más específico será cada uno de ellos y más precisas serán las recomendaciones aportadas aunque

hay que tener cuidado de no caer en sobreajuste.

Para llevar a cabo un filtro colaborativo basado en factores latentes, uno de los métodos es la factorización de matrices. Mediante los patrones que se infieren de las valoraciones que realizan los usuarios, la factorización de matrices aporta información tanto de los usuarios como de los items a través de vectores de factores. La facilidad de escalado que ofrece junto con la flexibilidad y los resultados tan precisos hacen de este un método cada vez más en uso. Además, pueden ser utilizados en sistemas de tiempo real.

La información más valiosa para este tipo de sistema de recomendación es la información explícita que proporciona un usuario mediante la valoración de los items, llevar a cabo formularios o encuestas, o indicar sus preferencias, entre otros. Usualmente estos datos se reflejan en una matriz cuyas dimensiones son los usuarios y los items. El problema principal asociado a estas matrices es que son muy dispersas, es decir, hay muchos espacios en blanco ya que suele existir una gran mayoría de items con los que un usuario no ha interactuado. Alguno modelos basados en factorización de matrices permiten la inclusión de información implícita, entre las que se incluye el historial de compras o los patrones de búsqueda de un usuario, entre otros. Esta información implícita se representa mediante una matriz booleana que representa la presencia o ausencia de un evento, por lo que no es una 'sparse matrix'. Existen numerosas formas de llevar a cabo una factorización de matrices para realizar un filtro colaborativo [5].

- **SVD**: La descomposición en valores singulares (SVD) es una de las formas más conocidas a la hora de factorizar matrices. La matriz de ratings, R como matriz $n \times m$, puede ser descompuesta en tres matrices de menor dimensión:

$$\mathbf{R} = X\Sigma Y^{\mathrm{T}},$$

  donde Σ es una matriz diagonal de dimensión $n \times m$ cuyos elementos son los autovalores de la matriz $\mathbf{R^T R}$, y $\mathbf{X}$ e $\mathbf{Y}$ son matrices ortogonales de dimensiones $n \times n$ y $m \times m$, respectivamente.

  Sin embargo, la descomposición en valores singulares convencional requiere que la matriz sea semidefinida positiva, cosa que la matriz de ratings no cumple ya que es una matriz dispersa. Una solución al problema de la 'sparsity' es intentar rellenar los huecos vacíos mediante técnicas de imputación, como puede ser utilizar la media, moda u otro estadístico, o realizar una regresión. Estos métodos de imputación suelen ser muy costosos.

  Uno de estos métodos es el visto en la asignatura de *Métodos Estadísticos de Computación Intensiva*, llamado *softimpute*. Este es un método de completado de matrices. Su funcionamiento se basa en sustituir los elementos

vacíos de la matriz por los obtenidos por una descomposición en valores singulares de rango bajo. Esto es una aproximación al SVD en la que solo se toman un número reducido de autovalores y sus correspondientes autovectores.

El SVD utilizado en el campo de los filtros colaborativos es una aproximación de rango bajo al SVD convencional, también se conoce como *FunkSVD*, en honor a Simon Funk, la persona encargada de popularizarlo. El objetivo es conseguir dos matrices de rango menor que la matriz original que contengan los factores latentes de los usuarios y los items, respectivamente. El resultado de esta factorización es el siguiente:

$$R = U \cdot V$$

Así se consigue una matriz para los usuarios, U, y otra para los items, V. Estas matrices indican el valor de cada factor f que representa cada usuario e item, respectivamente, es decir, contienen los vectores de cada factor latente para los usuarios y para los items. Se puede ver la relación entre la matriz de ratings y las de usuarios e items en la figura **2**. Esta constituye la factorización más simple de todas.

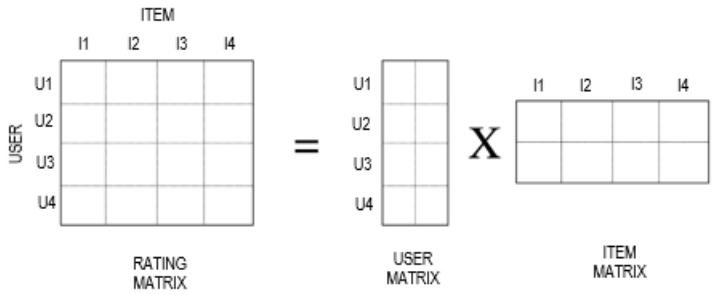


Figura 2: Factorización

El objetivo de los modelos de factorización de matrices será hallar las predicciones para cada valoración aportada por cada usuario a cada item.

$$r_{ij} = u_i \cdot v_j, \tag{1}$$

siendo u_i el vector de factores latentes del usuario i y v_j el vector de factores latentes del item j.

Para conseguir estas predicciones, se intenta minimizar la función de pérdida del error cuadrático medio. Para lidiar con la 'sparsity' de las matrices, solo se tienen en cuenta aquellas valoraciones observadas. Con el fin de evitar un posible sobreajuste, se introducen en el modelo términos de regularización L_2, que consiste en la suma de la norma de los vectores al cuadrado, multiplicado por un coeficiente de regularización. Los algoritmos usados para minimizar esta función objetivo se explican más adelante.

$$L = \sum_{(i,j)\epsilon K} (r_{ij} - u_i \cdot v_j)^2 + \lambda(||u_i||^2 + ||v_j||^2) \tag{2}$$

El término K hace referencia al conjunto de pares de usuarios e items para los que existe una valoración y λ es la constante que controla la regularización, la cual penaliza las magnitudes de los parámetros, tratando de reducir el sobreajuste. Este último término se puede determinar mediante validación cruzada.

Pero este modelo es demasiado sencillo y no tiene mucha capacidad de predicción. En (1) se trata de caputar las interacciones entre los usuarios y los items pero mucha de la variación que se recoge tiene que ver con diferentes efectos de los usuarios o los items, conocidos como **sesgos**. Un ejemplo claro es que no todos los usuarios valoran de la misma manera un item aunque tengan una percepción parecida de cuánto les ha gustado. Por esto, (1) es demasiado escueta para explicar por completo las valoraciones de los usuarios a los items, por lo que normalmente se añaden términos que intenten explicar estos sesgos para los usuarios o los items individualmente.

$$b_{ij} = \mu + b_i + b_j$$

Aquí, b_{ij} es el sesgo relacionado con los ratings r_{ij} y tiene en cuenta los efectos de los items y los usuarios. La media de las valoraciones se denota como μ y b_i y b_j reflejan las desviaciones de los usuarios y los items sobre la media, respectivamente. Es decir, si por ejemplo $b_i = 0{,}2$ quiere decir que este usuario hace unas valoraciones 0.2 por encima de la media. Estos sesgos se añaden a (1):

$$r_{ij} = u_i v_j + \mu + b_i + b_j \tag{3}$$

La ecuación se desglosa en los términos *interacción usuario - item*, *media global*, *sesgo del item* y *sesgo del usuario*. De esta manera, cada término

explica solo la parte referida a él. El sistema minimizará el error cuadrático regularizado que se da en la siguiente función.

$$L = \sum_{(i,j)\epsilon k} (r_{ij} - u_i \cdot v_j - \mu - b_i - b_j)^2 + \lambda(||u_i||^2 + ||v_j||^2 + {b_i}^2 + {b_j}^2) \quad (4)$$

– **SVD++**: Como se mencionó en la introducción de este apartado, existen algunas formas de factorizar matrices que incluyen información implícita acerca de los usuarios [2, 5]. Esta información puede ser, por ejemplo, el historial de búsqueda de un consumidor. Pero también se puede recabar otra información acerca de un usuario, como el género, la localización, el salario, etc. Por ejemplo, se podría configurar una variable N(u) booleana referida a un conjunto de items para los que el usuario u ha mostrado interés de forma implícita, por ejemplo, visitando el enlace de los mismos. Es necesario un nuevo conjunto de factores para los items, de manera que el item j está asociado a la variable $\mathrm{x_j} \epsilon \mathbb{R}^{\mathrm{f}}$. De esta manera, un usuario que ha mostrado interés de forma implícita por los items en N(u) está reflejado de la siguiente manera:

$$\sum_{j\epsilon N(u)} x_j$$

Es recomendable normalizar, de lo contrario los usuarios que hayan interaccionado con más items de forma implícita se verán representados por vectores con valores mayores. Una forma de hacerlo es la siguiente:

$$|N(u)|^{\text{-}0.5} \sum_{j\epsilon N(u)} {x_j}^{4.5}$$

Esta es información implícita que tiene en cuenta interacciones entre usuarios e items, pero también se puede incluir información exclusivamente acerca de un usuario. Se introduce una nueva variable lógica A(u), que describe los atributos considerados del usuario: género, grupo de edad, nivel de ingresos, etc. De nuevo, es necesario crear un vector de factores, $\mathrm{y_a} \epsilon \mathbb{R}^{\mathrm{f}}$, para cada atributo que describa a un usuario:

$$\sum_{a\epsilon A(u)} y_a$$

El modelo de factores latentes final debe incluir estos términos referidos a datos de fuentes de información adicionales.

$$r_{ij} = [u_i + |N(i)|^{-0.5} \sum_{j \epsilon N(i)} {x_j}^{4.5} + \sum_{a \epsilon A(i)} y_a] \cdot v_j + \mu + b_i + b_j$$

Este modelo trata de mejorar la percepción de los usuarios incluyendo información implícita acerca de estos pero también se puede aplicar este desarrollo a la inclusión de más cantidad de datos sobre los items.

Existen otros tipos de factorización de matrices como AsymmetricSVD, enfoques híbridos o factorización basada en redes neuronales y Deep Learning.

Uno de los aspectos más positivos del filtro basado en factores latentes es su flexibilidad a la hora de trabajar con diferentes características de los datos y otros requerimientos específicos. Por eso, los modelos expuestos pueden ser mejorados incluyendo otros aspectos de los datos.

Una de las casuísticas a tratar es el tiempo. Hasta el momento, todos los modelos vistos son estáticos, pero en la realidad las preferencias de los usuarios pueden variar y el número de items que se van consumiendo es cada vez mayor. Esto se puede añadir fácilmente a la factorización de matrices y puede ayudar a mejorar la precisión.

Para añadir la temporalidad a la ecuación hay que estudiar qué términos pueden variar en el tiempo. Se va a partir de (3), que incluye los sesgos y es la más completa sin llegar a una complicación excesiva. El término v_j no varía en el tiempo ya que, al referirse a los factores para un item, se entiende que no se modifican en el tiempo porque las características de estos no varían. El término u_i, en cambio, sí se modifica con el paso del tiempo ya que los gustos de las personas pueden variar o noticias relacionadas con ciertos items pueden cambiar la percepción que los usuarios tienen de estos. En cuanto a los términos del sesgo, μ evidentemente no cambia con el paso del tiempo ya que se refiere a la media de las valoraciones que se da a los items, no debería variar. El sesgo para los items sí que cambia ya que la popularidad de los items se rige, a grandes rasgos, por modas, y las valoraciones de estos pueden cambiar. Lo mismo pasa con el sesgo de los usuarios, los cuales pueden modificar los ratings que proporcionan y desviarse de la media en una u otra dirección. Estas modificaciones dejarían la ecuación mencionada de la siguiente forma:

$$r_{ij}(t) = u_i(t) \cdot v_j + \mu + b_i(t) + b_j(t)$$

Otro de los aspectos a tratar es la confianza en la información que con la que se trabaja. Hay ocasiones en las que no se puede saber con seguridad si a un

usuario realmente le ha gustado un item, por ejemplo, cuando un producto ha sido anunciado masivamente o en el caso de que solo se trabaje con información implícita. En el segundo caso se suele tener en cuenta el número de visitas que recibe un item determinado por el usuario en cuestión, es decir, si recibe una visita esporádica no se puede saber si le gusta pero si las visitas son constantes se va aumentando la confianza en que realmente le guste. Por esto, se puede añadir en el modelo la confianza que se tiene de que un rating proporcionado por el usuario i al item j refleje la realidad, se denota como c_{ij}.

$$min_{u,v,b} \sum_{(i,j)\epsilon K} c_{ij}(r_{ij} - u_i \cdot v_j - \mu - b_i - b_j)^2 + \lambda(||u_i||^2 + ||v_j||^2 + {b_i}^2 + {b_j}^2)$$

3.2.3 Minimizar una función de pérdida

Quedaba en el aire la forma en que se optimizan las funciones objetivo de los distintos modelos de factorización de matrices. Se van a exponer las formas que se utilizan actualmente para entrenar estos modelos. Principalmente, se pueden encontrar dos enfoques diferentes:

1. Descenso de gradiente estocástico (SGD) [6]: el denominado SGD sirve para optimizar una función objetivo con las pertinentes restricciones de suavizado. Es una aproximación estocástica al descenso de gradiente, el cual es otro método de optimización con grandes necesidades computacionales. El SGD reduce esta complejidad al llevar a cabo iteraciones menos exigentes, sin embargo, el ratio de convergencia es algo menor. En el SGD, los parámetros y el factor de aprendizaje del modelo se van actualizando cada vez que se trata una observación del conjunto de entrenamiento. Sin embargo, en el descenso de gradiente se requiere el paso por todas las observaciones del conjunto para actualizar los parámetros del modelo.

   Se opera sobre cada caso del conjunto de entrenamiento para poder obtener el error cuadrático medio entre el rating verdadero y el estimado.

   $$e_{ij} = r_{ij} - u_i \cdot v_j$$

   Para cada iteración se actualizan los valores de los vectores de factores latentes. A partir de (2), se resta el valor de la derivada respecto a cada uno de estos dos elementos multiplicado por un factor de aprendizaje γ. Para u_i, el procedimiento es el siguiente:

   $$u_i = u_i - \gamma \cdot \frac{\partial L}{\partial u_i}$$

   $$\frac{\partial L}{\partial u_i} = -2 \cdot (r_{ij} - u_i \cdot v_j) \cdot v_j + 2 \cdot \lambda \cdot u_i$$

El 2 que está multiplicando se recoge en el factor de aprendizaje. Así, los vectores de factores latentes para los usuarios se actualizan en cada iteración de la siguiente forma:

$$u_i = u_i + \gamma \cdot (e_{ij} \cdot v_j - \lambda \cdot u_i)$$

Siguiendo el mismo procedimiento se actualizan los vector de factores latentes para los items.

$$v_j = v_j + \gamma \cdot (e_{ij} \cdot u_i - \lambda \cdot v_j)$$

Este tipo de optimización solo asegura un mínimo local. Este enfoque presenta una gran sencillez para su implementación además de una ejecución muy rápida.

2. Alternating Least Squares (ALS): en (2) se encuentra la función a minimizar, la cual se corresponde con la suma de los errores. Al ser desconocidos tanto u_i como v_j, esta función de coste es no convexa, lo que indica que tiene numerosas soluciones posibles. Por esta razón, es muy difícil operar sobre ella.

   La técnica de Alternating Least Squares [7] trata de solventar el hecho de que este sea un problema no convexo. Para ello, se fija U de manera que se tiene un problema convexo para V. De igual forma, se podría fijar V para operar sobre la función para U.

   A continuación se detallan los pasos a seguir para llevar a cabo el enfoque de ALS.

   - Se inicializan U y V de forma aleatoria.
   - Se fija una de las dos matrices, en este caso, V. Para cada usuario i, se actualizan los valores de la matriz U

     $$u_i = (\sum_{r_{ij} \epsilon r_{i\cdot}} v_j v_j^T + \lambda I_f)^{-1} \sum_{r_{ij} \epsilon r_{i\cdot}} r_{ij} v_j$$

     f es el número de factores latentes e I es la matriz identidad.
   - Se fija la otra matriz, U, y se actualizan los pesos de la matriz V para cada item j.

     $$v_j = (\sum_{r_{ij} \epsilon r_{\cdot j}} u_i u_i^T + \lambda I_f)^{-1} \sum_{r_{ij} \epsilon r_{\cdot j}} r_{ij} u_i$$

     f es el número de factores latentes e I es la matriz identidad.

En vez de fijar primero V y luego U, se podría haber hecho en el orden inverso.

Normalmente se usa el descenso estocástico del gradiente por su facilidad y rapidez pero hay dos casos en los que puede ser recomendable hacer uso de ALS.

(a) Cuando el sistema puede fijar uno de los dos términos u_i o v_j para calcular el otro, es decir, cuando el sistema permita la paralelización.

(b) Cuando se usa exclusivamente información implícita. SGD no es tan eficiente en este caso porque itera sobre cada observación del conjunto de entrenamiento debido a que la matriz no es dispersa.

- **Modelos probabilísticos**
  Existen formas de implementar sistemas de recomendación basados en la teoría de la probabilidad [3]. Una manera simple de hacerlo es extrapolando el problema a uno de clasificación de manera que la tarea sea asignar un item a una categoría de entre unas dadas. Este tipo de recomendadores son los que se pueden usar a la hora de encontrar spam en el correo electrónico. Para llevarlo a cabo se tienen en cuenta las palabras que aparecen en mensaje o en el asunto del mismo para proporcionar la probabilidad de que sea o no spam.

  Otra forma de implementar un sistema de recomendación es mediante clasificadores bayesianos. En este caso se hace uso del teorema de Bayes para hallar la probabilidad de que un item reciba una nota concreta por parte de un usuario determinado, la *probabilidad a posteriori*. Si se quiere hallar la probabilidad de que esa nota sea 1, $P(Y = 1|X)$, se necesita la probabilidad de que se evalúe con un 1 ese item, $P(Y = 1)$, la probabilidad condicionada $P(X|Y)$ y la probabilidad de X, $P(X)$. La variable X refleja el resto de valoraciones aportadas por el usuario en cuestión. Por lo tanto, el teorema de Bayes es:

$$P(Y|X) = \frac{P(X|Y) \times P(Y)}{P(X)}$$

  Aunque no lo sean, se asume que las valoraciones proporcionadas por los usuarios son independientes ya que así se puede aplicar un clasificador *Naive Bayes*, el cual funciona bien a pesar de este hecho. Con este clasificador se puede hallar la probabilidad a posteriori para cada valor de Y, siendo d el número posible de valores.

$$P(Y|X) = \frac{\prod_{i=1}^{d} P(X_i|Y) \times P(Y)}{P(X)}$$

  $P(X)$ es fija porque se trata de las valoraciones ya conocidas del usuario que se quiere estudiar, por lo que se puede eliminar. Para hallar $P(Y)$ basta con conocer el resto de puntuaciones de todos los usuarios. Por ejemplo, si se quiere saber $P(Y = 1)$, hay que saber qué ratio del resto de usuarios han proporcionado esa

nota, esa será la probabilidad. Para hallar las probabilidades a priori, se necesitan las probabilidades condicionando para el valor de Y que se desee. Por ejemplo, $P(X|Y = 1)$ se halla multiplicando todas las probabilidades condicionando a que $Y = 1$. Por ejemplo, si se tienen otros dos items aparte de Y cuyos valores posibles van de 1 a 3, $P(X|Y = 1) = P(X_1 = 1|Y = 1) \times P(X_2 = 1|Y = 1) \times P(X_1 = 2|Y = 1) \times P(X_2 = 2|Y = 1) \times P(X_1 = 3|Y = 1) \times P(X_2 = 3|Y = 1)$. Cada una de las probabilidades que componen la fórmula corresponde con el ratio de usuarios que ha proporcionado esa nota, es decir, si ha habido 5 usuarios que han dado un 1 a Y, de los cuales 2 han dado un 1 a X_1, $P(X_1 = 1|Y = 1) = \frac{2}{5}$. Aplicando el teorema de Bayes, el sistema proporcionará como predicción aquella que tenga mayor probabilidad a posteriori.

Existen otros muchas formas de implementar modelos probabilísticos para filtros colaborativos. El que se ha explicado es tan solo un pequeño ejemplo de cómo se podría realizar pero en la realidad son modelos con una complejidad mayor.

En general, los modelos probabilísticos suelen obtener mejores resultados, especialmente en algunos dominios. Aunque para conjuntos de datos referidos a valoraciones de películas suelen funcionar otro tipo de filtros colaborativos, como los de memoria basados en usuario o los de factorización de matrices.

Los modelos probabilísticos que utlizan clasificadores Naive Bayes requieren menos tiempo que los basados en usuario y son capaces de ignorar atributos irrelevantes además de ser resistentes al sobreajuste.

- **Clustering**
  El clustering es una técnica perteneciente al grupo de aprendizaje no supervisado ya que no se tiene conocimiento a priori, es decir, un atributo que diga de qué clase es cada observación. Trata de agrupar las diferentes observaciones según las similaridades. Por esto, la idea detrás de los algoritmos de clustering en los sistemas de recomendación es la misma que la de los filtros colaborativos basados en memoria.

  El objetivo es calcular las similaridades entre los usuarios y/o items para usarlas como pesos a la hora de realizar las predicciones de valoraciones. La diferencia frente al enfoque basado en memoria radica en la forma en la que se hallan las similaridades y, por tanto, los vecinos más cercanos. En el caso del clustering, se usan algoritmos de aprendizaje no supervisado como *k nearest neighbours* [6] en vez de la correlación de Pearson que usa en el filtrado basado en memoria.

- **Redes neuronales**
  También han surgido, como en todos los ámbitos de computación, redes neuronales para implementar sistemas de recomendación. El funcionamiento es similar al de la factorización de matrices. La diferencia es que se sustituye el producto de las matrices de usuarios e items por una arquitectura neuronal que aprenda una función sobre los datos. Esta red se conoce como *Neural Collaborative Filtering*

[8], y puede servir para generalizar la factorización de matrices. Sin embargo, este tipo de redes neuronales no tienen tanto reconocimiento como la factorización de matrices a pesar de que existen artículos que demuestran una precisión superior.

En el artículo [8] se describe una red neuronal que consta de 4 capas. La primera es la capa de entrada, la cual contiene lo que pueden considerarse como vectores de usuarios e items utilizando el one-hot encoding. De esta pasan a la capa que contiene los vectores de factores latentes de usuarios e items. Estas capas alimentan a una red neuronal multicapa, las cuales infieren las relaciones usuario-item. Por último, se tiene la capa de salida que proporciona la valoración esperada por el usuario estudiado para el item que se introdujo enla red.

Los filtros colaborativos basados en modelos tienen la ventaja principal frente a otros enfoques de que no se ven tan afectados por la 'sparsity' de las matrices o la ausencia de valores. En cambio, la complejidad aumenta considerablemente debido al carácter desconocido de los factores.

4 Ejemplo práctico

Para poder ilustrar de una mejor forma cómo funciona uno de estos sistemas de recomendación, se procede a mostrar un ejemplo práctico. La primera tarea que se pretendía llevar a cabo en este TFG consiste en analizar los diferentes conjuntos de datos que se pueden encontrar por Internet y elegir uno o varios para trabajar. También se pretendía elaborar el sistema teniendo en cuenta el software del que se hace uso. El siguiente paso, más concreto, contempla la elección del tipo de sistema y, una vez implementado, se realizan gráficos y tablas para poder analizar mejor los resultados y compararlos según la configuración de los parámetros del sistema.

4.1 Datos

La elección del conjunto de datos es una tarea fundamental en el correcto desarrollo de un ejemplo de este tipo. Lo más fácil de encontrar son datos acerca de valoraciones de películas. A raíz del concurso organizado por Netflix, han surgido una gran cantidad de conjuntos de este tipo.

Se pueden encontrar conjuntos como el de la radio alemana Last FM, que contiene valoraciones de usuarios a diferentes canciones. Existe otro dataset un tanto curioso que consta de valoraciones de unos usuarios a un conjunto de chistes, este conjunto se llama Jester. La gran mayoría de los datasets que se pueden encontrar contienen información explícita, como son las valoraciones. Finalmente, para la demostración se ha escogido el conocido conjunto de *MovieLens*, el cual representa valoraciones aportadas por los distintos usuarios a una serie de películas.

Entre las razones que han llevado a la elección de este conjunto están la facilidad de interpretación, ya que se puede determinar de forma clara cuando ha gustado la

película y cuando no, así como la posibilidad de elegir diferentes tamaños. De esta forma, se pueden realizar varios análisis diferentes sobre un mismo tipo de conjunto.

Los conjuntos de MovieLens constan de cuatro archivos. Dos de ellos, *links* y *tags* no se van a usar. Muestran de donde se han obtenido las valoraciones y cómo describen las películas, respectivamente.

Los otros dos archivos son *ratings* y *movies*. El primero de ellos es el principal, contiene las valoraciones que los usuarios proporcionan a las diferentes películas. Tiene la forma que se ve en la figura **3**.

userId	movieId	rating	timestamp
1	1	4	964982703
1	3	4	964981247
1	6	4	964982224
1	47	5	964983815
1	50	5	964982931

Figura 3: Archivo **ratings**

La primera columna contiene los identificadores de usuario, la segunda los identificadores de las películas, la tercera la propia valoración (de 0 a 5) y la cuarta columna contiene el momento en que se realizó la valoración (no se usará en este ejemplo). El otro archivo relaciona los identificadores de las películas con su nombre además de proporcionar los estilos en los que se enmarcan estas. Sus columnas son: identificador de película, nombre de la película y géneros. Se ve una pequeña muestra en la figura **4**.

movieId	title	genres.
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy;
2	Jumanji (1995)	Adventure|Children|Fantasy;
3	Grumpier Old Men (1995)	Comedy|Romance;
4	Waiting to Exhale (1995)	Comedy|Drama|Romance;
5	Father of the Bride Part II (1995)	Comedy;

Figura 4: Archivo **movies**

Este último archivo dio problemas ya que es un **csv** separado por comas, pero también aparecían comas en el título de algunas películas. Fue necesario hacer uso del comando de linux **sed** junto con expresiones regulares para poder eliminar todas esas comas innecesarias que impedían leer el archivo de forma correcta.

La propia implementación del sistema de recomendación se realiza sobre el archivo de *ratings*, el cual contiene las valoraciones y los identificadores de los usuarios y las películas, lo necesario para poder hacer un sistema de recomendación basado en

información explícita. El de *movies* sirve para, una vez realizado todo el proceso, poder relacionar los diferentes usuarios con una lista de películas favoritas o poder hacer nuevas recomendaciones. También se puede buscar el significado de los factores latentes observando cuáles tienen pesos mayores para según qué películas o usuarios y, posteriormente, asociarlos a un género.

4.2 Software

Para la correcta realización de este apartado práctico se han contemplado principalmente dos lenguajes de programación: *pyhton* y *R*. Ambos han sido de los más trabajados durante la formación académica, razón por la que han sido los considerados esencialmente para este trabajo. Al ser este un problema estrictamente matemático/estadístico en el que no se necesita de paquetes concretos o difíciles de encontrar, se toma la decisión de realizar la implementación final con R. Se ha probado la implementación básica en ambos lenguajes y, a pesar de que los tiempos de lectura son mejores en Python, la ejecución del propio algoritmo es más eficiente en R.

Como ya se ha mencionado, la implementación se ha realizado en lenguaje *R* a través del entorno de RStudio. Este entorno permite, de una manera sencilla, la administración de los paquetes necesarios para llevar a cabo la tarea. Además, se pueden visualizar los archivos con los que trabajar y las variables que están siendo utilizadas para la creación del sistema de recomendación.

4.3 Tipo de sistema

Otra de las cuestiones que más hay que meditar en este caso es cuál de todos los sistemas de recomendación que se han explicado en este documento se quiere implementar. Sin duda alguna, los más sencillos son los sistemas de recomendación colaborativos basados en memoria ya que se basa en hallar las correlaciones entre los usuarios e items para clasificarlos según un grado de similaridad. Lo mismo pasa con los algoritmos de clustering. En cuanto a los sistemas de recomendación basados en contenido, además de que no gozan de tanta popularidad porque su rendimiento suele ser algo menor, para este problema concreto no funcionaría de una manera idónea al tratarse de valoraciones sobre películas sin ningún contenido textual. Se podría usar en el caso de querer clasificar los usuarios o los items según su similaridad de gustos, pero para el caso de predecir una valoración hay otras posibilidades mejores.

Los sistemas restantes se encuadran todos dentro de los filtros colaborativos basados en modelos. Debido a la fama adquirida al ser el sistema vencedor en el consurso de Netflix, se ha optado por realizar un filtro colaborativo mediante la factorización de matrices. Además, supone un reto a la hora de implementarlo y llegar a entender su funcionamiento completo. Como ya se ha mencionado, existen dos formas diferentes de llevar a cabo la optimización de este algoritmo: *alternating least squares* y *descenso de gradiente*. Se realizado la segunda forma por su sencillez y menor tiempo de ejecución, además de su mayor uso en otras áreas.

4.4 Implementación

Una vez en este punto en el que se ha determinado que se va a realizar un filtro colaborativo basado en factorización de matrices sobre los archivos de MovieLens y haciendo uso de R, es necesario codificar las instrucciones y explicar cuáles han sido las decisiones tomadas en cuanto a los parámetros del modelo.

La implementación se ha realizado a partir de los contenidos de [9].Se realiza la implementación de un sistema capaz de predecir las valoraciones que los usuarios darían a ciertas películas. Se parte del modelo que incluye los sesgos de los usuarios y de los items, (3). Ya se conocen las fórmulas para la actualización de los vectores de factores latentes, pero en esta ocasión también es necesario actualizar los vectores de los sesgos. Para llevar a cabo esta tarea hay que realizar el mismo procedimiento que se usó en el caso de las matrices de factores latentes.

$$b_i = b_i - \gamma \cdot \frac{\partial L}{\partial b_i}$$

$$\frac{\partial L}{\partial b_i} = -2 \cdot (r_{ij} - u_i \cdot v_j - \mu - b_i - b_j) + 2\lambda \cdot b_i$$

$$e_{ij} = r_{ij} - u_i \cdot v_j - \mu - b_i - b_j$$

$$b_i = b_i + \gamma \cdot (e_{ij} - \lambda \cdot b_i)$$
$$b_j = b_j + \gamma \cdot (e_{ij} - \lambda \cdot b_j)$$

Las actualizaciones para los vectores de factores latentes quedan de la misma forma que en el modelo sencillo explicado anteriormente:

$$u_i = u_i + \gamma \cdot (e_{ij} \cdot v_j - \lambda \cdot u_i)$$
$$v_j = v_j + \gamma \cdot (e_{ij} \cdot u_i - \lambda \cdot v_j)$$

Los valores otorgados a la constante de regularización, λ, y al factor de aprendizaje, γ, se elegirán mediante un proceso de repetición de las diferentes ejecuciones similar a validación cruzada. Se ha considerado llevar a cabo un proceso de validación cruzada pero era necesaria la fragmentación de los conjuntos según según cada usuario y no asegura un mejor resultado. Se ha considerado que el factor de aprendizaje se reduzca al finalizar cada paso con el fin de hacer las transformaciones menores cada vez.

Respecto a la inicialización de las matrices U y V, las que contienen los vectores de factores latentes, se hará de forma aleatoria con valores entre 0 y 1. Hay que recordar que U tiene n filas y f columnas mientras que V tiene f filas y m columnas, siendo **n** el número de usuarios, m el número de películas y f el número de factores. Los valores de los sesgos se inicializan a 0, siendo b_i un vector de tamaño n y b_j tiene tamaño m.

El último paso antes de realizar la propia implementación del sistema de recomendación es escoger los conjuntos de entrenamiento y test. Para ello se han tomado

muestras aleatorias correspondientes al 10 % de todas las películas evaluadas por cada usuario. Se toma esta decisión ya que muestras mayores del conjunto de entrenamiento no conducían a resultados mejores, como se verá más adelante. También se ha probado la opción de tener en cuenta solo aquellos usuarios que han valorado más de 20 películas con el fin de reducir los efectos del cold start, pero los resultados eran peores, seguramente por la reducción de muestras de entrenamiento.

Uno de los grandes inconvenientes en este tipo de problemas es la 'sparsity' de las matrices, como ya se ha mencionado en varias ocasiones. El archivo de *ratings* no presenta este problema ya que viene en un formato diferente pero para la ejecución del sistema de recomendación es necesario que se lleve esta información a una matriz con tantas filas como usuarios y columnas como películas. En este caso sí que se da el problema de la dispersión. No es un gran inconveniente, el tipo de SVD implementado toma solo los pares (usuario, item) con valores observados, actualizando los factores latentes y los vectores de sesgos correspondientes, ignorando aquellas observaciones que presentan `NAN`.

En este punto, se plantean dos apartados diferentes con el fin de sacar el máximo provecho de los datos y poder comparar la implementación realizada con una real.

4.4.1 Comparación del número de factores y análisis de usuarios

Una de las ideas principales del proyecto es conocer cuántos factores es necesario utilizar para conseguir los mejores resultados. Se ha escogido el archivo de MovieLens más pequeño por las complicaciones computacionales, de nombre *ml-latest-small*. Este cuenta con 100836 valoraciones diferentes proporcionadas por 610 usuarios sobre 9742 películas. Se realiza con este archivo con el objetivo de minimizar los tiempos de ejecución ya que lo que prima en este apartado es la comparación capturando un número diferente de factores.

Dado que la inicialización de las matrices de factores latentes y vectores de los sesgos influye en el funcionamiento del sistema, se han realizado varias ejecuciones para la misma configuración de los parámetros. Una vez se tenga la mejor de ellas, se llevará a cabo un análisis de algunos usuarios, teniendo en cuenta cómo de acertado ha sido el sistema y qué recomendaciones se le pueden enviar a estos usuarios. Además, también se intenta interpretar el significado de algún factor, en el caso de que el sistema más eficaz no tenga demasiados.

4.4.2 Comparación con un sistema de recomendación ya existente

El paquete de R *recommenderlab* contiene funciones capaces de realizar un sistema de recomendación de diferentes maneras, desde filtros colaborativos basados en usuarios o items hasta factorización de matrices, como el caso que se ha implementado en el proyecto.

El objetivo es realizar alguna ejecución con funciones de este paquete y comparar los resultados proporcionados con los del sistema implementado para poder ver cómo

de diferente es la ejecución y si los parámetros utilizados son similares.

4.5 Resultados y análisis

A continuación se muestran los resultados obtenidos a partir del sistema y la forma en que se ha llegado a ellos. Se comparan distintas configuraciones para el filtro colaborativo implementado escogiendo las mejores para comprobar su comportamiento con ciertos usuarios y analizar nuevas recomendaciones que se les pueden sugerir.

4.5.1 Comparación del número de factores y otros parámetros

El objetivo principal de este apartado práctico es comprobar cuál es el número más adecuado de factores latentes que incluir en un modelo de factorización de matrices a la hora de implementar un filtro colaborativo.

La primera tarea de la que hay que hacerse cargo es de los conjuntos de entrenamiento y test. Para cada usuario, se añade al conjunto de entrenamiento una muestra aleatoria del 10 % del total de películas a las que ha establecido una valoración. En la tabla 1 se muestran los RECM de las 5 mejores combinaciones tomando una muestra del 20 %. En estas solo se ha realizado una ejecución por configuración para agilizar el proceso, pero se comprueba que los valores hallados están en torno a los valores hallados con 10 % que se verán a continuación.

Factores	γ	λ	**RECM**
30	0.030	0.15	0.8324415
30	0.025	0.15	0.8366446
100	0.020	0.10	0.8380039
100	0.020	0.15	0.8397046
30	0.015	0.15	0.8399644

Tabla 1: RECM con muestra de test de 20 % de tamaño

A cada paso que recorre el sistema, es decir, cada vez que ha pasado por todas las observaciones del conjunto de entrenamiento, se disminuye el valor del factor de aprendizaje para llevar a cabo un proceso adaptativo ya que a medida que se vaya avanzando en la ejecución las diferencias con respecto a los valores óptimos serán menores, haciendo necesarias modificaciones menos drásticas. Para llevar a cabo esta disminución se multiplica por 0.99 haciéndolo ligeramente más pequeño. El número de pasos que se recorren en cada ejecución se ha fijado en 50 ya que, como se ve posteriormente, a partir de esta iteración o incluso antes el error adquirido apenas mejora. Además, se ha incluido la condición de que si la diferencia entre el RECM de un paso y el anterior es menor que 0.00001, la ejecución para y se devuelven las matrices de factores latentes y vectores de sesgos conseguidos hasta ese momento. Se ha hecho esto porque el algoritmo SGD solo encuentra óptimos locales por lo que no es capaz de mejorar sustancialmente a partir de este punto y ya no encuentra nada significantemente mejor.

Hay ciertas cosas a tener en cuenta antes de comenzar a comparar el número de factores. En un principio, la idea era realizar una plantilla con varios valores diferentes para γ y para λ y ver cuál era la mejor combinación. Pero no todos los valores diferentes de factores latentes se comportan igual en lo referido a γ y λ. Por esto, ha sido necesario realizar diversas ejecuciones para cada valor de factores teniendo en cuenta diferentes valores para estos parámetros. Otro aspecto que no se debe escapar es que la precisión del sistema puede depender de la inicialización de las matrices de factores latentes y vectores de sesgos y de los conjuntos de entrenamiento y test, por ello se han realizado 5 ejecuciones para cada configuración de parámetros de manera que se pueda disponer de una perspectiva más específica de cada una de ellas.

Por lo tanto, sabiendo esto, se ha establecido una función que recorra las diferentes configuraciones y llame a la función principal que ejecuta el sistema. Los valores probados para γ son los valores comprendidos entre 0.015 y 0.03, con un paso de 0.005. En cuanto a los valores de λ, son los correspondientes a la secuencia entre 0.1 y 0.25 con paso de 0.05. Se han probado 5, 30 y 100 factores latentes.

En la figura 5 se muestran la media de los mejores RECM obtenidos para cada combinación del factor de aprendizaje y constante de regularización cuando el número de factores latentes es 5. No parece que exista ninguna relación entre las diferentes configuraciones de parámetros. El mínimo se alcanza cuando γ =0.03 y λ =0.15, obteniendo 0.847.

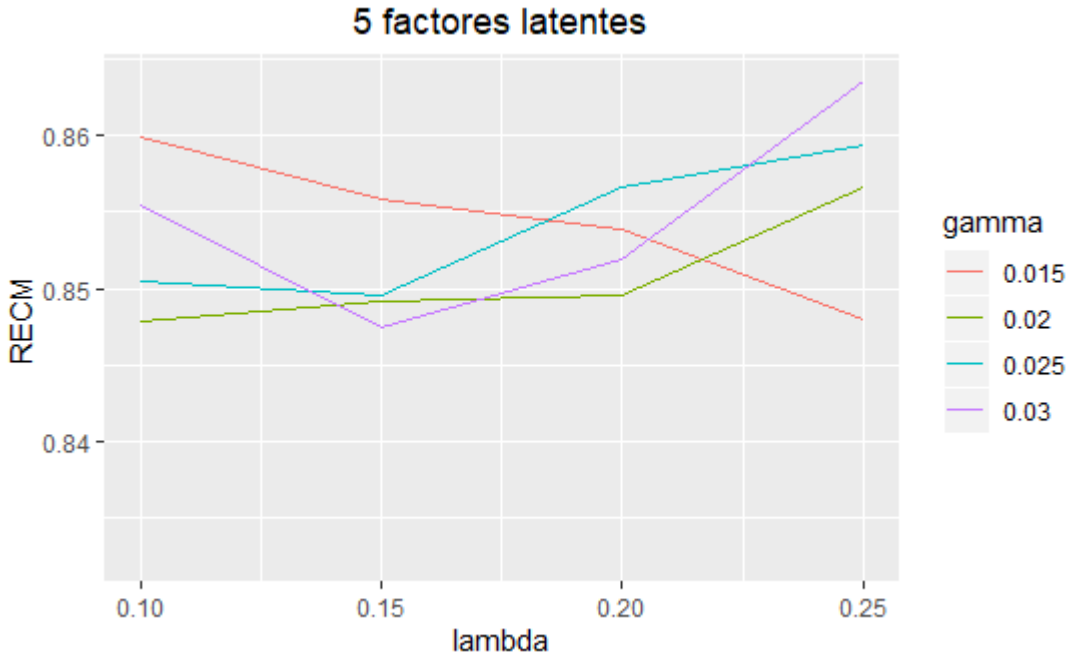


Figura 5: RECM con 5 factores latentes

La figura 6 presenta los errores para las diferentes combinaciones fijando el número

de factores latentes en 30. En esta ocasión, se intuye una relación entre todas las combinaciones siendo mayor entre las que tienen un factor de aprendizaje de 0.015, 0.02 y 0.03. Se distingue que la de 0.03 es algo mejor que las otras configuraciones. Todas alcanzan el mínimo en $\lambda = 0.15$ siendo la menor de ellas las de γ =0.03 con un RECM de 0.832.

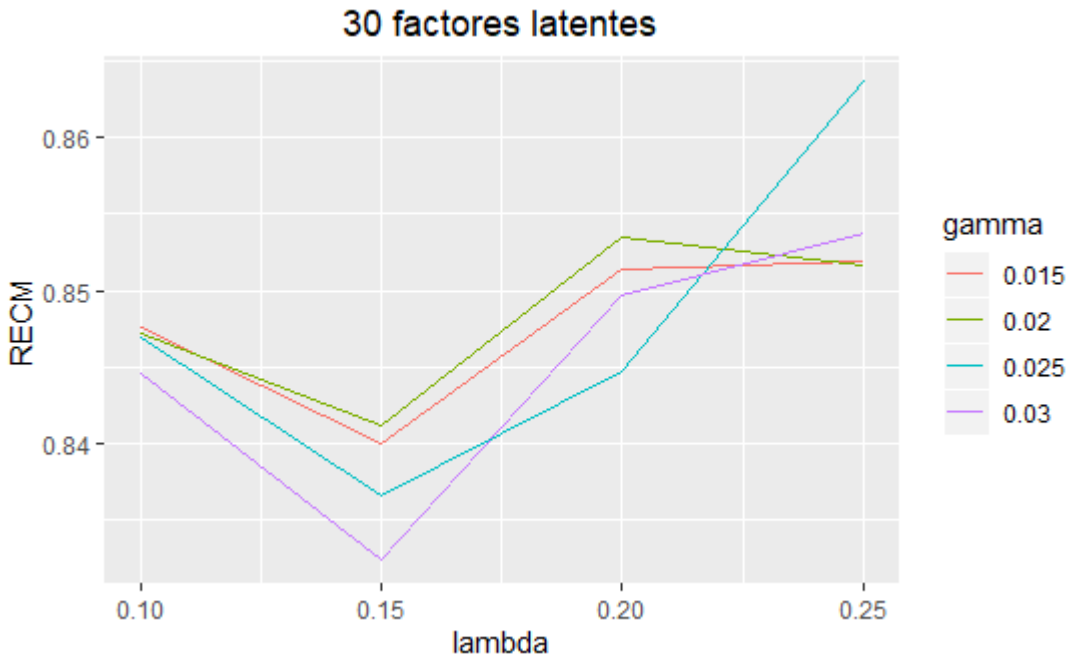


Figura 6: RECM con 30 factores latentes

Por último, se estudia el impacto de las distintas configuraciones cuando el número de factores latentes es igual a 100. Se puede ver en la figura 7. En esta ocasión solo ha sido posible analizar 3 valores de γ ya que, al tener tantos factores latentes, los primeros errores hallados para realizar las modificaciones del SGD cuando el factor de aprendizaje es 0.03 son muy grandes y no es capaz de computarlos. Se aprecia una ligera relación en la tendencia de los errores, especialmente cuando el factor de aprendizaje es 0.015 y 0.02. En esta ocasión, el mejor resultado se alcanza en γ =0.02 y λ =0.10.

Generalmente, el menor RECM se alcanza cuando la constante de regularización es igual a 0.15. En cuanto a γ, no parece haber un valor que sea mejor que el resto. Se puede sacar como conclusión de esta primera parte del análisis que contar con 5 factores latentes para este archivo es insuficiente, los errores mínimos alcanzados distan mucho de los alcanzados con 30 y 100 factores.

4.5.2 Elección del mejor modelo

De entre las mejores combinaciones, a continuación se realiza un análisis según los pasos que se van recorriendo y los errores obtenidos. En la tabla 2 se muestran las

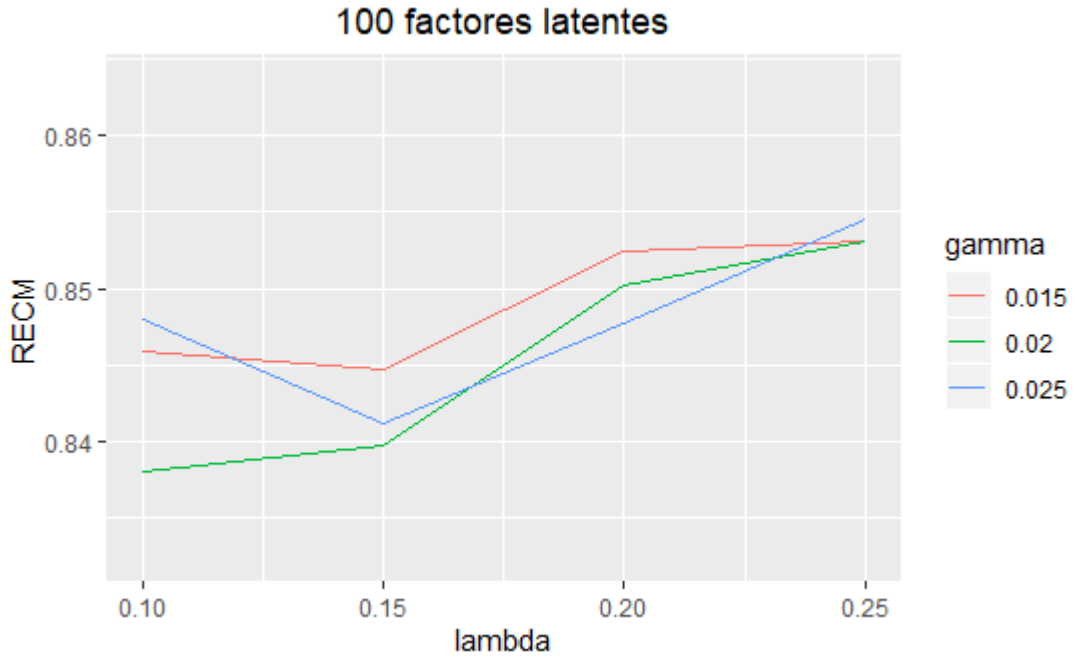


Figura 7: RECM con 100 factores latentes

3 mejores combinaciones de parámetros, dos de ellas presentan 30 factores habiendo una de 100. Para discernir cuál de estas es la mejor combinación, se vuelven a ejecutar usando un mismo conjunto de entrenamiento y test.

Factores	γ	λ	**RECM**
30	0.030	0.15	0.8324415
30	0.025	0.15	0.8366446
100	0.020	0.10	0.8380039

Tabla 2: Mejores combinaciones

En la figura 8 se muestra la evolución del RECM para cada uno de estos valores. Para este análisis se han llevado a cabo 100 iteraciones con el fin de conseguir el menor error posible aunque en muchas ocasiones la ejecución se detiene antes de llegar a esa cifra dado que las mejoras pueden no ser muy significativas. Como primer comentario, los modelos que tienen 30 factores comienzan con un error más bajo, es lógico dado que al ser la inicialización aleatoria y tener menos factores, la diferencia entre las predicciones y valores reales son más pequeñas. A medida que avanzan los pasos, las diferentes distribuciones se acercan. En la gráfica de la figura 9 se muestran las últimas 25 iteraciones, se aprecia que ya no existen cambios significativos y los errores oscilan buscando el óptimo, el modelo 2 ha finalizado antes de llegar a las 100 iteraciones. Como conclusión, los tres modelos se comportan de una forma muy similar, todos consiguen unas predicciones que tienen una correlación de en torno a 0.6 con los valores reales del conjunto de test.

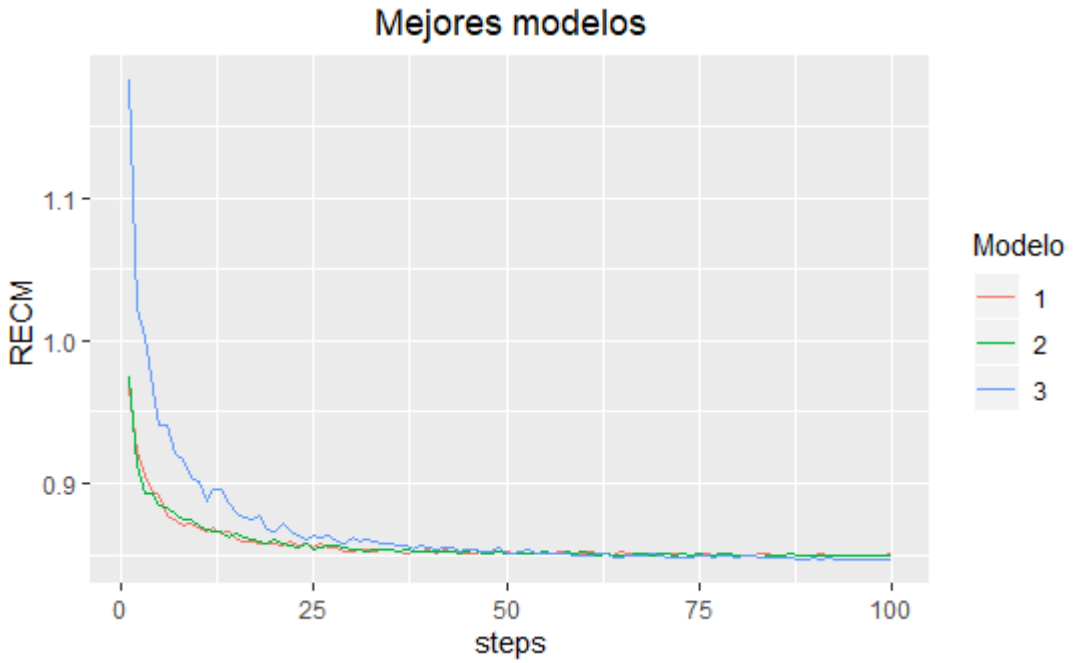


Figura 8: Comparación de los 3 mejores modelos

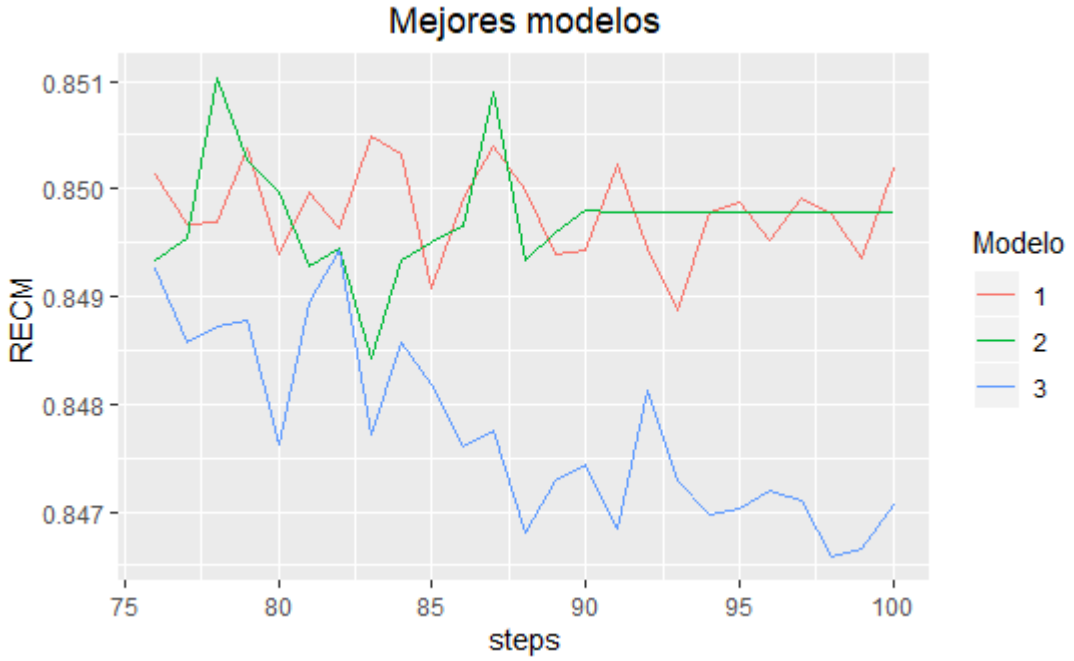


Figura 9: 25 últimas iteraciones

Dado que ninguno de los modelos en este análisis resulta ser especialmente mejor

que el resto, se van a utilizar los 3 para realizar un pequeño estudio de algunos usuarios.

4.5.3 Comparación de las predicciones y el test para varios usuarios

En este apartado se va a estudiar cuáles son las películas favoritas de 3 de los usuarios y ver en qué medida son similares a las predicciones de los 3 modelos elegidos anteriormente. Los usuarios estudiados son el 1, el 315 y el 512. Para ver qué películas son las favoritas se han seleccionado aquellas que han obtenida una valoración superior a 4, razón por la cual algunos modelos proporcionan un número mayor de predicciones que otros.

Las películas que han obtenido una nota superior a 4 en el conjunto de test para el usuario 1 son las mostradas en la figura 10. La figura 11 muestra las predicciones de los tres modelos. Las predicciones son muy similares entre ellas, tanto que los modelos 1 y 2 comparten todas las predicciones excepto una. Respecto al tercer modelo se presenta alguna diferencia, pero también existen muchas similiradidades. En comparación con las películas favoritas reales, todos los modelos aciertan 5 de las 10 películas aunque no todos seleccionan las mismas. La película *Bedknobs and Broomsticks* solo la escoge el tercer modelo mientras que *The Terminator* aparece en las predicciones de los dos primeros.

Desperado (1995)
Bedknobs and Broomsticks (1971)
E.T. the Extra-Terrestrial (1982)
Terminator The (1984)
Duck Soup (1933)
Lord of the Rings The (1978)
Enemy of the State (1998)
SLC Punk! (1998)
American Beauty (1999)
Live and Let Die (1973)

Figura 10: Películas favoritas del usuario 1

Apocalypse Now (1979)
American Beauty (1999)
Longest Day The (1962)
Toy Story (1995)
E.T. the Extra-Terrestrial (1982)
Terminator The (1984)
Dances with Wolves (1990)
Duck Soup (1933)
Live and Let Die (1973)
Jurassic Park (1993)

Modelo 1.

Apocalypse Now (1979)
Longest Day The (1962)
American Beauty (1999)
Toy Story (1995)
E.T. the Extra-Terrestrial (1982)
Dances with Wolves (1990)
Terminator The (1984)
Duck Soup (1933)
Live and Let Die (1973)
Ed Wood (1994)

Modelo 2.

Nico Icon (1995)
Apocalypse Now (1979)
Duck Soup (1933)
Live and Let Die (1973)
E.T. the Extra-Terrestrial (1982)
Dances with Wolves (1990)
Toy Story (1995)
Jurassic Park (1993)
Bedknobs and Broomsticks (1971)
American Beauty (1999)

Modelo 3.

Figura 11: Predicciones para el usuario 1.

El siguiente usuario, el 315, tiene como películas favoritas las que aparecen en la figura 12. Las predicciones proporcionadas por los tres modelos aparecen en la figura 13. Aquí podemos apreciar una tónica parecida que para el usuario anterior. Las predicciones son muy similares entre sí, en este caso los modelos 1 y 2 comparten todas las predicciones y 8 de las 10 respecto al tercer modelo. En relación a las películas favoritas reales del usuario, los dos primeros modelos predicen bien 7 de ellas mientras que el tercero predicen bien 5 de ellas.

Wallace & Gromit: The Best of Aardman Animation (1996)
Back to the Future (1985)
Creature Comforts (1989)
Requiem for a Dream (2000)
Snatch (2000)
Rules of Attraction The (2002)
Confessions of a Dangerous Mind (2002)
Kiss Kiss Bang Bang (2005)
Superbad (2007)

Figura 12: Películas favoritas del usuario 315

Modelo 1	Modelo 2	Modelo 3
Rocket Science (2007)	Rocket Science (2007)	Wallace & Gromit: (1996)
Wallace & Gromit: (1996)	Wallace & Gromit: (1996)	Creature Comforts (1989)
Creature Comforts (1989)	Requiem for a Dream (2000)	Rocket Science (2007)
Kiss Kiss Bang Bang (2005)	Kiss Kiss Bang Bang (2005)	After Hours (1985)
Requiem for a Dream (2000)	Forgetting Sarah Marshall (2008)	Kiss Kiss Bang Bang (2005)
Juno (2007)	Creature Comforts (1989)	Strange Brew (1983)
Superbad (2007)	Snatch (2000)	Requiem for a Dream (2000)
Back to the Future (1985)	Juno (2007)	Juno (2007)
Forgetting Sarah Marshall (2008)	Superbad (2007)	Forgetting Sarah Marshall (2008)
Snatch (2000)	Back to the Future (1985)	Snatch (2000)

Figura 13: Predicciones para el usuario 315

Por último, queda el análisis del usuario 512 para el que se muestran las películas que mejor ha valorado en la figura 14 y las predicciones de los modelos en la 15. Los dos primeros modelos comparten 8 películas predichas. Las predicciones del tercer modelo se diferencian en un item de las del modelo 1 y en dos de las del segundo modelo. En relación con las películas del test, todos los modelos aciertan las mismas 5 películas.

No existe un modelo que se comporte mejor que el resto de manera significativa en cuanto a las predicciones halladas. Se puede observar que las predicciones entre los dos modelos de 30 factores tienen algo más de similaridad entre sí que con el de 100, pero los tres reportan resultados muy parecidos, todos ellos muy válidos.

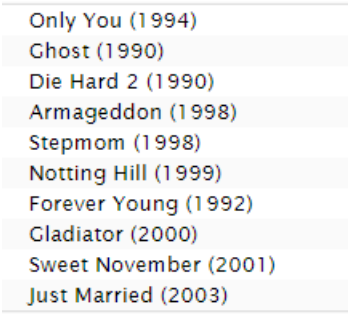
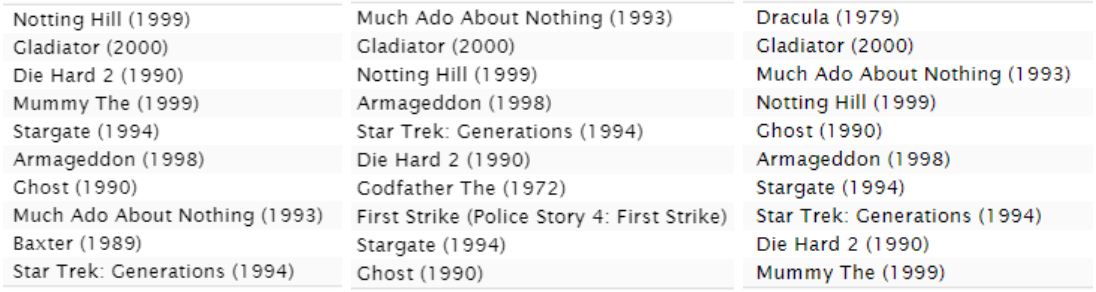


Figura 14: Películas favoritas del usuario 512



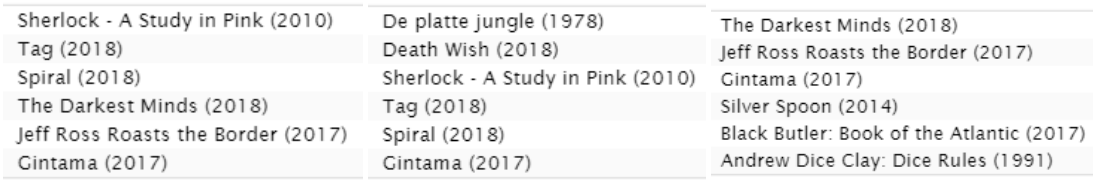
Modelo 1 Modelo 2 Modelo 3

Figura 15: Predicciones para el usuario 512

4.5.4 Nuevas recomendaciones para varios usuarios

Siguiendo con los mismos usuarios del apartado anterior, la mejor forma de llevar a cabo predicciones quizás sea tener en cuenta varios modelos y recomendar las películas que más se repitan. A continuación se muestran las 6 películas que han obtenido una mejor valoración por cada uno de los modelos para los 3 usuarios estudiados.

En la figura 16, se muestran las películas predichas por los tres modelos para el usuario 1. Se podría recomendar la película *Gintama* ya que se repite en los tres modelos y, en menor medida, películas como *Tag* o *The Darkest Minds* ya que están presentes en las recomendaciones de dos de los modelos.



Modelo 1 Modelo 2 Modelo 3

Figura 16: Recomendaciones para el usuario 1.

Para el usuario 315, las mejores recomendaciones son *Dragon Ball Z Gaiden* y *World of Tomorrow* ya que estas películas aparecen en las recomendaciones de los tres modelos. También se puede recomendar la película *Paterson* porque está en dos de las recomendaciones. Esta información está presente en la figura 17.

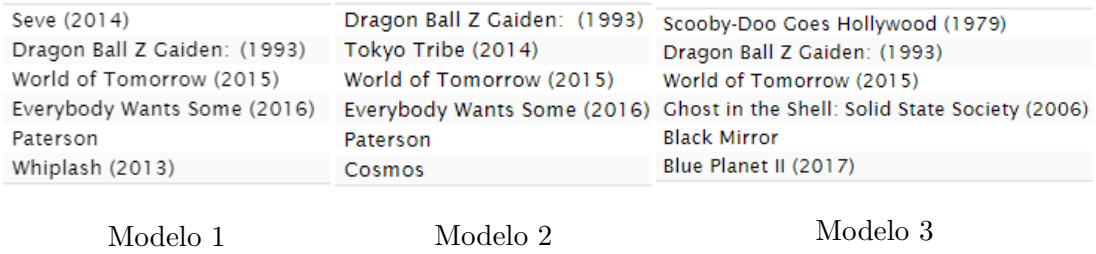


Figura 17: Recomendaciones para el usuario 315.

En cuanto al usuario 512, del que se ven las recomendaciones en la figura 18, no existe una película que aparezca en la lista de recomendaciones de los tres modelos. Sin embargo, tanto la película *Cosmos* como *De platte jungle* se repiten por lo que son las mejores recomendaciones que se pueden obtener.

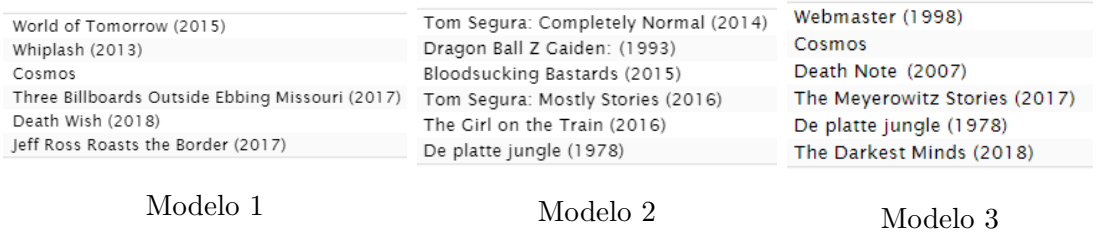


Figura 18: Recomendaciones para el usuario 512.

4.5.5 Comparación con paquete de R

Como se ha comentado con anterioridad, existe un paquete en R llamado *recommenderlab*, el cual implementa sistemas de recomendación de diferente tipo. En este apartado se ha ejecutado este paquete con los conjuntos de entrenamiento y test utilizados a la hora de realizar el estudio de los usuarios. Uno de los tipos de sistema que se puede poner en funcionamiento y que de hecho es el utilizado es el *SDVF*, al que llama Funk Singular Value Decomposition. Este se diferencia de la implementación creada en que no utiliza sesgos sino que normaliza las valoraciones de las matrices de entrenamiento y test.

Los resultados proporcionados por este paquete son muy parecidos a los que resultan de la implementación manual dado que consigue un RECM de 0.8388, y la correlación entre el test y la predicción es 0.6092. Son resultados prácticamente idénticos a los obtenidos en el sistema implementado a mano.

4.6 Explicación del código

A continuación se explica de forma detallada qué tarea o tareas se encarga de realizar cada porción del código utilizado para implementar el sistema de recomendación hecho a partir de un filtro colaborativo basado en la factorización de matrices.

La función `funcion_train_test` es la encargada de crear los conjuntos dedicados a entrenamiento y test, respectivamente. Recibe como argumentos la matriz de `valoraciones`, el número de usuarios (n) y el de items (m). A partir de estos se crean conjuntos de entrenamiento y test con tantas filas como usuarios diferentes hay y tantas columnas como items distintos. Se recorren todos los usuarios diferentes de forma iterativa y se toman todas las películas que ha evaluado cada uno almacenándolas en `items_valorados`. A continuación se toma una muestra aleatoria con un tamaño correspondiente al 10 % de las películas evaluadas por ese usuario. Las valoraciones de las películas elegidas se trasladan al conjunto de test mientras que se mantienen como `NA` en el conjunto de entrenamiento.

```
funcion_train_test<-function(valoraciones,n,m){
  train<-valoraciones
  test<-matrix(rep(NA,n*m),ncol=m)

  for(i in 1:n){
    items_valorados<-which(!is.na(valoraciones[i,]))
    muestra_items<-sample(items_valorados,
    round(length(items_valorados)*0.1))

    train[i,muestra_items]<-NA
    test[i,muestra_items]<-valoraciones[i,muestra_items]
  }
  return(list(train,test))
}
```

Otra forma posible de tomar los conjuntos de entrenamiento y test consistiría en tomar muestras aleatorias de todas las valoraciones. El inconveniente de este enfoque es que no se asegura que todos los usuarios estén representados en la misma proporción que en el conjunto de datos original, por eso se ha realizado de la forma mostrada.

La siguiente función implementada en el programa es `calcular_error`, la cual se encargada de calcular la raíz cuadrada del error cuadrático medio. Para ello, toma como argumentos un conjunto de datos (generalmente el de test), las matrices U y V y los términos correspondientes con los sesgos μ, b_i y b_j. Primero se hallan las predicciones haciendo uso de la función `pred`, la cual se explica más adelante, a la que se le pasan las matrices de factores de latentes y los términos correspondientes a los sesgos. Se halla a continuación la media de las diferencias al cuadrado entre las observaciones presentes en el conjunto de datos pasado como argumento y las predicciones obtenidas previamente para esos mismas observaciones. Después, se devuelve la raíz cuadrada de

este valor.

```
calcular_error<-function(datos,u,v,mu,bi,bj){
  predicho<-pred(u,v,mu,bi,bj)
  error<-mean((datos[!is.na(datos)]-
          predicho[!is.na(datos)])**2)
  return(sqrt(error))
}
```

Se ha tomado este error porque es la manera más usada en este tipo de sistemas para poder comparar lo bien o mal que actúa cada uno de ellos respecto al resto. Es además la métrica que se usaba en el concurso de Netflix.

Como se ha mencionado anteriormente, la función `pred` se encarga de obtener las predicciones de las valoraciones aportadas por cada usuario a cada película. Para llevarlo a cabo, se hace uso de (3), es decir, la predicción para cada par (usuario, item) viene dada por el producto de los vectores de factores latentes del usuario e item añadiendo el valor de la valoración media de las películas, la valoración media aportada por el usuario y la valoración media de la película en concreto.

En el código, primero se halla la matriz resultante de multiplicar las matrices de factores latentes y la media de todas las películas (`aux`). A esto hay que sumarle la media de cada usuario y cada película. Para ello, se crean dos matrices de las mismas dimensiones que la matriz auxiliar (`mat_bi` y `mat_bj`). La primera contiene la media de valoraciones de cada usuario de manera que, por ejemplo, la primera fila será el valor repetido tantas veces como películas hay de la media de valoraciones del primer usuario. Lo mismo pasa con la otra matriz, la cual incluye en cada columna la valoración media de cada película. Se realiza la suma de la matriz auxiliar y las matrices de sesgos de los usuarios y películas. Por último, aquellas valoraciones con una nota superior a 5 se establecen como 5 y, aquellas que tienen nota menor que 0, se cambian a 0.

```
pred<-function(u,v,mu,bi,bj){
  aux<-u%*%v+mu
  n<-dim(u)[1]
  m<-dim(v)[2]
  mat_bi<-matrix(rep(bi,each=m),nrow=n,byrow=T)
  mat_bj<-matrix(rep(bj,each=n),ncol=m)
  res<-aux+mat_bi+mat_bj
  res[res>5]<-5
  res[res<0]<-0
  return(res)
}
```

A continuación se detalla la función principal del sistema, se va a explicar por partes.

El algoritmo recibe como argumentos los conjuntos de datos de entrenamiento y test, el número de pasos a realizar (`pasos`), es decir, la cantidad de ocasiones que se recorre cada par (usuario, item). También recibe el número de factores latentes del modelo (`factores`) así como los parámetros γ y λ, los cuales son el factor de aprendizaje y la constante de regularización, respectivamente. Por último, el algoritmo necesita la cantidad de usuarios (n) e items (m). En esta porción de código también se muestra la inicialización de algunos parámetros. Primero se halla la media de todas las películas, μ y se crea la matriz que contendrá los RECM y que tiene tantas filas como factores latentes se van a probar y tantas columnas como pasos. Después se inicializan a 0 los parámetros correspondientes a las matrices y vectores que reporten un valor menor de RECM (las variables que comienzan con `mejor`) y se crean las variables `mult` y `minim`, esta última servirá para comparar los errores en cada iteración y comprobar si ha sido el menor hasta el momento. Tras la impresión por pantalla del número de factores considerado, se inicializan las matrices de factores latentes de forma aleatoria y los vectores de los sesgos. Por último, se reestablece el valor del factor de aprendizaje (`fact_aprend`) ya que se va modificando en cada paso multiplicándolo por `mult`.

```
algoritmo<-function(train,test,steps,factor,gamma,lambda,n,m){
  mu<-mean(train[!is.na(train)])
  error<-rep(NA,steps)
  mejor_u<-0
  mejor_v<-0
  mejor_bi<-0
  mejor_bj<-0
  minim<--1000
  mult<--0.99
  cat("\n",factor," factores\n")
  u<-matrix(runif(n*factor),nrow = n)
  v<-matrix(runif(factor*m),ncol = m)
  bi<-rep(0,n)
  bj<-rep(0,m)
  fact_aprend<-gamma
```

La siguiente parte del algoritmo consiste en la implementación del bucle principal que itera sobre los diferentes pasos. Dentro se desordenan los usuarios y los items que ha evaluado cada uno de ellos. Después se implementan las ecuaciones propias del SGD del apartado de 4.4 en las que se modifican las matrices de factores latentes y vectores de sesgos.

```
  for(s in 1:pasos){
    muestra_us<-sample(n)
    for(i in muestra_us){
    muestra_it<-sample(which(!is.na(train[i,])))
      for(j in muestra_it){
        r_ij<-u[i,]%*%v[,j]+mu+bi[i]+bj[j]
```

```
        e_ij<-train[i,j]-r_ij
        bi[i]<-bi[i]+fact_aprend*(c(e_ij)-lambda*bi[i])
        bj[j]<-bj[j]+fact_aprend*(c(e_ij)-lambda*bj[j])
        u[i,]<-u[i,]+fact_aprend*(c(e_ij)*v[,j]-lambda*u[i,])
        v[,j]<-v[,j]+fact_aprend*(c(e_ij)*u[i,]-lambda*v[,j])
      }
    }
```

A continuación se muestra la última porción de la función principal del sistema de recomendación. En esta se actualiza el factor de aprendizaje multiplicándolo por el valor dado (`mult`). Se llama a la función que calcula el RECM y se introduce el valor en el vector de errores para poder comparar una vez acabada la ejecución. Se introduce una condición para mostrar por pantalla la evolución de las iteraciones y errores hasta el momento y se comprueba si el RECM producido en esta iteración es el menor. En ese caso, se actualizan las matrices y vectores que recogen las matrices de factores latentes y vectores de sesgos de esa iteración (`u_i`, `v_j`, `bi` y `bj`) y el valor de *minim*. Por útlimo, se comprueba que la variación obtenida en el transcurso del paso respecto del anterior sea mayor que 0.00001, en caso contrario, se finaliza la ejecución. Esta función devuelve el vector que contiene los RECM para cada paso así como las matrices de factores latentes y vectores de sesgos conseguidos en la ejecución que reportó el mínimo RECM.

```
    fact_aprend<-fact_aprend*mult
    error[s]<-calcular_error(test,u,v,mu,bi,bj)
    if(error[s]<minim){
      minim<-error[s]
      mejor_u<-u
      mejor_v<-v
      mejor_bi<-bi
      mejor_bj<-bj
    }
    if(s%%10==0 ||s==1){
      cat("\tIteracion: ",s)
      cat("\n\t\tError: ",error[s],"\n")
    }
    if(s>1){
      if(abs(error[s]-error[s-1])<0.00001){
        error[(s+1):steps]<-error[s]
        cat("\n\tIteracion: ",s)
        cat("\n\t\tError: ",error[s])
        break;
      }
    }
  }
  return(list(error,best_u,best_v,mu,best_bi,best_bj))
}
```

Tras esta función se ha implementado otra que vaya recorriendo los diferentes niveles de factores latentes, factores de aprendizaje y constantes de regularización. Además ejecuta cada combinación `ejecucion` veces, cada una con un conjunto de entrenamiento y test diferente para conseguir una idea más realista del funcionamiento del sistema. Los resultados de cada ejecución se almacenan en una variable mediante el comando `assign`.

```
ejecucion<-function(pasos,fact,gam,lam,n,m,ex){
  for(f in fact){
    for(i in gam){
      for(j in lam){
        for(e in 1:ex){
          t<-funcion_train_test(valoraciones,n,m)
          train<-t[[1]]
          test<-t[[2]]
          cat("\ngamma:", i, ", lambda: ",j)
          resultado<-algoritmo(train,test,pasos,f,i,j,n,m)
          assign(paste("res",f,i,j,e,sep="_"),resultado,pos=1)
        }
      }
    }
  }
}
```

Esto es todo en lo referente a la definición de las funciones que se usan en la ejecución del sistema de recomendación. A continuación se refleja una muestra de una ejecución.

Lo primero de todo es leer los datos para lo que se hace uso de la función de R `read.table`. Una vez leído el conjunto de datos de ratings, se elimina la cuarta columna correspondiente a la fecha en la que se estableció la valoración, ya que no se va a usar esta información. Dado que este archivo tiene el formato de la figura 3, es necesaria hacer una transformación para conseguir una 'sparse matrix'. Para ello se halla el número de usuarios y películas diferentes que hay y se crean sendos vectores que mapean cada usuario e item con la posición que ocupan en el conjunto de usuarios e items, respectivamente. Esto sirve ya que existen saltos de números en los vectores de usuarios e items, es decir, pueden estar presentes en el archivo los usuarios 5 y 7 pero no el 6. Por lo tanto, es necesaria una forma de poder relacionar cada usuario y película con la posición que ocupan en la matriz de ratings que se crea a continaución y con las matrices de factores latentes y vectores de sesgos. Cuando se tienen los vectores con el orden de los usuarios e items, se crea la matriz de ratings con dimensión n×m y se introducen los valores del archivo original mediante un bucle.

```
datos<-read.table("ruta/ratings.csv",sep=",",header = T)
r<-datos[,-4]
```

```
n<-length(unique(r[,1]))
m<-length(unique(r[,2]))

orden_us<-sort(unique(r[,1]))
orden_it<-sort(unique(r[,2]))

valoraciones<-matrix(rep(NA,n*m),ncol=m)

for(i in 1:nrow(r)){
  valoraciones[which(orden_it==r[i,1]),
               which(orden_it==r[i,2])]<-r[i,3]
}
```

Ya se tienen los datos en el formato adeacuado así que solo falta determinar los parámetros del sistema. Es necesario conocer el número de factores latentes, los factores de aprendizaje, las constantes de regularización, los pasos y el número de repeticiones de cada combinación. Con esto se puede llamar a la función `ejecucion`, que realizará las llamadas pertinentes a la función principal.

```
factores_lat<-c(5,30,100)
gammas<-seq(0.015,0.03,0.005)
lambdas<-seq(0.1,0.25,0.05)
pasos<-50
n_ejecuciones<-5
ejecucion(pasos,factores,gammas,lambdas,n,m,n_ejecuciones)
```

Dado que se han realizado varias ejecuciones para cada combinación de parámetros, se crea un dataframe que incluye el número de factores latentes, γ, λ y la media de RECM de todas las ejecuciones.

```
errores<-data.frame(factores=c(),gamma=c(),lambda=c(),
error=c())
for(f in 1:length(factores_lat)){
  for(i in 1:length(gammas)){
    for(j in 1:length(lambdas)){
      err_vec<-c()
      results<-paste("res",factores_lat[f],gammas[i],
      lambdas[j],1:n_ejecuciones,sep="_")
      for(r in results){
        err_vec<-c(err_vec,min(get(r)[[1]]))
      }
      err<-mean(err_vec)
      errores<-rbind(errores,data.frame(factores=
             factores_lat[f],gamma=gammas[i],
             lambda = lambdas[j],error=err))
    }
```

```
  }
}
```

Para realizar los gráficos se ha utilizado la librería `ggplot2`. Para realizar los gráficos de comparación de factores y parámetros del sistema se usa `geom_line` estableciendo λ en el eje de abscisas, el RECM en el de coordenadas y γ como color de las líneas. Se han realizado tres gráficos, como se vio anteriormente, uno para cada valor considerado de factores. A continuación está el código usado para realizar el gráfico de 5 factores.

```
err5<-errores[(errores$factores==5),]
ggplot(err5,aes(x=lambda,y=error,color=factor(gamma)))
+ geom_line() +labs(title="5 factores latentes",
y='RECM',color="gamma")+theme(plot.title =
element_text(hjust = 0.5))+ coord_cartesian(ylim =
range(errores$error))
```

El otro enfoque que toma el trabajo consiste en el análisis de algunos usuarios para discernir qué películas han sido las favoritas y cómo se ajustan estos resultados a los predichos, además de descubrir qué nuevas recomendaciones se les pueden hacer. Hay que leer el archivo de `movies`, el cual muestra los identificadores de las películas junto con los géneros en los que se encuadra. La tarea a realizar consiste en conseguir aquellas películas, de entre las ya visualizadas, que han obtenido una nota superior a 4, considerándolas favoritas, para cada usuario estudiado. Una vez se tiene este conjunto de películas favoritas, se utiliza el conjunto de datos de `movies` para conseguir el nombre de estas películas. A continuación se muestra el código utilizado para obtener esta serie de películas utilizando el primer modelo para el usuario 1, donde `pred1` son las predicciones del primer modelo.

```
usuario1.1<-pred1[1,]
no_Nan1.1<-usuario1.1[!is.na(test[1,])]
mejores_peliculas_usuario1.1<-
        match(sort(no_Nan1.1[no_Nan1.1>4],
        decreasing=T),usuario1.1)
usuario1.1_indice<-orden_it[mejores_peliculas_usuario1.1]

usuario1.1_nombres<-movies[match(usuario1.1_indice,
        movies[,1]),2:3]
print(usuario1.1_nombres)
```

Con el fin de obtener nuevas recomendaciones, simplemente se seleccionan las que han obtenido una mejor nota en cada modelo, emparejándolo con el archivo que contiene el nombre de las películas. Aquí se ve cómo se obtienen las recomendaciones para el usuario 1 con las predicciones del primer modelo.

```
print(movies[match(orden_it[tail(sort(pred1[1,],
        index.return=TRUE)$ix)],movies[,1]),2:3])
```

Para llevar a cabo la ejecución del sistema de recomendación implementado en la librería `recommenderlab`, es necesario transformar tanto el conjunto de entrenamiento como el de test al formato `ratingMatrix`, que es el que utiliza la propia función de la librería. Esto se realiza con el comando `as`. Una vez hallados los dos conjuntos se entrena el sistema con el comando `Recommender` tomando como método el `SVDF`, el más similar a lo implementado en el proyecto. Después se predicen los resultados para el conjunto de test y se transforma al formato `matrix` estándar, para que sea legible. Por último, se calculan tanto la raíz del error cuadrático medio como la correlación entre las predicciones y los valores reales.

```
trainRec<-as(train,"realRatingMatrix")
testRec<-as(test,"realRatingMatrix")
recom<-Recommender(trainRec,method="SVDF")

aux_pred <- predict(recom, testRec,type="ratingMatrix")

rat_pred<-as(aux_pred, "matrix")

cat("RECM: ",sqrt(mean((test[!is.na(test)]-
                rat_pred[!is.na(test)])**2)))
cat("Correlacion: ",cor(test[!is.na(test)],
                rat_pred[!is.na(test)]))
```

5 Conclusiones

Este trabajo aporta una idea general sobre gran parte de los tipos de sistemas de recomendación que existen en el momento actual. Existen formas de realizar sistemas de recomendación que no se han considerado en este trabajo o no se ha hecho demasiado hincapié, como el terreno de las redes neuronales. No todos los sistemas valen para cualquier problema. Por ejemplo, se ha comentado que los filtros basados en contenido trabajan especialmente bien con textos, como libros o artículos o cómo los filtros basados en conocimiento se utilizan cuando no se posee una gran cantidad de datos acerca de usuarios e items.

El concurso creado por Netflix supuso un avance notorio en este ámbito. Nuevas formas de llevar a cabo un sistema de recomendación fueron puestas a punto, como la factorización de matrices. Esta forma permite analizar la información implícita de los conjuntos así como datos más escondidos como pueden ser los sesgos de los usuarios e items. También permitiría realizar análisis y predicciones teniendo en cuenta los cambios en los gustos de los usuarios, al introducir una componente temporal en los modelos.

Como se ha visto en el apartado práctico, para llevar a cabo un filtro colaborativo es necesario un conjunto de datos de gran tamaño y, además, es recomendable que los usuarios hayan evaluado un número suficiente de películas para evitar caer en el

problema del cold start, el cual se da cuando no se pueden realizar recomendaciones óptimas debido a la falta de datos con los que reflejar la relación de un usuario o item con los factores latentes, en el caso de filtros colaborativos.

Los sistemas de recomendación han contribuido a una ligera transformación en la forma en que consumen los usuarios. Esto ocurre ya que están presentes en casi cualquier sector económico, especialmente de ocio. Puede servir para recomendar música, series y películas, restaurantes, usuarios a los que seguir en redes sociales o productos que comprar en webs como Amazon.

El hecho de que se proporcionen sugerencias tan personalizadas puede levantar una duda respecto al carácter ético de que esto se produzca. Puede haber personas que no deseen que se entrometan en su privacidad, que en el fondo, es lo que se realiza en los sistemas de recomendación con el fin de crear modelos capaces de predecir los items que más van a gustar, aunque se haga de manera anónima. Por otra parte, supone una forma innovadora y eficiente de sugerir contenido a un usuario evitando que este tenga que realizar una búsqueda intentado hallar algo que se asemeje a sus gustos. Incluso puede ayudar a descubrir nuevos gustos a los usuarios ya que, como se planteó en los filtros colaborativos, se puede recomendar contenido en función de los usuarios similares y no solo en lo que ha consumido ese usuario.

6 Referencias

[1] I. Netflix, "Netflix prize." URL: netflixprize.com Fecha de acceso: 26-04-2020.

[2] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, p. 42–49, 2009.

[3] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich, "Part II: Introduction to basic concepts," in *Recommender Systems: An Introduction*, USA: Cambridge University Press, 1st ed., 2010.

[4] A. Felfernig, M. Jeran, G. Ninaus, F. Reinfrank, S. Reiterer, and M. Stettinger, "Basic approaches in recommendation systems," *Recommendation Systems in Software Engineering*, p. 15–37, 2013.

[5] Y. Koren, "Factorization meets the neighborhood," *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 08*, 2008.

[6] J. Zeng, *Latent Factor Models for Recommender Systems and Market Segmentation Through Clustering*. PhD thesis, The Ohio State University, 2017.

[7] H. Li, B. He, M. Lublin, and Y. Perez, "Matrix completion via alternating least square(ALS)." Lecture in Institute for Computational and Mathematical Engineering at University of Stanford, 2015.

[8] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, (Republic and Canton of Geneva, CHE), p. 173–182, International World Wide Web Conferences Steering Committee, 2017.

[9] Insight, "Explicit matrix factorization: ALS, SGD, and all that jazz," Aug 2016. URL: blog.insightdatascience.com/explicit-matrix-factorization Fecha de acceso: 20-04-2020.