UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

Evaluación de sistemas de recomendación utilizando la medida error entre rankings estimados y rankings originales (MERERO)

T E S I S

QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

P R E S E N T A :
O S C A R E S C A M I L L A G O N Z Á L E Z

Director de Tesis:
Dr. Sergio Rogelio Marcellin Jacques
Posgrado en Ciencias e Ingeniería de la Computación, UNAM

Ciudad Universitaria, CD. MX. Diciembre, 2017

Agradecimientos

A mi madre por todo el amor, comprensión y apoyo que me ha dado a lo largo de mi vida. A mi mujer Aidee por el amor que me ha brindado, todo el apoyo que recibí y por compartir desvelos durante estos tres años de mi maestría. A mis hermanas por el apoyo recibido; en especial a mi hermana Irma. Al Dr. José Luis Abreu por ser una fuente de inspiración a lo largo de tantos años de conocernos. Al Dr. Sergio Marcellin, por el valioso acompañamiento en esta aventura de tesis de maestría, así como al Dr. Miguel Tomasena, por su asesoría en el tema.
También a mis sinodales Mtra Guadalupe Ibargüengoitia, Dr. Fabiola Miroslaba y Dr. Francisco Valdez por sus valiosas aportaciones a mi trabajo. Al excoordinador del posgrado, el Dr. Jorge Luis Ortega y al coordinador Dr. Javier Gómez por aconsejarme y estar atentos a mis avances. Asimismo, a Amalia Arriaga, Lourdes González y Cecilia Mandujano por todo su apoyo en cuestiones tanto administrativas como personales.
Además quiero agradecer a todos mis amigos y compañeros con los cuales compartí durante el tiempo que curse mi maestría.
Y por último al CONACyT; gracias a su apoyo que me permitió realizar mis estudios de maestría y la UNAM, por ser mi alma máter y brindarme tantas experiencias extraordinarias en mi vida académica.

Atte. *Oscar Escamilla González*

Índice general

Introducción **1**

1. Estado del arte de los sistemas de recomendación **3**
1.1. Sistemas de recomendación 4
1.1.1. Filtrado basado en contenido 4
1.1.2. Filtrado colaborativo 6
1.1.3. Sistemas de recomendación híbridos 13
1.2. Comparación de técnicas de filtrado 14
1.3. Ejemplos de implementación de SR 16
1.4. Otras áreas de investigación en SR 17

2. Métricas de evaluación de SR **19**
2.1. Métricas de evaluación 19
2.1.1. MAE y RMSE 19
2.1.2. Precisión, *Recall* y valor-F 20
2.1.3. Cobertura 20

3. Propuesta de evaluación de SR usando rankings **21**
3.1. Antecedentes 21
3.2. El modelo MERERO 22
3.2.1. Comparando rankings 23
3.3. SR para la validación del modelo de evaluación 23
3.3.1. BestSR 24
3.3.2. FlipSR 25
3.3.3. MaxMSESR 31

4. Experimentación **34**
4.1. Implementación 34
4.1.1. Recommender101 34
4.2. Experimentos 40
4.2.1. Bases de datos para las pruebas 41
4.2.2. Ejemplo simplificado de experimentos 42
4.2.3. Segmentación de los datos para los experimentos 44
4.2.4. Resultados 44

Conclusiones **47**

A. Código implementado **48**
A.1. Clases en Recommender101 48
A.1.1. PredictionEvaluator 48
A.1.2. RecommendationListEvaluator 48
A.1.3. Experiment 50
A.2. Clases en MERERO 51

A.2.1. AbstractTestRecommender 51
A.2.2. BestRecommender 53
A.2.3. FlipRecommender 54
A.2.4. MaxMAERecommender 55
A.2.5. MERREROEvaluator 56
A.2.6. MinimalTest 58

B. Salida de los experimentos **62**
B.1. Salida de los experimentos 62

Resumen

Los sistemas de recomendación (*SR*) tienen como objetivo principal mostrar ítems al usuario que aún son desconocidos para éste (llamados *ítems recomendables*) y que puedan ser de relevancia para él. Para cumplir este objetivo, los SR realizan estimaciones sobre el nivel de relevancia de cada ítem del conjunto de ítems recomendables dentro del sistema para el usuario. Con base en estas estimaciones se crea un ranking de ítems personalizado y se recomiendan aquellos que se encuentren en las primeras k posiciones[1, 2].

Para medir la efectividad de los SR existen dos tipos de pruebas que de manera general se clasifican en las realizadas en línea (*online*) y en las realizadas fuera de línea (*offline*). Las pruebas *en línea* son aquellas que se realizan con el sistema en funcionamiento e interactuando con usuarios. En las pruebas fuera de línea, se tiene un conjunto de datos históricos por el cual se sabe el nivel de interés real del usuario sobre los ítems[3].

Actualmente los métodos de evaluación *fuera de línea* evalúan a los SR únicamente como estimadores sin tomar en cuenta el orden del ranking resultante. Para subsanar este hecho, la presente tesis propone un modelo de evaluación *fuera de línea* para SR, el cual llamaremos ***M**edida de **E**rror entre **R**ankings **E**stimados y **R**ankings **O**riginales* (*MERERO*), el cual compara los rankings generados, con aquellos que resultan de ordenar los ítems respecto al nivel de relevancia registrado en el conjunto de datos históricos.

Como parte de MERERO, se propone una representación de rankings que nos permita compararlos utilizando la distancia Manhattan[4] y así obtener un valor cuantitativo de las diferencias entre los rankings además de utilizar una variante del MSE[5, 6] para sumarizar las diferencias. Además se proponen tres SR sintéticos como control para validar los resultados obtenidos por MERERO, los cuales son: BestRS, MaxMSESR, FlipSR.

Teniendo los SR de control se ejecutaron pruebas con MERERO, MSE y RMSE con el fin de analizar el comportamiento de cada modelo con respecto a los SR de control. Las pruebas se implementaron haciendo uso del *framework* Recommender101[7] utilizando los datos históricos del sistema de MovieLens[8]. Estas pruebas arrojaron resultados positivos ya que al evaluar los tres SR propuestos como control con el modelo MERERO, se pudo comprobar que el modelo ofrece una evaluación que se enfoca en los rankings generados para medir la calidad de las recomendaciones generadas.

Palabras clave: *Sistemas de recomendación, Filtrado colaborativo basado en confianza, Filtrado basado en contenido, Filtrado colaborativo basado en usuarios, Filtrado basado en modelos, Filtrado colaborativo, Filtrado colaborativo basado en ítems, Sistemas de recomendación híbridos, Evaluación de sistemas de recomendación*

Introducción

Los sistemas de recomendación (que a lo largo de la tesis se nombran como *SR*) tienen como objetivo principal mostrar ítems al usuario que aún son desconocidos para este (llamados *ítems recomendables*) y que puedan ser de relevancia para él. Para cumplir este objetivo, los SR realizan estimaciones sobre el nivel de relevancia de cada ítem del conjunto de ítems recomendables dentro del sistema para el usuario. Con base en estas estimaciones se crea un ranking de ítems personalizado, entendiendo éste como una lista de ítems ordenada por el nivel de relevancia estimado, de tal manera que se recomiendan aquellos que se encuentren en las primeras k posiciones[1, 2].

Para medir la efectividad de los SR existen dos tipos de pruebas que de manera general se dividen como en línea (*online*) y fuera de línea (*offline*).

Las pruebas *en línea* son aquellas que se realizan con el sistema en funcionamiento e interactuando con usuarios. Por ejemplo, supongamos que se requiere medir la satisfacción del usuario con los ítems recomendados. En una situación como esta se podría medir la satisfacción del usuario con la lista opinión sobre las recomendaciones en un cuestionario o de forma implícita observando si el usuario interactúa con los ítems recomendados.

Por otro lado, en las pruebas *fuera de línea*, se cuenta con un conjunto de datos históricos con el cual se sabe el nivel de interés real del usuario sobre los ítems[3].

Actualmente los métodos de evaluación *fuera de línea* consideran a los SR únicamente como estimadores, es decir, evalúan solo la precisión de la estimación de cada pareja usuario - ítem. Para obtener la evaluación global de la precisión del SR se utilizan técnicas como el *error cuadrático medio* (MSE[5, 6]) o la *raíz del error cuadrático medio* (RMSE[5, 6]) que son medidas estadísticas para sumarizar los errores cometidos por estimadores de algún tipo. En el contexto de los SR se utilizan para medir la efectividad de éstos con respecto a los errores puntales en las estimaciones de cada pareja usuario - ítem. Este enfoque no considera los rankings personalizados de ítems generados por el SR, es decir que a pesar de que las estimaciones hechas por el SR reflejen el nivel de relevancia relativa entre los ítems para el usuario, si las estimaciones puntuales son malas, el SR obtendría una calificación baja. Supongamos que las calificaciones del usuario indican que el ítem a tiene un nivel de relevancia mayor al del ítem b y b tiene un nivel de relevancia mayor al de c (sin tomar en cuenta el valor nominal de la relevancia de estos), si las estimaciones hechas conservan el hecho de que:

$$a > b > c$$

se puede decir que el SR es adecuado para el conjunto de ítems a, b, c, independientemente de los valores puntuales de las estimaciones hechas para cada ítem.

Para subsanar este hecho, el trabajo de tesis muestra un modelo de evaluación *fuera de línea* para SR, el cual llamaremos ***M**edida de **E**rror entre **R**ankings **E**stimados y **R**ankinks **O**rignales* (*MERERO*). En el modelo MERERO se comparan los rankings generados usando las estimaciones hechas por el SR, que llamaremos el ranking estimado, con aquellos que resultan de ordenar los ítems respecto al nivel de relevancia registrado en el conjunto de datos históricos, que desde ahora llamaremos el ranking original.

Como parte de MERERO, se propone una representación de los rankings como vectores para después compararlos y obtener un valor cuantitativo de las diferencias entre éstos utilizando la distancia Manhattan[4], que es una medida de distancia entre vectores.

Para completar el modelo MERERO se propone una variante del MSE[5, 6] cambiando el valor absoluto de la diferencia ($|a - b|$), por la distancia propuesta en el modelo de tal manera que podamos

sumarizar las diferencias entre uno de los rankings estimados con el ranking original y tengamos un valor cuantitativo que describa el desempeño del SR evaluado y nos permita compararlos con otro SR.

Además del modelo MERERO en esta tesis se proponen tres SR sintéticos como control para validar el modelo de evaluación MERERO. Al decir que son SR sintéticos nos referimos que en lugar de tratar de estimar el valor de relevancia de los ítems utilizan los valores reales para calcular valores que nos permitan saber de antemano su comportamiento durante la evaluación.

Estos SR son:

BestSR Este representa un SR perfecto. El valor de la relevancia de un ítem i para un usuario u es estimado de manera exacta, por lo que los rankings generados deben ser idénticos a los rankings originales. Dada esta premisa, este SR debe de quedar evaluado de manera más alta con respecto a cualquier otro.

MaxMSESR Este ejemplifica al peor SR para los modelos de evaluación *fuera de línea* basados en estimaciones puntuales aunque está definido de tal manera que los rankings estimados correspondan a los originales.

FlipSR Este SR hace que los rankings estimados queden en orden inverso con respecto a los originales con lo que se maximiza la distancia de los rankings estimados con respecto a los rankings originales.

Bajo esta definición y al medir la efectividad de cada uno de los SR arriba descritos, BestSR deberá ser evaluado como mejor, MaxMSESR quedaría entre los peores, pero FlipSR debería ser calificado como el peor de los tres ya que estima rankings que son contrarios al nivel de relevancia real de los ítems para el usuario.

Con los tres SR sintéticos de control se ejecutaron varias pruebas usando los modelos tradicionales de evaluación (MSE y RMSE) además del aquí propuesto con el fin de analizar el comportamiento de cada de ellos con respecto a los SR de control. Las pruebas se implementaron utilizando el *framework* Recommender101[7] utilizando los datos históricos del sistema de MovieLens[8].

Estas pruebas arrojaron resultados alentadores ya que al evaluar los tres SR propuestos como control con el modelo MERERO, se pudo comprobar que el modelo ofrece una evaluación que se enfoca en la calidad de las recomendaciones, es decir, en la lista de ítems que recomienda junto con el orden de esta y no tanto en las estimaciones hechas. Esto bajo la evidencia de que evaluándolos con MERERO, BestSR es el mejor calificado, FlipSR es el peor calificado y MaxMSERS queda en un nivel intermedio.

La tesis se divide en cinco capítulos. En el primero capítulo se presenta el estado del arte de los SR con el objetivo de dar un panorama general sobre las distintas técnicas desarrolladas en cuestión de SR y algunas áreas de estudio alrededor de éstos. El capítulo dos muestra los modelos comúnmente usados para evaluar SR de tal manera que se tenga un punto de referencia para comparar el modelo propuesto. EL capítulo tres describe a detalle el modelo de evaluación propuesto junto con los SR de control para verificarlos. El cuarto capítulo habla sobre la implementación de las pruebas haciendo uso del *framework Recommender101* antes mencionado. Además toca los temas técnicos de la implementación como por ejemplo la manipulación de los datos originales de MovieLens y cambios realizados al código fuente del propio *framework*. El quinto capítulo está dedicado al análisis de los datos obtenidos de las pruebas, así como también a las conclusiones generales de esta tesis. Finalmente, se integra un anexo que expone de manera detallada los cambios al código fuente del *framework* además de las clases y utilidades implementadas para realizar las pruebas.

Capítulo 1

Estado del arte de los sistemas de recomendación

Actualmente la cantidad de información a la que podemos acceder por medio de la Internet excede la capacidad de una persona para poder procesarla en términos de lo que un usuario necesita encontrar. Por ejemplo, supongamos que tenemos un usuario que es un estudiante que está buscando artículos académicos relacionados con su tema de tesis. La cantidad de artículos que el usuario puede encontrar relacionados es exorbitante[9]. Aun así, por lo general sólo unos pocos artículos de este conjunto son relevantes o están acordes a las necesidades de su investigación.

Otro ejemplo típico es la adquisición de productos o servicios por medio de la Internet. La cantidad de posibles resultados al hacer una búsqueda es colosal, de tal manera que puede que los resultados que se obtengan sean servicios o productos que en realidad no se ajustan del todo a las necesidades, o en la multitud de resultados se pasen por alto algunas opciones que pudieran ser de interés. Este no es sólo un problema para los usuarios sino también para los proveedores de estos bienes o servicios, ya que estos pueden perder oportunidades de ventas por el simple hecho de que el usuario no encuentre el producto que el proveedor ofrece en la multitud de opciones, aun cuando este éste sea de total interés para el usuario.

Por otro lado, ante la falta de conocimiento de las alternativas disponibles o la inexperiencia en el tema, de manera natural nos apoyamos de las recomendaciones de terceros para poder llevar a cabo decisiones de mayor provecho. Pero también el hecho de encontrar este tipo de información auxiliar puede resultar complicado y muchas veces esta información es de carácter subjetivo.

Por estas razones se ha vuelto necesario la construcción de herramientas automáticas o semiautomáticas que provean algún tipo de recomendación, ayudando a los usuarios a encontrar información, productos, servicios, etc. de mejor manera, filtrando la información del universo disponible logrando así un mejor uso de ella[10]. Además, desde el punto de vista de los proveedores, del ejemplo de la venta línea (o *e-commerce*), se tiene un interés creciente por el desarrollo de este tipo herramientas, las cuales faciliten la vinculación entre sus productos y los usuarios que los necesitan e incluso aumenten las oportunidades de venta al recomendar productos que, si bien el usuario no está buscando en este momento, puedan ser suficientemente interesantes para él como para adquirirlos en ese mismo momento. Uno de los ejemplos más famosos de este tipo de proveedores es la compañía *Netflix*, que en 2007[11] organizó un concurso ofreciendo un premio de 1,000,000 USD a quien pudiera implementar un sistema que fuera mejor en al menos en un 10 % que el sistema que ellos ya tenían implementado e incluso cuando la meta no fue alcanzada hasta 2009, se le daba un premio de 50,000 USD al equipo que lograra batir la marca anterior en un 1 %.

El estudio de los RSs es relativamente nuevo y se independizó como un campo de investigación a mediados de los noventas y desde el 2007 se ha incrementado el interés por éstos[12]. Como muestra de ésto, los RSs son parte fundamental en importantes sitios Web entre los que destacan: Amazon, YouTube, Netflix, Yahoo, Tripadvisor, Last.fm, e IMDb.

1.1. Sistemas de recomendación

Los sistemas de recomendación (***SR***) están muy relacionados con sistemas de "búsqueda o recuperación de información", dado que ambos están pensados para que a partir de un conjunto de datos se obtenga información relevante para el usuario[1]. Pero a su vez tienen diferencias fundamentales. Por ejemplo, en los sistemas de búsqueda se espera un uso puntual de éstos, mientras que los SR se enfocan en usos repetidos a lo largo de tiempo. Otra diferencia es que en los sistemas de búsqueda los criterios de filtrado son expresados por el usuario de manera explícita cada vez que interactúa con él, mientras que en los SR, estos criterios se obtienen de forma implícita del *perfil del usuario*. El hecho de que sea necesario un perfil del usuario para realizar el filtrado, implica que se debe obtener y almacenarse en alguna parte, siendo esta necesidad otra de las diferencias con los sistemas de búsqueda. En los sistemas de búsqueda el usuario puede interactuar de forma anónima mientras que en los SR, el usuario necesita tener un perfil asociado.

Además tienen relación con la minería de datos ya que los SR se pueden ver como un problema de estimación de datos perdidos. De los ítems que el usuario aún no ha contemplado (*y por lo tanto desconocemos el nivel de interés del usuario sobre ellos*), queremos estimar el interés que el usuario puede tener sobre éstos a partir de observaciones previas o información extra y recomendar al usuario los que se estime sean de su interés, de tal manera que incluso, algunas técnicas de minería de datos se aprovechan directamente (ver sección "*1.1.2 Basado en modelos*") en los SR.

Por otro lado, los SR también están relacionados con los sistemas expertos. En ambos casos se trata de dar cierto soporte a la toma de decisiones. Con base en un conocimiento, se busca reducir el número de posibles opciones o dicho de otra forma, recomendar las opciones más plausibles dentro del contexto en el cual se está trabajando. Pero su diferencia fundamental es que en los SR se intenta no depender de expertos a los cuales extraer el conocimiento de forma directa. En los SR se utiliza a los usuarios como expertos y se intenta extraer el conocimiento de manera implícita a partir de sus perfiles.

Así los SR son sistemas de filtrado de información y su objetivo es mostrar al usuario ítems *recomendables*, de tal manera que sean de su interés, pero que además el usuario no haya tomado en cuenta. Por ejemplo, no tiene gran impacto recomendar al usuario un producto (en este caso nuestros ítems son productos) que ya compró, dado que ya lo conoce y además suponemos que es de su agrado y por ello hizo la compra.

Para cumplir este objetivo, los SR deben estimar de alguna manera el interés que el usuario tiene sobre cada ítem recomendable y seleccionar los de mayor interés para éste con base en las estimaciones hechas. Por lo tanto un SR debe de tener una función que dado un usuario y un ítem, pueda estimar el nivel de interés de ese usuario para el ítem dado. Esto de manera más formal lo podemos definir como en [1]:

Sea U el conjunto de todos los usuarios e I el conjunto de todos los posibles ítems recomendables. Definimos $\overline{r}$ como una función que estima la utilidad de un ítem i para el usuario u, es decir:

$$\overline{r} : U \times I \to R,\ R \text{ es conjunto ordenado} \tag{1.1}$$

En general dentro de los SR, no sólo se escoge un único ítem, sino que se crea un ranking de ítems basándose en el nivel de interés estimado y se seleccionan los n mejores ítems[1].

Como se mencionó en párrafos anteriores, se han desarrollado varias técnicas para implementar la función $\overline{r}$, y podemos clasificar a los SR con base en la técnica seleccionada para ser usada en el SR. En la figura 1.1 se muestran las técnicas más usadas para implementar un SR y a continuación se describen cada una de éstas.

1.1.1. Filtrado basado en contenido

El filtrado basado en contenido (**CBF**) recomienda ítems que estén dentro del perfil del usuario[13]. Este perfil se puede construir de manera explícita a partir de información solicitada al usuario, como por ejemplo usando formularios donde el usuario expresa preferencias o de manera implícita, extrayendo información de los ítems a los que el usuario ha mostrado interés anteriormente.

Este tipo de SR depende mucho del contexto ya que se requiere que los ítems tengan un conjunto de atributos (también llamados *metadatos*) que lo describan. Estos atributos son especificados manualmente

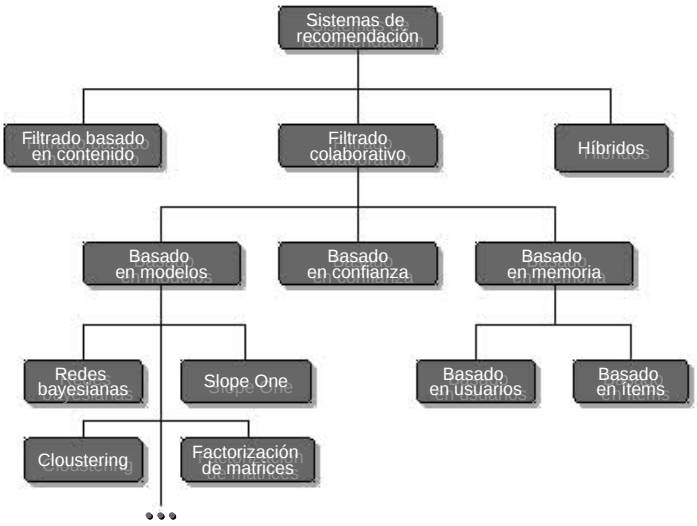

Figura 1.1: Clasificación de los **RS**

o se obtienen analizando información complementaria, cómo *tags*, comentarios, descripciones textuales o contenido multimedia, como imágenes, audio o video por ejemplo. Por otro lado, se necesita que el formato del perfil del usuario se pueda relacionar con los atributos de los ítems de tal manera que permita obtener una estimación del interés que el usuario puede tener sobre cada ítem.

La figura 1.2 [1] muestra de manera general el proceso de los SR usando CBF, donde los pasos son (1) extraer los atributos de los ítems, (2) comparar éstos con el perfil del usuario y (3) recomendar aquellos ítems que encajen mejor con el perfil del usuario[14].

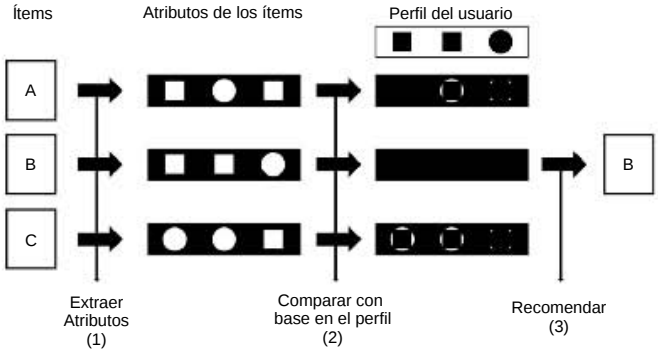

Figura 1.2: Esquema general del filtrado basado en contenido.

Esta técnica es la que mejores resultados genera en las estimación, ya que se tiene información precisa tanto del perfil del usuario como de los ítems, pero a su vez es de las más difíciles de implementar. Por un lado está el trabajo de dar mantenimiento a la información de los ítems, ya que esto puede requerir de personas especializadas en el contexto o de una investigación extra y esto multiplicado por la cantidad de ítems que se registren en el sistema. Por otro lado, no todos los usuarios están dispuestos a llenar un perfil con sus preferencias o no se requiere de un análisis extra para definir qué actividad del usuario conviene registrar para construir su perfil de manera implícita.

[1] Adaptada de la figura 8 (Pag. 122) del artículo "*Recommender systems survey*"[14]

1.1.2. Filtrado colaborativo

El filtrado colaborativo (**CF**) es el conjunto de técnicas más populares para desarrollar SR[1]. En este conjunto de técnicas se intenta hacer la estimación y recomendaciones con base en el comportamiento o en las calificaciones que los usuarios hacen sobre los ítems.

Algunas de las técnicas de CF suponen que las opiniones de otros usuarios pueden ser utilizadas de alguna manera para poder hacer una estimación razonablemente buena sobre las preferencias del usuario al cual se quieren hacer recomendaciones. La idea intuitiva de esto es que si un conjunto de usuarios está en cierta medida de acuerdo en el nivel de interés que tienen sobre un conjunto de ítems, entonces deberían coincidir en la misma medida en sus preferencias[15]. La idea intuitiva de estas técnicas es que existe una relación entre los gustos de los usuarios y que encontrando ésta se puede estimar el nivel de interés de un usuario sobre cada ítem.

La principal diferencia entre CBF y CF es que en este último no se requiere información o *metadatos* de los ítems, además de que los perfiles del usuario se reducen a tripletas del estilo *(usuario, ítem, calificación)*, donde *calificación* es un valor que refleja el nivel de interés del usuario, sobre un ítem dado. Los mecanismos que se usan para obtener esta calificación son muy variados, desde situaciones donde el usuario califica explícitamente un ítem como en *www.facebook.com*, donde el usuario determina si una publicación le gusta, o sitios donde se le otorga una calificación numérica como *www.amazon.com*, donde los usuarios pueden otorgar estrellas a los productos y cuántas estrellas se le otorgue es el nivel de agrado del producto dado.

Siendo los perfiles de usuario un conjunto de tripletas, de manera natural se pueden representar de forma matricial, donde los usuarios son las filas, los ítems las columnas y cada entrada de la matriz es la calificación correspondiente. En caso de que un usuario aún no produzca su respectiva calificación sobre un ítem tendremos un hueco en la matriz, tal y como se muestra en el ejemplo de la figura 1.3.

		Ítems			
		A	**B**	**C**	**D**
Usuarios	U_1	3	2	2	1
	U_2	3	1	2	5
	U_3		5		2
	U_4	5	4	2	4
	U_5	2	3	4	2

Figura 1.3: Matriz de calificaciones

Basado en memoria

Las técnicas de CF basadas en memoria utilizan algoritmos que trabajan con el conjunto completo de tripletas para estimar el nivel de interés de un usuario sobre un ítem dado. Para realizar la estimación, se utilizan funciones de agregación, de tal modo que si tenemos una calificación desconocida ($r_{u,i} = \oslash$) del usuario u sobre el ítem i, podemos hacer una estimación de su valor ($\overline{r}_{u,i}$) ya sea utilizando las calificaciones de otros usuarios *similares* u que sí han calificado i o usando las calificaciones que el usuario u ha hecho sobre ítems similares a i.

Basado en usuarios Como su nombre lo indica, esta técnica hace recomendaciones utilizando funciones de agregación sobre los usuarios. La idea intuitiva de esta técnica es que si un grupo de usuarios calificó de manera similar a u un conjunto de ítems, las calificaciones de este grupo a un ítem i, desconocido para u, deberían ser similares a la calificación que u haría a ese ítem. De tal manera que podemos estimar la calificación de u al ítem i con base en las calificaciones de este grupo de usuarios.

Esta es la técnica más usada para el CF y el algoritmo de los *k-vecinos* es el referente para las técnicas que engloba el CF basados en memoria. De manera general esta técnica consiste[14] en:

1. Se calcula la similitud entre el usuario al cual se desea hacer recomendaciones (usuario u) y cada uno de los usuarios utilizando una función $sim(u, u')$.
2. Para cada ítem i recomendable al usuario u, se selecciona el conjunto de los k usuarios más similares (también llamados *vecinos*) a éste que han calificado a i. Con base en este conjunto se estima la calificación de i ($\overline{r}_{ui}$) utilizando una función de agregación.
3. Se recomiendan los m ítems que mejor calificación tengan con base en nuestras estimaciones.

Algunos ejemplos de estas funciones de agregación que se utilizan para el CF con base en usuarios son[14]:

$$\overline{r}_{u,i} = \frac{1}{N} \sum_{u' \in \hat{U}} r_{u',i} \tag{1.2}$$

$$\overline{r}_{u,i} = c \sum_{u' \in \hat{U}} sim(u, u') \times r_{u',i} \tag{1.3}$$

$$\overline{r}_{u,i} = \overline{r}_u + c \sum_{u' \in \hat{U}} sim(u, u') \times (r_{u',i} - \overline{r}_{u'}) \tag{1.4}$$

Donde $\hat{U}$ es el conjunto de los N usuarios más similares a u que sí cuentan con calificaciones sobre i. El término $\overline{r}_u$ es el promedio de las calificaciones del usuario u y se define como:

$$\overline{r}_u = \left(\frac{1}{|I_u|} \right) \sum_{i \in I_u} r_{u,i} \quad \text{donde} \quad I_u = \{ i \in I | r_{u,i} \neq \oslash \}$$

Aquí I se refiere al conjunto de todos los ítems. Y por último tenemos el término c que es un factor de normalización y en usualmente se toma como:

$$c = \left(\frac{1}{\sum_{u' \in \hat{U}} |sim(u, u')|} \right)$$

El término $sim(u, u')$ se refiere una función que mide la similitud entre los usuarios u y u'. Como se puede observar esta función $sim(u_1, u_2)$ es parte fundamental para esta técnica más allá de la función de agregación que se escoja, por lo que en la literatura se puede encontrar una amplia gama de funciones para calcular la similitud entre usuario[1, 14, 16, 17], las más usadas son:

Correlación de Pearson (COR)

$$sim(a, b) = \frac{\sum_{i \in C_{ab}} (r_{ai} - \overline{r}_a)(r_{bi} - \overline{r}_b)}{\sqrt{\sum_{i \in C_{ab}} (r_{ai} - \overline{r}_a)^2} \sqrt{\sum_{i \in C_{ab}} (r_{bi} - \overline{r}_b)^2}} \tag{1.5}$$

Coseno (COS)

$$sim(a,b) = \frac{\sum_{i \in C_{ab}} (r_{ai} r_{bi})}{\sqrt{\sum_{i \in C_{ab}} r_{ai}^2} \sqrt{\sum_{i \in C_{ab}} r_{bi}^2}} \tag{1.6}$$

COR constreñido (CPC)

$$sim(a,b) = \frac{\sum_{i \in C_{ab}} (r_{ai} - \bar{r})(r_{bi} - \bar{r})}{\sqrt{\sum_{i \in C_{ab}} (r_{ai} - \bar{r})^2} \sqrt{\sum_{i \in C_{ab}} (r_{bi} - \bar{r})^2}} \tag{1.7}$$

Correlación de rankeo de Spearman (SRC)

$$sim(a,b) = \frac{6}{n(n^2-1)} \sum_{i \in C_{ab}} |r_{ai} - r_{bi}| \tag{1.8}$$

Error cuadrático medio (MSE)

$$sim(a,b) = 1 - \frac{1}{n} \sum_{i \in C_{ab}} |r_{ai} - r_{bi}|^2 \tag{1.9}$$

Donde a y b son los usuarios a comparar. C_{ab} es el conjunto de ítems que han sido calificados por ambos usuarios y n es la cardinalidad de C_{ab}, es decir, el número de elementos en C_{ab}. Los términos r_{ai} y r_{bi} son las calificaciones dadas al ítem i por a y b respectivamente. Los otros términos $\bar{r}_a$ y $\bar{r}_b$, son el promedio de todas las calificaciones hechas por a, b y $\bar{r}$ es el promedio de todos los usuarios.

En la figura 1.4 se muestra un ejemplo de predicciones utilizando CF basado en usuarios utilizando tres vecinos ($k = 3$) y queremos recomendar tres ítems ($m = 3$). En este ejemplo se utilizó el promedio como función de agregación (ecuación 1.2) y el error cuadrático medio para calcular la similitud entre usuarios (ecuación 1.9).

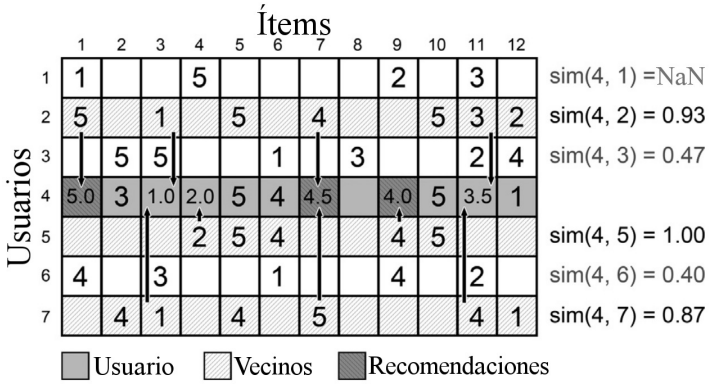

Figura 1.4: Ejemplo del algoritmo *k-vecinos*

En este ejemplo queremos estimar las calificaciones que el usuario 4 asignaría a los ítems 1, 3, 4, 7, 8, 9 y 11, que son los ítems que este usuario aun no califica. Iniciamos calculando la similitud del usuario 4 con los demás utilizando el error cuadrático medio. Como ejemplo tomemos la similitud entre los usuarios 4 y 5:

$$C_{ab} = \{5, 6, 10\}$$
$$n = 3 = |C_{ab}|$$

Por lo tanto :

$$\begin{aligned} sim(4,5) &= 1 - \frac{1}{n}\sum_{i \in C_{ab}} |r_{4,i} - r_{5,i}|^2 \\ &= 1 - \frac{|r_{4,5} - r_{5,5}|^2 + \cdots + |r_{4,10} - r_{5,10}|^2}{3} \\ &= 1 - \frac{|5-5|^2 + |4-4|^2 + |5-5|^2}{3} \\ &= 1 - \frac{0}{3} \\ &= 1 \end{aligned}$$

Entonces seleccionamos a los k usuarios más cercanos, en este caso 2, 5 y 7. Con estos usuarios estimamos las calificaciones desconocidas usando la función de agregación seleccionada. Tomemos como ejemplo el ítem 3:

En este caso tenemos que:

$$\hat{U} = \{2, 7\}$$

Por lo tanto:

$$\begin{aligned} \overline{r}_{4,3} &= \frac{1}{2}\sum_{u' \in \hat{U}} r_{u',3} \\ &= \frac{1}{2}(r_{2,3} + r_{7,3}) \\ &= \frac{1}{2}(1+1) = \frac{2}{2} \\ &= 1 \end{aligned}$$

Por último seleccionamos los m ítems con las estimaciones más altas para formar nuestra lista de recomendaciones, es decir, recomendamos los ítems 1, 7 y 9.

Cabe destacar que no se puede calcular la similitud entre los usuarios 4 y 1 con la función de similitud seleccionada. Esto debido a que la expresión $\frac{1}{n}$ queda indeterminada al no existir ítems que ambos usuarios hayan calificado, por lo que el usuario 1 no se tomó en cuenta. De igual forma, no se puede estimar la calificación para el ítem 8 debido a que ninguno de los k usuarios más cercanos ha calificado dicho ítem.

En muchos casos para ahorrar tiempo de procesamiento se calcula de antemano una matriz de similitud entre usuarios, de tal forma que cuando se desee hacer recomendaciones, solo se consulta esta tabla en lugar de procesar cada vez la similitud entre cada usuario. Aunque esto tiene la desventaja de que cada vez que se agrega un ítem, usuario o calificación, se requiere volver a calcular la matriz de similitudes.

Basado en ítems El CF basado en ítems es casi idéntico al basado en usuarios, pero como su nombre indica, las funciones de similitud y de agregación se realizan sobre los ítems. En este caso para estimar el valor de $\overline{r}_{ui}$, en lugar de tomar los usuarios parecidos a u que han calificado i se toman los ítems más parecidos a i que el usuario u a calificado.

En este caso el proceso a seguir para recomendar ítems al usuario sería:

1. Para cada ítem i recomendable al usuario u :
   - *a*) Se calcula su similitud con los ítems que u ha calificado, usando una función de similitud entre ítems ($sim(i, i')$).
   - *b*) De los ítems que u ha calificado se seleccionan los k más similares a i y se utiliza una función de agregación para estimar la calificación del ítem i ($\overline{r}_{ui}$)

2. Se recomiendan los m ítems que mejor calificación tengan con base en nuestras estimaciones.

En el ejemplo de la figura 1.5 se puede observar la estimación de la calificación del usuario 1 al ítem 1 ($\overline{r}_{1,1}$).

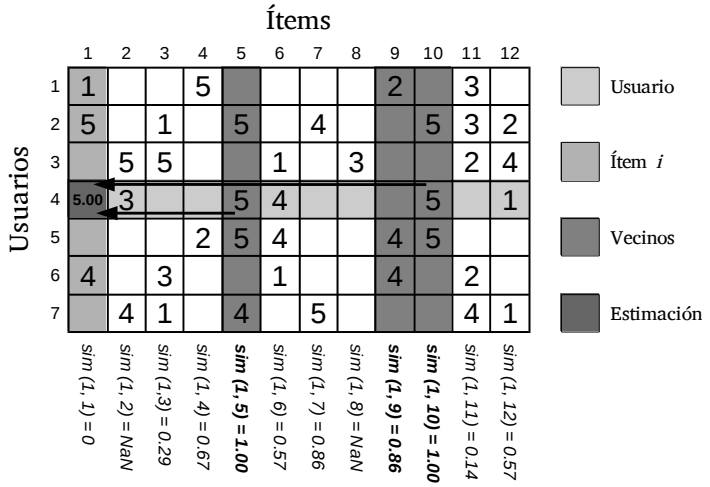

Figura 1.5: Estimación para el ítem de la calificación de i ($\overline{r}_{ui}$) usando el algoritmo *k-vecinos* basado en ítems.

De igual forma que en el CF basado en usuarios, es común que se almacene una matriz de similitud entre ítems para optimizar el tiempo de respuesta al calcular la lista de ítems recomendados.

Basado en confianza

También llamados recomendaciones sociales. En esta técnica las recomendaciones se basan en la confianza entre usuarios. Es similar al filtrado colaborativo con base en usuarios pero en este caso la noción de *vecinos* y *similitud* entre usuarios se remplaza por *amigos* y *confianza* entre usuarios respectivamente. Bajo esta perspectiva, es el propio usuario quien elige a sus *vecinos*, nombrados ahora *amigos*. Y ahora se agrega una función de confianza para emular el escenario de amigos de mis amigos. Esto se basa en la idea de que las personas confían en las opiniones de sus amigos y confían en las opiniones de amigos de sus amigos en mayor o menor medida[18].

De tal manera que se puede modificar la técnica de CF con base en usuarios descrita en la sección "*1.1.2 Basado en usuarios*" y remplazar las funciones de *similitud* por una nueva función que determine la confianza entre usuarios.

De qué manera modelar la confianza y así poder determinar ésta entre dos usuarios se ha convertido en una tarea indispensable en muchas ramas de la web social, como la seguridad, la computación en nube e incluso en el ámbito de los SR. Se han propuesto varios modelos para determinar la confianza entre usuarios e incorporarlos dentro de técnicas de SR. Algunos ejemplo de éstos modelos son: Modelo de Simon[19], Modelo de Abdul- Rahman[20], Modelo de Charif Alchiekh Haydar[18], MoleTrust[21], TidalTrust[22], Modelo de O'Donovan[23]

Al igual que en todas las áreas de la web, la confianza en los sistemas de recomendación se ha aplicado en dos áreas principales: la confianza como una relación entre dos individuos (local), la confianza de confianza como un resumen de la opinión de la comunidad respecto de un individuo (la confianza general o reputación). La primera forma (la confianza local) es la más común en los sistemas de recomendación.

Basado en modelos

A diferencia de los SR basados en memoria, los SR basados en modelos hacen recomendaciones de ítems construyendo previamente un modelo con base a las calificaciones de los usuarios. En este caso, se ataca el problema de las estimaciones como un problema de datos perdidos o de clasificación[24] y se utilizan algoritmos de *machine learning* como redes neuronales, redes bayesianas, clustering o incluso toman inspiración de algoritmos de otro tipo de problemas como factorización de matrices.

Una ventaja de estas técnicas con respecto a las basadas en memoria, es que son más rápidas al momento de estimar la relevancia de los ítems ya que tienen calculado un modelo previamente, aunque su desventaja es que al agregar nuevos datos, ya sea usuarios, ítems o calificaciones, el modelo necesita ser actualizado, mientras que en las técnicas basadas en memoria siempre se cuenta con la información actualizada.

Slope One La técnica de *Slope One*[25] considera que se puede estimar una calificación desconocida a partir de una función lineal de otro valor conocido, de tal forma que tenemos:

$$\overline{r}_{ui} = f(r_{uj}) = r_{uj} + \delta_{ij}$$

Por lo que el problema se reduce a encontrar δ_{ij} para cada par de ítems. Formalmente, dados dos vectores $v = (v_1, \ldots, v_n)$ y $w = (w_1, \ldots, w_n)$, buscamos una función $f(x) = x + \delta$ para predecir w a partir de v minimizando la expresión $\sum_{i=1}^{n} (v_i + \delta - w_i)^2$. Derivando la expresión con respecto a δ, igualando a cero y despejando tenemos que $\delta = \frac{1}{n} \sum_{i=1}^{n} (w_i - v_i)$. Por lo tanto δ es el promedio de la diferencia de cada entrada.

Ahora para encontrar δ_{ij} necesitamos un conjunto de usuarios de entrenamiento $\hat{U}_{ij}$ tal que éstos hayan calificado ambos ítems (el ítem i y el ítem j) y calculamos el promedio de las diferencias :

$$\delta_{ij} = \frac{1}{|\hat{U}_{ij}|} \sum_{u \ in \hat{U}_{ij}} (r_{uj} - r_{ui})$$

Dado que ya tenemos un estimador de $\overline{r}_{uj}$ dado r_{ui} podemos sacar la estimación promedio usando el conjunto (R_u) de todos los ítems que ha calificado el usuario u:

$$\overline{r}_{uj} = \frac{1}{|R_u|} \sum_{i \in R_u} (r_{ui} + \delta_{ij})$$

Redes Bayesianas Al igual que en los RS basados en memoria los SR que utilizan modelos basados en redes bayesianas estiman el valor de calificación $\overline{r}_{ui}$ como una función de agregación de las calificaciones que se tienen de otros usuarios, pero en este caso se hace una suma ponderada por probabilidad, es decir, se busca el valor esperado para $\overline{r}_{ui}$ [26]. Tomemos X_i como la variable aleatoria de la calificación que puede tener el ítem i, entonces el valor de r_{ui} es:

$$r_{ui} = E(X_i) = \sum_{\lambda \in R} P\left(X_i = \lambda | X_{I-\{i\}}\right) \times \lambda$$

Donde $X_{I-\{i\}}$ el conjunto de todas las X definidas por los ítems excepto i, es decir, el conjunto $I - \{i\}$. El conjunto R corresponde a los posibles valores que pueden tener las calificaciones, por ejemplo, si nuestro sistemas maneja calificaciones de una a cinco estrellas, $R = \{1, 2, 3, 4, 5\}$.

Por lo tanto podemos, hacer las estimaciones $\overline{r}_{ui}$ si conocemos los valores de $P\left(X_i = \lambda | X_{I-\{i\}}\right)$ y estos los podemos estimar utilizando un modelo de red bayesiana.

Existen varios métodos para estimar las correlaciones dentro una red bayesiana como por ejemplo, los que se proponen en [26]: 'Maximización del valor esperado' o 'Model Likelihood'.

Cloustering La técnica de *clustering* o *k-medias* divide el conjunto de usuarios en k subconjuntos de tal manera que los elementos de cada conjunto estén lo más cerca posible en relación a una medida de distancia dada. Cada conjunto C_j (*clouster* C_j) está definido por N_j elementos y un centroide λ_j. Este centroide λ_j es un punto donde se minimiza la suma de las distancias de éste con todos los elementos que pertenecen al clouster C_j[6].

Por lo tanto se puede implementar un proceso donde se seleccione aleatoriamente cada λ_j y de manera iterativa mueva elementos de un conjunto a otro hasta minimizar la ecuación:

$$E = \sum_{j=1}^{k} \sum_{x_n \in C_j} d(x_n, \lambda_j)$$

Donde x_n es el vector que representa el $n-$esimo elemento del clouster C_j.

Trasladándolo a los SR, cada elemento x_n representa las calificaciones del usuario n sobre el conjunto de ítems. Cada clouster C_j representaría un grupo de usuario con gustos similares y el controide λ_j serían las calificaciones representativas de este grupo. Por lo tanto podríamos estimar el valor de r_{ui}, buscando el clouster C_j al que pertenece el usuario u y tomar el valor de la posición i del vector λ_j.

Factorización de matrices La idea detrás de esta técnica es que existen factores latentes que pueden explicar por qué un usuario le da cierta calificación a un ítem dado. En caso de hablar de películas, estos factores pueden ser el género, los actores, la historia etc. de tal manera que la calificación que el usuario u da a un ítem i tomando en cuenta estos f factores, se puede expresar como:

$$r_{ui} = q_i p_u^T$$

Donde q_i y p_u son vectores en $\mathbb{R}^f$ donde cada entrada se refiere a cada uno de estos factores latentes. En el caso de q_i, cada elemento refleja el grado de apego al factor dado y cada elemento de p_u refleja la relevancia de ese factor para los gustos del usuario.

Si hacemos esto para cada usuario y cada ítem, se puede construir una matriz R de tal forma que:

$$R = QP^T$$

Donde cada fila de Q corresponde a cada uno de los vectores q_i y las filas de P son cada uno de los vectores p_u. Entonces el problema se convierte en tratar de estimar las matrices Q y P tomando como base la matriz de calificaciones (como en el caso del filtrado colaborativo). Uno de los primeros acercamientos que se plantearon es usar la *descomposición en valores singulares* (SVD), pero dado que en general las matrices de calificaciones que se tienen, cuentan con un gran número de entradas perdidas es probable que la descomposición se obtenga este sobre especializada o incluso puede que la factorización no se pueda obtener con los algoritmos propuestos para estimar la descomposición.

Se han propuesto otras alternativas para tratar sortear estos problemas, por ejemplo en [27] se propone utilizar sólo las observaciones conocidas (los valores conocidos de la matriz de calificaciones) y buscar un conjunto de p_is y q_us que minimicen el error cuadrático como en la siguiente expresión:

$$\min_{p*,q*} \sum_{(u,i) \in K} (r_{ui} - q_i {p_u}^T)^2 + \lambda(|q_i|^2 + |p_u|^2)$$

Donde K es el conjunto de parejas de *usuario-ítem* (u,i) de los cuales si conocemos el valor de $r_u i$. Y para evitar la sobre especialización agregamos un parámetro λ para penalizar la longitud del vector. Una implementación de como minimizar esta expresión puede verse en la página web "*Matrix Factorization: A Simple Tutorial and Implementation in Python*[2] ".

[2] http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/

1.1.3. Sistemas de recomendación híbridos

Los SR híbridos combinan varias técnicas de SR para mejorar las recomendaciones y mitigar los problemas particulares que presenta cada una de ellas. La idea principal de esta aproximación es combinar distintas técnicas de SR de tal manera que preserven las virtudes de cada una y que las desventajas de una técnica particular pueden ser mitigadas con las propiedades de las otras[16, 18].

Existen varias aproximaciones para combinar diferentes técnicas de SR, como ejemplo de las comúnmente usadas tenemos las siguientes técnicas:

Hibridación ponderada

En esta técnica de hibridación se combinan las estimaciones de dos o más SR utilizando una combinación lineal entre ellas de tal forma que se puede ponderar la importancia o relevancia que tendrán las estimaciones hechas por cada SR que se están combinando al asignar distintos pesos (coeficientes) a cada uno de ellas[18]. De tal manera que tendríamos:

$$\overline{r_{h}}_{ui} = \sum_{j \in \hat{SR}} \alpha_j \overline{r_j}_{ui}$$

Donde:

$\overline{r_{h}}_{ui}$ Es la estimación final de nuestro RS híbrido

$\overline{r_j}_{ui}$ Es la estimación del SR j del conjunto de SR que estamos combinando ($\hat{SR}$)

α_j Es el factor de ponderación para el SR j.
Además:

$$\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1$$
$$0 < \alpha_j < 1 \ \forall j \in \hat{SR}$$

Hibridación por selección

En este caso, se calcula la estimación de cada uno de los sistemas de recomendación a combinar y con base en un conjunto de condiciones se selecciona el valor para la estimación global[28]. De tal manera que tenemos:

$$\overline{r_{h}}_{ui} = \begin{cases} \overline{r_1}_{ui} & \textbf{si} \quad cond_1(\overline{r_1}_{ui}, \ \ldots, \ \overline{r_n}_{ui}) \\ \overline{r_2}_{ui} & \textbf{si} \quad cond_2(\overline{r_1}_{ui}, \ \ldots, \ \overline{r_n}_{ui}) \\ & \vdots \\ \overline{r_n}_{ui} & \textbf{si} \quad cond_n(\overline{r_1}_{ui}, \ \ldots, \ \overline{r_n}_{ui}) \end{cases}$$

Donde $cond_j(\overline{r_1}_{ui}, \ \ldots, \ \overline{r_n}_{ui})$ es la condición que debe cumplirse para seleccionar la estimación del SR j ($\overline{r_j}_{ui}$). Esta notación es para generalizar la estructura y a pesar de que indicamos que se utilizan todas las estimaciones para calcular la condición, puede ser que éstas sólo involucren algunas de las estimaciones, como por ejemplo:

$$\overline{r_{h}}_{ui} = \begin{cases} \overline{r_1}_{ui} & \textbf{si} \quad \overline{r_1}_{ui} \neq null \\ \overline{r_2}_{ui} & \textbf{si} \quad \overline{r_1}_{ui} = null \end{cases}$$

Hibridación por mezcla

Esta técnica de hibridación mezcla las recomendaciones de varios SR en una lista de recomendaciones global. Trabaja sobre los rankings generados a partir de las estimaciones del nivel de interés del usuario sobre los ítems recomendables.

Supongamos que queremos combinar tres SR (SR_1, SR_2 ySR_3) y que tenemos tres ítems recomendables (A,B y C) de los cuales se obtuvieron las estimaciones que se muestran en la tabla (a) para el

usuario u. Al calcular los rankings tenemos los resultados como se muestra en la tabla (b) y utilizando el algoritmo de votación de Hare[29, p. 322] obtenemos como resultado el ranking de la tabla (c):

	A	B	C
SR_1	3	5	4
SR_2	2	4	3
SR_3	4	6	3

(a)

	Ranking		
SR_1	B	C	A
SR_2	B	C	A
SR_3	B	A	C

(b)

	Ranking		
SR_h	B	C	A

(c)

Hibridación en cascada

La técnica de hibridación en cascada consiste en filtrar los ítems de manera escalonada. Se toma uno de los SR (SR_1) de los que se desea combinar y se calcula un conjunto de ítems candidatos usando las estimaciones de éste. Se vuelve a filtrar el conjunto anterior con la siguiente técnica de SR elegida (SR_2) y se recomienda el conjunto de ítems resultante.

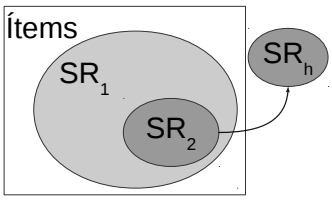

Figura 1.6: Ejemplo de SR híbrido en cascada. Donde SR_h es el conjunto de ítems recomendados.

1.2. Comparación de técnicas de filtrado

Al intentar estimar el nivel de interés que el usuario puede tener sobre el conjunto de ítems recomendables, se presentan una serie de problemas colaterales que han propiciado el desarrollo de una gran variedad de técnicas para intentar subsanar uno o varios de estos. A continuación se listan los más importantes:

Serendipia La serendipia se define como la ocurrencia de un evento afortunado poco probable. Cuando el SR carece de ésta, se puede dar el caso de que a pesar de existir ítems de interés para el usuario, éstos se dejen de lado por tener muy poca similitud con los ítems por los que el o los usuarios han mostrado interés[30]. Este rasgo se considera una especie de sobre especialización. Por ejemplo, supongamos que tenemos un sistema donde los ítems son canciones. El usuario gusta de canciones de Rock de los años 70, pero sólo ha reproducido canciones del genero Pop, por lo que el SR no recomendaría las canciones de Rock ya que el usuario no ha mostrado interés en ellas.

Arranque en frío También llamado "*cold start*", ocurre cuando no se tiene suficiente información para generar una buena aproximación. Ya sea porque no se cuenta con un conjunto de calificaciones sobre un ítem dado o no se cuenta con información sobre los gustos del usuario. Este es un caso frecuente ya que se produce cuando un sitio es puesto en línea por primera vez o se agrega un nuevo usuario/ítem al sistema[14].

Usuarios maliciosos Se produce cuando usuarios intentan manipular el SR para que oculte/muestre ciertos ítems. Por ejemplo, un usuario podría dar una buena calificación a un libro que él mismo escribió y crear varios usuarios ficticios para votar por su libro. De esta manera, podría sesgar el SR para mostrar más a menudo dicho libro.[21]

Escalabilidad Este problema se presenta cuando el volumen de los datos (número de usuario y/o número de ítems) comienza a crecer[16]. Una técnica de SR puede funcionar bien bajo un conjunto

de datos limitado, pero la eficiencia y desempeño puede decaer al punto de ser no ser plausible cuando la cantidad de datos crece.

Contexto del usuario En muchas de las técnicas utilizadas por los SR se utiliza la calificación de otros usuarios sobre un ítem dado para estimar el nivel de interés que el usuario objetivo tiene sobre éste. Se busca los usuarios *similares*, entendiendo similar como que ha demostrado apreciaciones similares en varios ítems, pero dentro de esta similitud no se toma en cuenta el trasfondo de la calificación. Por ejemplo, puede que a dos usuarios les guste la misma película y que hayan asignado la misma calificación a ésta, pero mientras el primero sólo está interesado en la fotografía, el otro la considera una buena película por los actores involucrados[31].

Dificultad de implementación En muchos casos el SR resulta difícil de implementar ya que por ejemplo puede que requiera de un esfuerzo extra por parte de los usuarios, como es el caso del filtrado con base en contenido. O porque se requiere de un entendimiento profundo del contexto de los ítems o de los usuarios a pesar de que en la parte computacional sea relativamente sencillo de procesar. En otro casos, la solución computacional exige herramientas y conocimientos extra, como es el caso de los SR basados en modelos.

En la tabla de la figura 1.7 se desglosan brevemente los problemas comunes en SR y si afecta o no a cada técnica descrita.

			Serendipia	Arranque en frío	Usuarios maliciosos	Escalabilidad	Contexto del usuario	Dificultad al implentar
Basado en contenido			N	N	N	N	N	S
Filtrado colaborativo	Basado en memoria	Usuarios	S	S	S	S	S	N
		Ítems	S	S	S	S	S	N
	Basado en confianza		N	S	N	S	S	M
	Basado en Modelos	Red bayesiana	S	N	S	N	N	M
		Slope One	S	N	S	N	N	N
		Cloustering	S	N	S	N	N	M
		Factorización de matrices	N	N	S	N	N	M
Híbridos			N	N	N	N	N	S

N = no afecta, **S** = sí afecta, **M** = un punto intermedio

Figura 1.7: Comparación de sistemas de recomendación

1.3. Ejemplos de implementación de SR

A continuación se presentan algunos sitios web que implementan SR junto con el tipo de técnica que utilizan para ofrecer recomendaciones. En la mayoría de los casos no se hace explicito el tipo de SR que utilizan pero en las notas al pie se puede encontrar los artículos con el material de donde se dedujo está información.

Sitio	**Tipo de SR implementado**
Amazon	Filtrado Colaborativo[3]
Tinder	Híbrido[4]: Demográfico Basado en contenido Filtrado colaborativo
Pandora	Basado en contenido[5]
Last.fm	Basado en confianza[6]
Spotify	Filtrado colaborativo[7]
Tripadvisor	Híbrido[8]: Basado en contenido Filtrado colaborativo
Yelp	Híbrido[9]: Demográfico Factorización de Matrices Filtrado colaborativo con base en usuarios
LinkedIn	Híbrido[10]: Filtrado basado en contenido Filtrado colaborativo con base en usuarios
Netflix	Híbrido[11]: Demográfico Factorización de Matrices Filtrado basado en contenido etc.
eBay	Factorización de matrices[12] *Buscan factores latentes para producir un RS basado en contenido*
Youtube	Filtrado colaborativo con base en ítems[13]
Jester	Filtrado colaborativo[14]

[3] https://www.google.com/patents/US6266649
[4] http://highscalability.com/blog/2016/1/27/tinder-how-does-one-of-the-largest-recommendation-engines-de.html
[5] http://blog.stevekrause.org/2006/01/pandora-and-lastfm-nature-vs-nurture-in.html
[6] http://blog.stevekrause.org/2006/01/pandora-and-lastfm-nature-vs-nurture-in.html
[7] http://www.slideshare.net/erikbern/collaborative-filtering-at-spotify-16182818
[8] http://www.hermesthemes.com/tripadvisors-2016-ranking-algorithm-update/
[9] https://www.math.uci.edu/icamp/summer/research/student_research/recommender_systems_slides.pdf
[10] https://www.quora.com/How-does-LinkedIns-recommendation-system-work
[11] https://pdfs.semanticscholar.org/e9dd/899f0e599eafb4fe47696c83d07d971c0088.pdf
[12] https://es.slideshare.net/planetcassandra/e-bay-nyc
[13] https://pdfs.semanticscholar.org/e7d5/3f538f5239739d1f943c81d17e4a167c65c6.pdf
[14] http://www.ieor.berkeley.edu/~goldberg/pubs/eigentaste.pdf

1.4. Otras áreas de investigación en SR

Recomendaciones con base en dominios cruzados

La idea detrás de esta área de investigación es que se pueden mejorar las recomendaciones haciendo uso de la información recolectada en distintos sistemas (nombrados en esta área como *dominios*), de tal modo que se pueda construir un perfil unificado y más completo del usuario juntando la información recolectada en cada sistema.

Las principales líneas de investigación se enfocan en como homologar la información proveniente de distintos dominios, que funciones de agregación usar para conjuntar la información en un solo perfil y como identificar el contexto en el cual se requieren las recomendaciones, es decir, cual es el conjunto de ítems a recomendar. Otro aspecto importante es la privacidad del usuario ya que se tiene que compartir información de éste entre distintos sistemas.

Por ejemplo supongamos que tenemos un usuario el cual tiene cuenta en un sitio de música, otra para un sitio de libros y extra para un sitio de películas. Conjuntando la información de los tres sitios se podría tener un mejor perfil del usuario y hacer recomendaciones más relevantes, claro esta que en cada uno de los sistemas se deben de recomendar ítems de distintos tipos, canciones en el de música, novelas en el de libros y películas en el último.

Sistemas sociales de etiquetado STS (Social Tagging Systems)

Hoy en día existen muchos sitios y aplicaciones Web que permiten al usuario publicar y compartir contenido propio, como por ejemplo: Flickr, Snapchat, Delicius, Youtube o Facebook. Muchos de estos sitios cuentan con algún mecanismo que les permite a los usuarios caracterizar los contenidos con anotaciones, o dicho de otra maneara, etiquetarlos con texto sin ninguna restricción en los términos a utilizar. Esto tiene la ventaja de que permite organizar y clasificar la información a los usuarios de una manera cercana al modelo que tiene en su mente. Pero a su vez esto se convierte en un problema ya que los modelos mentales pueden ser tan heterogéneos que dificultan la tarea de recuperar y compartir estos contenidos. Por lo tanto se busca alguna manera de hacer recomendaciones a los usuarios sobre etiquetas relevantes para el contenido que éste está publicando. Esta tarea se desarrolla bajo la idea de que dos contenidos similares deberían tener etiquetas similares y que el sistema puede hacer uso de las etiquetas que los usuarios han asignado y de como se han utilizado para extraer las más relevantes para el usuario en orden de etiquetar su contenido[32].

A pesar de que este problema es muy parecido al que intenta solucionar los SR, en este caso tenemos una dimensión más, ya que en lugar de tener *(usuario, ítem, calificación)*, tenemos *(usuario, contenido, etiqueta, calificación)*. Además de esto, a diferencia de los SR tradicionales, no es necesario que las recomendaciones no se repitan, es decir, es posible que recomendemos una misma etiqueta a un usuario para distintos contenidos.

Problemas de privacidad en SR

Los RS recolectan información acerca de los usuarios a lo largo del tiempo de tal manera que las recomendaciones personalizadas van siendo mejores, pero esto claramente tiene un impacto negativo en la privacidad de los usuarios de tal manera que éstos comiencen a sentir que sistema sabe demasiado acerca de ellos y que gente externa pueda saber información sensible de ellos, como por ejemplo sus hábitos de compra, sus gustos personales, etc.

Incluso aunque la empresa que tenga las mejores intenciones en resguardar la información de sus usuarios, siempre existen situaciones en las que esta información puede ser difundida sin consentimiento explicito de los usuarios, por ejemplo en un ataque cibernético. Otro ejemplo tiene que ver con el hecho de que la información ya se considera como un bien con valor monetario y por lo tanto puede ser vendida o en caso de que empresas entren en bancarrota, la información o recursos informáticos que ha acumulado es susceptible a ser embargada[33].

Dado este tipo de situaciones ha surgido la necesidad de crear mecanismos que mantengan la privacidad de los datos del usuario, evitando que ésta sea visible para agentes maliciosos y al mismo tiempo permita al RS hacer recomendaciones acertadas y de interés para sus usuarios.

Integración de las preferencias a corto y largo plazo

Actualmente los SR hacen sus recomendaciones utilizando el conjunto completo de los datos sobre los intereses del usuario que se han recolectado hasta el momento sin tomar en cuenta que éstos pueden diferir a lo largo del tiempo. Supongamos, por ejemplo, que tenemos un RS para libros y un usuarios que por mucho tiempo ha estado interesado principalmente en novelas de ciencia ficción, pero que en fechas recientes el mismo usuario ha estado interesado más en novelas del viejo oeste. En este caso, si nuestro SR hace recomendaciones en este momento, debería mostrar novelas del viejo oeste aunque a largo plazo, sea más probable que este usuario retome las novelas de ciencia ficción y en su caso, para futuras recomendaciones, se le muestren principalmente novelas de este tipo[12].

Capítulo 2

Métricas de evaluación de SR

La calidad de una técnica de SR se ha evaluado de manera tradicional sobre el poder de predicción, es decir, qué tan cercano es el valor estimado a la selección del usuario. Si bien es cierto que es importante que la técnica de SR cumpla esto, también se puede evaluar a los SR desde el punto de vista de qué tan novedosos para el usuario son los ítems recomendados o qué cantidad de ítems diferentes son recomendados a los usuarios. Para cada uno de estos casos se han propuesto una serie de métricas para evaluar estos aspectos[6].

Los distintos tipos de evaluaciones se pueden catalogar en dos grandes grupos, de tal forma que una técnica de SR puede ser evaluada en línea (*online*) o fuera de línea (*offline*). Típicamente las evaluaciones *en línea* tratan de investigar cómo los usuarios se comportan o reaccionan ante las recomendaciones hechas por el SR, por lo que la evaluación se hace sobre éste ya trabajando con usuarios reales. La recopilación de datos puede ser hecha de manera explícita, pidiendo a los usuarios que califiquen las recomendaciones hechas por medio de un formulario o de manera implícita, analizando el comportamiento del usuario, por ejemplo si éste interactúa y con cuál de los ítems recomendados. En la parte de las evaluaciones *fuera de línea*, se trata de contrastar las predicciones hechas por el SR con los valores reales de un conjunto de calificaciones colectado previamente.

2.1. Métricas de evaluación

Para cada tipo de evaluación se han desarrollado distintas métricas para poder analizar el comportamiento de los resultados que la técnica de SR arroja. Las más comúnmente usadas son:

2.1.1. MAE y RMSE

Las métricas "raíz del error cuadrático medio" (RMSE) y el "error absoluto medio" (MAE) están pensadas para medir la precisión de las predicciones de manera estadística. Estas son medidas ampliamente utilizadas, no sólo en SR, sino en cualquier área en la que se requiera evaluar predicción de valores[5, 6].

Éstas se definen como:

$$MAE = \frac{1}{n} \sum_{\substack{i \in I \\ u \in U}} |r_{ui} - \overline{r}_{ui}| \tag{2.1}$$

$$RMSE = \sqrt{\frac{1}{n} \sum_{\substack{i \in I \\ u \in U}} (r_{ui} - \overline{r}_{ui})^2} \tag{2.2}$$

Donde I y U son el conjunto de ítems y usuarios sobre los cuales se hizo la evaluación del SR, de tal forma que r_{ui} es la calificación real que el usuario u le dio al ítem i y $\overline{r}_{ui}$ es la estimación de este valor hecha por el SR.

2.1.2. Precisión, *Recall* y valor-F

La precisión y *recall* (sensibilidad o exhaustividad) son medidas que se usan para medir la calidad de sistemas de búsqueda y recuperación de información y reconocimiento de patrones. Estas métricas toman en cuenta el número (t) total de ítems recomendados, el número (g) de esos ítems que fueron de utilidad y el total que son de utilidad (T) dentro del conjunto completo de ítems[5, 6]. Éstas métricas se define como:

$$\text{Precisión} = \frac{g}{t} \qquad \text{Recall} = \frac{g}{T}$$

La situación ideal es que la precisión y la exhaustividad tengan un valor alto, es decir que sean muy cercanos a 1, de tal manera que todas las recomendaciones hechas por el SR sean de utilidad y que recomiende la mayoría de los ítems que pueden ser útiles para el usuario.

Además tenemos la métrica *valor-F* que nos indica en qué medida el SR está cumpliendo con las dos anteriores de manera simultánea al combinar las medidas de precisión y *recall*. Esta métrica está definida por:

$$\textit{valor-F} = \frac{\text{Precisión} \times \text{Recall}}{\text{Precisión} + \text{Recall}}$$

Una cuestión interesante de estas métricas es qué se tiene que definir que significa que un ítem sea de utilidad para el usuario. Puede ser que un ítem sea de utilidad para un usuario pero para otro no o incluso se puede argumentar que el hecho de que un usuario exprese que un ítem es de utilidad para él, esta valoración puede ser subjetiva e incluso presentar inconsistencias. Por lo que esto es dependiente del contexto donde el SR se esté usando.

2.1.3. Cobertura

Con esta métrica se trata de analizar la porción de ítems que el sistema puede recomendar. Esta métrica es usada en las evaluaciones *fuera de línea* e intenta reflejar la proporción de ítems que se pueden recomendar con base en los que fueron recomendados en los experimentos[5, 6]. Para medir la cobertura en los RS se utiliza el coeficiente Gini, que es una medida estadística de dispersión y está definido como:

$$G = \frac{1}{n-1}\sum_{j=i}^{n}(2j - n - 1)p(i_j)$$

Donde $p(i_j)$ es la proporción de veces que los usuarios han seleccionado el ítem i_j y $i_1, i_2, \ldots, i_n$ es la lista de ítems ordenados de mayor a menor utilizando el valor de $p(i_j)$. Este índice es cero cuando todos los ítems son seleccionados un número equitativo de veces, mientras que un valor de uno indica que un solo ítem es seleccionado siempre.

Otra medida que se utiliza a menudo para evaluar la cobertura es el índice de entropía de Shannon:

$$H = -\sum_{i \in I} p(i) \times log\,(\ p(i)\ )$$

Donde I es el conjunto de ítems y nuevamente $p(i)$ es la proporción de veces que los usuarios han seleccionado el ítem i. Si el valor de la entropía es cero significa los usuarios siempre seleccionan (*o el SR recomienda*) el mismo ítem y un valor de $log(|I|)$ indica que todos los ítems se seleccionan un número equitativo de veces.

Capítulo 3

Propuesta de evaluación de SR usando rankings

3.1. Antecedentes

La mayoría de los métodos de evaluación *fuera de línea* tradicionales sólo toman en cuenta el valor puntual de las estimaciones producidas por el SR, dejando de lado el análisis de los rankings generado a partir de estas estimaciones. Si bien el hecho de que los errores puntuales sean pequeños implica que los rankings generados a partir de las estimaciones y por los valores reales son muy parecidos, el hecho de que estos rankings sean muy parecidos no implica que los errores puntales sean pequeños.

Consideremos el siguiente ejemplo. Se tiene el valor real y las estimaciones de la relevancia de los ítems $\{A, B, C\}$ para un usuario (u) usando dos técnicas de SR diferentes:

A	B	C
8	2	6

Real

A	B	C
4	1.2	3

Estimado SR_1

A	B	C
9	1.8	5.6

Estimado SR_2

Figura 3.1: Valor real dado por el usuario y estimaciones hechas por dos SR distintos

Si comparamos los dos SR usando el criterio del error cuadrático medio (MSE) tendríamos :
Para SR_1

$$\frac{1}{3}\left((8-4)^2+(2-1{,}2)^2+(6-3)^2\right)=\frac{16+0{,}64+9}{3}=8{,}54$$

Para SR_2

$$\frac{1}{3}\left((8-9)^2+(2-1{,}8)^2+(6-5{,}6)^2\right)=\frac{1+0{,}04+0{,}16}{3}=0{,}4$$

Como se puede observar, si utilizamos MSE para comparar la precisión en nuestras estimaciones, se considera que SR_2 es mejor que SR_1. Pero al observar los rankings generados, tenemos que son iguales

	SR_1	SR_2
MSE	8.54	0.4

Tabla 3.1: MSE para SR_1 y SR_2

	Ranking
SR_1	[A, C, B]
SR_2	[A, C, B]

Tabla 3.2: Rankings generados a partir de las estimaciones hechas por SR_1 y SR_2

como se puede apreciar en la Tabla 3.2 y en ambos casos se recomendarían los ítems A y C, además de que en ambos casos las recomendaciones son acertadas.

Dado que el objetivo final de los SR es la ponderación de los ítems dentro del sistema para hacer recomendación, se puede obtener una evaluación del desempeño de este analizando las estimaciones hechas como el nivel de apreciación del usuario sobre los ítems de manera relativa en lugar de analizar el valor puntual de cada estimación. Dicho de otra manera y tomando conceptos de la lógica difusa lo que queremos es medir la precisión de significado en vez de solo medir la precisión de valor sobre las recomendaciones que emite un SR[34].

Retomando el ejemplo anterior, ambos SR estiman que el usuario valora más el ítem A sobre los otros dos y al ítem B sobre el C y esto es coherente con las estimaciones reales expresadas por el usuario.

3.2. El modelo MERERO

Con el fin de evaluar las recomendaciones hechas por un SR comparando el nivel relativo de aprecio sobre los ítems como se menciona en la sección anterior, se propone el modelo de evaluación fuera de línea MERERO (**M**edida de **E** rror entre **R**ankings **E**stimados y **R**ankings **O**rignales), en el cual se comparan los rankings generados a partir de las estimaciones hechas por el SR contra los rankings generados con los valores reales del aprecio el usuario por los ítems.

Para evaluar un SR con este modelo, necesitamos antes generar los rankings a partir de las estimaciones hechas por el SR. Se hace la suposición de que un número mayor obtenido en la estimación implica que se estima un mayor nivel de interés del usuario por ese ítem, por lo que el ranking se genera ordenando los ítems de los cuales se hace la estimación de manera descendente. Tomemos como ejemplo la estimación hecha por el sistema de recomendación SR_1 en la Figura 3.1.

A	B	C
4	1.2	3

$\xrightarrow{Ordenando}$

A	C	B
4	3	1.2

En caso de que dos o más ítems, tengan el mismo nivel de interés (*real o estimado*), se toman el identificador de los ítems para decidir su posición en el ranking. Extendiendo el ejemplo anterior con un ítem con el identificador D tendríamos:

A	B	C	D
4	1.2	3	4

$\xrightarrow{Ordenando}$

A	D	C	B
4	4	3	1.2

Por otro lado necesitamos utilizar una estructura que nos permita la comparación de estos rankings. En este trabajo de tesis se utilizó una estructura de vector de n entradas, donde el valor de n es distinto para cada usuario ya que el vector se compone de ítems recomendables para el usuario y este conjunto diferente para cada usuario y por lo tanto también el número de estos puede variar. Cada entrada del vector hace referencia a uno de los ítems recomendables y su valor es la posición en el ranking del ítem respectivo. Retomando el ejemplo de la Figura 3.1 y con base en los valores reales del interés del usuario sobre los ítems tendríamos:

$$Ranking_u = [A, C, B]$$
$$R_u = (\underset{A}{1}, \underset{B}{3}, \underset{C}{2})$$

3.2.1. Comparando rankings

Para comparar dos rankings utilizaremos la distancia Manhattan[4] dada su simplicidad y bajo costo computacional. Usando esta métrica podemos determinar que tan distante, es decir, que tan diferente o parecido es un ranking a otro utilizando la estructura de vector antes mencionada.

Entonces para obtener la distancia entre dos rankings haciendo uso de la distancia Manhattan tendríamos:

$$d_m(P, Q) = \sum_{i=1}^{n} |p_i - q_i| \tag{3.1}$$

Donde P y Q dos rankings expresados en la estructura de vector descrita en la sección anterior y estos están definidos como:

$$P = (p_1, p_2, \ldots, p_n)$$
$$Q = (q_1, q_2, \ldots, q_n)$$

De tal forma que si se desea calcular la distancia entre los rankings $R_1 = (1, 3, 2)$ y $R_2 = (2, 3, 1)$ tendríamos:

$$d_m(R_1, R_2) = |1 - 2| + |3 - 3| + |2 - 1| = 2$$

Con está medida tenemos forma de medir de manera cuantitativa la diferencias entre los rankings generados con base en los valores reales del interés del usuario y los generados por las estimaciones hechas por el SR a evaluar. Con base en esto, podemos adaptar el MSE y el RMSE para medir de manera cuantitativa el desempeño del SR y así podemos compararlo con otros SR.

Así adaptando las fórmulas para calcular MSE y RMSE ahora tenemos:

$$MSE_{ranking} = \frac{\sum_{u \in U'} d_m(R_u, \overline{R_u})}{|U'|} \tag{3.2}$$

Donde:

U'	Es el conjunto de usuarios a los cuales estamos intentando recomendar en el experimento.
R_u	Es el ranking generado con los valores reales para el usuario u
$\overline{R_u}$	Es el ranking generado con las estimaciones hechas por el SR para el usuario u
$d_m(R_u, \overline{R_u})$	Es la distancia de Manhattan entre los rankings R_u y $\overline{R_u}$

Y de manera similar tenemos el RMSE :

$$RMSE_{ranking} = \sqrt{\frac{\sum_{u \in U'} d_m(R_u, \overline{R_u})}{|U'|}} \tag{3.3}$$

3.3. SR para la validación del modelo de evaluación

Para validar que el modelo de evaluación MERERO mide la efectividad de los SR de la manera en que se planteó en la sección anterior se requieren SR de referencia. Se proponen los siguientes SR sintéticos como casos de referencia de tal manera que nos permitan verificar el comportamiento del modelo de evaluación propuesto. En este caso definimos un SR sintético como uno que no pretende

estimar del nivel de relevancia de los ítems, sino que utiliza el valor real para calcular un valor el cual nos ayude a saber exactamente como debería ser calificado por los modelos de evaluación de SR y por lo tanto los podemos usar como referencia.

Los tres SR que se proponen como referencia son:

BestSR Este representa un SR perfecto. El valor de la relevancia de un ítem i para un usuario u es estimado de manera exacta, por lo que los rankings generados deben ser idénticos a los rankings originales. Dada esta premisa, este SR debe de quedar evaluado de manera más alta con respecto a cualquier otro.

MaxMSESR Este ejemplifica al peor SR para los modelos de evaluación *fuera de línea* basados en estimaciones puntuales aunque está definido de tal manera que los rankings estimados correspondan a los originales.

FlipSR Las estimaciones hechas por este SR hace que los rankings estimados queden en orden inverso con respecto a los originales con lo que se maximiza la distancia de los rankings estimados con respecto a los rankings originales.

Definidos de esta manera, al medir y comparar la efectividad de cada uno, deberían suceder que el BestSR quedara como el mejor, FlipSR como el peor y MaxMSESR entre ambos SR, es decir mejor que FlipSR pero peor calificado que BestSR. Esto es debido a que FlipSR estima rankings que son completamente contrarios a los rankings que se generarían con los valores reales del nivel de relevancia de los ítems para cada usuario. Por lo tanto un modelo de evaluación que mida la efectividad de los SR con respecto a la estimación del nivel de relevancia relativo de los ítems para los usuario debería calificar a BestSR, MaxMSESR y FlipSR de la manera aquí descrita.

Antes de proseguir cabe aclarar que se hace la suposición que estos SR tienen acceso a los valores reales del nivel de relevancia de cada ítems para los usuarios dado que nuestro modelo de evaluación es *fuera de línea* y por definición se cuenta con esta información.

A continuación se definirán los símbolos comunes que se utilizarán a lo largo de la descripción de los SR de referencia:

Definición 1 (Simbología a utilizar)**.**

r_{ui}	Es el valor real del interés del usuario u sobre el ítem i
$\overline{r_{ui}}$	Valor estimado del SR evaluado
r_{max}	El valor máximo de las calificaciones que puede tener r_{ui} en el SR
r_{min}	El valor mínimo de las calificaciones que puede tener r_{ui} en el SR
r_{med}	El valor medio de las calificaciones, es decir $\frac{1}{2}(r_{min} + r_{max})$

3.3.1. BestSR

El primer RS de control, es aquel que estima de manera exacta el nivel de interés que el usuario tiene por cada ítem.

Definición 2 (BestSR)**.** La función de estimación para el sistema de recomendación *BestSR* se define como:

$$\overline{r_{ui}} = r_{ui} \tag{3.4}$$

Por definición es el mejor RS ya que las estimaciones realizadas por él corresponden exactamente al valor expresado por el usuario.

3.3.2. FlipSR

Lo que se busca con este SR artificial es que los elementos en el ranking generados con las estimaciones estén en el orden inverso con respecto al ranking generado usando los valores reales de tal manera que se maximice la distancia entre ellos al compararlos con la distancia d_m. Entonces si tenemos el ranking original:

$$\begin{aligned} Ranking_u &= [A, C, B, D] \\ R_u &= (\underset{A}{1}, \underset{B}{3}, \underset{C}{2}, \underset{D}{4}) \end{aligned}$$

Al hacer las estimaciones con este SR, el ranking deberá quedar:

$$\begin{aligned} \overline{Ranking_u} &= [D, B, C, A] \\ \overline{R_u} &= (\underset{A}{4}, \underset{B}{2}, \underset{C}{3}, \underset{D}{1}) \end{aligned}$$

Es decir, que si un ítem a está en la posición i en el ranking original de n elementos, este ítem debe quedar en la posición $(n+1-i)$ dentro del ranking generado con base en las estimaciones del SR.

Definición 3 (FlipSR)**.** La función de estimación para el sistema de recomendación *FlipSR* se define como:

$$\overline{r_{ui}} = r_{min} + (r_{max} - r_{ui}) \tag{3.5}$$

Por demostrar: con esta función de estimación los elementos en los rankings generados se encuentran en el orden inverso con respecto al ranking generado con los valores reales.

Demostración. Supongamos que nuestro conjunto de ítems recomendables es $\{a_1, a_2, \ldots, a_n\}$ y el ranking de ítems original es:

$$\begin{aligned} Ranking_u &= [a_{k_1}, a_{k_2}, ..., a_{k_n}] \\ R_u &= (\underset{a_1}{p_1}, \underset{a_2}{p_2}, \ldots, \underset{a_n}{p_n}) \end{aligned}$$

Donde $p_i = j$ tal que $a_i = a_{k_j}$ y $p_c \neq p_d \ \forall c, d$ tal que $c \neq d$, es decir, p_i es la posición en el ranking del ítem a_i.

Además nombremos $r_{ua_{k_i}}$ el nivel de interés del usuario u sobre el ítem a_{k_i}, con lo cual sabemos que:

$$r_{ua_{k_1}} \geq r_{ua_{k_2}} \cdots \geq r_{ua_{k_n}}$$

Si tomamos cualesquiera dos elementos del ranking a_{k_i} y a_{k_j} tal que $r_{ua_{k_j}} \geq r_{ua_{k_i}}$

$$\begin{aligned} &\implies -r_{ua_{k_j}} \leq -r_{ua_{k_i}} \\ &\implies r_{max} - r_{ua_{k_j}} \leq r_{max} - r_{ua_{k_i}} \\ &\implies r_{max} + r_{min} - r_{ua_{k_j}} \leq r_{max} + r_{min} - r_{ua_{k_i}} \\ &\implies r_{max} + \left(r_{min} - r_{ua_{k_j}}\right) \leq r_{max} + \left(r_{min} - r_{ua_{k_i}}\right) \\ &\implies \overline{r_{ua_{k_j}}} \leq \overline{r_{ua_{k_i}}} \quad \forall a_{k_j}, a_{k_i} \in Ranking_u : r_{ua_{k_j}} \geq r_{ua_{k_i}} \\ &\implies \overline{r_{ua_{k_n}}} \geq \cdots \geq \overline{r_{ua_{k_2}}} \geq \overline{r_{ua_{k_1}}} \\ &\implies \overline{Ranking_u} = [a_{k_n}, ..., a_{k_2}, a_{k_1}] \end{aligned}$$

Entonces el elemento a_{k_j} que estaba en la posición j dentro de $Ranking_u$, queda en la posición $(n+1-j)$ dentro de $\overline{Ranking_u}$, por lo que:

$$\overline{R_u} = (\underbrace{n+1-p_1}_{a_1},\ \underbrace{n+1-p_2}_{a_2}, \ldots,\ \underbrace{n+1-p_n}_{a_n}) \text{ con } p_i = j \text{ tal que } a_i = a_{k_j} \tag{3.6}$$

□

Observaciones: La distancia entre el ranking original (R_u) y el ranking generado ($\overline{R_u}$) está dada por:

$$d_m(R_u, \overline{R_u}) = \begin{cases} \frac{1}{2}(n^2) & \text{si } (n \text{ es par}) \\ \\ \frac{1}{2}(n^2-1) & \text{si } (n \text{ es impar}) \end{cases} \tag{3.7}$$

Demostración. Supongamos que $R_u = (p_1, p_2, \ldots, p_n)$, con lo cual:

$$\begin{aligned} R_u = (p_1, p_2, \ldots, p_n) &\implies \overline{R_u} = (n+1-p_1, n+1-p_2, \ldots, n+1-p_n) \\ &\implies d_m(R_u, \overline{R_u}) = \sum_{i=1}^{n} |n+1-p_i-p_i| \\ &\implies d_m(R_u, \overline{R_u}) = \sum_{i=1}^{n} |n+1-2i| \end{aligned} \tag{3.8}$$

Por otro lado cada p_i es un número entero del conjunto $\{1, 2, \ldots, n\}$ y todas son distintas por lo que alguna $p_i = 1$, para otra i distinta $p_i = 2$ y así hasta tener los n enteros. Dado esto, podemos ordenar la suma de la siguiente manera:

$$\begin{aligned} \sum_{i=1}^{n} |n+1-2p_i| &= |n+1-2p_1| + |n+1-2p_2| + \cdots + |n+1-2(1)| + \cdots + |n+1-2p_n| \\ &= |n+1-2(1)| + |n+1-2(2)| + \cdots + |n+1-2(n)| \\ &= \sum_{i=1}^{n} |n+1-2i| \end{aligned}$$

Aquí tenemos dos casos, cuando n es par u cuando n es impar. Si n es par entonces $\left(n' = \frac{n}{2}\right)$ es un entero positivo y sabemos que:

$$\begin{aligned} i \leq n' &\implies (n+1-2i) \geq 0 \\ i > n' &\implies -(n+1-2i) = (2i-(n+1)) \geq 0 \end{aligned}$$

Entonces si partimos la suma y simplificamos términos :

$$\begin{aligned}
\sum_{i=1}^{n}|n+1-2i| &= \sum_{i=1}^{n'}|n+1-2i| + \sum_{i=n'+1}^{n}|n+1-2i| \\
&= \sum_{i=1}^{n'}(n+1)-2i + \sum_{i=n'+1}^{n}|2i-(n+1)| \\
&= \sum_{i=1}^{n'}(n+1) - \sum_{i=1}^{n'}2i + \sum_{i=n'+1}^{n}(2i-(n+1)) \\
&\quad \text{sustituyendo } i \text{ por } (n'+j) \text{ y ordenando la numeración} \\
&\quad \text{de la tercera suma} \\
&= \sum_{i=1}^{n'}(n+1) - 2\sum_{i=1}^{n'}i + \sum_{j=1}^{n'}(2(n'+j)-(n+1)) \\
&= \sum_{i=1}^{n'}(n+1) - 2\sum_{i=1}^{n'}i + \sum_{j=1}^{n'}(2n'+2j-(n+1)) \\
&= \sum_{i=1}^{n'}(n+1) - 2\sum_{i=1}^{n'}i + \sum_{j=1}^{n'}2n' + \sum_{j=1}^{n'}2j - \sum_{j=1}^{n'}(n+1) \\
&= \sum_{i=1}^{n'}(n+1) - 2\sum_{i=1}^{n'}i + 2\sum_{j=1}^{n'}n' + 2\sum_{j=1}^{n'}j - \sum_{j=1}^{n'}(n+1) \\
&= \sum_{i=1}^{n'}(n+1) - \sum_{j=1}^{n'}(n+1) - 2\sum_{i=1}^{n'}i + 2\sum_{j=1}^{n'}j + 2\sum_{j=1}^{n'}n' \\
&= 2\sum_{j=1}^{n'}n' \\
&= 2(n')n' = 2(n')^2 = 2\left(\frac{n}{2}\right)^2 \\
\sum_{i=1}^{n}|n+1-2i| &= \frac{n^2}{2}
\end{aligned} \tag{3.9}$$

Si n es impar entonces $\left(n_1 = \frac{n-1}{2}\right)$ y $\left(n_2 = \frac{n+1}{2}\right)$ son números enteros positivos y al igual que el caso anterior sabemos:

$$\begin{aligned}
i \leq n' &\implies (n+1-2i) \geq 0 \\
i > n' &\implies -(n+1-2i) = (2i-(n+1)) \geq 0
\end{aligned}$$

Entonces si de nuevo partimos la suma y simplificamos términos :

$$\begin{aligned}
\sum_{i=1}^{n}|n+1-2i| &= \sum_{i=1}^{n_1}|n+1-2i| + |n+1-2n_2| + \sum_{i=n_2+1}^{n}|n+1-2i| \\
&= \sum_{i=1}^{n_1}|n+1-2i| + \left|n+1-2\left(\frac{n+1}{2}\right)\right| + \sum_{i=n_2+1}^{n}|2i-(n+1)| \\
&= \sum_{i=1}^{n_1}(n+1-2i) + |n+1-n+1| + \sum_{i=n_2+1}^{n}(2i-(n+1)) \\
&= \sum_{i=1}^{n_1}(n+1) - 2\sum_{i=1}^{n_1} i + |0| + \sum_{i=n_2+1}^{n}(2i-(n+1))
\end{aligned}$$

es fácil ver que de n_2 a n tenemos n_1 elementos por lo que de nuevo podemos sustituir i reordenar la numeración en la tercera suma:

$$\begin{aligned}
&= \sum_{i=1}^{n_1}(n+1) - 2\sum_{i=1}^{n_1} i + \sum_{j=1}^{n_1}(2(n_2+j)-(n+1)) \\
&= \sum_{i=1}^{n_1}(n+1) - 2\sum_{i=1}^{n_1} i + \sum_{j=1}^{n_1}(2n_2+2j-(n+1)) \\
&= \sum_{i=1}^{n_1}(n+1) - 2\sum_{i=1}^{n_1} i + \sum_{j=1}^{n_1}2n_2 + \sum_{j=1}^{n_1}2j - \sum_{j=1}^{n_1}(n+1) \\
&= \sum_{i=1}^{n_1}(n+1) - \sum_{j=1}^{n_1}(n+1) - 2\sum_{i=1}^{n_1} i + 2\sum_{j=1}^{n_1} j + 2\sum_{j=1}^{n_1} n_2 \\
&= 2\sum_{j=1}^{n_1} n_2 \\
&= 2(n_1)(n_2) = 2\left(\frac{n-1}{2}\right)\left(\frac{n+1}{2}\right) \\
&= 2\left(\frac{n^2-1}{2^2}\right) \\
\sum_{i=1}^{n}|n+1-2i| &= \frac{n^2-1}{2}
\end{aligned} \tag{3.10}$$

Por lo tanto la distancia entre el ranking original (R_u) y el ranking generado ($\overline{R_u}$) utilizando FlipSR está dada por:

$$d_m(R_u, \overline{R_u}) = \begin{cases} \frac{1}{2}(n^2) & \text{si } (n \text{ es par}) \\ \frac{1}{2}(n^2-1) & \text{si } (n \text{ es impar}) \end{cases} \tag{3.11}$$

□

Falta demostrar que la distancia d_m entre el ranking original (R_u) y el ranking generado ($\overline{R_u}$) es mayor o igual a la distancia con cualquier otro ranking con los mismos ítems, es decir:

$$
\begin{aligned}
&R_u = (p_1, p_2, \ldots, p_n) \\
&\overline{R_u} = (\overline{p_1}, \overline{p_2}, \ldots, \overline{p_n}) \\
&Q_u = (q_1, q_2, \ldots, q_n) \text{ con } q_i, p_i, \overline{p_i} \in \{1, \ldots, n\}
\end{aligned}
$$

$$
\implies d_m(R_u, \overline{R_u}) \geq d_m(R_u, Q_u) \; \forall \; Q_u
$$

O lo que es lo mismo:

$$
\sum_{i=1}^{n} |p_i - \overline{p_i}| \geq \sum_{i=1}^{n} |p_i - q_i| \tag{3.12}
$$

Dada la ecuación (3.7), podemos deducir:

$$
\begin{aligned}
\text{Si } n \text{ es par} &\implies \sum_{i=1}^{n} |p_i - \overline{p_i}| \geq \sum_{i=1}^{n} |p_i - q_i| \iff \left(\frac{n^2}{2}\right) \geq \sum_{i=1}^{n} |p_i - q_i| \\
\text{Si } n \text{ es impar} &\implies \sum_{i=1}^{n} |p_i - \overline{p_i}| \geq \sum_{i=1}^{n} |p_i - q_i| \iff \left(\frac{n^2 - 1}{2}\right) \geq \sum_{i=1}^{n} |p_i - q_i|
\end{aligned}
$$

Por lo que la demostración se reduce a demostrar que:

$$
\text{Si } n \text{ es par} \sum_{i=1}^{n} |p_i - q_i| \leq \left(\frac{n^2}{2}\right) \tag{3.13}
$$

$$
\text{Si } n \text{ es impar} \sum_{i=1}^{n} |p_i - q_i| \leq \left(\frac{n^2 - 1}{2}\right) \tag{3.14}
$$

Podemos simplificar aún más la demostración reordenando la suma ($\sum_{i=1}^{n} |p_i - q_i|$) como en la sección anterior donde se demostró (3.7):

$$
\sum_{i=1}^{n} |p_i - q_i| = |p_1 - q_1| + |p_2 - q_2| + \cdots + |p_n - q_n|
$$

Pero sabemos que ($p_{k_1} = 1$) para alguna k_1, ($p_{k_2} = 2$) para alguna k_2 y así sucesivamente hasta k_n tal que ($p_{k_n} = n$). Por lo tanto:

$$
\begin{aligned}
\sum_{i=1}^{n} |p_i - q_i| &= |1 - q_{k_1}| + |2 - q_{k_2}| + \cdots + |n - q_{k_n}| \\
&= \sum_{i=1}^{n} |i - q_{k_i}| \qquad (3.15)
\end{aligned}
$$

Donde q_{k_i} es la correspondiente entrada de Q_u para la expresión $|p_j - q_j|$ donde ($p_j = i$). Entonces combinando (3.13), (3.14) y (3.15) tenemos:

$$
d_m(R_u, \overline{R_u}) \geq d_m(R_u, Q_u) \; \forall \; Q_u \iff \begin{cases} \text{Si } n \text{ es par} & \sum_{i=1}^{n} |i - q_{k_i}| \leq \left(\frac{n^2}{2}\right) \\ \text{Si } n \text{ es impar} & \sum_{i=1}^{n} |i - q_{k_i}| \leq \left(\frac{n^2 - 1}{2}\right) \end{cases} \tag{3.16}
$$

Demostración. Para presentar los casos en una sola demostración usando el método de inducción, primero vamos a demostrar que si n es impar, por lo que se debe demostrar que se cumple que $\sum_{i=1}^{n} |i - q_{k_i}| \leq \left(\frac{n^2-1}{2}\right)$, dado que si se cumple para este caso, también se cumple para el otro ya que:

$$\frac{n^2-1}{2} \leq \frac{n^2}{2}$$

Entonces nuestro caso base es cuando $n = 3$, por lo que los posibles rankings son:

$$\begin{array}{ll} Q_{1u} = (1,2,3) & Q_{4u} = (2,3,1) \\ Q_{2u} = (1,3,2) & Q_{5u} = (3,1,2) \\ Q_{3u} = (2,1,3) & Q_{6u} = (3,2,1) \end{array}$$

Y todas las posibles distancias $\left(d_m(Q_{iu}, Q_{ju})\right)$ están en la siguiente tabla:

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6
Q_1	0	2	2	4	4	4
Q_2	2	0	4	2	4	4
Q_3	2	4	0	4	2	4
Q_4	4	2	4	0	4	2
Q_5	4	4	2	4	0	2
Q_6	4	4	4	2	2	0

No importa que Q_{iu} tomemos como R_u, $d_m(R_u, \overline{R_u})$ es :

$$\begin{aligned} d_m(R_u, \overline{R_u}) &= \left(\frac{n^2-1}{2}\right) \\ &= \left(\frac{3^2-1}{2}\right) \\ &= \left(\frac{9-1}{2}\right) = 4 \end{aligned}$$

Que es mayor igual que cualquiera de las posibles distancias que tenemos, por lo que queda demostrado el caso base. Por lo tanto, nuestra hipótesis de inducción es que se cumple:

$$\sum_{i=1}^{n} |i - q_{k_i}| \leq \left(\frac{n^2-1}{2}\right)$$

Para demostrar para $(n+1)$ tenemos que agregar el termino $|(n+1) - q_{k_{(n+1)}}|$, pero al ser $q_{k_{(n+1)}}$ una posición dentro de un ranking de $(n+1)$ elementos implica que $1 \leq q_{k_{(n+1)}} \leq (n+1)$, entonces:

$$\left|(n+1) - q_{k_{(n+1)}}\right| \leq n$$

Sumando esto con nuestra hipótesis de inducción:

$$\sum_{i=1}^{n} |i - q_{k_i}| + \left|(n+1) - q_{k_{(n+1)}}\right| \leq \left(\frac{n^2-1}{2}\right) + n$$

$$\sum_{i=1}^{n+1} |i - q_{k_i}| \leq \left(\frac{n^2-1+2n}{2}\right)$$

$$\leq \left(\frac{n^2+2n+1-1-1}{2}\right)$$

$$\leq \left(\frac{n^2+2n+1-1-1}{2}\right)$$

$$\leq \left(\frac{(n+1)^2-1-1}{2}\right)$$

$$\leq \left(\frac{(n+1)^2-1}{2} - \frac{1}{2}\right)$$

$$\leq \left(\frac{(n+1)^2-1}{2} - \frac{1}{2}\right) \leq \left(\frac{(n+1)^2-1}{2}\right)$$

$$\sum_{i=1}^{n+1} |i - q_{k_i}| \leq \left(\frac{(n+1)^2-1}{2}\right)$$

Con lo que queda demostrado (3.16) y como consecuencia queda demostrado (3.12).

□

3.3.3. MaxMSESR

Este SR maximiza el MSE (*ver Sección 2.1*) con respecto a los valores reales respetando que los valores de estas estimaciones estén dentro del rango $[r_{min}, r_{max}]$. Está diseñado para ser el peor calificado con las técnicas tradicionales de evaluación *fuera de línea* de SR.

Definición 4 (MaxMSESR)**.** La función de estimación para el sistema de recomendación *MaxMSESR* se define como:

$$\overline{r_{ui}} = \begin{cases} r_{max} & \text{si } (r_{ui} < r_{med}) \\ r_{min} & \text{si } (r_{ui} \geq r_{med}) \end{cases} \tag{3.17}$$

Por demostrar que el SR *MaxMSERS* maximiza MSE. Para esto hay que demostrar que para cada estimación puntal hecha la diferencia entre esta ($\overline{r_{ui}}$) y el valor real (r_{ui}) es la máxima posible dentro del rango $[r_{min}, r_{max}]$, por lo que se tiene que demostrar:

$$\text{Sí } (r_{ui} \geq r_{med}) \implies |r_{ui} - y|^2 \leq |r_{ui} - r_{min}|^2 \quad \forall\, y \in [r_{min}, r_{max}] \tag{3.18}$$

$$\text{Sí } (r_{ui} < r_{med}) \implies |r_{ui} - y|^2 \leq |r_{max} - r_{ui}|^2 \quad \forall\, y \in [r_{min}, r_{max}] \tag{3.19}$$

Demostración. Tenemos que separar la demostración en dos casos, por lo que empecemos por el caso cuando $(r_{ui} \geq r_{med})$. Sin pérdida de generalidad, podemos eliminar el operador de valor absoluto dado que $(r_{ui} \geq r_{med})$, por lo que solo tenemos que demostrar que:

$$|r_{ui} - y|^2 \leq (r_{ui} - r_{min})^2 \quad \forall\, y \in [r_{min}, r_{max}]$$

Observaciones: r_{ui} y y se pueden reescribir como:

$$r_{ui} = r_{min} + k$$
$$y = r_{min} + q$$

con $k, q \geq 0$ y $k, q \in [0, d_{max}]$, tal que $d_{max} = (r_{max} - r_{min})$. Sustituyendo en ambos lados de la inecuación que queremos demostrar:

$$|r_{ui} - y|^2 = |r_{min} + k - (r_{min} + q)|^2 = |k - q|^2$$
$$(r_{ui} - r_{min})^2 = (r_{min} + k - r_{min})^2 = k^2$$

Por lo tanto

$$|r_{ui} - y|^2 \leq (r_{ui} - r_{min})^2 \iff |k - q|^2 \leq k^2$$

Entonces lo que hay que demostrar:

$$|k - q|^2 \leq k^2$$

Otra observación es que :

$$\begin{aligned} &(r_{ui} \geq r_{med}) \\ &\quad \implies (r_{min} + k) \geq r_{med} \\ &\quad \implies k \geq (r_{med} - r_{min}) \end{aligned} \tag{3.20}$$

Por definición $0 \leq q \leq d_{max} = (r_{max} - r_{min})$ y $(r_{med} - r_{min} = r_{max} - r_{med})$ entonces:

$$0 \leq q \implies -k \leq q - k \tag{3.21}$$

$$\begin{aligned} q \leq (r_{max} - r_{min}) &= (r_{max} - r_{med} + r_{med} - r_{min}) \\ &= (r_{med} - r_{min} + r_{med} - r_{min}) \\ &= 2(r_{med} - r_{min}) \\ &= 2(r_{med} - r_{min}) \leq 2k \quad \text{[por (3.20)]} \\ \implies q &\leq 2k \\ \implies q - k &\leq k \end{aligned} \tag{3.22}$$

Entonces por (3.21) y (3.22) tenemos que:

$$\begin{aligned} -k \leq q - k \leq k &\implies |q - k| \leq k \\ &\implies |k - q| \leq k \\ &\implies |k - q|^2 \leq k^2 \\ &\implies |r_{ui} - y|^2 \leq (r_{ui} - r_{min})^2 \quad \forall\ y \in [r_{min}, r_{max}] \end{aligned}$$

Con lo que queda demostrada la inecuación (3.18).
Para demostrar (3.18), utilizamos las mismas sustituciones que en el caso anterior y nos queda que:

$$|r_{ui} - y| = |r_{min} + k - (r_{min} + q)| = |k - q|$$
$$(r_{max} - r_{ui}) = (r_{max} - (r_{min} + k)) = d_{max} - k$$

Además

$$\begin{aligned} &(r_{ui} < r_{med}) \\ &\quad \implies (r_{min} + k) < r_{med} \\ &\quad \implies k < (r_{med} - r_{min}) \end{aligned} \tag{3.23}$$

Por lo tanto

$$|r_{ui} - y|^2 \leq (r_{max} - r_{ui})^2 \iff |k - q|^2 \leq (d_{max} - k)^2$$

$$\begin{aligned} q \leq d_{max} &\implies q - k \leq d_{max} - k \\ &\implies k - d_{max} \leq k - q \\ &\implies -(d_{max} - k) \leq k - q \end{aligned} \tag{3.24}$$

Por (3.24) sabemos que $(d_{max} - k \leq q - k)$ y dado que $(k \leq d_{max})$

$$\begin{aligned} (k \leq d_{max}) &\implies 0 \leq d_{max} - k \\ &\implies k - d_{max} \leq 0 \\ &\implies k - d_{max} \leq 0 \leq d_{max} - k \leq q - k \\ &\implies k - d_{max} \leq q - k \\ &\implies k - q \leq d_{max} - k \end{aligned} \tag{3.25}$$

$$\begin{aligned} -(d_{max} - k) \leq k - q \leq d_{max} - k &\implies |k - q| \leq d_{max} - k \\ &\implies |k - q|^2 \leq (d_{max} - k)^2 \\ &\implies |r_{ui} - y|^2 \leq (r_{max} - r_{ui})^2 \end{aligned}$$

Con lo que queda demostrada la inecuación (3.19). □

Capítulo 4

Experimentación

4.1. Implementación

Se utilizó el framework Recommender101 (ver figura 4.1), para implementar y poner a prueba el modelo de evaluación MERERO junto con los SR BestSR, MaxMSESR y FlipSR que sirven como control para las pruebas, además del SVD en su implementación de FunkSVD[35] como referencia. Como parte de la implementación y para permitir la implementación de las funcionalidades de los SR para control, se realizaron cambios menores directamente a algunas de las clases del framework ya que BestSR, MaxMSESR y FlipSR necesitaban de más información de la que ofrecía por omisión al utilizar el API del fremework. En las siguientes secciones se detalla un poco más el framework además de las consideraciones para la implementación tanto del modelo MERERO, como de los SR de control propuestos en esta tesis.

4.1.1. Recommender101

Es un *framework* escrito en Java para llevar a cabo experimentos fuera de línea con SR. Sus principales ventajas es que es ligero y relativamente fácil de usar y extender. Como parte del framework, Recommender101 ofrece un conjunto de SR y de métricas de evaluación ya implementadas, como por ejemplo:

Sistemas de recomendación

- Filtrado colaborativo (kNN), con base en usuarios o en ítems.
- SlopeOne.
- Métodos de factorización de matriz, por ejemplo, FunkSVD, SVD y SVD++.
- Redes Bayesianas.
- Máquinas de soporte vectorial.
- Filtrado con basado en contenido.

Métricas de evaluación

- Precisión
- Recall
- MAE
- RMSE

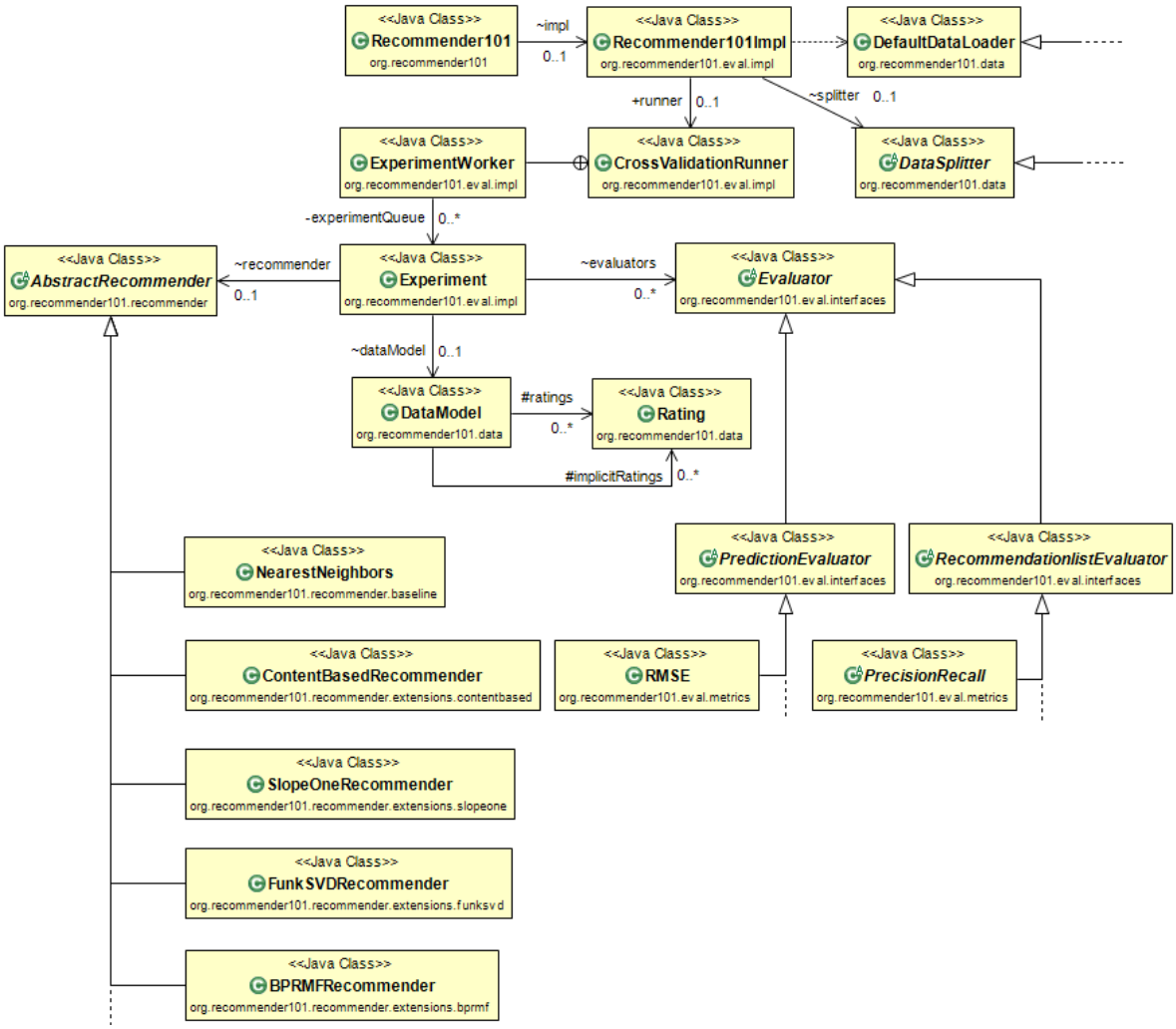

Figura 4.1: Estructura da clases del framework Recommender101

Configuración de pruebas

El framework está pensado para realizar experimentos para evaluar distintos sistemas de recomendación. En este sentido, Recommender101 contiene un conjunto de clases y archivos de configuración que permiten realizar experimentos personalizados con sistemas de recomendación y evaluadores personalizados de tal manera que uno puede implementar sus propios sistemas de recomendación y evaluarlos con métodos de evaluación propios.

Para simplificar las cosas, Recommender101 cuenta con la clase ejecutable `Recommender101`[1] la cual ejecuta los experimentos configurados en un archivo en formato *properties*[2] de java. A continuación se muestra un ejemplo minimalista de la configuración con este archivo seguido por la explicación de cada parametro en la tabla 4.1.

[1] `org.recommender101.Recommender101`
[2] `https://en.wikipedia.org/wiki/.properties`

```
###### Minimal Sample ######

## Data ##
DataLoaderClass=\
        org.recommender101.data.DefaultDataLoader:\
                filename=data/bookcrossing/bx_UserItemRating.csv|\
                separatorString=,|\
                sampleNUsers=1000

GlobalSettings.minRating = 1
GlobalSettings.maxRating = 10

# Specify the minimum rating that will be considered a hit.
GlobalSettings.listMetricsRelevanceMinRating = 1

# The class to split the data into train and test splits. It must
# implement the DataSplitterInterface.
# The default behavior is n-fold cross-validation.
DataSplitterClass=\
        org.recommender101.data.DefaultDataSplitter:\
                nbFolds=10

## Algorithms ##

# List the algorithms that should be evaluated. They must extend
# the AbstractRecommender class.
# Parameters can be set as arguments. If an argument is missing,
# a default value is used.
AlgorithmClasses=\
        oeg.merero.recommenders.BestRecommender, \
        org.recommender101.recommender.extensions.funksvd.FunkSVDRecommender:\
                numFeatures=50|initialSteps=50

## Metrics ##

# Specify the global setting for top-N that will be used by, e.g.,
# precision and recall,
# It can be set individually for each metric as an argument
GlobalSettings.topN = 10

# List the metrics to be measured. They must implement either
# the PredictionEvaluator or RecommendationListEvaluator interface
Metrics =\
        oeg.merero.recommenders.evaluator.MERERROEvaluator,\
        org.recommender101.eval.metrics.RMSE
```

Listing 4.1: Ejemplo de archivo de configuración

Entrada	Descripción
DataLoaderClass	Especifica la clase la cual se encargada de leer los datos para realizar los experimentos. Se pueden definir el valor de parámetros extra para la clase utilizando el delimitador "**:**" seguido de una lista de elementos `llave=valor` separados por el caracter "|". El framework cuenta con un lector de datos que contempla la lectura de un archivo en formato *csv*[3] y cuenta con los siguientes parámetros: **filename** La ruta a un archivo *csv* separado por `<separatorString>` donde están almacenado los datos. **separatorString** El carácter que delimita cada uno de los campos de cada fila. **sampleNUsers** La cantidad máxima de filas (es decir, *usuarios*) que serán leídos del archivo.
GlobalSettings.minRating GlobalSettings.maxRating	Se puede especificar el valor mínimo y máximo de las calificaciones que se quieren leer.
GlobalSettings. listMetricsRelevanceMinRating	Especifica el mínimo valor para que una estimación hecha por algún SR se consideré dentro de la evaluación
DataSplitterClass	Especifica la clase que se utilizará para dividir el conjunto de datos en el subconjunto para realizar el entrenamiento (si el SR requiere de una etapa de entrenamiento) y el subconjunto para realizar las pruebas. Recomender101 implementa una clase pensada para hacer pruebas de validación cruzada con n iteraciones. Más adelante en la sección "*4.2.3 Segmentación de los datos para los experimentos*" se explica más de esta técnica de dividir los datos para pruebas y entrenamiento. Al usar esta clase se puede configurar el número de iteraciones a realizar para la validación asignandole un valor a la variable `nbFolds`.
AlgorithmClasses	Con este parámetro se configuran los distintos algoritmos de SR que se quieren poner a prueba. Esta es una lista separada por comas con los nombres de las clases que implementas los algoritmos de SR. Opcionalmente se pueden pasar parámetros a las clases de la misma manera que en el parámetro de configuración `DataLoaderClass`. En la configuración de ejemplo tenemos dentro de la lista la clase : `org.recommender101.recommender.extensions.funksvd.FunkSVDRecommender` La cual implementa el método de factorización de matrices SVD y en la cual podemos configurar el número de características que tendrán las matrices con las que queremos aproximar nuestra matriz original `numFeatures`) y el numero de iteraciones para crear el modelo (`initialSteps`).

[3]Archivo en formato de valores separados por comas y salto de lineas o **c***omma* **s***eparated* **v***alores* aunque de manera genérica pueden ser separados por cualquier carácter.

GlobalSettings.topN	Configura el valor global para el tamaño de la lista de recomendaciones a obtener. De tal manera que solo se recomendaran los *topN* ítems con los valores más altos en la estimación de la relevancia. Cabe aclarar que este valor se puede configurar para cada métrica de evaluación que se van a realizar.
Metrics	Con este parámetro se configura la lista de métricas con las cuales se van a evaluar los SR configurados en `AlgorithmClasses`. De nuevo es una lista separada por comas de las clases que implementan cada una de las métricas de evaluación.

Tabla 4.1: Definición de entradas el archivo de configuración para `Recommender101`

Implementando nuevos SR

Como se observa en la figura 4.1, para implementar nuevos SR se debe de extender la clase abstracta `org.recommender101.recommender.AbstractRecommender`. Ademas estas implementaciones deben contener las siguientes funciones:

`public abstract void init() throws Exception`

Esta función es invocada al inicio de los experimentos para que el SR haga todos los cálculos necesarios para poder iniciar con el cálculo de las estimaciones. Por ejemplo en el caso de un SR colaborativo basado en usuarios, dentro de de esta función es donde se debería calcular la matriz de similitud entre usuario utilizando el conjunto de datos de entrenamiento. Este conjunto puede ser obtenido por medio de la función `public DataModel getDataModel()` que esta implementada en la clase `AbstractRecommender`.

`public abstract float predictRating(int user, int item)`

Debe realizar la estimación del nivel de relevancia del ítem (*item*) para el usuario (*user*), por lo que en esta se debe de implementar los algoritmos necesarios para obtener esta estimación.

`public abstract List<Integer>recommendItems(int user)`

Tiene que regresar una lista con los ítems que el SR estime deben ser recomendados al usuario *user* ordenados por el nivel de relevancia que se estimó para cada uno de los ítems que la forman.

Implementando nuevas métricas de evaluación

Para implementar nuevas métricas de evaluación de SR se tienen que extender alguna de las siguientes dos clases que se encuentran dentro del paquete `org.recommender101.eval.interfaces`:

- `PredictionEvaluator`
  (ver sección "*A.1 Clase `PredictionEvaluator`*")
- `RecommendationListEvaluator`
  (ver sección "*A.2 Clase `RecommendationListEvaluator`*")

La primera clase se extiende si se desea crear una métrica de evaluación que se enfoque en evaluar las estimaciones puntuales de las estimaciones como es el caso del MSE, el MAE o RMSE como se observa en la figura 4.1. Por otro lado la segunda clase está pensada para ser extendida por implementaciones de métricas de evaluación que trabajen sobre la lista de recomendaciones como es nuestro caso. Al extender esta clase es necesario implementar las funciones:

`public abstract void addRecommendations(Integer user, List<Integer>list)`

Cada que se genera una lista de recomendaciones por el SR a evaluar se le pasa a esta función para que sea evaluada y se vaya acumulando de manera interna para ir construyendo la evaluación global del SR. La lista contiene solo los identificadores numéricos de los ítems a recomendar

ordenados por el nivel de relevancia estimado por el SR. Cabe destacar que la lista contiene todos los ítems de los cuales el SR pudo hacer la estimación, por lo que hay que tomar el parámetro `topN` que se obtiene por medio de la función `int getTopN()` que es el tamaño máximo que puede tener la lista de recomendaciones.

`public abstract float getEvaluationResult();`

Esta función es llamada cuando se termino de evaluar las estimaciones y la lista de recomendaciones para la métrica de evaluación que se está implementando, de tal manera que se debe regresar el valor sumarizado de las evaluaciones de las listas de recomenaciones con las que fue llamada la función anterior (es decir la función `addRecommendations`).

Modificaciones realizadas a Recommender101

Como se mencionó anteriormente, fue necesario hacer algunas modificaciones directamente a las clases de Recommender101. Específicamente se modificó la clase `Experiment`[4], ya que en la implementación original, los experimentos proveen a los SR a ser puestos a prueba solo el conjunto de datos de entrenamiento, pero en el caso de nuestros SR de control, necesitamos conocer el conjunto completo ya que en estos se asume que al hacer la estimación del nivel de relevancia del ítem i para el usuario u se conoce el valor real del nivel de relevancia.

Para cumplir con este requisito, se utiliza un mecanismo previsto por la clase `AbstractRecommender` (ver sección "*A.1.3 Experiment*") en el cual se puede asignar información extra además de los datos de entrenamiento a la instancia del SR que se está poniendo a prueba, de tal manera que se le asigna una referencia no solo al conjunto de datos de entrenamiento sino también al conjunto de datos con el que se realizarán los experimentos y una referencia al conjunto de datos completo, es decir, el conjunto de entrenamiento más el conjunto para realizar pruebas.

Implementación del modelo MERERO

Para implementar las clases correspondientes a los tres sistemas de recomendación que se utilizan como control se desarrollo la clase abstracta intermedia:

```
oeg.merero.recommenders.AbstractTestRecommender
```

Esta engloba las funcionalidades comunes que se requieren para la implementación de cada uno de estos SR, de tal manera que en su correspondiente función `public void init()` (ver sección "*A.4 Implementación de la clase `AbstractTestRecommender`*") se encarga de calcular los valores para r_{max}, r_{med}, r_{min} que son utilizados le las clases correspondientes a los SR que se usan como control para las pruebas, de tal forma que estos últimos solo tienes que implementar la función:

```
public float predictRating(int user, int item)
```

Por otro lado esta clase sobre escribe la función:

```
public List<Integer>recommendItemsByRatingPrediction(int user)
```

Originalmente está función solo toma en cuenta los ítems que dentro del conjunto de entrenamiento que tengan por lo menos una calificación, de tal manera que si un ítem no cuenta con calificaciones en el conjunto de entrenamiento, no se tomará en cuenta para las pruebas a pesar de que esté presente con calificaciones en el conjunto de datos para pruebas. Así que, esta clase busca el conjunto de datos completo haciendo uso de la función `getExtraInformation(String key)` con la llave *`OriginalTextData`*. Esta información es asignada por la clase `Experiment` y es el cambio que se mencionó en la sección "*4.1.1 Modificaciones realizadas a Recommender101*". Además esta clase da acceso a estos datos a las clases que la extiendan por medio de la implementación de la función:

```
protected DataModel getOriginalDataModel()
```

[4] `org.recommender101.eval.impl.Experiment`

Sistema de recomendación BestSR La clase que implementa el SR BestSR es `BestRecommender` (ver sección "*A.5 Implementación de la clase `BestRecommender`*"). Esta clase extiende la clase abstracta `AbstractTestRecommender` y para obtener el valor de relevancia del usuario sobre cada ítem, es decir: implementar la función `public abstract float predictRating(int user, int item)`, utiliza la función heredada `getOriginalDataModel()` para obtener los datos reales y simplemente regresa directamente el valor que marca el conjunto de datos.

Sistema de recomendación MaxMSESR Para implementar la función `predictRating` se utiliza la función `getOriginalDataModel()` para obtener el valor real y calcular el valor de la estimación según lo definido en la sección "*3.3.3 MaxMSESR*". La clase correspondiente a la implementación de este SR está en la clase `MaxMSERecommender` (ver sección A.7).

Sistema de recomendación FlipSR Al igual que en la implementación de los SR anteriores se utiliza la función `getOriginalDataModel()` para obtener el valor real del nivel de relevancia para cada pareja de ítem - usuario y así poder implementar la función `predictRating` según lo definido en la sección "*3.3.2 FlipSR*". Aunque necesitamos conocer los elementos que forman parte del ranking original generado por el conjunto de datos de prueba para saber que elementos debemos dejar fuera para que los elementos de ambos rankings (el original y el estimado) contengan los mismos elementos aunque en orden distinto, por lo que dentro de la implementación tenemos la función:

```
private List<Integer>createOriginalRankingFor(int user)
```

La implementación de este SR está en la clase `FlipRecommender` y se puede ver en la sección A.6.

Modelo de evaluación MERERO La clase que implementa nuestro sistema de evaluación extiende de `RecommendationListEvaluator` y tiene como nombre:

```
oeg.merero.recommenders.evaluator.MERREROEvaluator.
```

de tal manera que está clase (ver sección"*A.2 Clase `RecommendationListEvaluator`*") implementa las funciones:

- `public abstract void addRecommendations(Integer user, List<Integer>list)`
- `public abstract float getEvaluationResult();`

Para realizar la evaluación necesaria de la lista de ítems recomendados *list* al usuario *user*, `MERREROEvaluator` implementa dos funciones auxiliares para llevar a cabo esta tarea según el modelo definido en esta tesis y estas funciones son:

`private Map<Integer, Integer>convertRankingToPositionVector(List<Integer>ranking)`

Esta función convierte la lista de ítems en formato de ranking a el formato de vector definido por MERERO en la sección "*3.2 El modelo MERERO*".

`private float calcDistance(List<Integer>a, List<Integer>b)`

Una vez convertidos al formato propuesto los rankings, es decir las listas de ítems con las que se invoca `addRecommendations(Integer user, List<Integer>list)`, esta función se encarga de compararlos utilizando la distancia Manhattan.

4.2. Experimentos

Para hacer las pruebas se decidió utilizar la técnica de validación cruzada en n-iteraciones, por lo cual se requirió de seleccionar un conjunto de datos. Se analizaron tres bases de datos provenientes distintos desarrollos de SR los cuales son: Bookcrossing, Jester y MovieLens. De estos se utilizó la base de datos de MovieLens ya que entre otras cosas es la que más calificaciones de usuario contiene. A partir de esto

de corrieron tres distintos experimentos con muestreos de 100, 1000, y 138,493 usuarios. Para cada muestra se evaluaron cuatro SR (BestRecommender, FunkSVDRecommender, MaxMSERecommender y FlipRecommender) utilizando los modelos de evaluación MAE, RMSE y MERERO con el objetivo de validar que este último es el que evalúa de manera correcta a los SR con respecto a lo definido en la sección "*3.3 SR para la validación del modelo de evaluación*". Al analizar los resultados de los experimentos de cada muestreo se encontró que estos muestran que el modelo de evaluación MERERO es valido.

4.2.1. Bases de datos para las pruebas

Para realizar las pruebas se analizaron tres posibles conjuntos de datos: Bookcrossing, Jester y MovieLens. De cada conjunto se obtuvo el número de usuarios ($|U|$), el número de ítems ($|I|$) y el número de calificaciones ($|R_{ui}|$), con lo cual se calculó la densidad de los datos de la siguiente manera:

$$\text{densidad} = \frac{|R_{ui}|}{|U| \times |I|}$$

La densidad nos indica el porcentaje de calificaciones que tenemos con respecto a todas las que se podrían tener, es decir, el número de calificaciones que se tendrían si se supiera la relevancia de cada ítems para cada usuario. Visto como una matriz de usuarios × ítems, el porcentaje de valores conocidos para de la matriz.

Por otro lado se obtuvo el la cantidad mínima y máxima de valores conocidos por ítem y por usuario ("Min. # valores/ítem","Min. # valores/usuario","Max. # valores/ítem","Max. # valores/usuario") además de propiedades estadísticas como la media, la desviación estándar, la curtosis[5], la asimetría[6]y el coeficiente Gini[7]de cada conjunto de datos

De estos se escogió hacer los experimentos sobre la base de datos de MovieLens ya que es el conjunto de datos más grande y tiene el coeficiente Gini más alto de los tres conjuntos de datos, como se puede ver en la descripción que se hace a continuación de cada uno de ellos.

Bookcrossing

Bookcrossing[36] es un sistema pensado para conectar usuarios amantes de la lectura que desean compartir sus libros. Aquí se pueden registrar, calificar, hacer reseñas de libros y llevar un monitoreo para informar de posiciones geográficas de donde se han dejado libros que alguien más puede recoger para leer. Además los miembros pueden registrar que han tomado algún libro o si lo han liberado y donde lo han puesto.

El conjunto de datos tiene calificaciones explicitas en el intervalo $[1, 10]$ además de calificaciones implícitas representadas por el valor 0. Este valor indica que el usuario no calificó el libro pero sí adquirió este.

Usuarios	105,283
Ítems	340,556
Calificaciones	1,149,780
Rango de calif.	$[0, 10]$
Densidad	0.1 %
Min. # valores/ítem	1
Min. # valores/usuario	1
Max. # valores/ítem	13,602
Max. # valores/usuario	2,502

Media	0
Desviacion std.	3.854
Curtosis	-1.24
Asimetría	0.734
Coeficiente Gini	0.689328...

[5] Es una medida que analiza el grado de concentración de la muestra alrededor de la media. Para más información consultar: https://es.wikipedia.org/wiki/Curtosis

[6] Esta medida indica el nivel asimetría que tiene la muestra con respecto a la media. https://es.wikipedia.org/wiki/Asimetr%C3%ADa_estad%C3%ADstica

[7] Es una medida estadística usada en economía para medir la desigualdad de los ingresos entre individuos de una población, pero puede utilizarse para medir cualquier forma de distribución desigual, e nuestro caso, la distribución de la asignación de calificaciones para nuestro conjunto de ítems. Para más información consultar: https://en.wikipedia.org/wiki/Gini_coefficient

Jester

Jester[37] es un sistema de recomendación de bromas desarrollado por la universidad de Berkeley para estudiar filtrado de información social.

Las calificaciones de este conjunto de datos son valores reales en -10 y 10 y todos fueron asignados de manera explicita por los usuarios.

Usuarios	50,692
Ítems	140
Calificaciones	1,728,875
Rango de calif.	$[-10, 10]$
Densidad	24.4 %
Min. # valores/ítem	166
Min. # valores/usuario	8
Max. # valores/ítem	50,692
Max. # valores/usuario	140

Media	2.25
Desviacion std.	5.27
Curtosis	-0.673
Asimetría	-0.419
Coeficiente Gini	0.22616419...

MovieLens Dataset

MovieLens[8] es sitio especializado en películas. Dentro del sistema los usuarios pueden buscar pelicular, calificarlas y recomienda películas en base al perfil del usuario.

En este caso, las calificaciones fueron asignadas de manera explicita por los usuario y estas están son números enteros en el intervalo $[1, 5]$.

Usuarios	138,493
ítems	26,744
Calificaciones	20,000,263
Rango de calif.	$[1, 5]$
Densidad	0.53 %
Min. # valores/ítem	1
Min. # valores/usuario	20
Max. # valores/ítem	67,310
Max. # valores/usuario	9,254

Media	3.5
Desviacion std.	1.052
Curtosis	0.137
Asimetría	-0.655
Coeficiente Gini	0.455656...

4.2.2. Ejemplo simplificado de experimentos

A continuación se presenta un experimento simplificado como prueba de concepto y con fines puramente didácticos para mostrar el proceso de evaluación con el modelo MERERO en contraste con el error cuadrático medio (MSE). Cabe señalar que este experimento es solo para fines didácticos e ilustra como se calcula en el modelo MERERO las diferencias de rankings.

Conjunto de datos y usuario de prueba

Para realizar la prueba se tomó una muestra de 100,000 usuarios y las calificaciones emitidas por estos del conjunto de datos de MovieLens y se hizo la evaluación sobre un sólo usuario, tomando como conjunto de prueba diez de las calificaciones que este usuario emitió dejando el resto del conjunto de datos como entrenamiento. Se escogió el usuario con identificador 13 que cuenta con 639 ítems calificados, que es el número máximo de ítems calificados con respecto a los demás usuarios de la muestra.

Sistemas de recomendación evaluados

Al igual que los experimentos completos, se evaluaron los SR:

- BestSR
- FlipSR
- MaxMSESR
- FunkSVD

Resultado

A continuación se incluye la salida del programa donde se implementó esta prueba cuyo código se puede consultar en la sección "*A.2.6 MinimalTest*". Para cada SR evaluado se detalla la calificación obtenida en MERERO así como un desglose del ranking de ítems, su representación como vector y su comparación con el original usando la distancia Manhattan en el campo *diferencias*.

Ejemplos de cómo se calcula en el modelo MERERO las diferencias de rankings

```
;*************TIEMPOR ENTRENAMIENTO******************;
BestRecommender : 0 ms
FlipRecommender : 0 ms
MaxMSERecommender       : 0 ms
FunkSVDRecommender      : 30117 ms

;*************EVALUACIÓN******************;
BestRecommender * [merero: 0.0, mse: 0.0]:
        Estimaciones (<idItem>=<valorEstimado>):
                {525=5.0, 272=4.0, 739=4.0, 770=4.0, 38=3.0, 79=3.0, 755=3.0, 866=3.0, 312=1.0, 561=1.0}
        Rankings:
                Original: [525, 272, 739, 770, 38, 79, 755, 866, 312, 561]
                Estimado: [525, 272, 739, 770, 38, 79, 755, 866, 312, 561]
        Vectores:
                Original   : {38=5, 79=6, 272=2, 312=9, 525=1, 561=10, 739=3, 755=7, 770=4, 866=8}
                Estimado   : {38=5, 79=6, 272=2, 312=9, 525=1, 561=10, 739=3, 755=7, 770=4, 866=8}
                Diferencias:  |5-5|+|6-6|+ |2-2|+ |9-9|+ |1-1|+|10-10|+ |3-3|+ |7-7|+ |4-4|+ |8-8|
                           :    0  +  0  +   0  +   0  +   0  +   0   +   0  +   0  +   0  +   0
                           : 0.0

FlipRecommender * [merero: 38.0, mse: 5.9999743]:
        Estimaciones (<idItem>=<valorEstimado>):
                {561=5.0000334, 312=5.0000186, 866=3.0000515, 755=3.0000448, 79=3.0000048, 38=3.0000021,
                 770=2.0000458, 739=2.0000439, 272=2.0000162, 525=1.0000312}
        Rankings:
                Original: [525, 272, 739, 770, 38, 79, 755, 866, 312, 561]
                Estimado: [561, 312, 866, 755, 79, 38, 770, 739, 272, 525]
        Vectores:
                Original   : {38=5, 79=6, 272=2, 312=9, 525= 1, 561=10, 739=3, 755=7, 770=4, 866=8}
                Estimado   : {38=6, 79=5, 272=9, 312=2, 525=10, 561= 1, 739=8, 755=4, 770=7, 866=3}
                Diferencias:  |6-5|+|5-6|+ |9-2|+ |2-9|+|10-1|+  |1-10|+ |8-3|+ |4-7|+ |7-4|+ |3-8|
                           :    1  +  1  +   7  +   7  +   9  +    9   +   5  +   3  +   3  +   5
                           : 38.0

MaxMSERecommender * [merero: 40.0, mse: 9.1]:
        Estimaciones (<idItem>=<valorEstimado>):
                {38=5.0, 79=5.0, 312=5.0, 561=5.0, 755=5.0, 866=5.0, 272=1.0, 525=1.0, 739=1.0, 770=1.0}
        Rankings:
                Original: [525, 272, 739, 770, 38, 79, 755, 866, 312, 561]
                Estimado: [38, 79, 312, 561, 755, 866, 272, 525, 739, 770]
        Vectores:
                Original   : {38=5, 79=6, 272=2, 312=9, 525=1, 561=10, 739=3, 755=7, 770=4, 866=8}
                Estimado   : {38=1, 79=2, 272=7, 312=3, 525=8, 561=4, 739=9, 755=5, 770=10, 866=6}
                Diferencias:  |1-5|+|2-6|+ |7-2|+ |3-9|+ |8-1|+ |4-10|+|9-3|+ |5-7|+|10-4|+  |6-8|
                           :    4  +  4  +   5  +   6  +   7  +   6   +  6  +   2  +   6  +    2
                           : 40.0

FunkSVDRecommender * [merero: 16.0, mse: 1.0423208]:
        Estimaciones (<idItem>=<valorEstimado>):
                {79=4.445184, 272=3.8572583, 525=3.8080056, 770=3.546683, 755=3.1970682, 739=3.032273,
                 38=2.9505274, 312=2.6953278, 561=2.6137474, 866=2.5185099}
```

```
Rankings:
        Original: [525, 272, 739, 770,  38,  79, 755, 866, 312, 561]
        Estimado: [ 79, 272, 525, 770, 755, 739,  38, 312, 561, 866]
Vectores:
        Original   : {38=5, 79=6, 272=2, 312=9, 525=1, 561=10, 739=3, 755=7, 770=4, 866=8}
        Estimado   : {38=7, 79=1, 272=2, 312=8, 525=3, 561= 9, 739=6, 755=5, 770=4, 866=10}
        Diferencias:  |7-5|+|1-6|+ |2-2|+ |8-9|+ |3-1|+ |9-10|+ |6-3|+ |5-7|+ |4-4|+|10-8|
                    :    2  +  5  +   0  +   1  +   2  +   1   +   3  +   2  +   0  +   2
                    : 16.0
```

Salida 4.2: Ejemplos del calculo de diferencias de rankings por el modelo MERERO

4.2.3. Segmentación de los datos para los experimentos

Los experimentos se realizaron utilizando la técnica de validación cruzada en n-iteraciones, pero además se siguió la práctica común en experimentos para evaluación de estimadores donde se segmenta los datos aleatoriamente en 80 % para entrenamiento y 20 % para realizar pruebas, por lo que se asignó $n = 5$ como se muestra en la figura 4.2.

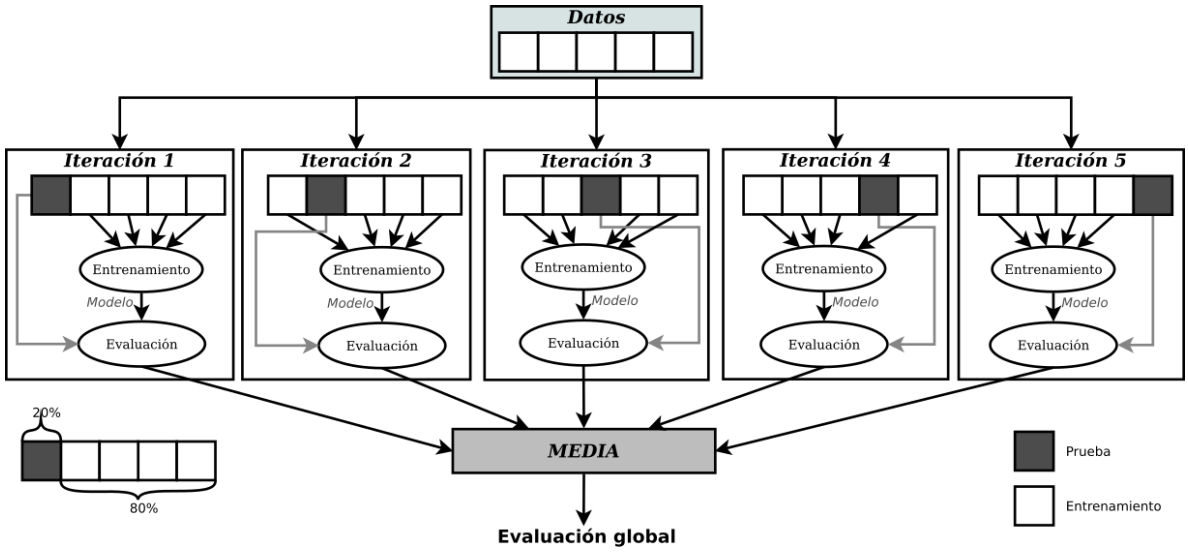

Figura 4.2: Técnica de evaluación por validación cruzada con n-iteraciones ($n = 5$)

Por lo tanto para realizar los experimentos se asignó la siguiente configuración a la variable `DataSplitterClass`:

```
DataSplitterClass=org.recommender101.data.DefaultDataSplitter:nbFolds=5
```

Ya que la clase `DefaultDataSplitter` implementa la segmentación necesaria para la realizar la validación cruzada con n-iteraciones y por medio del parámetro `nbFolds` asignamos que se hagan cinco iteraciones.

4.2.4. Resultados

Los resultados muestran que nuestro modelo de evaluación se comporta como se había predicho, dejando en último lugar al SR FlipSR y mostrando el problema de la evaluación como estimador. En el caso del RMSE, MaxMSERS queda por debajo de FlipSR a pesar de que este último ofrece una lista de ítems completamente opuesta a las preferencias reales del usuario. En las siguientes tablas muestran los resultados de tres experimentos donde se tomaron distintos número de usuarios para el muestreo en 100, 1000 y para el conjunto de datos completo, es decir 138,493 usuarios. En la sección "*B Salida de los experimentos*" se incluye el texto de salida que emitió el programa de los experimentos en cada uno de ellos.

Resultados para muestra de 100 usuarios

Como se puede observar en la tabla 4.3, para una muestra de usuario pequeña los tres modelos de evaluación califican los SR de referencia de manera correcta dejando a FlipSR por debajo de MaxMSERS.

		Calificación en		
ID	Sistema de recomendación	MERERO	RMSE	MAE
BSR	BestRecommender	0	0.069	0.009
FSVD	FunkSVDRecommender	2.663	0.722	0.530
MSR	MaxMSERecommender	2.680	3.005	2.694
FSR	FlipRecommender	2.776	3.075	2.988

Tabla 4.2: Errores reportados por MERERO, RMSE y MAE. Experimento con una muestra de 100 usuarios.

Ranking en		
MERERO	RMSE	MAE
BSR	BSR	BSR
FSVD	FSVD	FSVD
MSR	MSR	MSR
FSR	FSR	FSR

Tabla 4.3: SR ordenados respecto su calificación en cada modelo de evaluación. Experimento con una muestra de 100 usuarios.

Resultados para muestra de 1000 usuarios

Al incrementar la muestra se observa que los modelos de evaluación que se enfocan en estimaciones puntuales dejan en último lugar al SR MaxMSESR por debajo de FlipSR (tabla 4.5) aún a pesar de que este último ofrece recomendaciones completamente contrarias a las preferencias del usuario.

		Calificación en		
ID	Sistema de recomendación	MERERO	RMSE	MAE
BSR	BestRecommender	0	0.05	0.007
FSVD	FunkSVDRecommender	0.525	0.759	0.583
MSR	MaxMSERecommender	2.617	3.033	2.978
FSR	FlipRecommender	2.695	2.992	2.876

Tabla 4.4: Errores reportados por MERERO, RMSE y MAE. Experimento con una muestra de 1000 usuarios.

Ranking en		
MERERO	RMSE	MAE
BSR	BSR	BSR
FSVD	FSVD	FSVD
MSR	FSR	FSR
FSR	MSR	MSR

Tabla 4.5: SR ordenados respecto su calificación en cada modelo de evaluación. Experimento con una muestra de 1000 usuarios.

Resultados para muestra de 138,493 usuarios

Al utilizar el conjunto completo de los datos se mantiene el mismo comportamiento para los modelos de evaluación que se enfocan en estimaciones puntuales, siendo MERERO el único que deja en último lugar al SR FlipSR como se observa en la tabla 4.7.

		Calificación en		
ID	Sistema de recomendación	MERERO	RMSE	MAE
BSR	BestRecommender	0	0.055	0
FSVD	FunkSVDRecommender	2.593	0.655	0.655
MSR	MaxMSERecommender	2.615	3.030	3.128
FSR	FlipRecommender	2.685	2.981	2.983

Tabla 4.6: Errores reportados por MERERO, RMSE y MAE. Experimento con una muestra de 138,493 usuarios.

Ranking en		
MERERO	RMSE	MAE
BSR	BSR	BSR
FSVD	FSVD	FSVD
MSR	FSR	FSR
FSR	MSR	MSR

Tabla 4.7: SR ordenados respecto su calificación en cada modelo de evaluación. Experimento con una muestra de 138,493 usuarios.

Conclusiones

Al investigar el estado del arte de los SR junto con los modelos de evaluación existentes se identificó qué estos últimos se enfocaban en las estimaciones puntuales dando una evaluación muy estricta, dejando de lado el hecho de que si el SR evaluado presenta un ranking de ítems parecido al generado por la apreciación real del usuario sobre los ítems, se puede decir que este SR ofrece buenas recomendaciones a pesar de que en las estimaciones puntuales para cada ítem pudieran ser erradas como se ejemplificó en "*3 Propuesta de evaluación de SR usando rankings*".

Por tanto se propuso un nuevo modelo de evaluación (MERERO) el cual se enfoca en medir las diferencias entre los dos rankings (original y estimado) para cada usuario de tal manera que se evalúa si el SR preserva la relevancia relativa de los ítems para cada usuario, en lugar de enfocarse en el hecho de si el SR logra estimar de manera precisa el nivel de relevancia de cada ítem para cada usuario.

Como parte del modelo se definieron tres SR sintéticos (ver "*3.3 SR para la validación del modelo de evaluación*") para tener puntos de referencias, es decir, SR que de antemano, se sabe cómo serán evaluados, de tal manera que se pueda validar si el modelo MERERO evalúa de manera correcta. Además como valor agregado, estos SR pueden ser usados en un futuro para validar otros modelos de evaluación.

Los experimentos realizados en esta tesis comprueban que el modelo MERERO sí evalúa los SR con respecto a los rankings generados ya que este calificó a los SR de referencia de la manera que se definió como correcta. Por otro lado parece ser que MERERO ofrece una evaluación menos estricta en comparación con los modelos de evaluación actuales, pero que a su vez evalúa a los SR en su objetivo final, que es el ranking que se le presenta al usuario o dicho de otra manera, evalúa la precisión de significado, en lugar de la precisión de valor haciendo referencia a terminología de lógica difusa[34].

Como producto derivado de los dos primeros capítulos de la tesis se creó el artículo "*Estado del arte en los sistemas de recomendación*", el cual fue presentado en el evento COMIA y fue publicado en el volumen 135 en la revista *Research in Computing Science*[8] la cual se encuentra registrada en los índices DBLP[9], LatIndex[10] y Periodica[11].

Los SR como parte de los avances tecnológicos, están cada vez más presentes en todas las áreas de la vida del ser humano. Las personas los usan consciente o inconscientemente para encontrar productos, bienes y servicios tales como libros, música, noticias, viajes e incluso relaciones románticas. Constantemente se incrementa el número de sistemas informáticos que ofrecen productos, servicios o simplemente información que contienen algún tipo de SR como apoyo a sus usuarios en la elección entre las innumerables alternativas que éstos ofrecen. El desarrollo de este tipo de sistemas informáticos ha creado una comunidad de investigación creciente, que intenta innovar y solventar los muchos problemas que aún se encuentran por resolver en esta área.

Los dos posibles trabajos a futuro serían: el comprobar la hipótesis de que MERERO evalúa el valor de significado de los SR y crear SR inspirados por el modelo MERERO en los cuales se intente minimizar las diferencias entre los rankings (original y estimado) para cada usuario utilizando una perspectiva diferente a solo tratar de disminuir los errores puntuales de las estimaciones, con lo cual se abrirían nuevos caminos para intentar ofrecer mejores recomendaciones.

[8] http://www.rcs.cic.ipn.mx/
[9] http://dblp.org/db/journals/rcs/index.html
[10] http://www.latindex.org
[11] http://periodica.unam.mx

Apéndice A

Código implementado

A.1. Clases en Recommender101

A.1.1. org.recommender101.eval.interfaces.PredictionEvaluator

```
1  /** DJ **/
2  package org.recommender101.eval.interfaces;
3
4  import org.recommender101.data.Rating;
5  import org.recommender101.gui.annotations.R101HideFromGui;
6
7
8  /**
9   * An abstract for the implementation of prediction evaluation metrics
10  * @author DJ
11  *
12  */
13 @R101HideFromGui
14 public abstract class PredictionEvaluator extends Evaluator {
15
16
17   /**
18    * This method is called for every prediction and can be used to accumulate things
19    * @param user the user id
20    * @param item the item id
21    * @param prediction the predicted value (or Float.NaN in case the recommender could
22    * not make a prediction)
23    *
24    */
25   public abstract void addTestPrediction(Rating r, float prediction);
26
27   /**
28    * Returns the final number
29    * @return
30    */
31   public abstract float getPredictionAccuracy();
32
33 }
```

Código fuente A.1: Clase `PredictionEvaluator`

A.1.2. org.recommender101.eval.interfaces.RecommendationListEvaluator

```
1  /** DJ **/
2  package org.recommender101.eval.interfaces;
3
4  import java.util.List;
5
6  import org.recommender101.eval.impl.Recommender101Impl;
```

```
7  import org.recommender101.gui.annotations.R101HideFromGui;
8  import org.recommender101.gui.annotations.R101Setting;
9  import org.recommender101.gui.annotations.R101Setting.SettingsType;
10 import org.recommender101.tools.Utilities101;
11
12
13 /**
14  * An abstract class that can be used to evaluate recommendation lists
15  * @author DJ
16  *
17  */
18 @R101HideFromGui
19 public abstract class RecommendationlistEvaluator extends Evaluator {
20
21   /**
22    * Add a recommendation list for a user. The data is accumulated internally
23    * @param user the user for whom the recommendation is made
24    * @param list the list of recommended items. Can also be null or empty
25    */
26   public abstract void addRecommendations(Integer user, List<Integer> list);
27
28   /**
29    * Calculates and returns the result at the end.
30    * @return
31    */
32   public abstract float getEvaluationResult();
33
34   /**
35    * Set the number of items to be retrieved in the evaluation
36    * @param n
37    */
38   @R101Setting(displayName="Top N",minValue=0, type=SettingsType.INTEGER,
       description="The top N value for this metric", defaultValue="10")
39   public void setTopN(String n) {
40     topN = Integer.parseInt(n);
41   };
42
43   /** Returns the number of items used */
44   public int getTopN() {
45     return topN;
46   };
47
48
49   /**
50    * How do we calculate precision, default is only relevant items in test set
51    */
52   String targetSet = null;
53
54
55   /**
56    * Returns the target set for the calculations
57    * @return
58    */
59   public String getTargetSet() {
60     return targetSet;
61   }
62
63   /**
64    * Setter for the target set used by class instantiator
65    * @param targetSet
66    */
67   @R101Setting(displayName="Target set", type=SettingsType.ARRAY,
68       description="The target set", values={"allrelevantintestset", "allintestset",
       "positioninrandomset"})
69   public void setTargetSet(String targetSet) {
70     this.targetSet = targetSet;
71   }
72
```

```
73
74    /**
75     * How long is the topN list
76     */
77    public int topN = Recommender101Impl.TOP_N;
78
79  //=========================================================================
80    /**
81     * Determines, if an item is relevant for the user or not. An item is relevant when
82     * it is above a defined threshold, which is either the user's average or (in case
83     * the parameter is set) if it is x percent above the user's average.
84     * Alternatively, if the minRatingForRelevance parameter is set, this is used as
85     * threshold
86     * @param item the item id
87     * @param user the user id
88     * @return true if the item is relevant
89     */
90    public boolean isItemRelevant(int item, int user) {
91      return Utilities101.isItemRelevant(item, user, getTestDataModel());
92    }
93
94    /*
95     * A variant that counts the actually irrelevant based on a threshold which is not
        the
96     * strict opposite of relevant. (Relevant can be: only the five-stars; bad might be
        the one stars or two stars less)
97     * TODO: for rating-based methods, we can compute the max difference between
        prediction and test data
98     * The method name is kept to integrate the measure into precision / recall
99     */
100   public boolean isItemRelevant(int item, int user, int countBadOnes) {
101     if (countBadOnes != -1) {
102       float rating = getTestDataModel().getRating(user, item);
103       // Old variant: Absolute values
104       if (rating <= countBadOnes) {
105         return true;
106       }
107       else {
108         return false;
109       }
110     }
111     else {
112       return  Utilities101.isItemRelevant(item, user, getTestDataModel());
113     }
114   }
115
116 }
```

Código fuente A.2: Clase `RecommendationListEvaluator`

A.1.3. org.recommender101.eval.impl.Experiment

```
1  /** DJ **/
2  package org.recommender101.eval.impl;
   ⋮
33 public class Experiment {
   ⋮
77   /**
78    * A constructor that accepts a recommender and a data model as well as a set
79    * of evaluators
80    *
81    * @param recommender
82    * @param dataModel
83    * @param evals
```

```
84      *           the evaluator classes
85      */
86     @SuppressWarnings("unchecked")
87     public Experiment(AbstractRecommender recommender,
88         DataModel trainingDataModel, DataModel testDataModel, String evals,
89         int evalRound) throws Exception {
90       super();
91       this.recommender = recommender;
92       this.evaluators = new ArrayList<Evaluator>();
93       this.dataModel = testDataModel;
94       this.evaluationRound = evalRound;
95
96       //TODO: Edited by OEG, 2017/01/26. Make the original data accessible for the
         recommender system to be tested
97       Set<Rating> data = testDataModel.getRatings();
98       data.addAll(trainingDataModel.getRatings());
99       data = cloneRatingsSet(data);
100
101      DataModel allData = new DataModel(data, this.dataModel.getExtraInformationMap(),
         this.dataModel.getMinRatingValue(), this.dataModel.getMaxRatingValue());
102      allData.recalculateUserAverages();
103
104      DataModel tmpTestModel = new DataModel(
105            cloneRatingsSet(testDataModel.getRatings()),
106            testDataModel.getExtraInformationMap(),
107            testDataModel.getMinRatingValue(),
108            testDataModel.getMaxRatingValue());
109
110      recommender.setExtraInformation("OriginalTestData", allData.unmodifiable());
111      recommender.setExtraInformation("TestData", tmpTestModel.unmodifiable());
    ⋮
187      }
    ⋮
318  }
```

Código fuente A.3: Cambios realizados en la clase `Experiment`

A.2. Clases en MERERO

A.2.1. oeg.merero.recommenders.AbstractTestRecommender

```
1  /**
2   *
3   */
4  package oeg.merero.recommenders;
5
6  import java.util.ArrayList;
7  import java.util.Collections;
8  import java.util.HashMap;
9  import java.util.HashSet;
10 import java.util.List;
11 import java.util.Map;
12 import java.util.Set;
13
14 import org.recommender101.data.DataModel;
15 import org.recommender101.data.Rating;
16 import org.recommender101.recommender.AbstractRecommender;
17 import org.recommender101.tools.Utilities101;
18
19 /**
20  * @author oescamil
21  *
```

```
22   */
23  abstract public class AbstractTestRecommender extends AbstractRecommender {
24    static final long serialVersionUID = -3073503454386601618L;
25    /**
26     * The mean of the
27     */
28    protected float r_med = 0;
29    protected float r_max = 0;
30    protected float r_min = 0;
31
32    /**
33     *
34     * @return
35     */
36    protected DataModel getOriginalDataModel(){
37      return (DataModel) this.getExtraInformation("OriginalTestData");
38    }
39
40
41
42    /* (non-Javadoc)
43     * @see org.recommender101.recommender.AbstractRecommender#recommendItems(int)
44     */
45    @Override
46    public List<Integer> recommendItems(int user) {
47      DataModel origModel = getOriginalDataModel();
48      Set<Rating> userRatings = origModel.getRatingsOfUser(user);
49
50      Map<Integer, Float> mapURatings = new HashMap<Integer, Float>();
51      for(Rating r : userRatings){
52        mapURatings.put(r.item, r.rating);
53      }
54
55
56
57      List<Integer> result = this.recommendItemsByRatingPrediction(user);
58      return result;
59    }
60
61    /**
62     * A general method for ranking items according to their rating prediction.
63     * The method should be overwritten in case the recommender cannot make
64     * rating predictions or when a better heuristic is needed, which for
65     * example takes the popularity of the recommendations into account.
66     * @param user the user for which a recommendation is sought
67     * @return the ranked list of items
68     */
69    @Override
70    public List<Integer> recommendItemsByRatingPrediction(int user) {
71      List<Integer> result = new ArrayList<Integer>();
72
73      // If there are no ratings for the user in the test set,
74      // there is no point of making a recommendation.
75      Map<Integer, Set<Rating>> ratings = getDataModel().getRatingsPerUser();
76      // If we have no ratings...
77      if (ratings == null || ratings.size() == 0) {
78        return Collections.emptyList();
79      }
80
81
82      DataModel origModel = getOriginalDataModel();
83      // Calculate rating predictions for all items we know
84      Map<Integer, Float> predictions = new HashMap<Integer, Float>();
85      //byte rating = -1;
86      float pred = Float.NaN;
87      // Go through all the items
88      for (Integer item : origModel.getItems()) {
89
```

```
90          float rating = getDataModel().getRating(user, item);
91          // We will not recommend items repeatedly here-
92          if (rating == -1 || Float.isNaN(rating)) {
93            // make a prediction and remember it in case the recommender
94            // could make one
95            pred = predictRating(user, item);
96            if (!Float.isNaN(pred)) {
97              predictions.put(item, pred);
98            }
99          }
100       }
101
102       predictions = filterElementsByRelevanceThreshold(predictions, user);
103       predictions = Utilities101.sortByKeyAscending(predictions);
104       predictions = Utilities101.sortByValueDescending(predictions);
105
106       for (Integer item : predictions.keySet()) {
107         result.add(item);
108       }
109       return result;
110     }
111
112     /**
113      * Initialize the recommender. Calculates the mean from the ratings
114      * @see org.recommender101.recommender.AbstractRecommender#init()
115      */
116     @Override
117     public void init() throws Exception {
118       DataModel dataModel = this.getDataModel();
119
120       r_max = dataModel.getMaxRatingValue();
121       r_min = dataModel.getMinRatingValue();
122       r_med = (r_max + r_min)/2.0f;
123
124
125     }
126   }
```

Código fuente A.4: Implementación de la clase `AbstractTestRecommender`

A.2.2. oeg.merero.recommenders.BestRecommender

```
1   /**
2    *
3    */
4   package oeg.merero.recommenders;
5
6   import org.recommender101.data.DataModel;
7
8   /**
9    * @author oescamil
10   *
11   */
12  public class BestRecommender extends AbstractTestRecommender {
13    static final long serialVersionUID = -3073503454386601618L;
14
15
16
17
18    /* (non-Javadoc)
19     * @see org.recommender101.recommender.AbstractRecommender#predictRating(int, int)
20     */
21    @Override
22    public float predictRating(int user, int item) {
23      DataModel origModel = this.getOriginalDataModel();
24      float r_ui = origModel.getRating(user, item);
25
26
```

```
27      return r_ui;
28    }
29  }
```

Código fuente A.5: Implementación de la clase `BestRecommender`

A.2.3. oeg.merero.recommenders.FlipRecommender

```
1   /**
2    *
3    */
4   package oeg.merero.recommenders;
5
6   import java.util.ArrayList;
7   import java.util.HashMap;
8   import java.util.List;
9   import java.util.Map;
10  import java.util.Set;
11
12  import org.recommender101.data.DataModel;
13  import org.recommender101.data.Rating;
14  import org.recommender101.eval.impl.Recommender101Impl;
15  import org.recommender101.tools.Utilities101;
16
17  /**
18   * @author oescamil
19   *
20   */
21  public class FlipRecommender extends AbstractTestRecommender {
22    static final long serialVersionUID = -3073503454386601618L;
23    private HashMap<Integer, List<Integer>> orinalRankings;
24    private int itemCount = 0;
25
26    /* (non-Javadoc)
27     * @see org.recommender101.recommender.AbstractRecommender#predictRating(int, int)
28     */
29    @Override
30    public void init() throws Exception {
31      super.init();
32      orinalRankings = new HashMap<Integer, List<Integer>>();
33      DataModel origModel = this.getOriginalDataModel();
34      this.itemCount = origModel.getItems().size();
35    }
36
37
38
39
40    @Override
41    public float predictRating(int user, int item) {
42      DataModel origModel = this.getOriginalDataModel();
43
44      if(orinalRankings.get(user) == null) {
45        orinalRankings.put(user,createOriginalRankingFor(user));
46      }
47
48      List<Integer> origRank = orinalRankings.get(user);
49      if(origRank.indexOf(item) < 0 ) {
50        return Float.NaN;
51      }
52
53      float r_ui = origModel.getRating(user, item);
54      if(r_ui <= -1){
55        return Float.NaN;
56      }
57
58      float newRui= r_min+(r_max-r_ui)+((float)item/(float)itemCount)*0.0001f;
59      return newRui;
60    }
```

```
61
62    /**
63     * Crea el ranking original de items del usuario, solo toma en cuenta
64     * las primeras Recommender101Impl.TOP_N posiciones del ranking.
65     * @param user El usuario de cual queremos el ranking original
66     * @return
67     */
68    private List<Integer> createOriginalRankingFor(int user) {
69      int topN = Recommender101Impl.TOP_N;
70      DataModel testModel = (DataModel) this.getExtraInformation("TestData");
71      DataModel trainingM = this.getDataModel();
72      Set<Rating> userRatings = testModel.getRatingsOfUser(user);
73
74      // Get rating for all items we know
75      Map<Integer, Float> originalRatings = new HashMap<Integer, Float>();
76      for(Rating r : userRatings){
77        float trainingR = trainingM.getRating(r.user, r.item);
78        if(trainingR == -1) // Just for items which recommender can recommend
79          originalRatings.put(r.item, r.rating);
80      }
81      originalRatings = Utilities101.sortByKeyAscending(originalRatings);
82      originalRatings = Utilities101.sortByValueDescending(originalRatings);
83      List<Integer> originalRanking = new ArrayList<Integer>();
84
85      topN = Math.min(originalRatings.size(), topN);
86      int counter = 0 ;
87      for (Integer item : originalRatings.keySet()) {
88        originalRanking.add(item);
89        counter++;
90        if(counter >= topN)
91          break;
92      }
93      return originalRanking.subList(0, topN);
94    }
95  }
```

Código fuente A.6: Implementación de la clase `FlipRecommender`

A.2.4. oeg.merero.recommenders.MaxMAERecommender

```
1   /**
2    *
3    */
4   package oeg.merero.recommenders;
5
6   import java.util.List;
7
8   import org.recommender101.data.DataModel;
9
10  /**
11   * @author oescamil
12   *
13   */
14  public class MaxMAERecommender extends AbstractTestRecommender {
15    static final long serialVersionUID = 108152434558655334L;
16
17
18    /* (non-Javadoc)
19     * @see org.recommender101.recommender.AbstractRecommender#predictRating(int, int)
20     */
21    @Override
22    public float predictRating(int user, int item) {
23      DataModel origModel = this.getOriginalDataModel();
24      float r_ui = origModel.getRating(user, item);
25
26      if(r_ui <= -1){
27        return Float.NaN;
28      }
```

```
29
30      return (r_ui <= r_med)?r_max:r_min;
31    }
32
33
34
35  }
```

Código fuente A.7: Implementación de la clase `MaxMAERecommender`

A.2.5. oeg.merero.recommenders.evaluator.MEREROEvaluator

```
1   /**
2    *
3    */
4   package oeg.merero.recommenders.evaluator;
5
6   import java.util.ArrayList;
7   import java.util.HashMap;
8   import java.util.Iterator;
9   import java.util.List;
10  import java.util.Map;
11  import java.util.Set;
12
13  import javax.management.ImmutableDescriptor;
14  import javax.print.attribute.IntegerSyntax;
15
16  import org.recommender101.data.DataModel;
17  import org.recommender101.data.Rating;
18  import org.recommender101.eval.interfaces.RecommendationlistEvaluator;
19  import org.recommender101.tools.Utilities101;
20
21  import oeg.rs.utils.DamerauLevenshteinDistance;
22
23  /**
24   * @author oescamil
25   *
26   */
27  public class MEREROEvaluator extends RecommendationlistEvaluator {
28
29    protected float errorAccumulator = 0 ;
30    protected int rankingCounter = 0;
31    @Override
32    public void initialize() {
33      super.initialize();
34      this.errorAccumulator = 0;
35      this.rankingCounter = 0;
36    }
37
38
39    /* (non-Javadoc)
40     * @see
        org.recommender101.eval.interfaces.RecommendationlistEvaluator#addRecommendations(java.lang.Integer,
        java.util.List)
41     */
42    @Override
43    public void addRecommendations(Integer user, List<Integer> list) {
44      DataModel testModel = getTestDataModel();
45      DataModel trainingModel = getTrainingDataModel();
46      Set<Rating> userRatings = testModel.getRatingsOfUser(user);
47
48      // Get rating for all items we know
49      Map<Integer, Float> originalRatings = new HashMap<Integer, Float>();
50      for(Rating r : userRatings){
51        float trainingR = trainingModel.getRating(r.user, r.item);
52        if(trainingR == -1) // Just for items which recommender can recommend
53          originalRatings.put(r.item, r.rating);
54      }
```

```
55      originalRatings = Utilities101.sortByKeyAscending(originalRatings);
56      originalRatings = Utilities101.sortByValueDescending(originalRatings);
57      List<Integer> originalRanking = new ArrayList<Integer>();
58
59      int topN = Math.min(this.getTopN(),list.size());
60      int counter = 0 ;
61      for (Integer item : originalRatings.keySet()) {
62        originalRanking.add(item);
63        counter++;
64        if(counter >= topN)
65          break;
66      }
67      topN = Math.min(topN,originalRanking.size());
68
69      list = list.subList(0, topN);
70      originalRanking = originalRanking.subList(0, topN);
71
72      float error = calcDistance(originalRanking, list);
73
74      this.errorAccumulator += error;
75      this.rankingCounter++;
76    }
77
78
79
80
81    /* (non-Javadoc)
82     * @see
        org.recommender101.eval.interfaces.RecommendationlistEvaluator#getEvaluationResult()
83     */
84    @Override
85    public float getEvaluationResult() {
86      if(this.rankingCounter <= 0)
87        return 0;
88
89      return (float) Math.sqrt(this.errorAccumulator/this.rankingCounter);
90    }
91
92
93
94    /**
95     * Used for evaluation aggregation
96     */
97    public String toString() {
98      return "Ranking Evaluator";
99    }
100
101
102
103
104
105   /**
106    * Calculates the distance between two rankings (a and b)
107    * @param a Ranked items
108    * @param b Ranked items
109    * @return
110    */
111   private float calcDistance(List<Integer> a, List<Integer> b){
112
113
114     // For Manhattan distance
115     Map<Integer, Integer> mapA = convertRankingToPositionVector(a);
116     Map<Integer, Integer> mapB = convertRankingToPositionVector(b);
117
118
119     DamerauLevenshteinDistance dlDistance = new DamerauLevenshteinDistance();
120     return dlDistance.calc(mapA.values().toArray(),mapB.values().toArray());
121   }
```

```
122
123
124     /**
125      *
126      * @param ranking
127      * @return
128      */
129     private Map<Integer, Integer> convertRankingToPositionVector(List<Integer> ranking){
130       Map<Integer, Integer> res = new HashMap<Integer,Integer>();
131       for (int i = 0; i < ranking.size();i++) {
132         res.put(ranking.get(i), i);
133       }
134       return res;
135     }
136
137
138
139 }
```

Código fuente A.8: Implementación de la clase `MEREROEvaluator`

A.2.6. oeg.merero.MinimalTest

```
1  package oeg.merero;
2
3  import java.util.ArrayList;
4  import java.util.Date;
5  import java.util.HashMap;
6  import java.util.Hashtable;
7  import java.util.Iterator;
8  import java.util.List;
9  import java.util.Map;
10 import java.util.Set;
11
12 import org.recommender101.data.DataModel;
13 import org.recommender101.data.DefaultDataLoader;
14 import org.recommender101.data.Rating;
15 import org.recommender101.eval.impl.Recommender101Impl;
16 import org.recommender101.recommender.AbstractRecommender;
17 import org.recommender101.recommender.extensions.funksvd.FunkSVDRecommender;
18 import org.recommender101.tools.Utilities101;
19
20 import oeg.merero.recommenders.BestRecommender;
21 import oeg.merero.recommenders.FlipRecommender;
22 import oeg.merero.recommenders.MaxMSERecommender;
23 import oeg.merero.recommenders.evaluator.MEREROEvaluator;
24
25
26 public class MinimalTest {
27
28   public static void main(String[] args) {
29     try {
30
31       Recommender101Impl.TOP_N = 100000;
32       int minRank = 1;
33       int maxRank = 5;
34       /*
35        * *********************************
36        * Leemos nuestros datos
37        * *********************************
38        */
39       DefaultDataLoader dataLoader = new DefaultDataLoader();
40       DataModel allData = new DataModel();
41
42       allData.setMinRatingValue(minRank);
43       allData.setMaxRatingValue(maxRank);
44
45       dataLoader.setFilename("data/movielens/MovieLens100kRatings.txt");
```

```
46          dataLoader.setSampleNUsers("100000");
47          dataLoader.loadData(allData);
48
49          /*
50           * ************************************
51           * Vamos a probar sobre el usuario 13 es
52           * el que tiene mas calificaciones emitidas
53           * ****************************************
54           * */
55          int testUser = 13;
56          Set<Rating> uRatings = allData.getRatingsOfUser(testUser);
57          System.out.println("EL usuairo tiene n :" + uRatings.size());
58
59
60
61          /*
62           * ********************************************
63           * Creamos el conjunto de entrenamuento
64           *
65           * Todos - los del items de usuario 13 que
66           * queremos estimar
67           * ********************************************
68           */
69          DataModel testDataModel = new DataModel();
70          DataModel trainingDataModel = new DataModel(allData);
71
72          trainingDataModel.setMinRatingValue(minRank);
73          trainingDataModel.setMaxRatingValue(maxRank);
74
75
76
77
78          // Tomamos 10 items al azar para las pruebas y
79          // los quitamos del conjunto de entrenamiento
80          int[] itemsToEval = {561, 312, 866, 755, 79, 38, 770, 739, 272, 525};
81          //ArrayList<Rating> arrayRate = new ArrayList<Rating>(uRatings);
82          //Collections.shuffle(arrayRate);
83          for (int i = 0; i < itemsToEval.length; i++) {
84            //Rating rate = arrayRate.get(i);
85            int idItem = itemsToEval[i];
86            float rateV = trainingDataModel.getRating(testUser, idItem);
87            Rating rate = new Rating(testUser, idItem, rateV);
88
89            testDataModel.addRating(rate);
90            trainingDataModel.removeRating(rate);
91          }
92
93
94          /*
95           * ***********************************
96           * Instanciamos nuestros SR de prueba
97           * ***********************************
98           */
99          FlipRecommender    flip   = new FlipRecommender();
100         MaxMSERecommender maxMSE  = new MaxMSERecommender();
101         BestRecommender    best   = new BestRecommender();
102         FunkSVDRecommender  funkSVD = new FunkSVDRecommender();
103         funkSVD.setNumFeatures("50");
104         funkSVD.setInitialSteps("50");
105
106
107
108         Hashtable<String, Map> expResults = new Hashtable<String, Map>();
109         AbstractRecommender[] recommenderList = new AbstractRecommender[] { best, flip,
        maxMSE, funkSVD };
110         String txtOutTraining = ";************TIEMPOR
        ENTRENAMIENTO*****************;\n";
111         String txtOutRanking = ";************EVALUACION*****************;\n";
```

```
112
113
114         System.out.println("Entrenando modelos ...");
115         String origName = best.getClass().getSimpleName();
116         for (int i = 0; i < recommenderList.length; i++) {
117           AbstractRecommender recommender = recommenderList[i];
118           String rsName = recommender.getClass().getSimpleName();
119           /*
120            * **********************************
121            * Entrenamos el SR
122            * **********************************
123            */
124             long time = (new Date()).getTime();
125           recommender.setDataModel(trainingDataModel);
126           recommender.setExtraInformation("OriginalTestData", allData);
127           recommender.setExtraInformation("TestData", testDataModel);
128           recommender.init();
129             time = (new Date()).getTime()- time;
130
131           txtOutTraining += rsName+"\t: "+time+" ms\n";
132
133           // Donde guardamos las estimaciones
134           Map<Integer, Float> estimedRatings = new HashMap<Integer, Float>();
135           // Los items que vamos a usar para le prueba
136           Iterator<Integer> testItems = testDataModel.getItems().iterator();
137           while(testItems.hasNext()) {
138             int tmpItem = testItems.next();
139             float r = recommender.predictRating(testUser, tmpItem);
140             estimedRatings.put(tmpItem, r);
141           }
142
143           estimedRatings = Utilities101.sortByKeyAscending(estimedRatings);
144           estimedRatings = Utilities101.sortByValueDescending(estimedRatings);
145
146           expResults.put(rsName, estimedRatings);
147
148           @SuppressWarnings("unchecked")
149           Map<Integer, Float> realValues = expResults.get(origName);
150
151
152           List<Integer> ranking = new ArrayList<Integer>(estimedRatings.keySet());
153           List<Integer> rankOrig = new ArrayList<Integer>(realValues.keySet());
154
155
156           float mseDiff = MinimalTest.calculaMSE(realValues, estimedRatings);
157           float mereroDiff = MEREROEvaluator.calcDistance(ranking, rankOrig);
158
159
160           Map<Integer, Integer> vectPredicted =
      Utilities101.sortByKeyAscending(MEREROEvaluator.convertRankingToPositionVector(ranking));
161           Map<Integer, Integer> vectOriginal =
      Utilities101.sortByKeyAscending(MEREROEvaluator.convertRankingToPositionVector(rankOrig));
162
163
164           List<Integer> tmpKeys = new ArrayList<Integer>(vectOriginal.keySet());
165           String sumaMh = "";
166           String sumaMh2 = "";
167           for (Integer itemID : tmpKeys) {
168             sumaMh += "|"+vectPredicted.get(itemID)+"-"+vectOriginal.get(itemID)+"|+";
169             sumaMh2 += Math.abs(vectPredicted.get(itemID)-vectOriginal.get(itemID))+" +
      ";
170           }
171
172
173
174           txtOutRanking += "\n"+rsName+ " * [merero:"+mereroDiff+",
      mse:"+mseDiff+"]:\n" +
175                 "\tEstimaciones (<idItem>=<valorEstimado>):\n"+
```

```
176             "\t\t"          +estimedRatings+"\n"+
177             "\tRankings:\n"+
178             "\t\tOriginal: "+rankOrig+"\n"+
179             "\t\tEstimado: "+ranking+"\n"+
180             " \tVectores: \n"+
181             "\t\tOriginal    : "+vectOriginal+"\n"+
182             "\t\tEstimado    : "+vectPredicted+"\n"+
183             "\t\tDiferencias: "+sumaMh+"\n"+
184             "\t\t            : "+sumaMh2+"\n"+
185             "\t\t            : "+mereroDiff+"\n"+
186             "";
187       }
188
189
190       /*
191        * ************************************************
192        * Imprimiendo resultados
193        * ************************************************
194        */
195       System.out.println(txtOutTraining);
196       System.out.println(txtOutRanking);
197     } catch (Exception e) {
198       // TODO Auto-generated catch block
199       e.printStackTrace();
200     }
201
202   }
203
204
205   /**
206    * Funcion auxiliar para calcular el error cuadratico medio
207    * @param original Algo como [it1=3, it2=3]
208    * @param predicted Algo como [it1=3.2, it2=4.5]
209    * @return El error cuadratico medio entre las dos listas o NaN si no todos los
       items de original estan en predicted o si original es vacio.
210    */
211   public static float calculaMSE(Map<Integer, Float> original, Map<Integer, Float>
       predicted) {
212     float mse = 0 ;
213     if(original.size() <= 0 || predicted.size() <= 0) {
214       return Float.NaN;
215     }
216
217     Iterator<Integer> items = original.keySet().iterator();
218     while(items.hasNext()) {
219       Integer idxItem = items.next();
220       Float origV =  original.get(idxItem);
221       Float predV = predicted.get(idxItem);
222       if(predV == null || predV < 0) { //No se encontro en el otro conjunto
223         return Float.NaN;
224       }
225
226       float error = origV - predV;
227       mse += error*error;
228     }
229     return mse/original.size();
230   }
231
232 }
```

Código fuente A.9: Clase `MinimalTest`. Prueba con fines didácticos que ejemplifica la evaluación con el modelo MERERO

Apéndice B

Salida de los experimentos

B.1. Salida de los experimentos

```
Recommender101 called with parameters: [config/recommender101.properties]
Looking for property file: config/recommender101.properties
Checking if test data has to be downloaded..
Checking and obtaining MovieLens 100k data
Target file data/movielens/MovieLens100kRatings.txt already exists.
Recommender101 v0.61, 2015-12-09
Tue Oct 17 06:23:12 CDT 2017
DataSet : data/movielens/ml-20m/ratings.txt
Basic data set statistics
-------------------------------
#Users:      100
#Items:      26744
#Ratings:      14170
Sparsity:      0.005
-------------------------------
Global avg:      3.436
Global median:     3.5
Standard deviation:    1.146
-------------------------------
Ratings freqs:
Value     Freq.    Pct.     Cum Pct.
0.5  268 2%  2%
1.0  867 6%  8%
1.5  162 1%  9%
2.0  1012  7%  16%
2.5  503 4%  20%
3.0  3112  22% 42%
3.5  1443  10% 52%
4.0  3811  27% 79%
4.5  843 6%  85%
5.0  2149  15% 100%
-------------------------------
Skewness:       -0.669
Kurtosis:       -0.059
Gini of freqs:     0.4440649151802063
Avg. Ratings/user:   141.7
Avg. Ratings/item:   0.53
Min. Ratings/user:   20
Min. Ratings/item:   1
Max. Ratings/user:   1785
Max. Ratings/item:   50
-------------------------------
--------------------------
Evaluation results:
--------------------------
MAE                              |     2.988 |MaxMSERecommender
MAE                              |     2.694 |FlipRecommender
MAE                              |      0.53 |FunkSVDRecommender:numFeatures=50|initialSteps=50
MAE                              |     0.009 |BestRecommender

MEREROEvaluator                  |     2.776 |FlipRecommender
MEREROEvaluator                  |      2.68 |MaxMSERecommender
MEREROEvaluator                  |     2.663 |FunkSVDRecommender:numFeatures=50|initialSteps=50
MEREROEvaluator                  |         0 |BestRecommender
```

```
55
56 RMSE                              |    3.075 |MaxMSERecommender
57 RMSE                              |    3.005 |FlipRecommender
58 RMSE                              |    0.722 |FunkSVDRecommender:numFeatures=50|initialSteps=50
59 RMSE                              |    0.069 |BestRecommender
60
61 ------------------------------
62 Runtime results:
63 ------------------------------
64 Training:        0ms |Predicting:   433359ms |FlipRecommender
65 Training:        0ms |Predicting:   551142ms |BestRecommender
66 Training:   119631ms |Predicting:    91466ms
       |FunkSVDRecommender:numFeatures=50|initialSteps=50
67 Training:        0ms |Predicting:   314873ms |MaxMSERecommender
68
69 Program ended
```

Salida B.1: Experimento con una muestra de 100 usuarios.

```
1  Recommender101 called with parameters: [config/recommender101.properties]
2  Looking for property file: config/recommender101.properties
3  Checking if test data has to be downloaded..
4  Checking and obtaining MovieLens 100k data
5  Target file data/movielens/MovieLens100kRatings.txt already exists.
6  Recommender101 v0.61, 2015-12-09
7  Tue Oct 17 02:25:14 CDT 2017
8  DataSet : data/movielens/ml-20m/ratings.txt
9  Basic data set statistics
10 ------------------------------
11 #Users:      1000
12 #Items:      26744
13 #Ratings:      146488
14 Sparsity:      0.005
15 ------------------------------
16 Global avg:      3.55
17 Global median:     4
18 Standard deviation:    1.042
19 ------------------------------
20 Ratings freqs:
21 Value     Freq.    Pct.     Cum Pct.
22 0.5 1470  1%  1%
23 1.0 4686  3%  4%
24 1.5 1985  1%  6%
25 2.0 10184 7%  13%
26 2.5 6615  5%  17%
27 3.0 31517 22% 39%
28 3.5 15814 11% 49%
29 4.0 40696 28% 77%
30 4.5 11145 8%  85%
31 5.0 22376 15% 100%
32 ------------------------------
33 Skewness:       -0.638
34 Kurtosis:       0.121
35 Gini of freqs:     0.4619941711425781
36 Avg. Ratings/user:   146.488
37 Avg. Ratings/item:   5.477
38 Min. Ratings/user:   20
39 Min. Ratings/item:   1
40 Max. Ratings/user:   4101
41 Max. Ratings/item:   495
42 ------------------------------
43 ------------------------------
44 Evaluation results:
45 ------------------------------
46 MEREROEvaluator                 |    2.695 |FlipRecommender
47 MEREROEvaluator                 |    2.617 |MaxMSERecommender
48 MEREROEvaluator                 |    0.525 |FunkSVDRecommender:numFeatures=50|initialSteps=50
49 MEREROEvaluator                 |        0 |BestRecommender
50
51 RMSE                            |    3.033 |MaxMSERecommender
52 RMSE                            |    2.992 |FlipRecommender
53 RMSE                            |    0.759 |FunkSVDRecommender:numFeatures=50|initialSteps=50
54 RMSE                            |     0.05 |BestRecommender
55
56 MAE                             |    2.978 |MaxMSERecommender
57 MAE                             |    2.876 |FlipRecommender
58 MAE                             |    0.583 |FunkSVDRecommender:numFeatures=50|initialSteps=50
59 MAE                             |    0.007 |BestRecommender
60
```

```
61
62
63  ------------------------------
64  Runtime results:
65  ------------------------------
66  Training:         0ms |Predicting:    36970ms |BestRecommender
67  Training:         0ms |Predicting:    14114ms |FlipRecommender
68  Training:         0ms |Predicting:    29240ms |MaxMSERecommende
69  Training:   218932ms |Predicting:     9466ms
        |FunkSVDRecommender:numFeatures=50|initialSteps=50
70
71  Program ended
```

Salida B.2: Experimento con una muestra de 1,000 usuarios.

```
1   Recommender101 called with parameters: [config/recommender101.properties]
2   Looking for property file: config/recommender101.properties
3   Checking if test data has to be downloaded..
4   Checking and obtaining MovieLens 100k data
5   Target file data/movielens/MovieLens100kRatings.txt already exists.
6   Recommender101 v0.61, 2015-12-09
7   Tue Oct 17 11:48:37 CDT 2017
8   DataSet : data/movielens/ml-20m/ratings.txt
9   Basic data set statistics
10  ------------------------------
11  #Users:       138493
12  #Items:       26744
13  #Ratings:     20000263
14  Sparsity:     -0.034
15  ------------------------------
16  Global avg:     3.526
17  Global median:    3.5
18  Standard deviation:   1.052
19  ------------------------------
20  Ratings freqs:
21  Value    Freq.    Pct.    Cum Pct.
22  0.5 239125  1%  1%
23  1.0 680732  3%  5%
24  1.5 279252  1%  6%
25  2.0 1430997 7%  13%
26  2.5 883398  4%  18%
27  3.0 4291193 21% 39%
28  3.5 2200156 11% 50%
29  4.0 5561926 28% 78%
30  4.5 1534824 8%  86%
31  5.0 2898660 14% 100%
32  ------------------------------
33  Skewness:     -0.655
34  Kurtosis:     0.137
35  Gini of freqs:    0.4556567072868347
36  Avg. Ratings/user:  144.414
37  Avg. Ratings/item:  747.841
38  Min. Ratings/user:  20
39  Min. Ratings/item:  1
40  Max. Ratings/user:  9254
41  Max. Ratings/item:  67310
42  ------------------------------
43
44  ------------------------------
45  NO ES EL CORRECTO
46  ------------------------------
47  ------------------------------
48  Evaluation results:
49  ------------------------------
50  MEREROEvaluator                 |    2.685 |FlipRecommender
51  MEREROEvaluator                 |    2.612 |MaxMSERecommender
52  MEREROEvaluator                 |    2.593 |FunkSVDRecommender:numFeatures=50|initialSteps=50
53  MEREROEvaluator                 |        0 |BestRecommender
54
55  RMSE                            |     3.03 |MaxMSERecommender
56  RMSE                            |    2.981 |FlipRecommender
57  RMSE                            |    2.981 |FunkSVDRecommender:numFeatures=50|initialSteps=50
58  RMSE                            |    0.055 |BestRecommender
59
60  MAE                             |    2.845 |FlipRecommender
61  MAE                             |    2.948 |MaxMSERecommender
62  MAE                             |    0.655 |FunkSVDRecommender:numFeatures=50|initialSteps=50
63  MAE                             |    0.006 |BestRecommender
64
```

```
65 ------------------------------
66 Runtime results:
67 ------------------------------
68 Training:          0ms | Predicting:   570365ms | BestRecommender
69 Training:          0ms | Predicting:   204753ms | FlipRecommender
70 Training:          0ms | Predicting:   463944ms | MaxMSERecommender
71 Training:  25923142ms | Predicting:    75622ms
       | FunkSVDRecommender: numFeatures=50| initialSteps=50
72
73 Program ended
```

Salida B.3: Experimento con una muestra de 138,493 usuarios.

```
<terminated> MinimalTest [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (03/12/2017 00:42:23)
EL usuairo tiene n :636
Entrenando modelos ...
;************TIEMPOR ENTRENAMIENTO******************;
BestRecommender : 0 ms
FlipRecommender : 0 ms
MaxMSERecommender       : 0 ms
FunkSVDRecommender      : 43286 ms

;************EVALUACIÓN******************;

BestRecommender * [merero:0.0, mse:0.0]:
        Estimaciones (<idItem>=<valorEstimado>):
                {525=5.0, 272=4.0, 739=4.0, 770=4.0, 38=3.0, 79=3.0, 755=3.0, 866=3.0, 312=1.0, 561=1.0}
        Rankings:
                Original: [525, 272, 739, 770, 38, 79, 755, 866, 312, 561]
                Estimado: [525, 272, 739, 770, 38, 79, 755, 866, 312, 561]
        Vectores:
                Original   : {38=5, 79=6, 272=2, 312=9, 525=1, 561=10, 739=3, 755=7, 770=4, 866=8}
                Estimado   : {38=5, 79=6, 272=2, 312=9, 525=1, 561=10, 739=3, 755=7, 770=4, 866=8}
                Diferencias: |5-5|+|6-6|+|2-2|+|9-9|+|1-1|+|10-10|+|3-3|+|7-7|+|4-4|+|8-8|+
                           : 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 +
                           : 0.0

FlipRecommender * [merero:38.0, mse:5.9999743]:
        Estimaciones (<idItem>=<valorEstimado>):
                {561=5.0000334, 312=5.0000186, 866=3.0000515, 755=3.0000448, 79=3.0000048, 38=3.0000021, 770=2.0000458, 739=2.0000439, 272=2.0000162, 525=1.0000312}
        Rankings:
                Original: [525, 272, 739, 770, 38, 79, 755, 866, 312, 561]
                Estimado: [561, 312, 866, 755, 79, 38, 770, 739, 272, 525]
        Vectores:
                Original   : {38=5, 79=6, 272=2, 312=9, 525=1, 561=10, 739=3, 755=7, 770=4, 866=8}
                Estimado   : {38=6, 79=5, 272=9, 312=2, 525=10, 561=1, 739=8, 755=4, 770=7, 866=3}
                Diferencias: |6-5|+|5-6|+|9-2|+|2-9|+|10-1|+|1-10|+|8-3|+|4-7|+|7-4|+|3-8|+
                           : 1 + 1 + 7 + 7 + 9 + 9 + 5 + 3 + 3 + 5 +
                           : 38.0

MaxMSERecommender * [merero:40.0, mse:9.1]:
        Estimaciones (<idItem>=<valorEstimado>):
                {38=5.0, 79=5.0, 312=5.0, 561=5.0, 755=5.0, 866=5.0, 272=1.0, 525=1.0, 739=1.0, 770=1.0}
        Rankings:
                Original: [525, 272, 739, 770, 38, 79, 755, 866, 312, 561]
                Estimado: [38, 79, 312, 561, 755, 866, 272, 525, 739, 770]
        Vectores:
                Original   : {38=5, 79=6, 272=2, 312=9, 525=1, 561=10, 739=3, 755=7, 770=4, 866=8}
                Estimado   : {38=1, 79=2, 272=7, 312=3, 525=8, 561=4, 739=9, 755=5, 770=10, 866=6}
                Diferencias: |1-5|+|2-6|+|7-2|+|3-9|+|8-1|+|4-10|+|9-3|+|5-7|+|10-4|+|6-8|+
                           : 4 + 4 + 5 + 6 + 7 + 6 + 6 + 2 + 6 + 2 +
                           : 40.0

FunkSVDRecommender * [merero:18.0, mse:1.0215076]:
        Estimaciones (<idItem>=<valorEstimado>):
                {79=4.3501096, 272=3.9637134, 525=3.9182944, 770=3.493242, 755=3.1054423, 739=2.9668357, 312=2.720731, 38=2.7147584, 561=2.639479, 866=2.605845}
        Rankings:
                Original: [525, 272, 739, 770, 38, 79, 755, 866, 312, 561]
                Estimado: [79, 272, 525, 770, 755, 739, 312, 38, 561, 866]
        Vectores:
                Original   : {38=5, 79=6, 272=2, 312=9, 525=1, 561=10, 739=3, 755=7, 770=4, 866=8}
                Estimado   : {38=8, 79=1, 272=2, 312=7, 525=3, 561=9, 739=6, 755=5, 770=4, 866=10}
                Diferencias: |8-5|+|1-6|+|2-2|+|7-9|+|3-1|+|9-10|+|6-3|+|5-7|+|4-4|+|10-8|+
                           : 3 + 5 + 0 + 2 + 2 + 1 + 3 + 2 + 0 + 2 +
                           : 18.0
```

Figura B.1: Salida de prueba de concepto de experimentos

Bibliografía

[1] Xavier Amatriain. Recommender Systems. *Machine Learning Summer School 2014 @ CMU*, July 2014.

[2] Deuk Hee Park, Hyea Kyeong Kim, Il Young Choi, and Jae Kyeong Kim. A Review and Classification of Recommender Systems Research. August 2011.

[3] Gunnar Schröder, Maik Thiele, and Wolfgang Lehner. Getting Goals and Choosing Metrics for RecommenderSystem Evaluations. August 2011.

[4] Wikipedia. Taxicab geometry. August 2017.

[5] Albin Bramstång Yaling Jin. Constructing a Context-aware Recommender System with Web Sessions. Master's thesis, Department of Computer Science and Engineering, Chalmers University of Technology, the Netherlands, June 2015.

[6] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*, volume 1. 2011.

[7] D. Jannach, L. Lerche, F. Gedikli, and G. Bonnin. What recommenders recommend - An analysis of accuracy popularity and sales diversity effects. In *Proc. 21st International Conference on User Modeling, Adaptation and Personalization (UMAP 2013)*, Rome, Italy, 2013.

[8] GroupLens Research. About MovieLens. August 2017.

[9] U. Hanani, B. Shapira, and P. Shoval. User Modeling and User-Adapted Interaction. *Information Filtering: Overview of Issues, Research and Systems*, 11, 2001.

[10] Nicholas J. Belkin. Helping People Find What They Dont Know. *Communications of the ACM*, 43(8):58–61, August 2000.

[11] Inc Wikimedia Foundation. Netflix Prize. February 2017.

[12] Lior Rokach Francesco Ricci and Bracha Shapira. Introduction to Recommender Systems Handbook. October 2010.

[13] Laurent Candillier, Kris Jack, Françoise Fessant, and Frank Meyer. *Collaborative and Social Information Retrieval and Access: Techniques for Improved User Modeling*. 2009.

[14] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 16(46):109–132, April 2013.

[15] Michael D. Ekstrand, John T. Riedl, and Joseph A. Konstan. Collaborative Filtering Recommender Systems. *NOW the essence of knowledge*, 4(2):81–173, 2011.

[16] NF.O. Isinkaye, Y.O. Folajimi, and B.A. Ojokoh. Recommendation systems: Principles methods and evaluation. *Egyptian Informatics Journal*, 16(20):262–271, Agosto 2015.

[17] Hyung Jun Ahn. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*, 1(178):37–51, 2007.

[18] Charif Alchiekh Haydar. *Les systèmes de recommandation à base de confiance*. PhD thesis, Université de Lorraine, France, September 2014.

[19] M. Simon, M. Lionel, and L. Frédérique. Recommandation sociale et locale basée sur la confiance. *Document numérique*, pages 33–56, 2012.

[20] A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, page 9, 2000.

[21] Paolo Massa and Paolo Avesani. Trust-aware Collaborative Filtering for Recommender Systems. *Via Sommarive*, 4(4):1–17, 2005.

[22] Jennifer Ann Golbeck. *Computing and Applying Trust in Web-based Social Networks*. PhD thesis, University of Maryland at College Park, 2005.

[23] J. O'Donovan and B Smyth. Trust in recommender systems. *Proceedings of the 10th international conference on intelligent user interfaces ACM*, page 167–174, 2005.

[24] Cantador Gutiérrez Iván. *Exploiting the conceptual space in hybrid recommender systems: a semantic-based approach*. PhD thesis, Universidad Autónoma de Marid, Espain, October 2008.

[25] Daniel Lemire and Anna Maclachlan. Slope One Predictors for Online Rating-Based Collaborative Filtering. February 2005.

[26] Bertrand Dechoux. Recommender Systems Naive Bayes Networks and the Netflix Prize. January 2009.

[27] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix Factorization techniques for recommender systems. 2009.

[28] Robin Burker. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 2002.

[29] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of Data Science*. 2015.

[30] Soanpet Sree Lakshmi and T. Adi Lakshmi. Recommendation Systems:Issues and challenges. *International Journal of Computer Science and Information Technologies*, 5(4):5771–5772, 2014.

[31] Gediminas Adomavicius and Alexander Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on knowledge and data engineering*, 4(6):734–749, 2005.

[32] Cataldo Musto, Fedelucio Narducci, Marco De Gemmis, Pasquale Lops, and Giovanni Semeraro. A Tag Recommender System Exploiting User and Community Behavior. October 2009.

[33] John Canny. Collaborative Filtering with Privacy via Factor Analysis. August 2002.

[34] Lotfi A. Zadeh. Is there a need for fuzzy logic? *Information Sciences*, 178:2751–2779, 2008.

[35] Simon Funk. Netflix Update: Try This at Home. 2006.

[36] BookCrossing.com. About BookCrossing. August 2017.

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