UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Framework orientado a algoritmos de recomendación basados en vecinos cercanos

Alejandro Gil Hernán
Tutor: Alejandro Bellogín Kouki
Ponente: Pablo Castells Azpilicueta

Febrero 2017

Framework orientado a algoritmos de recomendación basados en vecinos cercanos

AUTOR: Alejandro Gil Hernán
TUTOR: Alejandro Bellogín Kouki
PONENTE: Pablo Castells Azpilicueta

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Febrero 2017

Resumen (castellano)

Los sistemas de recomendación se han convertido en una pieza clave en internet, bien sea navegando a través de artículos científicos (Mendeley), música (Spotify, Last.fm), películas o vídeos (Netflix, IMDB, YouTube), personas (LinkedIn, Facebook, Twitter), así como plataformas online de venta de infinidad de productos (Amazon).
Dichos sistemas de recomendación están motivados a su vez por el crecimiento exponencial que ha experimentado la web en los últimos años y con la aparición de gran contenido digital y se caracterizan por observar la actividad de los usuarios y aprovecharla para predecir cuáles son los intereses de éstos, según los cuales se presentarán unos productos u otros, de una forma individual y personalizada.

En este TFG se explorará un tipo de algoritmo de recomendación muy habitual: aquellos basados en vecinos cercanos (KNN). Estos algoritmos se suelen utilizar para hacer recomendaciones basadas en similitudes entre usuarios o entre objetos, siendo de esta forma un tipo de filtrado colaborativo; sin embargo, si la similitud tiene en cuenta atributos de los usuarios o de los objetos, también se podría utilizar para algoritmos basados en contenido. Este potencial permite, en principio, que se puedan utilizar en multitud de dominios, teniendo la ventaja adicional de que su salida es fácil de interpretar y analizar.

En este trabajo se plantea diseñar e implementar un framework orientado a generar y evaluar recomendaciones basadas en este tipo de algoritmos, así como el estudio sobre los distintos parámetros a configurar para discernir cuál es la combinación que aporta mejores resultados. Por ello los objetivos principales serán obtener implementaciones generales, así como que se ejecuten de la forma más eficiente posible.

Palabras clave (castellano)

Sistema de recomendación, algoritmo, framework, vecinos cercanos (KNN), filtrado colaborativo, algoritmo basado en contenido.

Abstract (English)

Recommender systems have become a key element in the internet in different aspects such as scientific articles browsing (Mendeley), music (Spotify, Last.fm), movies (Netflix, IMDB), people (LinkedIn, Facebook, Twitter), even online selling platforms (Amazon). Such recommender systems are also motivated by the exponential growth experimented by the web in the past years and with the appearance of a substantial quantity of digital content; they are characterised by taking advantage of observing user's activity to predict their interests. The expected preferred products will be presented differently each time in an individual and personalised way.

In this bachelor thesis, it will be explored a very common kind of recommendation algorithm: k-nearest neighbours (KNN). These algorithms are typically used to make recommendations based on similarities between users or items, being that way a kind of collaborative filtering; however, if the similarity considers user's or item's attributes, it also will be possible to use it for content-based algorithms. This potential allows to use them in many domains, having the advantage of producing an easily interpretable and analysable output.

This work contemplates designing and implementing a framework oriented to generate and evaluate recommendations based on this kind of algorithms, as well as the study of the different parameter configurations to distinguish which is the most beneficial combination. Therefore, the principal objectives will be to obtain general implementations that will be executed in the most efficient way.

Keywords (inglés)

Recommender system, algorithm, framework, nearest neighbours (KNN), collaborative filtering, content-based.

Agradecimientos

Quiero agradecer en primer lugar a mi familia, sobre todo a mis padres y abuelos.
A mi madre por criarme y hacer de mí la persona que soy.
A mi padre por sus consejos.
A mis abuelos por tratarme como a un hijo.

A mi tutor Alejandro por guiarme en este trabajo y asignaturas de la carrera, por ser un profesor ejemplar con una dedicación como jamás he visto en el mundo docente.

Muchas gracias a mis amigos que son como hermanos y a Paula por estar siempre apoyándome.

Por último, dar las gracias a la Escuela Politécnica Superior por hacerme conocer a compañeros increíbles y formarme como profesional y como persona.

Alejandro Gil Hernán

ÍNDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1. Introducción

Los sistemas de recomendación (SR) son herramientas software y técnicas que proveen al usuario de elementos que pueden resultarle interesantes y afines a sus gustos, facilitando la labor de búsqueda entre la inmensa cantidad de información de la que dispone. Estas recomendaciones están relacionadas con las acciones que lleva a cabo el usuario: productos comprados, música que escucha, libros que lee, etc.

Hoy en día estamos más que acostumbrados a que prácticamente la totalidad de las aplicaciones informáticas de uso diario nos propongan contenido, ya sea porque Amazon nos recomiende un producto que comprar, Spotify una canción que escuchar o Youtube un vídeo que quizá nos guste. En definitiva, no nos resulta extraño que la propaganda *online* se adapte a nosotros con recomendaciones personalizadas.

1.1 Motivación

Los sistemas de recomendación nacen con el objetivo de facilitar la toma de decisiones al usuario, así como de aumentar el beneficio de empresas dedicadas al comercio *online*. Desde la aparición de la *World Wide Web* el comercio en internet ha experimentado un crecimiento exponencial, hasta el punto de que la comisión nacional de los mercados y la competencia determinó que la facturación de las tiendas *online* en España crecía a un 25% interanual.

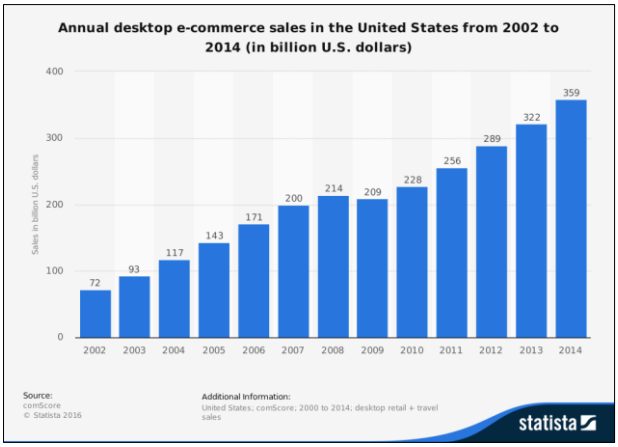


Figura 1. Valor de las ventas del comercio electrónico en EEUU

Para conseguir estos buenos resultados, los puntos más importantes de los SR son la valoración y comparación de los resultados obtenidos en las recomendaciones, lo que se conoce como evaluación. Durante mucho tiempo la manera natural de evaluar estos sistemas eran las métricas de error, pero es evidente que a lo largo de los años las tendencias varían; por ello, en la actualidad, están evolucionando hacia métricas basadas en ranking, ampliando un nuevo horizonte de estudio y desarrollo. Ahora queda

pendiente la comparativa de efectividad entre los distintos algoritmos y variantes dentro de éstos con este nuevo tipo de métricas.

Este trabajo de fin de grado se ha realizado con la intención de crear un framework donde sea posible la comparación de resultados para recomendaciones generadas con un algoritmo basado en vecinos cercanos (KNN) y todas sus variantes posibles, ya que como se ha mencionado anteriormente, esto viene motivado por el cambio en las métricas de evaluación (la manera en que se mide la tasa de error y, por tanto, la eficacia en una recomendación).

Esto es importante ya que los sistemas de recomendación ofrecen una serie de ventajas a las plataformas de productos online:

- Incrementar el número de ítems vendidos: ya que a cada usuario se le muestra primero lo que potencialmente más le atrae.
- Aumentar la diversidad: Una característica es generar diversidad en las recomendaciones, sugiriendo al usuario ítems similares a sus gustos con un índice de popularidad menor.
- Incrementar la satisfacción del usuario.
- Incrementar la fidelidad del usuario.

Es por esto que el método de evaluación en este trabajo tendrá presente las tres métricas que creemos más influyentes actualmente: precisión, recall y nDCG (*normalised Discounted Cumulative Gain*).

1.2 Objetivos

La meta general de este trabajo es averiguar cuál de las configuraciones implementadas del algoritmo KNN basado en filtrado colaborativo arroja mejores resultados.

De forma particular, los objetivos a llevar a cabo son los siguientes:

- Fijar el punto de partida con un conjunto de datos (*dataset*) concreto (no se va a tratar la recogida de información, por lo que se supone que ésta se obtiene de forma externa al proyecto).
- Estudio de las variantes del algoritmo KNN, así como de sus parámetros de entrada; con cada uno se implementará y evaluará al menos una variante.
- Estudio de métodos alternativos para el cálculo de vecinos, sin basarse en el procedimiento tradicional de los SR, para analizar su efecto en la eficiencia.
- Generar recomendaciones para los usuarios del sistema con la información del dataset inicial.

- Evaluar las recomendaciones generadas con las tres métricas mencionadas anteriormente.

- Sintetizar los resultados en una tabla comparativa.

Para contrastar las diferentes variaciones se realizarán experimentos exhaustivos donde se podrán comparar los valores de las distintas métricas obtenidos por cada variante, lo cual nos permitirá concluir qué alternativas son mejores en cada caso.

1.3 Organización de la memoria

El presente documento está dividido en cuatro partes. Primero, en el estado del arte, se hará una introducción a los sistemas de recomendación y la situación que atraviesan en la actualidad, a la vez que se presentan los distintos tipos de algoritmos que se van a estudiar. En el capítulo de diseño y desarrollo, se explica el proyecto realizado para el estudio de los sistemas de recomendación tomando como base la teoría ya explicada, y mostrando las estructuras y los algoritmos que se han implementado junto a los correspondientes diagramas. La sección de resultados expone los experimentos llevados a cabo, incluyendo tablas comparativas de todas las versiones para el conjunto de datos real estudiado. Para finalizar, se muestran las conclusiones obtenidas comentando los objetivos a llevar a cabo en un futuro. En última instancia, los anexos proporcionan más información acerca de secciones como los resultados completos o el manual de instalación de componentes necesarios.

2. Estado del arte

En la última década, los sistemas de recomendación están adquiriendo una importancia esencial debido al crecimiento del comercio electrónico y a la repercusión que Internet está teniendo en los consumidores.

Es común asociar el mundo de los SR al de buscadores, aunque en realidad son algo dispares: mientras que en un buscador es necesario que el usuario introduzca explícitamente lo que quiere buscar, un recomendador actúa como una entidad activa, recuperando información de manera implícita a través de las interacciones de los usuarios y sugiriendo ítems de una manera transparente.

A su vez, los SR comparten cualidades con el aprendizaje automático, ya que el sistema también aprende los intereses y gustos del usuario detectando patrones de comportamiento. Varias técnicas de los SR como el cálculo de vecinos, la factorización de matrices o las particiones del conjunto de datos utilizan conceptos y herramientas conocidas en ambos campos.

Los usuarios de un SR pueden tener características muy distintas; no obstante, para poder realizar recomendaciones personalizadas, lo esencial es estudiar la estructura que tiene la información que se posee. Ésta información puede estar estructurada de varias maneras, la selección de qué atributos son relevantes depende de la técnica de recomendación que se vaya a utilizar. Por ejemplo, en filtrado colaborativo, los usuarios son estructurados como una lista que contiene los diferentes ítems que ese usuario ha puntuado, aunque también pueden ser descritos por su patrón de comportamiento, como las acciones que llevan a cabo a la hora de navegar, por ejemplo, dónde se hace click, de forma que es posible conocer los intereses también de esta manera.

Aun así, las puntuaciones o *ratings* son la forma más común de recoger los datos en un SR. Pueden ser obtenidos de manera implícita (el número de veces que se escucha una canción) o explícita (la puntuación numérica que un usuario da a un ítem concreto).

Existen diferentes tipos de ratings explícitos:

- Numéricos: puntuación normalmente comprendida entre 1 y 5.
- Binarios: me gusta, no me gusta
- Ordinales (en desacuerdo, de acuerdo, muy de acuerdo…).

Por otro lado, los tipos de ratings implícitos dependen de la plataforma donde esté implementado el SR, por ejemplo, en el caso de canciones, el rating representa las veces que ha sido escuchada cada canción o cada artista; para productos, puede representar simplemente el haber hecho click, y para vídeos el tiempo de visualización, etc. Con este tipo de ratings, el hecho de que un usuario no haya interaccionado con un ítem concreto normalmente se entiende como que no conoce dicho ítem, mientras que, si el valor es muy bajo, se interpreta como que tiene poco interés en el mismo. En cambio, la ausencia de ratings explícitos aporta un matiz diferente a la hora de recomendar, ya que el usuario puede conocer el ítem, pero prefiere no invertir ese tiempo en dar dicha información al sistema.

Un evento a destacar ocurrido en los últimos años ha sido el *Netflix prize* (premio Netflix) [11], donde la famosa empresa multinacional de servicios multimedia (series y películas) propuso una competición abierta en busca del mejor algoritmo de filtrado colaborativo para generar recomendaciones de películas. En el reto podía participar cualquier persona excepto aquellas cercanas a la propia empresa.

El 21 de septiembre de 2009 el gran premio de US$1.000.000 fue otorgado al equipo de BellKor's Pragmatic Chaos, formado por la unión de los equipos Bellkor in BigChaos and Pragmatic Theory, quienes consiguieron una mejora de 10,06% sobre el algoritmo de Netflix.

Este reto es importante ya que junto a él surge la motivación de mejorar y optimizar los métodos de recomendación, así como la publicación de un dataset muy amplio y relevante a nivel mundial con 480.000 usuarios, 17.000 películas y más de 100 millones de ratings.

2.1 Tipos de algoritmos

Dependiendo de la forma en que los algoritmos tengan en cuenta los atributos de los ítems o usuarios, podemos concluir que existen dos tipos de algoritmos principales: basados en contenido y basados en filtrado colaborativo.

2.1.1 Algoritmos basados en contenido

Cuando se aplican métodos basados en contenido, el sistema aprende a recomendar ítems similares a los que le han gustado al usuario en un pasado [6]. Esta similitud es calculada basándose en las características asociadas a los ítems. En el caso de que un usuario haya puntuado con buena nota una película cuyo género es comedia, es muy probable que se generen recomendaciones de películas con este mismo género. En el caso de ítems más complejos como documentos de texto, se calcula el *TF-IDF* (Term Frequency-Inverse Document Frequency) que comprende las palabras o términos más relevantes.
Los algoritmos más comunes de éste género son *Rocchio* y KNN.

2.1.2 Algoritmos basados en filtrado colaborativo

A diferencia de los métodos basados en contenido, el filtrado colaborativo se basa en los ratings de los usuarios del sistema. La idea clave es que, si dos usuarios *u* y *v* son similares, el sistema recomendará a *u*, ítems que le hayan gustado a *v*, ya que en teoría sus gustos son similares.

El filtrado colaborativo soluciona algunas de las limitaciones de los algoritmos basados en contenido. Por ejemplo, los ítems cuyo contenido es difícil de obtener pueden seguir siendo recomendados a usuarios por la similitud con otros. Otra ventaja es que el sistema puede recomendar mucha más variedad de elementos, nunca produciéndose el encasillamiento que ocurre con los basados en contenido, ya que las tendencias de una comunidad de usuarios pueden ir variando, a la vez que el SR se adapta a ellas.

Basados en usuario (User KNN)

Se recomiendan al usuario los ítems que han gustado a usuarios similares a éste.

$$f(u,i) = \sum_{\substack{v \in N_k(u) \\ r(v,i) \neq \emptyset}} sim(u,v) * r(v,i)$$

Basado en ítem (Item KNN)

Se recomiendan al usuario los ítems que se parecen a ítems que le han gustado.

$$f(u,i) = \sum_{\substack{j \in N_k(i) \\ r(u,j) \neq \emptyset}} sim(i,j) * r(u,j)$$

Los algoritmos KNN requieren de un vecindario para realizar el cálculo de ratings [7]. Si se aplica un filtrado colaborativo basado en usuario, el vecindario se compondrá de un conjunto de usuarios ordenados por similitud, en orden decreciente, tomando como referencia el usuario al que se le quiere recomendar. En el caso de utilizar filtrado colaborativo basado en ítem, la estructura es la misma, teniendo ítems en vez de usuarios en el vecindario.

Dichas similitudes representan el factor de semejanza y se calculan todas para cada par de usuarios o ítems. Existen diferentes métodos para medirlas, en este trabajo se han estudiado las siguientes similitudes:

- **Coseno**: Quizá sea la más famosa de todas, su función es medir el ángulo entre los vectores (ratings) para cada par de usuarios o ítems. Si ambos vectores apuntan al mismo lugar (son iguales) el valor de la similitud tendrá como valor 1, en el caso contrario (los usuarios son ortogonales, por lo que el coseno se anula) se devuelve una similitud de 0.

$$\cos(u,v) = \frac{\sum_{i \in \mathcal{I}_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in \mathcal{I}_u} r_{ui}^2 \sum_{j \in \mathcal{I}_v} r_{vj}^2}}$$

En el caso de coseno basado en ítem:

$$\cos(i,j) = \frac{\sum_{u \in \mathcal{U}_{ij}} r_{ui} r_{uj}}{\sqrt{\sum_{u \in \mathcal{U}_i} r_{ui}^2 \sum_{u \in \mathcal{U}_j} r_{uj}^2}}$$

- **Pearson**: Muy similar al coseno, con la diferencia de que Pearson tiene en cuenta la puntuación media del usuario o ítem para suavizar las puntuaciones con valores extremos.

$$\text{PC } (u,\ v) = \frac{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \overline{r}_u)(r_{vi} - \overline{r}_v)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \overline{r}_u)^2 \sum_{i \in \mathcal{I}_{uv}} (r_{vi} - \overline{r}_v)^2}}$$

Para ítems:

$$\text{PC } (i,\ j) = \frac{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \overline{r}_i)(r_{uj} - \overline{r}_j)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \overline{r}_i)^2 \sum_{u \in \mathcal{U}_{ij}} (r_{uj} - \overline{r}_j)^2}}$$

Se han implementado dos variantes de ésta similitud, ya que a la hora de realizar el cálculo de la media ($\overline{r_u}, \overline{r_l}$) se pueden tomar todos los usuarios o ítems del dataset o simplemente aquellos que posean ratings en común.

- **Jaccard**: Mide el grado de similitud entre dos conjuntos como la cardinalidad de la intersección de ambos conjuntos dividida por la cardinalidad de su unión. Devuelve 0 si los conjuntos no poseen ningún valor en común, tendiendo a 1 a medida que aumenta el número de elementos compartidos:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Factorización de matrices

Existen otros métodos no estudiados en este trabajo [8], como los algoritmos de factorización de matrices, que son bastante exitosos. La mayoría de estos métodos caracterizan a los usuarios y a los ítems como vectores derivados de los patrones de ratings, dividiendo la matriz inicial en productos de diferentes matrices. Éstos métodos se han popularizado recientemente por tener un buen equilibrio entre escalabilidad y precisión en las recomendaciones, también son más flexibles a la hora de modelar situaciones del mundo real. Por norma general, los datos de entrada más convenientes para este tipo de algoritmos son ratings explícitos, pero en caso de no tenerlos, es posible añadir información adicional, utilizando ratings implícitos que reflejan indirectamente la opinión de los usuarios simplemente observando su comportamiento.

Hay diferentes variantes de algoritmos para obtener la factorización de matrices como pLSA (probabilistic Latent Semantic Analysis), SVD (Singular Value Decomposition), SVDN (SVD no-empty entries), HSVDN (SVD with Hypergraph transformation).

2.1.3 Algoritmos generales

Los algoritmos base o generales (*baseline*) no ofrecen recomendaciones personalizadas, pues no tienen en cuenta ningún dato acerca de los usuarios.

Popularidad

Recomienda los ítems más populares, todos los usuarios tienen el mismo ranking. La popularidad implica el número de interacciones totales con un ítem, ya sean reproducciones de un vídeo o canción, productos más vendidos o los ratings más altos.

Éste algoritmo es de los que más se usan a nivel comercial, sin necesidad de utilizar un sistema de recomendación, debido a su fácil implementación y aporte de información útil para el usuario.

Random

Recomienda ítems de manera aleatoria, representa la probabilidad de escoger un ítem en un conjunto de datos, por lo que cuanto más denso sea, más probabilidad de recomendar al usuario algo que le guste.

2.2 Normalización de ratings

La normalización se aplica para acotar el valor de un rating a una escala deseada o para tener (o no tener) en cuenta ciertas desviaciones en los datos [7]. Existen las siguientes variantes:

- **Mean-Centering**: se compara el rating calculado con la media del usuario objetivo. Así se puede saber la apreciación (positiva o negativa) de un cierto usuario por un ítem observando el rating normalizado.

$$\hat{r}_{ui} = \overline{r}_u + \frac{\sum_{v \in \mathcal{N}_i(u)} (r_{vi} - \overline{r}_v) w_{uv}}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|}$$

  Para ítems:

$$\hat{r}_{ui} = \overline{r}_i + \frac{\sum_{j \in \mathcal{N}_u(i)} (r_{uj} - \overline{r}_j) w_{ij}}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}$$

- **Z-score**: supóngase el caso donde se tienen dos usuarios: A y B, el primero de ellos, posee una dispersión de ratings alta (varían mucho), mientras que el segundo siempre puntúa con un rating igual a 3.

El hecho de que B puntúe un ítem con un rating de 5 implica más apreciación por parte de ese usuario que si dicha puntuación se la hubiera dado A. En mean-centering se eliminan estas desviaciones tomando únicamente la media.

Esta apreciación es la que mide z-score dividiendo el mean-centering del usuario entre la desviación típica de los ratings:

$$\overline{r}_u + \sigma_u \frac{\sum_{v \in \mathcal{N}_i(u)} (r_{vi} - \overline{r}_v) w_{uv} / \sigma_v}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|}$$

Para ítems:

$$\overline{r}_i + \sigma_i \frac{\sum_{j \in \mathcal{N}_u(i)} (r_{uj} - \overline{r}_j) w_{ij} / \sigma_j}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}$$

2.3 Selección de vecinos

El número de vecinos próximos y el criterio utilizado puede tener un gran impacto en la calidad del sistema de recomendación [7]. La selección de los vecinos se lleva a cabo en dos pasos:

- Filtrado global donde solamente los candidatos más afines son seleccionados.
- Una fase de predicción donde se escogen los posibles mejores candidatos.

Filtrado global de vecinos

- **Top-N**: Para cada usuario, se guardan los N vecinos más cercanos con sus respectivas similitudes. Es necesario escoger de manera cautelosa este N por motivos de rendimiento y precisión. Si N es muy grande, se necesitará gran cantidad de memoria para almacenar todo el vecindario con los datos asociados y la predicción será lenta. Sin embargo, escogiendo un N muy pequeño es probable que algunos ítems nunca sean recomendados debido a la baja cobertura (ya que, para recomendar un ítem, es necesario que al menos un vecino lo haya puntuado).
- **Threshold**: En vez de mantener un número fijo de vecinos (N), threshold establece un umbral, el cual deben superar los valores de las similitudes.

  Ésta técnica es algo más flexible que la anterior, pues solo los vecinos más significativos son escogidos, aun así, dicho umbral es más complicado de establecer.
- **Negative filtering**: Por lo general, las correlaciones negativas entre usuarios son menos fiables que las positivas, esto se debe intuitivamente a que una correlación positiva entre dos usuarios es un buen indicador de su correspondencia a un grupo determinado (adolescente, fans de un género determinado…).

Aun así, una correlación negativa puede implicar pertenencia a grupos opuestos, y dependiendo del conjunto de datos en el que se esté trabajando, se pueden ignorar.

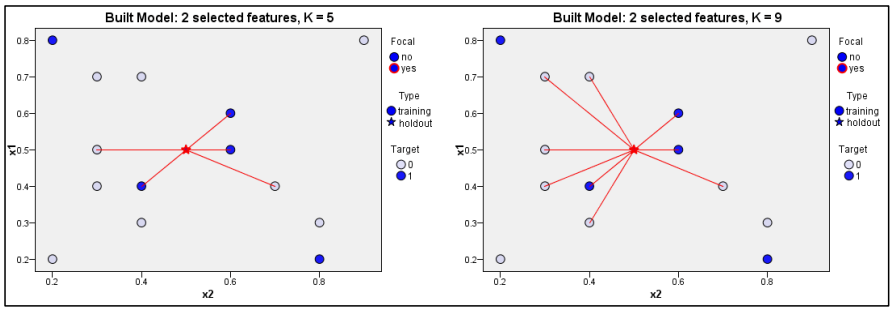


Figura 2. Elección de vecinos

Comentado [AGH1]: Hace tiempo me dijiste que esta foto igual no tenia mucho sentido, si lo ves necesario eliminala.

2.4 Comparación entre User KNN e Item KNN

Precisión

Depende del ratio entre el número de usuarios y los ítems del sistema, influyendo también si los ratings están distribuidos uniformemente sobre los ítems o no. En general, es preferible un número pequeño de vecinos fiables a una gran cantidad de vecinos con una similitud no tan alta.

Eficiencia

Tanto la memoria utilizada por un recomendador como el coste computacional dependen del ratio usuarios/ítems.

Por lo tanto, cuando el número de usuarios sobrepasa al número de ítems, correspondiendo al caso normal, la recomendación basada en ítem requiere mucha menos memoria y coste, ya que se tienen que calcular muchas menos similitudes. Aun así, el tiempo necesario para calcular una recomendación en la fase online, que depende solamente del nº de ítems relevantes y el nº máximo de vecinos es el mismo para los dos algoritmos.

En la práctica, el coste necesario para computar las similitudes es mucho menos costoso que el peor caso (teóricamente hablando) dado que los usuarios puntúan unos pocos ítems, y sólo se calculan las similitudes entre dos usuarios que hayan puntuado ítems en común, por lo que el conjunto de vecinos candidatos suele ser pequeño.

Estabilidad

La elección entre estos dos algoritmos se puede basar en la frecuencia y cantidad de cambios en los usuarios e ítems del sistema. Si la lista de ítems es bastante estática en

comparación a la de usuarios, una recomendación basada en ítem es preferible dado que las similitudes entre ítems se pueden calcular en intervalos de tiempo separados, pudiendo de esta manera realizar recomendaciones a usuarios nuevos.

Por el contrario, en sistemas donde el número de ítems cambia constantemente, los métodos basados en usuarios prueban ser más estables.

Un aspecto importante que diferencia a estos algoritmos es por ejemplo la variedad: mientras que una recomendación basada en ítem tendrá poca variedad, pues siempre recomendará ítems parecidos que le hayan gustado al usuario, provocando cierto estancamiento, por otra parte, una recomendación basada en usuario escogerá ítems algo más distantes a los gustos iniciales del usuario, que en ciertos casos es preferible al "estancamiento" antes mencionado.

2.5 Métricas de evaluación

Tradicionalmente, la eficacia de las recomendaciones se ha calculado mediante métricas de error como MAE (Mean Squared Error) o RMSE (Root Mean Squared Error) que representan la distancia entre la puntuación generada por el recomendador y la que realmente hizo el usuario. El problema reside en que este tipo de métricas están orientadas a la predicción de ratings, y éstos no están siempre disponibles, ya que es bastante probable que existan ítems que el usuario no ha puntuado. Existen a su vez métricas que tienen en cuenta la novedad y diversidad de los ítems, no estudiadas en este trabajo. Por ello, el foco de trabajo se ha dirigido hacia tres métricas basadas en ranking: precisión, recall y nDCG (*normalised Discounted Cumulative Gain*).

$$\text{Precision(L)} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|L(u) \cap T(u)|}{|L(u)|}$$

$$\text{Recall(L)} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|L(u) \cap T(u)|}{|T(u)|}$$

La métrica de *precisión* se obtiene a partir del conjunto intersección entre los ítems recomendados ($L(u)$) y los relevantes ($T(u)$) y dividiéndolo entre el conjunto total de recomendados. *Recall* por el contrario, divide la misma intersección por el total de ítems relevantes, éstos son los que el usuario ha puntuado, mientras que los recomendados hacen referencia a los que el sistema ha proporcionado para un usuario concreto.

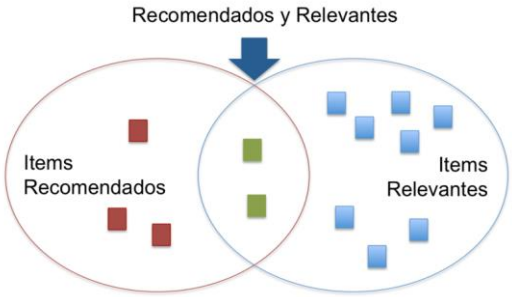


Figura 3. Conjunto intersección entre ítems recomendados y relevantes

Al aumentar el recall (la proporción de elementos relevantes) disminuye la precisión, ya que, si hay un número alto de relevantes, es menos probable llegar a recomendarlos todos, de esta manera, hay un compromiso entre ambas métricas.

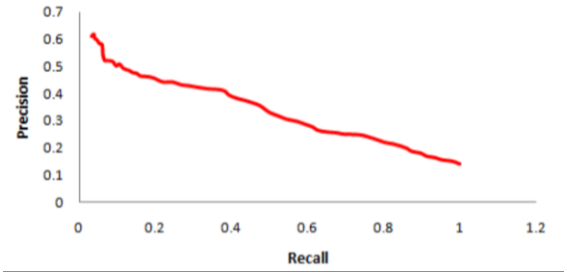


Figura 4. Precisión frente a recall

La última métrica por tratar (nDCG) se basa en la relevancia de los ítems teniendo en cuenta su posición en el ranking de recomendación, de modo que el valor de esta métrica será alto si los ítems más relevantes son los primeros posicionados.

$$nDCG = \frac{DCG}{iDCG}$$

$$DCG = \sum_{i}^{p} \frac{2^{rel_i} - 1}{log_2(1+i)}$$

Primero se calcula el *DCG* (*Discounted Cummulative Gain*), que representa la suma de grados de relevancia penalizados por lo tarde que aparezcan en el ranking, como se menciona anteriormente. *iDCG* hace referencia al valor ideal de *DCG*, es decir, es el valor de DCG que obtiene un ranking tal que todos los ítems devueltos estuvieran ordenados de acuerdo a las preferencias del usuario.

3. Diseño y desarrollo

En este capítulo se describen los aspectos técnicos del proyecto en cuanto a desarrollo de software, incluyendo la integración de librerías externas y un amplio rango de pruebas con su correspondiente análisis, para poder comparar el rendimiento de las diferentes versiones del algoritmo.

De manera general, se puede subdividir el proyecto en módulos diferenciándolos por funcionalidad:

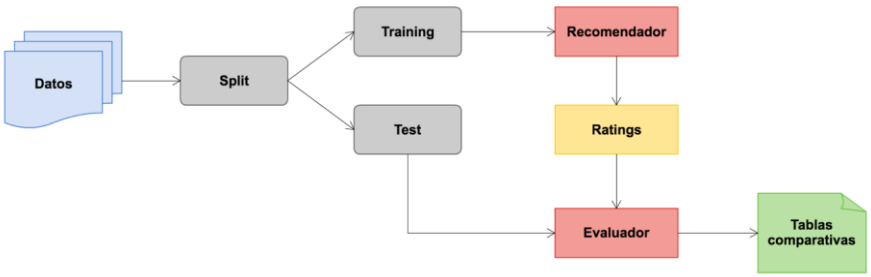


Figura 5. Esquema de arquitectura.

Todo parte de un conjunto de datos (*dataset*), que contiene los ficheros de usuarios, ítems, ratings... con sus correspondientes particiones de entrenamiento y test (*splits*). Este trabajo no se focaliza en la minería de datos, por lo que tomaremos dichos ficheros ajenos al sistema. Tampoco se realizan los splits del dataset ya que vienen incluidos.

Seguidamente, tenemos los dos grandes grupos de evaluación y recomendación, los cuales albergan el grueso del trabajo. Mientras que el recomendador se encarga de la generación de ficheros con las recomendaciones personalizadas de los usuarios a partir de cualquier combinación deseada, el evaluador toma esos ficheros para calcular la eficacia del algoritmo con la configuración de entrada. Este trabajo contiene todas las combinaciones posibles que permite el algoritmo para las parametrizaciones que han sido implementadas.

Una vez evaluadas las recomendaciones, ya disponemos de todos los datos necesarios para ensamblar nuestro framework, que serán extrapolados en formato de tablas para poder discernir la mejor solución con un simple vistazo.

3.1 Descripción del sistema

Teniendo en cuenta la funcionalidad que iba a adquirir el proyecto, así como los algoritmos que se iban a implementar, se optó por el uso de Java como lenguaje de programación al tener ventajas como:

- Lenguaje de alto nivel orientado a objetos.
- Gran soporte de librerías.
- Multiplataforma.
- Lenguaje conocido. Al ser un lenguaje altamente usado en la carrera, no requería un elevado esfuerzo de aprendizaje más allá de las novedades introducidas en la versión 1.8 (programación funcional), versión en la que se basa la principal librería del proyecto.

3.2 Requisitos del sistema

3.2.1 Requisitos funcionales

1. Permitir predicciones de rating basadas en usuario e ítem.
2. A la hora generar recomendaciones el sistema debe permitir la variación de distintos parámetros como:
    - Elegir la similitud del vecindario (Coseno, Jaccard o Pearson (con sus dos variantes)).
    - Elegir la similitud entre los usuarios o ítems en el momento de calcular el rating, puede ser diferente a la usada para generar el vecindario.
    - Permitir la normalización de la similitud.
    - Permitir la normalización de los ratings (*Standard* (sin normalizar), *Mean-Centering* o *Z-Score*, según lo explicado en la Sección 2.2*).*
    - Escoger el cálculo de vecinos mediante similitud o usando otras técnicas.
    - Permitir aplicar un umbral a la similitud (threshold).
3. Si ocurre algún fallo interno, el sistema informará del error ocasionado.
4. El sistema no dispondrá de interfaz gráfica.
5. El sistema generará ficheros de texto plano con los resultados de los experimentos.

3.2.2 Requisitos no funcionales

1. El sistema debe poder ser accedido desde un repositorio GIT (p.ej GitHub[1]).
2. El sistema funcionará en cualquier plataforma con versión de Java 1.8 a superior.

3.3 Diseño

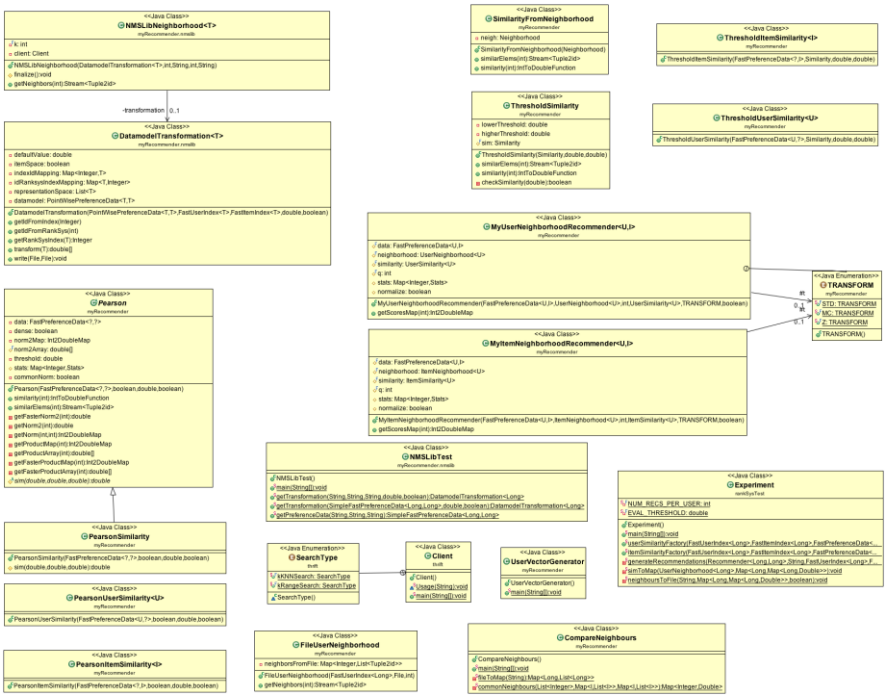


Figura 6. Diagrama de clases de la aplicación

Este diagrama muestra el diagrama de clases del sistema. La orientación a objetos ha tomado importancia a la hora de separar las clases e interfaces en distintos módulos a fin de maximizar la flexibilidad del código para añadir más algoritmos, datasets, métricas o librerías.

La metodología seguida a lo largo del proyecto ha sido un modelo de desarrollo clásico en cascada:

- Análisis de requisitos. Se sigue un estudio teórico del algoritmo con todas las posibles parametrizaciones posibles.
- Implementación. Tras el análisis inicial, se lleva a código.
- Validación. Una vez implementado se realizan pruebas con conjuntos de datos pequeños, además de pruebas de caja negra para comprobar el correcto funcionamiento.

- Evaluación y comparativas. Una vez se sabe que la implementación es correcta, se genera un script para poder ejecutar las pruebas de una sola vez. Estos resultados se muestran en tablas para facilitar su visualización.

Las clases a su vez está divididas en paquetes dependiendo de la funcionalidad. El primero (*myRecommender*) es el más grande en cuanto a contenido, posee principalmente el cómputo de la recomendación como se explica en 3.4, la implementación de la similitud de Pearson y la clase encargada de aplicar un threshold a la similitud.

En el segundo (el paquete *myRecommender.nmslib*) se incluyen las clases del recomendador necesarias para manejar el vecindario de NMSLIB (3.4.1).

La clase *compareNeighbours* nos permite saber el grado de semejanza entre distintas técnicas de cálculos de vecinos, como puede ser aquellos calculados con NMSLIB y los métodos de cálculo de vecinos por similitud; para ello se analiza el número de vecinos que tienen ambos en común.

Por otra parte, en *UserVectorGenerator* se generan los vectores de todos los usuarios, necesarios a la hora de calcular el vecindario con NMSLIB (3.4.2) esto es, trasladar a un fichero cada usuario con su lista de ítems puntuados o el valor por defecto en caso de un ítem sin rating. La idea es crear una matriz de *n* usuarios y *m* ítems, de manera que las filas representan los usuarios y las columnas los ítems.

Las dos últimas clases restantes en este paquete (*NMSLibNeighbourhood* y *DatamodelTransformation*) son utilizadas para invocar métodos de dicha librería. Mientras que en *NMSLibNeighbourhood* se conecta con el servidor para solicitar los vecinos, *DatamodelTransformation* mantiene el mapeo (control de los índices de usuarios e ítems entre RankSys y NMSLIB), ya que dichos índices son diferentes en ambos sitios, pudiéndose producir anomalías en ciertas ocasiones. Una vez obtenidos los vecinos, ya sea por medio de NMSLIB o por similitudes, podemos cargar el vecindario desde un fichero al sistema mediante *FileUserNeighbourhood*.

El paquete Thrift (3.4.4) contiene los ficheros necesarios para hacer funcionar Thrift, siendo *Client* la clase más importante.

Por último, todo es unificado en el paquete rankSysTest, donde se instancian los recomendadores para poder ejecutar cualquier configuración deseada dependiendo de los parámetros de entrada. La clase más significativa es *Experiment*, donde se aúnan las labores de recomendación y evaluación.

Podríamos concluir que la ejecución total del sistema se lleva a cabo en esta clase. La fase de predicción de rating se puede distribuir en tres partes. La carga de datos desde el dataset introduciendo en el sistema los usuarios e ítems los datos proporcionados por la partición de entrenamiento (*training*).

Una vez cargados los datos, se calculan los vecinos, dependiendo del método utilizado, de los datos introducidos previamente.

Finalmente, en la etapa de calcular las predicciones, se observa el conjunto de test, el cual determina las recomendaciones concretas que se deben calcular para evaluar la precisión de las mismas en la fase de evaluación, donde mediante diferentes métricas (2.5) se concreta lo acertado que ha resultado ser el recomendador con la configuración de entrada.

3.4 Desarrollo y codificación

El primer paso en el desarrollo es el estudio y la integración de la librería RankSys. De ella se han utilizado algunas funcionalidades como base para el proyecto principal como:

- Similitudes de coseno tradicional y Jaccard.
- Módulo de carga de datos del dataset.
- Estructuras de datos como Tuple2Id.

Sobre RankSys se han implementado distintas variantes del algoritmo kNN:

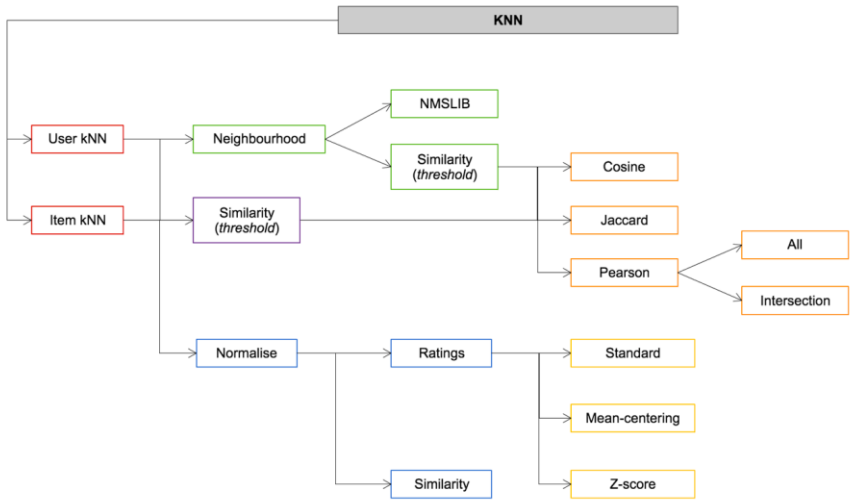


Figura 7. Esquema de las variantes en kNN

Si tenemos en cuenta la fórmula original del cálculo de ratings por ejemplo en filtrado colaborativo basado en usuarios, podemos realizar tres modificaciones generales tal como se muestra en el diagrama:

$$f(u,i) = \sum_{\substack{v \in N_k(u) \\ r(v,i) \neq \emptyset}} sim(u,v) * r(v,i)$$

En cuanto al vecindario, puede ser obtenido por dos vías, recibiendo los vecinos desde librería NMSLIB, explicado en 3.4.1 y 3.4.2 o bien por el método tradicional calculando la similitud entre los usuarios y obteniendo las k similitudes más altas.

Dentro de las similitudes existen diferentes opciones como se menciona en 2.1.2. Dos de ellas, coseno y Jaccard son utilizadas directamente desde la implementación que ofrece RankSys, mientras que Pearson es implementada desde el inicio ofreciendo a su vez otras dos variantes. También existe la posibilidad de aplicar un *threshold* a la similitud, esto es

un valor límite por debajo del cual no se van a considerar las similitudes, en el caso de escoger un threshold de 0.3, el vecindario no contendrá vecinos con un valor de similitud por debajo de 0.3, únicamente valores superiores.

Cuando se calcula una similitud entre dos usuarios se opera sobre los ítems que ambos han valorado, aunque también cabe la posibilidad de utilizar todo el conjunto de ítems, asignando un valor por defecto a los ítems no comunes. Normalmente este valor suele ser 0, pero un valor interesante podría ser la media del usuario para evitar distorsionar en exceso la similitud.

En el diagrama, "Pearson all" hace referencia a esto último, mientras que Pearson intersection implica el conjunto de ítems puntuados en común.

La segunda variante (Similarity threshold) puede parecer redundante, ya que se ha utilizado antes para el vecindario, pero no es así, pues aunque normalmente utilicemos la misma similitud para calcular los vecinos y la que multiplica al rating, según la fórmula *sim*(*u*,*v*) en 2.1.2, es posible utilizar diferentes para cada caso, pudiéndose aplicar threshold a las tres posibilidades anteriores.

La tercera y última variación es la normalización, bien de ratings (2.2) o de similitud. La normalización de similitudes implica dividir por el valor absoluto de la suma de las similitudes utilizadas en el sumatorio. Por otra parte, la normalización del rating implica establecer un intervalo fijo en el que se van a representar, normalmente [1,5]. En el caso de no aplicar normalización al rating, los valores de recomendación en vez de variar dentro del rango, tomarían valores muy altos, resultante de la suma de todos los ítems comunes.

Lo más común es que la normalización se aplique únicamente para presentarle al usuario los resultados, ya que internamente, no importa tener valores desproporcionados ya que hay que ordenarlos, siendo el valor más alto el ítem más recomendable.

De esta manera podríamos parametrizar la fórmula en tres puntos:

$$① + ② \sum_{\substack{v \in N_k(u) \\ r(v,i) \neq \emptyset}} sim(u,v) * r(v,i) \; - \; ③$$

- En el primero se suma la media del usuario tanto en Mean-Centering como en Z-score (2.2).
- En el segundo, se decide normalizar o no la similitud, y en el caso de Z-score, también se multitplica por la desviación típica del usuario.
- En el último se aplica la normalización de ratings, restando 0 en caso de STD, la media del vecino en Mean-Centering y la media del vecino dividida entre su desviación típica si aplicamos Z-score.

3.4.1 Librerías externas utilizadas

La librería principal en la que está basado el proyecto es RankSys[2], un framework para la implementación de algoritmos de recomendación y evaluación. Posee algoritmos báscos (popularidad, knn, factorización de matrices), similitudes (coseno, jaccard) para métodos basados en ítem y usuario. RankSys ha sido programada en Java 8, tomando ventajas de esta nueva versión, como las expresiones lambda o *streams*, así como la facilidad de paralelizar el código para obtener mayor rendimiento. Se han evaluado otras librerías como LensKit o Mahout pero no se obtenían los mismos resultados que con RankSys.

La otra librería utilizada se trata de NMSLIB, la cual realiza un cálculo de vecinos de manera aproximada para ahorrar tiempo de ejecución y recursos hardware, pues en lugar de ordenar la lista de todos los vecinos para tomar los k más cercanos, calcula la distancia entre los vectores (usuarios o ítems) en un espacio multidimensional. Para conectar estas dos librerías utilizamos una tercera: Apache Thrift (3.4.4)

3.4.2 Cálculo aproximado de vecinos

El cálculo de vecinos aproximados nace con la librería Annoy[3] creada por Erik Bernhardsson [2], [3] y [4]. La finalidad es la misma que el cómputo mediante similitudes, obtener los k vecinos más cercanos a un usuario en concreto.

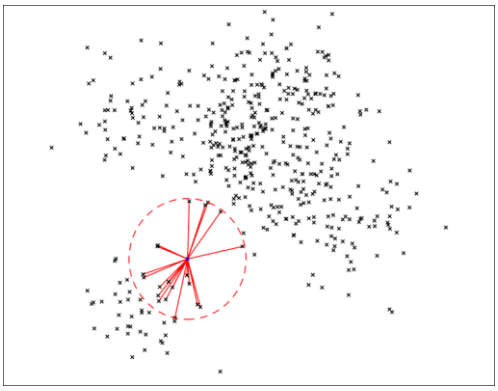


Figura 8. Mapa bidimensional de usuarios

La idea básica es representar objetos en un espacio donde la proximidad implique que sean parecidos entre ellos. En el caso de filtrado colaborativo, un dataset con 10 usuarios y 5 ítems compondrá un espacio de 5 dimensiones. La proximidad entre dos vectores normalmente se mide con la distancia euclídea al cuadrado:

$$\sum_{i=1}^{n}(q_i - p_i)^2$$

Por lo que en éste método se excluyen las similitudes y tomamos como valor de proximidad la distancia entre los vectores.

Volviendo al mapa de usuarios, el objetivo es calcular los vecinos más cercanos en el espacio. Nótese que se representa un espacio de dos dimensiones por un motivo meramente visual, ya que en realidad la mayoría de modelos vectoriales aplicados a sistemas de recomendación poseen dimensiones mucho mayores.

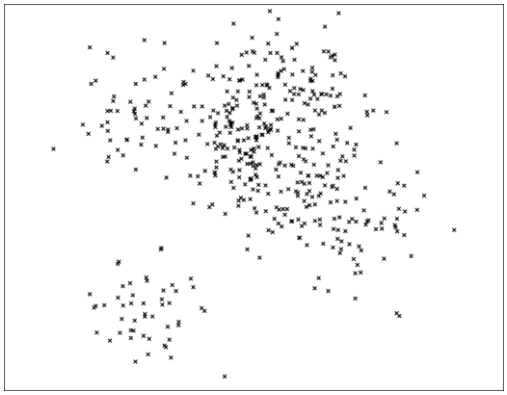


Figura 9. Mapa de distribución de usuarios en el espacio

El primer reto es establecer una estructura de datos que permita encontrar las distancias más cortas a un vector en un tiempo sublineal. Parece natural elegir un árbol para ello, pues su rendimiento es de orden $O(\log n)$.

Ésta es la manera en que funciona Annoy: un árbol binario donde cada nodo es una partición aleatoria del conjunto de usuarios y, de manera recursiva, se van tomando dos puntos (usuarios) de manera aleatoria y se traza un hiperplano equidistante a ambos.

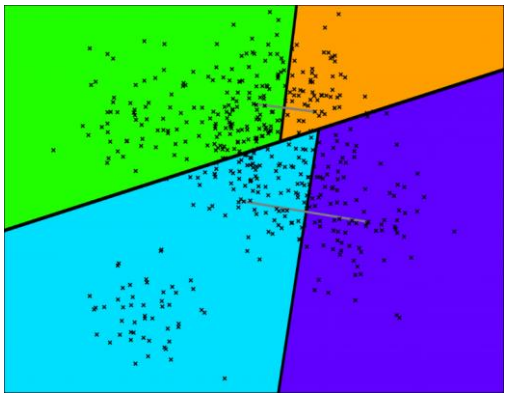


Figura 10. Mapa con tres particiones aleatorias

En este punto el árbol resultante sería el siguiente:

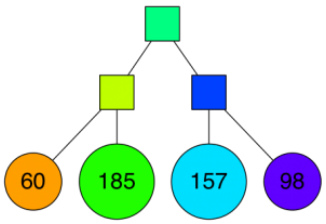


Figura 11. Árbol resultante de realizar tres particiones

Tras realizar sucesivas iteraciones hasta que en cada nodo queden 10 usuarios como mucho, nos quedan las siguientes particiones:

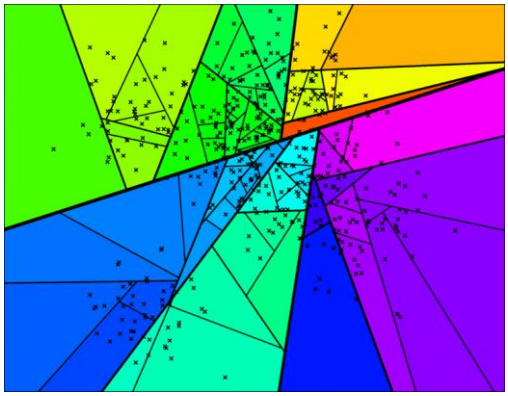


Figura 12. Mapa de particiones para k=10

Y el correspondiente árbol:

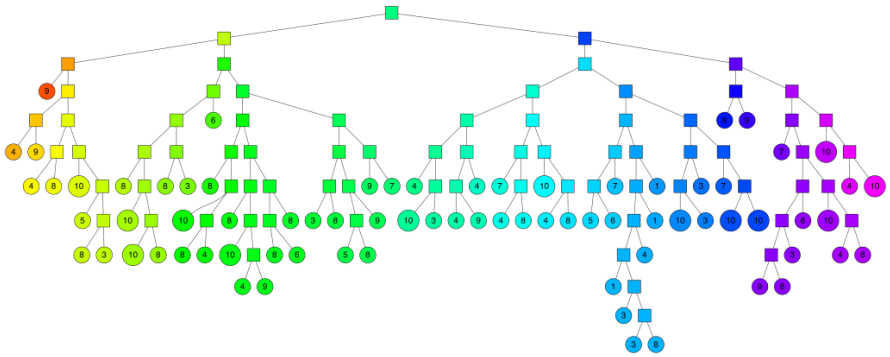


Figura 13. Árbol binario de particiones para k=10.

Una vez completado el árbol, se puede observar que, si dos puntos están cerca en el espacio, también estarán cerca en el árbol. Para buscar cualquier punto en el espacio, simplemente se ha de recorrer el árbol desde la raíz. Cada punto intermedio definido como cuadrado, representa un hiperplano, de manera que podemos saber hacia qué hijo deberemos dirigirnos a la hora de buscar.

Imaginemos que buscamos un usuario concreto que resulta ser el siguiente:

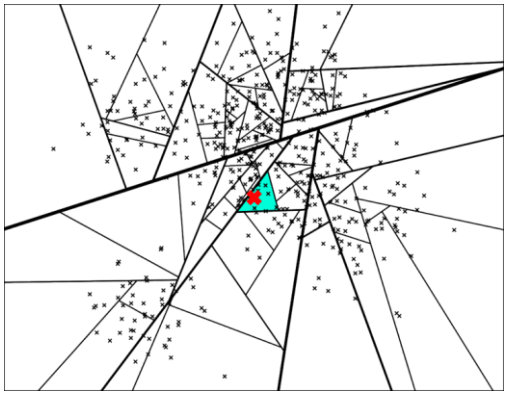


Figura 14. Búsqueda de un usuario en el espacio

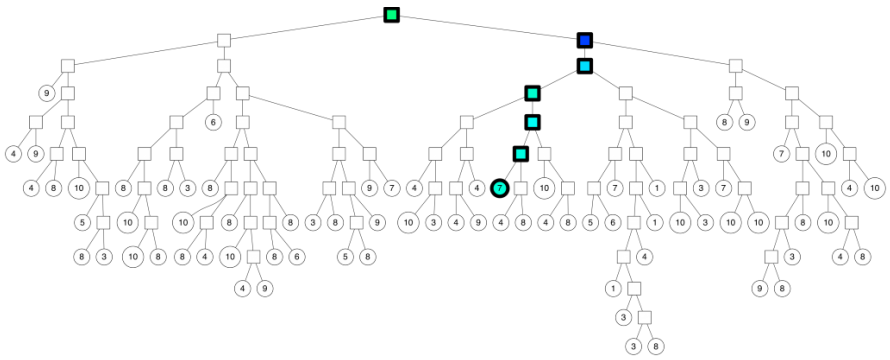


Figura 15. Árbol resultante de la búsqueda de un usuario

En el área que contiene al usuario en el espacio hay siete vecinos. Si observamos, éstos no resultan ser los más cercanos, pues el usuario está en la zona delimitada por un hiperplano quedando vecinos más cercanos en un área distinta.

Por lo tanto, hay dos problemas a resolver:

- Encontrar los vecinos realmente más cercanos
- Encontrar más de los siete vecinos que definen el área encontrada.

La primera solución pasa por ir explorando las áreas colindantes, profundizando en el árbol estableciendo un threshold (umbral) en las distancias para saber si estamos en la rama correcta. Para ello se utiliza una cola de prioridad para explorar nodos ordenados por la distancia al objetivo.

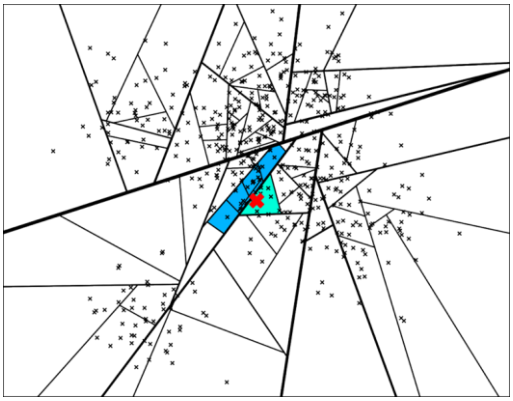


Figura 16. Mapa de ampliación en la búsqueda de vecinos

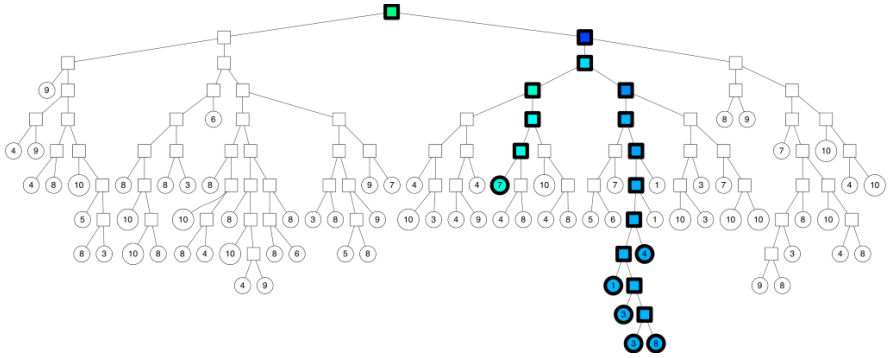


Figura 17. Árbol de ampliación en la búsqueda de vecinos

Otra posible solución pasa por crear muchos árboles (bosque), cada uno construido con distintas particiones, los cuales van a ser recorridos en la búsqueda al mismo tiempo.

Como tenemos la cola de prioridad ya configurada, es posible recorrer todos los árboles para obtener la unión de las hojas de los árboles (áreas) que contienen los puntos más cercanos. Ésta unión de áreas representa el vecindario:

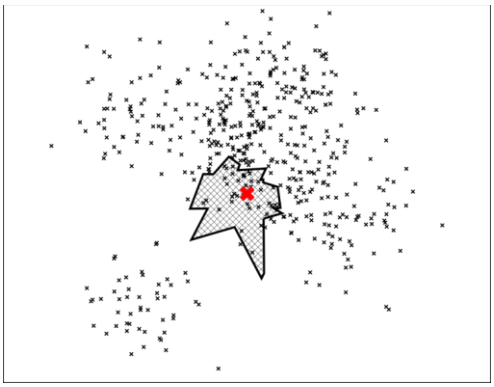


Figura 18. Vecindario de candidatos a vecinos más próximos en el espacio

Una vez obtenido el conjunto de candidatos, hay que calcular las distancias entre ellos, ordenándolas para obtener los más cercanos. Aunque si observamos con detenimiento, hay vecinos que se han escapado del área:

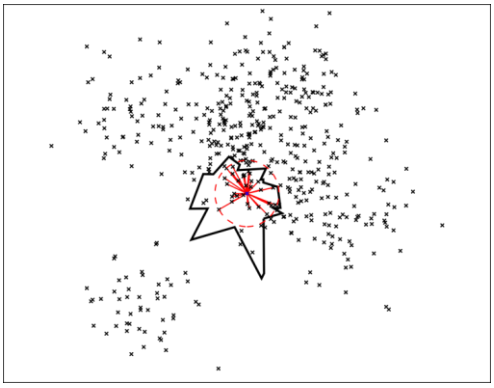


Figura 19. Radio que comprende los vecinos más próximos en el espacio

Por ello se denomina cálculo *aproximado* de vecinos, siempre existe la probabilidad de que queden excluidos vecinos realmente relevantes. La filosofía de este método yace en que, sacrificando un poco de precisión, somos capaces de obtener una gran mejora (en torno a 100 veces mejor) en el rendimiento cuanto más grande sea el conjunto de usuarios o ítems. Existe la posibilidad de aumentar la precisión, esto es, aumentando el número de árboles, pero perderíamos eficiencia en cuanto a memoria utilizada, por lo que hay que encontrar el compromiso entre estos factores.

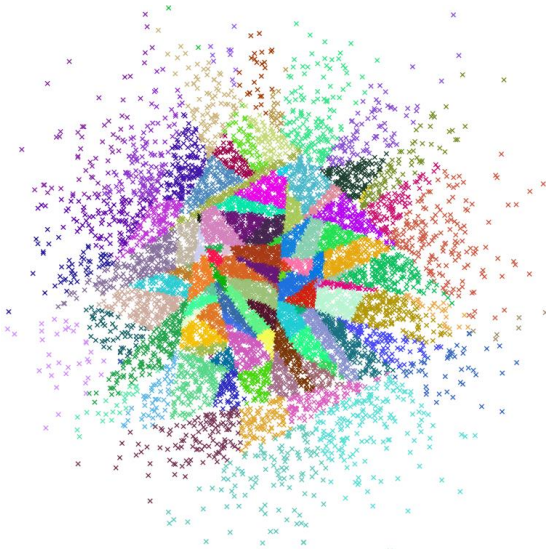


Figura 20. Mapa bidimensional de vecinos con particiones aleatorias

3.4.3 Annoy vs NMSLIB

La razón por la que en este proyecto se incluye la librería NMSLIB en vez de Annoy es debido a que, en mayo de 2016, NMSLIB le ha superado notablemente en términos de rendimiento y precisión.

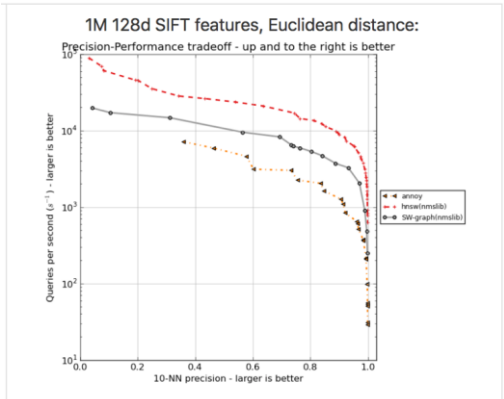


Figura 21. Annoy vs NMSLIB

3.4.4 Apache Thrift y NMSLIB

El principal problema del uso de la librería NMSLIB es que está escrita en el lenguaje C++, mientras que el proyecto es Java, lo que requiere de algún método para compatibilizar ambos lenguajes.

Mediante Apache Thrift, un framework que permite la creación y definición de servicios en múltiples lenguajes, usando RPC (*Remote Procedure Call*) y generación de código, es posible conectar aplicaciones escritas en multitud de lenguajes diferentes.

De esta manera, se instancia un servidor escrito en C++ con la librería NMSLIB y un cliente en Java que se conectará al servidor para ejecutar los métodos necesarios de manera remota (RPC). En este caso el servidor realizará el cálculo de vecinos devolviendo una lista de usuarios o ítems al cliente de manera totalmente transparente gracias a Thrift.

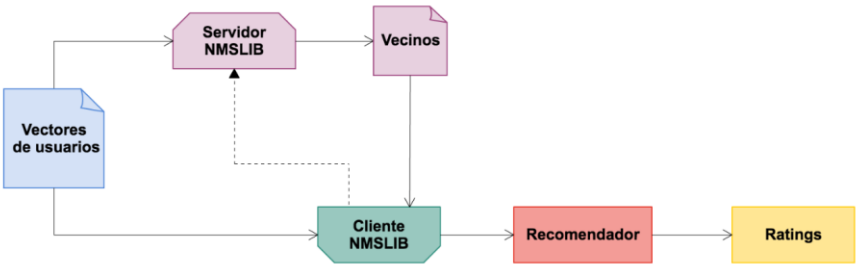


Figura 22. Esquema de empleo de NMSLIB

Una vez obtenidos los vecinos usando NMSLIB, son tratados por el programa cliente exportándolos a un fichero, el cual será la entrada del recomendador para la generación de recomendaciones.

4. Integración, pruebas y resultados

4.1 Pruebas unitarias

Para comprobar el correcto funcionamiento de los diferentes algoritmos implementados se han hecho test JUnit tanto para las similitudes (coseno, Jaccard, Pearson) como para las clases más generales (NMSLIB, *MyRecommender*…), además de las pruebas manuales habituales durante el desarrollo de todo el proyecto.

4.2 Datasets utilizados

Para la ejecución del sistema completo se ha utilizado un único dataset (Movielens), pues se ha preferido profundizar en el análisis de las variantes del algoritmo KNN que aumentar el número de datasets, ya que esta faceta es fácilmente extensible en el futuro. Movielens tiene un tamaño de 100.000 ratings (943 usuarios y 1682 ítems). El formato de las recomendaciones consta de cuatro campos: id de usuario, id de ítem, rating (en el intervalo [1,5]) y la fecha cuando se puntuó el artículo; en las pruebas no se tendrá en cuenta este último valor.

4.3 Resultados

Aunque a simple vista los sistemas de recomendación que utilizan KNN para generar las recomendaciones tengan un solo parámetro configurable, esto es, el número de vecinos, lo cierto es que existe un amplio abanico de modificaciones posibles como se ha visto anteriormente (3.4). Teniendo en cuenta todas las combinaciones posibles, incluyendo cinco repeticiones por combinación para obtener resultados significativos, se han ejecutado en torno a 13.000 experimentos.

En las tablas 4-8 (anexo A.B) se recogen de forma completa los resultados obtenidos tras la ejecución de todas las configuraciones posibles del algoritmo con valores de 10 y 100 en los *cutoffs* de las métricas. Debido al gran número de muestras recogidas, en este apartado sólo se mostrarán las configuraciones y versiones de las métricas más relevantes. Se trabajará con un 80% de los datos en entrenamiento (*train*) y un 20% para test, todo ello repetido 5 veces (5 *folds* por lo tanto), de esta manera, los valores de las tablas son una media de todas las particiones.

El formato de las tablas es el que sigue: cada fila representa una variante cuyo nombre hace referencia a los parámetros utilizados. El nombre *ub_cosine_false_MC* indica que se ha realizado el método basado en usuario, con similitud por coseno, la cual no está normalizada (false) y aplicando *Mean-Centering* como normalización de rating. Otra variante como *ib_jaccard_th_0.3_true_Z* implica una recomendación basada en ítem utilizando la similitud de Jaccard con un threshold de 0.3 y normalizada aplicando *Z-score* posteriormente. En cada columna se incluyen las métricas de evaluación, diferenciando el número de ítems del ranking a considerar en cada una (*cutoff*); de esta manera, prec@10 es la precisión en los 10 ítems con mayor importancia. Para cada métrica se incluyen cinco

valores de vecindario (10, 20, 40, 60 y 100) lo que nos permite analizar cómo se comportan los algoritmos cuando se utiliza un número diferente de vecinos (cuantos menos vecinos, más cercanos al usuario serán todos, pero se podrán generar menos posibles recomendaciones).

Los valores en todas las tablas se muestran en gradiente de color para su fácil distinción visual, siendo el color verde la mayor precisión y roja la peor, resaltando en una fuente de color blanco los valores máximos.

Promedio de v	ndcg@10					prec@10					recall@10				
Etiquetas de fila	10	20	40	60	100	10	20	40	60	100	10	20	40	60	100
NMSLIB_l1	0,118	0,130	0,138	0,142	0,146	0,141	0,153	0,162	0,166	0,171	0,077	0,086	0,095	0,100	0,103
NMSLIB_l2	0,116	0,127	0,135	0,141	0,145	0,139	0,152	0,160	0,167	0,171	0,075	0,083	0,092	0,097	0,101
ub_cosine_false_MC	0,279	0,291	0,295	0,295	0,297	0,280	0,286	0,290	0,288	0,298	0,144	0,149	0,151	0,152	0,149
ub_cosine_false_STD	0,323	0,341	0,348	0,347	0,349	0,344	0,357	0,363	0,362	0,374	0,194	0,202	0,205	0,204	0,198
ub_cosine_false_Z	0,278	0,291	0,294	0,300	0,295	0,278	0,286	0,289	0,299	0,294	0,143	0,149	0,150	0,148	0,146
ub_cosine_th_0.3_false_MC	0,263	0,273	0,277	0,276	0,281	0,264	0,271	0,274	0,274	0,285	0,135	0,138	0,141	0,140	0,138
ub_cosine_th_0.3_false_STD	0,305	0,319	0,325	0,325	0,330	0,326	0,338	0,344	0,343	0,357	0,182	0,189	0,192	0,192	0,187
ub_cosine_th_0.3_false_Z	0,261	0,273	0,276	0,281	0,280	0,263	0,270	0,274	0,286	0,284	0,134	0,139	0,139	0,137	0,137
ub_cosine_th_0.5_false_MC	0,058	0,059	0,059	0,059	0,057	0,056	0,056	0,057	0,057	0,057	0,039	0,039	0,039	0,039	0,037
ub_cosine_th_0.5_false_STD	0,065	0,065	0,065	0,065	0,064	0,064	0,064	0,064	0,064	0,064	0,045	0,045	0,045	0,045	0,044
ub_cosine_th_0.5_false_Z	0,059	0,059	0,059	0,058	0,058	0,056	0,057	0,057	0,057	0,057	0,039	0,039	0,039	0,038	0,038
ub_cosine_true_MC	0,040	0,019	0,006	0,003	0,001	0,048	0,023	0,009	0,005	0,003	0,033	0,015	0,005	0,002	0,001
ub_cosine_true_STD	0,032	0,015	0,004	0,002	0,001	0,038	0,018	0,006	0,003	0,002	0,030	0,014	0,004	0,001	0,001
ub_cosine_true_Z	0,042	0,020	0,006	0,003	0,001	0,050	0,024	0,009	0,004	0,002	0,034	0,016	0,005	0,002	0,001
ub_jaccard_false_MC	0,276	0,290	0,297	0,296	0,298	0,275	0,287	0,293	0,292	0,300	0,142	0,148	0,153	0,153	0,150
ub_jaccard_false_STD	0,325	0,341	0,347	0,346	0,346	0,345	0,359	0,363	0,362	0,373	0,194	0,203	0,206	0,205	0,199
ub_jaccard_false_Z	0,277	0,290	0,296	0,302	0,297	0,274	0,286	0,292	0,304	0,298	0,142	0,147	0,151	0,151	0,148
ub_jaccard_th_0.3_false_MC	0,085	0,086	0,086	0,086	0,084	0,084	0,085	0,085	0,085	0,086	0,051	0,051	0,051	0,051	0,049
ub_jaccard_th_0.3_false_STD	0,096	0,098	0,098	0,098	0,095	0,098	0,099	0,099	0,099	0,100	0,064	0,064	0,064	0,064	0,061
ub_jaccard_th_0.3_false_Z	0,085	0,086	0,086	0,084	0,084	0,084	0,085	0,085	0,086	0,086	0,051	0,051	0,051	0,049	0,049
ub_jaccard_th_0.5_false_MC	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001
ub_jaccard_th_0.5_false_STD	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001
ub_jaccard_th_0.5_false_Z	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001
ub_pearsoncn_false_MC	0,043	0,052	0,065	0,076	0,099	0,052	0,060	0,073	0,083	0,108	0,024	0,028	0,032	0,035	0,042
ub_pearsoncn_false_STD	0,053	0,064	0,077	0,085	0,104	0,069	0,081	0,096	0,106	0,129	0,032	0,039	0,047	0,051	0,058
ub_pearsoncn_false_Z	0,044	0,053	0,065	0,080	0,099	0,054	0,061	0,073	0,087	0,107	0,024	0,028	0,032	0,035	0,042
ub_pearson_false_MC	0,035	0,034	0,038	0,043	0,103	0,056	0,057	0,061	0,067	0,130	0,029	0,031	0,034	0,035	0,051
ub_pearson_false_STD	0,013	0,009	0,007	0,013	0,075	0,025	0,018	0,015	0,020	0,093	0,009	0,006	0,004	0,005	0,022
ub_pearson_false_Z	0,037	0,036	0,039	0,046	0,103	0,058	0,060	0,065	0,072	0,130	0,030	0,032	0,035	0,037	0,052
ub_pearson_th_0_false_MC	0,214	0,232	0,239	0,241	0,247	0,216	0,230	0,234	0,235	0,246	0,097	0,103	0,105	0,105	0,102
ub_pearson_th_0_false_STD	0,241	0,268	0,286	0,292	0,302	0,260	0,284	0,298	0,302	0,319	0,131	0,143	0,155	0,157	0,157
ub_pearson_th_0_false_Z	0,213	0,230	0,238	0,245	0,244	0,214	0,227	0,233	0,244	0,244	0,095	0,101	0,103	0,101	0,100

Tabla 1. Métricas de precisión para KNN basado en usuario

De forma particular, para las recomendaciones basadas en usuario se puede comprobar que las similitudes que mejor funcionan son coseno y Jaccard, siendo muy interesante a su vez los resultados ofrecidos por NMSLIB, los cuales están por encima de gran parte de las variantes.

Si indagamos un poco más en las variantes, encontramos bastantes malos resultados a la hora de normalizar las similitudes. Comparando *ub_cosine_false_STD* con *ub_cosine_true_STD* (que normaliza la similitud), se observa cómo la precisión en todas las métricas cae en un factor de diez veces, esto puede deberse a que normalizar implica igualar valores de predicción al intervalo [1,5] (rango de los ratings) independientemente del número de vecinos que se hayan visto involucrados en dicha predicción, perdiendo de

esta manera el concepto de *confianza en la predicción* que sí se tiene en la versión opuesta, ya que valores más altos están asociados a predicciones donde muchos vecinos han aportado su ratings. Esta característica se cumple en todos los casos que normalizan la similitud, por este motivo, se ha decidido obviar dichas variantes.

El siguiente factor a tener en cuenta es el threshold que se aplica a la similitud. Teniendo en cuenta el coseno, no se observa apenas diferencia entre cuando se aplica un threshold con valor de 0.3 y cuando no se aplica, esto es debido a que prácticamente la totalidad de las similitudes están por encima de ese valor tanto en 10, 20, 40, 60 y 100 vecinos. No sucede lo mismo cuando se aplica un threshold de 0.5, ya que existen muchas similitudes con valor mayor que 0.3 pero menor que 0.5 que son eliminadas en estas combinaciones. Esto implica que los vecinos con los que se opera son más fiables, pero hay muy pocos, por ese motivo los valores en las métricas son idénticos, ya que el vecindario es tan pequeño que es insuficiente para ofrecer una cobertura mínima para la recomendación.

Es interesante la diferencia que existe entre la versión estándar de Pearson y con un threshold de 0, pues en cualquier otra similitud serían iguales (ya que tanto coseno como Jaccard son siempre positivos), pero Pearson puede devolver valores negativos. En este caso, al no tenerlos en cuenta se eliminan usuarios con gustos opuestos. Es muy clara esta diferencia: mientras que la versión sin threshold arroja resultados realmente malos, la otra es realmente competitiva, posicionándose como tercer mejor método.

En cuanto a la normalización de ratings, parece que se cumple de nuevo la característica observada anteriormente, el método estándar (*STD*), es decir, no normalizar, ofrece ligeramente mejores resultados que las otras versiones (*MC* y *Z*), las cuales están muy a la par.

ib_cosine_false_MC	0,162	0,146	0,126	0,111	0,091	0,167	0,156	0,143	0,132	0,119	0,100	0,095	0,091	0,082	0,074
ib_cosine_false_STD	0,288	0,284	0,278	0,274	0,278	0,300	0,295	0,284	0,279	0,279	0,198	0,195	0,187	0,183	0,182
ib_cosine_false_Z	0,166	0,149	0,128	0,112	0,092	0,171	0,159	0,145	0,134	0,121	0,103	0,097	0,092	0,082	0,075
ib_cosine_th_0.3_false_MC	0,166	0,150	0,131	0,117	0,102	0,171	0,160	0,148	0,138	0,129	0,105	0,097	0,090	0,083	0,079
ib_cosine_th_0.3_false_STD	0,285	0,278	0,267	0,264	0,270	0,298	0,290	0,275	0,269	0,272	0,195	0,189	0,177	0,171	0,176
ib_cosine_th_0.3_false_Z	0,168	0,152	0,132	0,117	0,102	0,174	0,162	0,149	0,139	0,130	0,106	0,099	0,091	0,084	0,079
ib_cosine_th_0.5_false_MC	0,175	0,164	0,155	0,153	0,153	0,180	0,170	0,164	0,163	0,163	0,097	0,089	0,085	0,085	0,085
ib_cosine_th_0.5_false_STD	0,211	0,208	0,208	0,208	0,208	0,232	0,227	0,223	0,224	0,224	0,124	0,121	0,118	0,119	0,119
ib_cosine_th_0.5_false_Z	0,176	0,165	0,156	0,154	0,154	0,182	0,171	0,165	0,164	0,164	0,098	0,090	0,085	0,085	0,085
ib_jaccard_false_MC	0,162	0,146	0,125	0,114	0,094	0,167	0,155	0,140	0,132	0,120	0,095	0,089	0,079	0,075	0,069
ib_jaccard_false_STD	0,273	0,265	0,255	0,253	0,253	0,292	0,285	0,272	0,269	0,268	0,191	0,187	0,180	0,177	0,179
ib_jaccard_false_Z	0,169	0,152	0,129	0,118	0,097	0,173	0,160	0,144	0,135	0,123	0,099	0,092	0,081	0,077	0,071
ib_jaccard_th_0.3_false_MC	0,181	0,167	0,156	0,153	0,153	0,184	0,173	0,165	0,163	0,163	0,106	0,097	0,091	0,091	0,091
ib_jaccard_th_0.3_false_STD	0,224	0,213	0,211	0,212	0,212	0,244	0,230	0,225	0,226	0,226	0,138	0,130	0,127	0,128	0,128
ib_jaccard_th_0.3_false_Z	0,183	0,168	0,157	0,154	0,154	0,185	0,174	0,166	0,165	0,164	0,107	0,097	0,092	0,092	0,092
ib_jaccard_th_0.5_false_MC	0,054	0,054	0,054	0,054	0,054	0,033	0,033	0,033	0,033	0,033	0,017	0,017	0,017	0,017	0,017
ib_jaccard_th_0.5_false_STD	0,053	0,053	0,053	0,053	0,053	0,033	0,033	0,033	0,033	0,033	0,016	0,016	0,016	0,016	0,016
ib_jaccard_th_0.5_false_Z	0,054	0,054	0,054	0,054	0,054	0,033	0,033	0,033	0,033	0,033	0,017	0,017	0,017	0,017	0,017
ib_pearsoncn_false_MC	0,001	0,002	0,001	0,001	0,000	0,003	0,005	0,003	0,002	0,001	0,001	0,001	0,001	0,001	0,000
ib_pearsoncn_false_STD	0,001	0,003	0,002	0,001	0,001	0,003	0,006	0,004	0,002	0,001	0,001	0,002	0,001	0,001	0,001
ib_pearsoncn_false_Z	0,001	0,002	0,001	0,001	0,000	0,003	0,005	0,003	0,002	0,001	0,001	0,001	0,001	0,001	0,000
ib_pearson_false_MC	0,001	0,001	0,001	0,002	0,002	0,002	0,002	0,002	0,003	0,003	0,001	0,001	0,001	0,002	0,002
ib_pearson_false_STD	0,001	0,001	0,002	0,002	0,005	0,002	0,003	0,004	0,005	0,010	0,001	0,001	0,001	0,002	0,004
ib_pearson_false_Z	0,001	0,001	0,001	0,002	0,002	0,002	0,002	0,002	0,003	0,003	0,001	0,001	0,001	0,002	0,002
ib_pearson_th_0_false_MC	0,141	0,118	0,092	0,079	0,062	0,139	0,123	0,105	0,095	0,080	0,078	0,069	0,058	0,054	0,043
ib_pearson_th_0_false_STD	0,223	0,214	0,200	0,193	0,178	0,249	0,242	0,230	0,224	0,208	0,154	0,148	0,137	0,134	0,123
ib_pearson_th_0_false_Z	0,141	0,118	0,092	0,079	0,062	0,139	0,123	0,105	0,095	0,080	0,078	0,069	0,058	0,054	0,043

Tabla 2. Métricas de precisión para KNN basado en ítem

Para las recomendaciones basadas en ítems, no se ha ejecutado la librería NMSLIB por falta de tiempo. Como se puede comprobar, tenemos resultados algo diferentes a los anteriores. Por una parte, observamos que la precisión no llega a ser tan alta como en UB, también difiere el ranking de configuraciones, donde coseno en este caso siempre es mejor que Jaccard y Pearson, aunque por diferencias muy pequeñas. Como ya se hizo en UB, se omiten las variantes con normalizaciones de similitud, ya que en ningún caso superan a las que no se normalizan.

En el caso del threshold aplicado a coseno, se ha reducido la diferencia notablemente para los valores 0.3 y 0.5. Lo mismo pasa con Jaccard para valores iguales a 0 y 0.3. Podríamos afirmar que existen pocos ítems con similitudes inferiores a 0.5 y 0.3 entre los más relevantes respectivamente, debido a esto, la intersección de ambos vecindarios es muy parecida arrojando valores similares en las métricas. Sin embargo, la versión sin threshold sigue dando resultados muy poco precisos.

A diferencia con el método basado en usuarios, los resultados varían notablemente al aplicar una normalización u otra al rating. Claramente, lo que más precisión aporta es no normalizar (*STD*) frente a las otras dos, independientemente de la métrica de similitud o el uso del threshold.

Promedio de HitRate s de

	pearson_false_STD					jaccard_false_STD					NMSLIB_l1					NMSLIB_l2				
Rótulos de fila	10	20	40	60	100	10	20	40	60	100	10	20	40	60	100	10	20	40	60	100
cosine_false_STD	0.50	1.04	2.00	3.32	8.94	76.14	78.41	81.47	83.28	85.66	11.24	12.15	13.90	15.45	18.33	15.69	16.62	18.18	19.80	22.17
pearson_false_STD						0.47	1.10	2.10	3.47	9.11	0.45	1.41	2.69	4.28	8.12	0.38	1.27	2.65	4.24	8.61
jaccard_false_STD											10.29	11.16	13.36	14.85	18.31	14.35	15.55	17.73	19.34	22.35

Tabla 3**. Grado de semejanza de vecinos entre similitudes.**

La figura de arriba muestra la intersección de vecindarios entre distintas similitudes, con el objetivo de saber si están relacionadas entre ellas y en qué grado. Todas las versiones han sido calculadas sin normalización.

Como se puede observar, la semejanza entre las parejas coseno - Pearson, Pearson - Jaccard, Pearson - NMSLIB_l1 y Pearson - NMSLIB_l2 es muy baja, lo que implica que poseen muy pocos vecinos en común. Sin embargo, coseno y Jaccard tienen una intersección mucho mayor, llegando hasta un 85% de semejanza en un vecindario de tamaño 100. Esto puede se ve reflejado en las buenas precisiones que se obtienen en ambos métodos, por lo que podríamos concluir que esos vecinos comunes tienen un grado alto de relevancia.

Por otra parte, para los conjuntos de vecinos calculados de manera aproximada (NMSLIB) vemos que tienen poca semejanza con los métodos tradicionales con similitud, quizá este hecho justifique las precisiones algo más bajas que coseno y Jaccard, aunque es un suceso que no podemos afirmar, pues el hecho de que compartan pocos vecinos no implica que un método sea peor, ya que éstos son obtenidos y representados de formas distintas, o bien NMSLIB escoge vecinos relevantes distintos a coseno, o los únicos acertados son los que comparten. Puede darse el caso que con semejanzas tan bajas como las que obtenemos para este dataset, ambos métodos generen recomendaciones acertadas.

No se debe olvidar que NMSLIB realiza un cálculo aproximado con el fin de ganar eficiencia frente a precisión (en torno a 1000 veces más eficiente que métodos por similitud), sería muy interesante comparar los rendimientos de todos estos métodos para poder saber en qué casos interesa utilizar uno u otro.

5. Conclusiones y trabajo futuro

Ya ha quedado claro que los sistemas de recomendación son realmente útiles, tanto para usuarios como para organizaciones comerciales, en un mundo donde la cantidad de datos no para de crecer, llegando a un punto donde es difícil su manejo. Un sistema de recomendación ofrece un gran abanico de posibilidades, desde incrementar el número de ventas o clientes, hasta proporcionar una mejor experiencia de usuario, llegando al punto de no preocuparse de buscar entre millones de artículos, teniendo al alcance recomendaciones totalmente personalizadas.

Durante este trabajo se han ido desarrollando las facetas más básicas de los sistemas de recomendación, profundizando a su vez en el algoritmo de vecinos cercanos e incluyendo métodos novedosos como la funcionalidad ofrecida por NMSLIB, la cual se ha observado que funciona relativamente bien en comparación a coseno, como similitud más precisa.

Ha sido muy interesante trabajar con el nuevo método del cálculo aproximado de vecinos, que presumiblemente puede ofrecer buenos resultados aumentando en gran medida la eficiencia. Este es precisamente el aspecto principal a trabajar en un futuro, comparando tanto con las alternativas tradicionales como factorización de matrices junto a más datasets que el del presente proyecto. Otra posible meta sería comprobar cómo se comporta el recomendador con las parametrizaciones evaluadas para diferentes datos en proporción al ratio de usuarios/ítems, al tamaño del dataset o a una matriz de datos más completa (es decir, que cada usuario haya puntuado la mayor parte de los ítems). También sería interesante evaluar el recomendador con métricas de novedad y diversidad para ver principalmente cómo rinden las distintas variantes implementadas con el fin de esclarecer que métodos se comportan mejor en cada situación. El mismo proceso de análisis llevado a cabo en este proyecto se podría extrapolar a otras familias de algoritmos, pudiendo encontrar de esta manera nuevas configuraciones más interesantes que las utilizadas en la actualidad.

Queda pendiente el tratamiento de datos, actualmente las particiones de entrenamiento y test vienen incluidas como ficheros en el propio dataset. Esto puede llegar a ser un factor limitante, por lo que la idea es poder tomar el conjunto de datos completo y realizar las particiones necesarias para hacer una buena cobertura en la recomendación. Debido a la programación modular y el diseño genérico, sería fácilmente extensible. Otra alternativa sería la inclusión de una librería externa encargada de esta funcionalidad.

Por último, la URL del repositorio donde se encuentra el código fuente del sistema desarrollado es la siguiente: https://github.com/AlejandroGil/TFG

Referencias

[1] A. Bellogín, Recommender System performance evaluation and prediction: An information retrieval perspective, Thesis, October 2012, pp 17-35.

[2] E. Bernhardsson, Algorithms and data structures, September 2015.

[3] E. Bernhardsson, Curse of dimensionality, September 2015.

[4] E. Bernhardsson, Nearest neighbors and vector models, September 2015.

[5] F. Ricci, L. Rokach, B. Shapira, Paul B. Kantor (eds.). Recommender Systems Handbook, 1st edition, Springer, 2011, Chapter 1. Introduction to Recommender Systems Handbook.

[6] F. Ricci, L. Rokach, B. Shapira, Paul B. Kantor (eds.). Recommender Systems Handbook, 1st edition, Springer, 2011, Chapter 3. Content-based Recommender Systems: State of the Art and Trends.

[7] F. Ricci, L. Rokach, B. Shapira, Paul B. Kantor (eds.). Recommender Systems Handbook, 1st edition, Springer, 2011, Chapter 4. A Comprehensive Survey of Neighborhood-based Recommendation Methods.

[8] F. Ricci, L. Rokach, B. Shapira, Paul B. Kantor (eds.). Recommender Systems Handbook, 1st edition, Springer, 2011, Chapter 5.3. Matrix factorization models.

[9] F. Ricci, L. Rokach, B. Shapira, Paul B. Kantor (eds.). Recommender Systems Handbook, 1st edition, Springer, 2011, Chapter 8. Evaluating Recommendation Systems.

[10] P. Sánchez, Estudio y aplicación de algoritmos y estructuras de datos a los Sistemas de Recomendación, Trabajo Fin de Grado, Escuela Politécnica Superior, Universidad Autónoma de Madrid, May 2016.

[11] Premio netflix. http://www.netflixprize.com/

[12] S.Marina, Suite de algoritmos de recomendación en aplicaciones reales, Trabajo Fin de Grado, Escuela Politécnica Superior, Universidad Autónoma de Madrid, May 2014.

Comentado [ABK2]: No está en orden

Comentado [AGH3R2]: Están ordenados por nombre, hay que ordenar por apellido?

Comentado [AGH4R2]:

Glosario

- **Annoy**: Approximate nearest neighbours. Librería que calcula vecinos de manera aproximada.
- **Apache Thrift**: Framework
- **Dataset**: Conjunto de datos analizados
- **DCG**: Discounted cumulative gain. Métrica de evaluación.
- **Framework**: estructura que sirve como base para construir una extensión de ella o aumentar la funcionalidad.
- **iDCG**: ideal DCG. Métrica de evaluación.
- **kNN**: k nearest neighbours. Algoritmo que utiliza los k vecinos más cercanos.
- **MAE**: Mean absolute error. Métrica de evaluación.
- **NDCG**: Normalised discounted cumulative gain
- **NMSLB**: Non-metric space library. Librería basada en Annoy para el cálculo aproximado de vecinos.
- **RankSys**: Ranking system. Librería base del proyecto, aporta funcionalidad de generación de recomendaciones y evaluación.
- **Rating**: Puntuación que un usuario da a un ítem.
- **Recall**: Métrica de evaluación.
- **RMSE**: Root mean squared error. Métrica de evaluación.
- **RPC**: Remote procedural call. Llamada a procedimiento remoto, usado para ejecutar código en otra máquina (Thrift)
- **Split**: Partición del dataset, dividiendo en training y test.
- **SR**: Sistema de recomendación.
- **Test**: Partición que contiene las recomendaciones a evaluar.
- **Tf-idf**: Term frequency-inverse document frequency. Métrica de frecuencia de términos.
- **Threshold**: Umbral por debajo del cual las similitudes son descartadas.
- **Training**: Partición del dataset utilizada para "enseñar" al sistema de recomendación.
- **UB**: user-based. Basado en usuario

Anexos

A Manual de instalación

Instalación de apache Thrift y librerías para usar nmslib

1) Desde el repositorio en GitHub de NMSLIB[1] se descarga el .zip del proyecto

   Dependiendo del repositorio disponible instalar una de las dos:

   ```
   sudo apt-get install libboost1.58-all-dev
   sudo apt-get install libboost1.54-all-dev
   ```

   Instalar las siguientes librerías:

   ```
   sudo apt-get install libgsl0-dev libeigen3-dev
   sudo apt-get install cmake
   sudo apt-get install libboost-dev libboost-test-dev libboost-program-options-dev libboost-system-dev libboost-filesystem-dev libevent-dev automake libtool flex bison pkg-config g++ libssl-dev libboost-thread-dev make
   ```

   En la carpeta similarity_search ejecutar: `cmake . ; make`

2) Descargar Apache Thrift[2] (actualmente la librería funciona con Thrift 0.9.2).
   Para instalar:

   ```
   ./configure
   sudo make
   sudo make install
   ```

   Verificar instalación: `thrift -version`

3) Una vez están instalados todos los paquetes necesarios se puede compilar el servidor (directorio query_server/cpp_client_server): `make`

4) Y ejecutar el servidor (p.ej):

   ```
   ./query_server –i inputFile -s l2 -m sw-graph -c NN=10,efConstruction=200,initIndexAttempts=1 -p 10000
   ```

   NOTA: si al ejecutar el servidor salta el error "./query_server: error while loading shared libraries: libthrift-0.9.2.so: cannot open shared object file: No such file or directory". Ejecutar `sudo ldconfig`

   Antes de ejecutar el cliente es necesario compilar. Ya que es un proyecto maven, necesitaremos descargar las dependencias necesarias (especificadas en el pom.xml) para ello ejecutamos: `mvn package`

   Antes de ejecutar el cliente necesitamos un fichero de datos:
   `export DATA_FILE=../../sample_data/final8_10K.txt`

Para ejecutar el cliente:

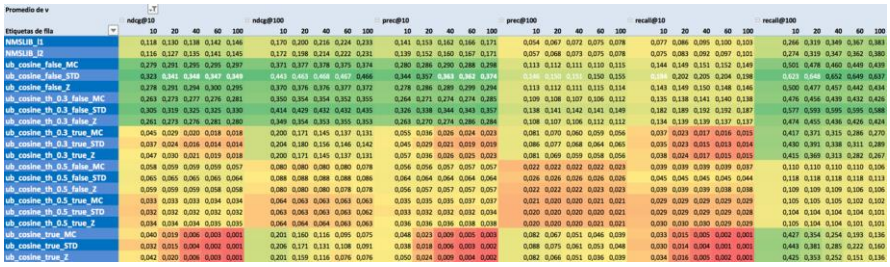
```
exec:java -Dexec.mainClass=thrift.Client -Dexec.args="-p 10000 -a localhost -k 500 -i
inputFile(usersRatingsVector) -out utFile(neighbours)"
```

Nos devolvería los 500 vecinos más cercanos al primer usuario

NOTA: tener inicializada la variable JAVA_HOME para ejecutar el cliente.

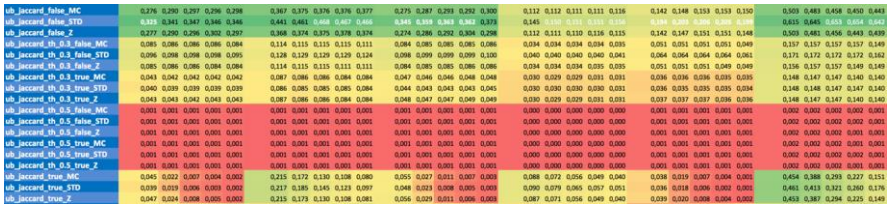
B Resultados métricas de ranking

A continuación se muestran los resultados completos de las métricas de ranking para el dataset Movielens. Si por cualquier motivo no se visualizan correctamente, también estarán accesibles en la URL del repositorio (https://github.com/AlejandroGil/TFG)



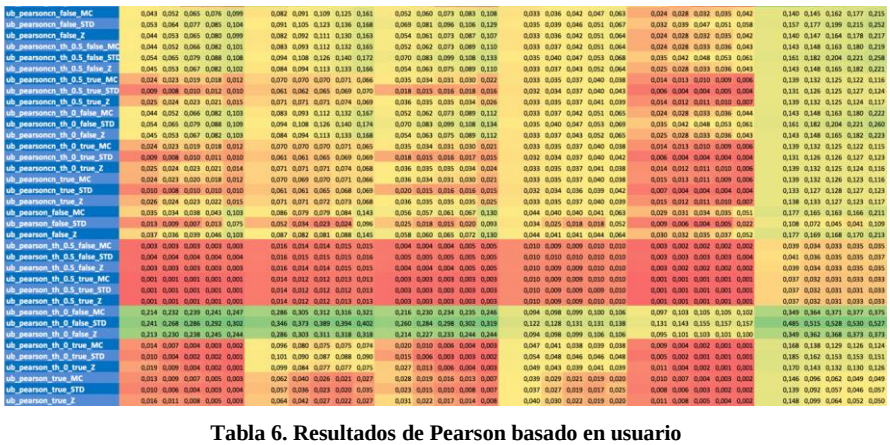
Promedio de v	ndcg@10					ndcg@100					prec@10					prec@100					recall@10					recall@100				
Etiquetas de fila	10	20	40	60	100	10	20	40	60	100	10	20	40	60	100	10	20	40	60	100	10	20	40	60	100	10	20	40	60	100
NMSLIB_l1	0,118	0,130	0,138	0,142	0,146	0,170	0,200	0,216	0,224	0,233	0,141	0,153	0,162	0,166	0,171	0,054	0,067	0,072	0,075	0,078	0,077	0,086	0,095	0,100	0,103	0,266	0,319	0,349	0,367	0,383
NMSLIB_l2	0,116	0,127	0,135	0,141	0,145	0,172	0,198	0,214	0,222	0,231	0,139	0,152	0,160	0,167	0,171	0,057	0,068	0,073	0,075	0,078	0,075	0,083	0,092	0,097	0,101	0,274	0,319	0,347	0,362	0,380
ub_cosine_false_MC	0,279	0,291	0,295	0,295	0,297	0,371	0,377	0,378	0,375	0,374	0,280	0,286	0,290	0,288	0,288	0,113	0,112	0,111	0,110	0,115	0,144	0,149	0,151	0,152	0,149	0,501	0,478	0,460	0,449	0,439
ub_cosine_false_STD	0,321	0,341	0,348	0,347	0,349	0,443	0,463	0,468	0,467	0,466	0,344	0,357	0,363	0,362	0,374	0,148	0,150	0,151	0,150	0,155	0,194	0,202	0,205	0,204	0,198	0,623	0,648	0,652	0,649	0,637
ub_cosine_false_Z	0,278	0,291	0,294	0,300	0,295	0,370	0,376	0,376	0,377	0,372	0,278	0,286	0,289	0,299	0,294	0,113	0,112	0,111	0,115	0,114	0,143	0,149	0,150	0,148	0,146	0,500	0,477	0,457	0,442	0,434
ub_cosine_th_0.3_false_MC	0,263	0,273	0,277	0,276	0,281	0,350	0,354	0,354	0,352	0,355	0,264	0,271	0,274	0,274	0,285	0,109	0,108	0,107	0,106	0,112	0,135	0,138	0,141	0,140	0,138	0,476	0,456	0,439	0,432	0,426
ub_cosine_th_0.3_false_STD	0,305	0,319	0,325	0,325	0,330	0,414	0,429	0,432	0,432	0,435	0,326	0,338	0,344	0,343	0,357	0,138	0,141	0,142	0,141	0,149	0,182	0,189	0,192	0,192	0,187	0,577	0,593	0,595	0,595	0,588
ub_cosine_th_0.3_false_Z	0,261	0,273	0,276	0,281	0,280	0,349	0,354	0,353	0,355	0,353	0,263	0,270	0,274	0,286	0,284	0,108	0,107	0,106	0,112	0,112	0,134	0,139	0,139	0,137	0,137	0,474	0,455	0,436	0,426	0,424
ub_cosine_th_0.3_true_MC	0,045	0,029	0,020	0,018	0,018	0,200	0,171	0,145	0,137	0,131	0,055	0,036	0,026	0,024	0,023	0,081	0,070	0,060	0,059	0,056	0,037	0,023	0,017	0,016	0,015	0,417	0,371	0,315	0,286	0,270
ub_cosine_th_0.3_true_STD	0,037	0,024	0,016	0,014	0,014	0,204	0,180	0,156	0,146	0,142	0,045	0,029	0,021	0,019	0,019	0,086	0,077	0,068	0,064	0,065	0,035	0,023	0,015	0,013	0,014	0,430	0,391	0,338	0,311	0,289
ub_cosine_th_0.3_true_Z	0,047	0,030	0,021	0,019	0,018	0,200	0,171	0,145	0,137	0,131	0,057	0,036	0,026	0,025	0,023	0,081	0,069	0,059	0,058	0,056	0,038	0,024	0,017	0,015	0,015	0,415	0,369	0,313	0,282	0,267
ub_cosine_th_0.5_false_MC	0,058	0,059	0,059	0,059	0,057	0,080	0,080	0,080	0,080	0,078	0,056	0,056	0,057	0,057	0,057	0,022	0,022	0,022	0,022	0,023	0,039	0,039	0,039	0,039	0,037	0,110	0,110	0,110	0,110	0,106
ub_cosine_th_0.5_false_STD	0,065	0,065	0,065	0,065	0,064	0,088	0,088	0,088	0,088	0,086	0,064	0,064	0,064	0,064	0,064	0,026	0,026	0,026	0,026	0,026	0,045	0,045	0,045	0,045	0,044	0,118	0,118	0,118	0,118	0,113
ub_cosine_th_0.5_false_Z	0,059	0,059	0,059	0,058	0,058	0,080	0,080	0,080	0,078	0,078	0,056	0,057	0,057	0,057	0,057	0,022	0,022	0,022	0,023	0,023	0,039	0,039	0,039	0,038	0,038	0,109	0,109	0,109	0,106	0,106
ub_cosine_th_0.5_true_MC	0,033	0,033	0,033	0,034	0,034	0,064	0,063	0,063	0,063	0,063	0,035	0,035	0,035	0,037	0,037	0,021	0,020	0,020	0,021	0,021	0,029	0,029	0,029	0,029	0,029	0,105	0,105	0,105	0,102	0,102
ub_cosine_th_0.5_true_STD	0,032	0,032	0,032	0,032	0,032	0,063	0,063	0,063	0,063	0,062	0,033	0,032	0,032	0,032	0,034	0,020	0,020	0,020	0,020	0,021	0,029	0,029	0,029	0,029	0,028	0,104	0,104	0,104	0,104	0,101
ub_cosine_th_0.5_true_Z	0,034	0,034	0,034	0,035	0,035	0,064	0,064	0,064	0,063	0,063	0,036	0,036	0,036	0,038	0,038	0,020	0,020	0,020	0,021	0,021	0,030	0,030	0,030	0,029	0,029	0,105	0,104	0,104	0,101	0,101
ub_cosine_true_MC	0,040	0,019	0,006	0,003	0,001	0,201	0,160	0,116	0,095	0,075	0,048	0,023	0,009	0,005	0,003	0,082	0,067	0,051	0,046	0,039	0,033	0,015	0,005	0,002	0,001	0,427	0,354	0,254	0,193	0,136
ub_cosine_true_STD	0,032	0,015	0,004	0,002	0,001	0,206	0,171	0,131	0,108	0,091	0,038	0,018	0,006	0,003	0,002	0,088	0,075	0,061	0,053	0,048	0,030	0,014	0,004	0,001	0,001	0,443	0,381	0,285	0,222	0,160
ub_cosine_true_Z	0,042	0,020	0,008	0,003	0,001	0,201	0,159	0,116	0,076	0,076	0,050	0,024	0,009	0,004	0,002	0,082	0,066	0,051	0,036	0,039	0,034	0,016	0,005	0,002	0,001	0,425	0,353	0,252	0,151	0,136

Tabla 4. Resultados de NMSLIB y coseno basados en usuario con cutoffs 10 y 100



ub_jaccard_false_MC	0,276	0,290	0,297	0,296	0,298	0,367	0,375	0,376	0,376	0,377	0,275	0,287	0,293	0,292	0,300	0,112	0,112	0,111	0,111	0,116	0,142	0,148	0,153	0,153	0,150	0,503	0,483	0,458	0,450	0,443
ub_jaccard_false_STD	0,321	0,341	0,347	0,346	0,346	0,441	0,461	0,468	0,467	0,466	0,345	0,359	0,363	0,362	0,373	0,145	0,150	0,151	0,151	0,156	0,194	0,203	0,206	0,205	0,198	0,615	0,645	0,653	0,654	0,642
ub_jaccard_false_Z	0,277	0,290	0,296	0,302	0,297	0,368	0,374	0,375	0,378	0,374	0,274	0,286	0,292	0,304	0,298	0,112	0,111	0,110	0,116	0,115	0,142	0,147	0,151	0,151	0,148	0,503	0,481	0,456	0,443	0,439
ub_jaccard_th_0.3_false_MC	0,085	0,086	0,086	0,086	0,084	0,114	0,115	0,115	0,115	0,111	0,084	0,085	0,085	0,085	0,086	0,034	0,034	0,034	0,034	0,035	0,051	0,051	0,051	0,051	0,049	0,157	0,157	0,157	0,157	0,149
ub_jaccard_th_0.3_false_STD	0,096	0,098	0,098	0,098	0,095	0,128	0,129	0,129	0,129	0,124	0,098	0,099	0,099	0,099	0,100	0,040	0,040	0,040	0,040	0,041	0,064	0,064	0,064	0,064	0,061	0,171	0,172	0,172	0,172	0,162
ub_jaccard_th_0.3_false_Z	0,085	0,086	0,086	0,084	0,084	0,114	0,115	0,115	0,111	0,111	0,084	0,085	0,085	0,086	0,086	0,034	0,034	0,034	0,035	0,035	0,051	0,051	0,051	0,049	0,049	0,156	0,157	0,157	0,149	0,149
ub_jaccard_th_0.3_true_MC	0,043	0,042	0,042	0,042	0,042	0,087	0,086	0,086	0,084	0,084	0,047	0,046	0,046	0,048	0,048	0,030	0,029	0,029	0,031	0,031	0,036	0,036	0,036	0,035	0,035	0,148	0,147	0,147	0,140	0,140
ub_jaccard_th_0.3_true_STD	0,040	0,039	0,039	0,039	0,039	0,086	0,085	0,085	0,085	0,084	0,044	0,043	0,043	0,043	0,045	0,030	0,030	0,030	0,030	0,031	0,036	0,035	0,035	0,035	0,034	0,148	0,148	0,147	0,147	0,140
ub_jaccard_th_0.3_true_Z	0,043	0,043	0,042	0,043	0,043	0,087	0,086	0,086	0,084	0,084	0,048	0,047	0,047	0,049	0,049	0,030	0,029	0,029	0,031	0,031	0,037	0,037	0,037	0,036	0,036	0,148	0,147	0,147	0,140	0,140
ub_jaccard_th_0.5_false_MC	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,000	0,000	0,000	0,000	0,000	0,001	0,001	0,001	0,001	0,001	0,002	0,002	0,002	0,002	0,001
ub_jaccard_th_0.5_false_STD	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,000	0,000	0,000	0,000	0,000	0,001	0,001	0,001	0,001	0,001	0,002	0,002	0,002	0,002	0,001
ub_jaccard_th_0.5_false_Z	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,000	0,000	0,000	0,000	0,000	0,001	0,001	0,001	0,001	0,001	0,002	0,002	0,002	0,001	0,001
ub_jaccard_th_0.5_true_MC	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,000	0,000	0,000	0,000	0,000	0,001	0,001	0,001	0,001	0,001	0,002	0,002	0,002	0,001	0,001
ub_jaccard_th_0.5_true_STD	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,000	0,000	0,000	0,000	0,000	0,001	0,001	0,001	0,001	0,001	0,002	0,002	0,002	0,002	0,001
ub_jaccard_th_0.5_true_Z	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,000	0,000	0,000	0,000	0,000	0,001	0,001	0,001	0,001	0,001	0,002	0,002	0,002	0,001	0,001
ub_jaccard_true_MC	0,045	0,022	0,007	0,004	0,002	0,215	0,172	0,130	0,108	0,080	0,055	0,027	0,011	0,007	0,003	0,088	0,072	0,056	0,049	0,040	0,038	0,019	0,007	0,004	0,001	0,454	0,388	0,293	0,227	0,151
ub_jaccard_true_STD	0,039	0,019	0,006	0,003	0,002	0,217	0,185	0,145	0,123	0,097	0,048	0,023	0,008	0,005	0,003	0,090	0,079	0,065	0,057	0,051	0,036	0,018	0,006	0,003	0,001	0,461	0,413	0,321	0,260	0,176
ub_jaccard_true_Z	0,047	0,024	0,008	0,005	0,002	0,215	0,173	0,130	0,108	0,081	0,056	0,029	0,011	0,006	0,003	0,087	0,071	0,056	0,049	0,040	0,039	0,020	0,008	0,004	0,002	0,453	0,387	0,294	0,225	0,149

Tabla 5. Resultados de Jaccard basado en usuario



ub_pearsoncn_false_MC	0,043	0,052	0,065	0,076	0,099	0,082	0,091	0,109	0,125	0,161	0,052	0,060	0,073	0,083	0,108	0,033	0,036	0,042	0,047	0,063	0,024	0,028	0,032	0,035	0,042	0,140	0,145	0,162	0,177	0,215
ub_pearsoncn_false_STD	0,053	0,064	0,077	0,085	0,104	0,091	0,105	0,123	0,136	0,168	0,069	0,081	0,096	0,106	0,129	0,035	0,046	0,051	0,057	0,067	0,032	0,039	0,047	0,051	0,058	0,157	0,177	0,199	0,215	0,252
ub_pearsoncn_false_Z	0,044	0,053	0,065	0,080	0,099	0,082	0,092	0,111	0,130	0,163	0,054	0,061	0,073	0,087	0,107	0,033	0,036	0,042	0,051	0,064	0,024	0,028	0,032	0,035	0,042	0,140	0,147	0,164	0,178	0,217
ub_pearsoncn_th_0.5_false_MC	0,044	0,052	0,066	0,082	0,101	0,083	0,093	0,112	0,132	0,165	0,052	0,062	0,073	0,089	0,110	0,033	0,037	0,042	0,051	0,064	0,024	0,028	0,033	0,036	0,043	0,143	0,148	0,163	0,180	0,219
ub_pearsoncn_th_0.5_false_STD	0,054	0,065	0,079	0,088	0,108	0,094	0,108	0,126	0,140	0,172	0,070	0,083	0,099	0,108	0,133	0,035	0,040	0,047	0,053	0,068	0,035	0,042	0,048	0,053	0,061	0,161	0,182	0,204	0,221	0,258
ub_pearsoncn_th_0.5_false_Z	0,045	0,053	0,067	0,082	0,102	0,084	0,094	0,113	0,133	0,166	0,054	0,063	0,075	0,089	0,110	0,033	0,037	0,043	0,052	0,064	0,025	0,028	0,033	0,036	0,043	0,143	0,148	0,165	0,182	0,221
ub_pearsoncn_th_0.5_true_MC	0,024	0,023	0,019	0,018	0,012	0,070	0,070	0,070	0,071	0,066	0,035	0,034	0,031	0,030	0,022	0,033	0,035	0,037	0,040	0,038	0,014	0,013	0,010	0,009	0,006	0,139	0,132	0,125	0,122	0,116
ub_pearsoncn_th_0.5_true_STD	0,009	0,008	0,010	0,012	0,010	0,061	0,062	0,065	0,069	0,070	0,018	0,015	0,016	0,018	0,016	0,032	0,034	0,037	0,040	0,043	0,006	0,004	0,004	0,005	0,004	0,131	0,126	0,125	0,127	0,124
ub_pearsoncn_th_0.5_true_Z	0,025	0,024	0,023	0,021	0,015	0,071	0,071	0,071	0,074	0,069	0,036	0,035	0,035	0,034	0,026	0,033	0,035	0,037	0,041	0,039	0,014	0,012	0,011	0,010	0,007	0,139	0,132	0,125	0,124	0,117
ub_pearsoncn_th_0_false_MC	0,044	0,052	0,066	0,082	0,103	0,083	0,093	0,112	0,132	0,167	0,052	0,062	0,073	0,089	0,112	0,033	0,037	0,042	0,051	0,065	0,024	0,028	0,033	0,036	0,044	0,143	0,148	0,163	0,180	0,222
ub_pearsoncn_th_0_false_STD	0,054	0,065	0,079	0,088	0,109	0,094	0,108	0,126	0,140	0,174	0,070	0,083	0,099	0,108	0,134	0,035	0,040	0,047	0,053	0,069	0,035	0,042	0,048	0,053	0,061	0,161	0,182	0,204	0,221	0,260
ub_pearsoncn_th_0_false_Z	0,045	0,053	0,067	0,082	0,103	0,084	0,094	0,113	0,133	0,168	0,054	0,063	0,075	0,089	0,112	0,033	0,037	0,043	0,052	0,065	0,025	0,028	0,033	0,036	0,043	0,143	0,148	0,165	0,182	0,223
ub_pearsoncn_th_0_true_MC	0,024	0,023	0,019	0,018	0,012	0,070	0,070	0,070	0,071	0,065	0,035	0,034	0,031	0,030	0,021	0,033	0,035	0,037	0,040	0,038	0,014	0,013	0,010	0,009	0,006	0,139	0,132	0,125	0,122	0,115
ub_pearsoncn_th_0_true_STD	0,009	0,008	0,010	0,011	0,010	0,061	0,061	0,065	0,069	0,069	0,018	0,015	0,016	0,017	0,015	0,032	0,034	0,037	0,040	0,042	0,006	0,004	0,004	0,004	0,004	0,131	0,126	0,126	0,127	0,123
ub_pearsoncn_th_0_true_Z	0,025	0,024	0,023	0,021	0,014	0,071	0,071	0,071	0,074	0,068	0,036	0,035	0,035	0,034	0,024	0,033	0,035	0,037	0,041	0,038	0,014	0,012	0,011	0,010	0,006	0,139	0,132	0,125	0,124	0,116
ub_pearsoncn_true_MC	0,024	0,023	0,020	0,018	0,012	0,070	0,069	0,070	0,071	0,066	0,036	0,034	0,031	0,030	0,021	0,033	0,035	0,037	0,040	0,038	0,015	0,013	0,011	0,009	0,006	0,139	0,132	0,126	0,123	0,116
ub_pearsoncn_true_STD	0,010	0,008	0,010	0,010	0,010	0,061	0,061	0,065	0,068	0,069	0,020	0,015	0,016	0,016	0,015	0,032	0,034	0,036	0,039	0,042	0,007	0,004	0,004	0,004	0,004	0,133	0,127	0,128	0,127	0,123
ub_pearsoncn_true_Z	0,026	0,024	0,023	0,022	0,015	0,071	0,071	0,072	0,073	0,068	0,036	0,035	0,035	0,035	0,025	0,033	0,035	0,037	0,040	0,039	0,015	0,012	0,011	0,010	0,007	0,138	0,133	0,127	0,123	0,117
ub_pearson_false_MC	0,035	0,034	0,038	0,043	0,103	0,086	0,079	0,079	0,084	0,143	0,056	0,057	0,061	0,067	0,130	0,044	0,040	0,040	0,041	0,063	0,029	0,031	0,034	0,035	0,051	0,177	0,165	0,163	0,166	0,211
ub_pearson_false_STD	0,013	0,009	0,007	0,013	0,075	0,052	0,034	0,023	0,024	0,096	0,025	0,018	0,015	0,020	0,093	0,034	0,025	0,018	0,018	0,052	0,009	0,006	0,004	0,005	0,022	0,108	0,072	0,045	0,041	0,109
ub_pearson_false_Z	0,037	0,036	0,039	0,046	0,103	0,087	0,082	0,081	0,088	0,145	0,058	0,060	0,065	0,072	0,130	0,044	0,041	0,041	0,044	0,064	0,030	0,032	0,035	0,037	0,052	0,177	0,169	0,168	0,170	0,213
ub_pearson_th_0.5_false_MC	0,003	0,003	0,003	0,003	0,003	0,016	0,014	0,014	0,015	0,015	0,004	0,004	0,004	0,005	0,005	0,010	0,009	0,009	0,010	0,010	0,003	0,002	0,002	0,002	0,002	0,039	0,034	0,033	0,035	0,035
ub_pearson_th_0.5_false_STD	0,004	0,004	0,004	0,004	0,004	0,016	0,015	0,015	0,015	0,016	0,005	0,005	0,005	0,005	0,005	0,010	0,010	0,010	0,010	0,010	0,003	0,003	0,003	0,003	0,004	0,041	0,036	0,035	0,035	0,037
ub_pearson_th_0.5_false_Z	0,003	0,003	0,003	0,003	0,003	0,016	0,014	0,014	0,015	0,015	0,004	0,004	0,004	0,005	0,005	0,010	0,009	0,009	0,010	0,010	0,003	0,002	0,002	0,002	0,002	0,039	0,034	0,033	0,035	0,035
ub_pearson_th_0.5_true_MC	0,001	0,001	0,001	0,001	0,001	0,014	0,012	0,012	0,013	0,013	0,003	0,003	0,003	0,003	0,003	0,010	0,009	0,009	0,010	0,010	0,001	0,001	0,001	0,001	0,001	0,037	0,032	0,031	0,033	0,033
ub_pearson_th_0.5_true_STD	0,001	0,001	0,001	0,001	0,001	0,014	0,012	0,012	0,012	0,013	0,003	0,003	0,003	0,003	0,003	0,010	0,009	0,009	0,010	0,010	0,001	0,001	0,001	0,001	0,001	0,037	0,032	0,031	0,033	0,033
ub_pearson_th_0.5_true_Z	0,001	0,001	0,001	0,001	0,001	0,014	0,012	0,012	0,013	0,013	0,003	0,003	0,003	0,003	0,003	0,010	0,009	0,009	0,010	0,010	0,001	0,001	0,001	0,001	0,001	0,037	0,032	0,031	0,033	0,033
ub_pearson_th_0_false_MC	0,214	0,232	0,239	0,241	0,247	0,286	0,305	0,312	0,316	0,321	0,216	0,230	0,234	0,235	0,246	0,094	0,098	0,099	0,100	0,106	0,097	0,103	0,105	0,105	0,102	0,349	0,364	0,371	0,377	0,375
ub_pearson_th_0_false_STD	0,241	0,268	0,286	0,292	0,302	0,346	0,373	0,389	0,394	0,402	0,260	0,284	0,298	0,302	0,319	0,122	0,128	0,131	0,131	0,138	0,131	0,143	0,155	0,157	0,157	0,485	0,515	0,528	0,530	0,527
ub_pearson_th_0_false_Z	0,213	0,230	0,238	0,245	0,244	0,286	0,303	0,311	0,318	0,318	0,214	0,227	0,233	0,244	0,244	0,094	0,098	0,099	0,106	0,106	0,095	0,101	0,103	0,103	0,100	0,349	0,362	0,368	0,373	0,373
ub_pearson_th_0_true_MC	0,014	0,027	0,004	0,003	0,002	0,096	0,080	0,075	0,075	0,074	0,020	0,010	0,006	0,004	0,003	0,047	0,041	0,038	0,039	0,038	0,009	0,004	0,002	0,001	0,001	0,168	0,138	0,129	0,126	0,124
ub_pearson_th_0_true_STD	0,010	0,004	0,002	0,002	0,001	0,101	0,090	0,087	0,088	0,090	0,015	0,006	0,003	0,003	0,002	0,054	0,048	0,046	0,046	0,048	0,005	0,002	0,001	0,001	0,001	0,185	0,162	0,153	0,153	0,151
ub_pearson_th_0_true_Z	0,019	0,009	0,004	0,002	0,001	0,099	0,084	0,077	0,077	0,075	0,027	0,013	0,006	0,004	0,003	0,049	0,043	0,039	0,041	0,039	0,011	0,004	0,002	0,001	0,001	0,170	0,143	0,132	0,130	0,126
ub_pearson_true_MC	0,013	0,009	0,007	0,005	0,003	0,062	0,040	0,026	0,021	0,027	0,028	0,019	0,016	0,013	0,007	0,039	0,029	0,021	0,019	0,020	0,010	0,007	0,004	0,003	0,002	0,146	0,096	0,062	0,049	0,049
ub_pearson_true_STD	0,010	0,006	0,004	0,003	0,004	0,057	0,036	0,023	0,020	0,035	0,023	0,015	0,010	0,008	0,007	0,037	0,027	0,019	0,017	0,025	0,008	0,006	0,003	0,002	0,002	0,139	0,092	0,057	0,046	0,057
ub_pearson_true_Z	0,016	0,011	0,008	0,005	0,003	0,064	0,042	0,027	0,022	0,027	0,031	0,022	0,017	0,014	0,008	0,040	0,030	0,022	0,019	0,020	0,011	0,008	0,005	0,004	0,002	0,148	0,099	0,064	0,052	0,050

Tabla 6. Resultados de Pearson basado en usuario

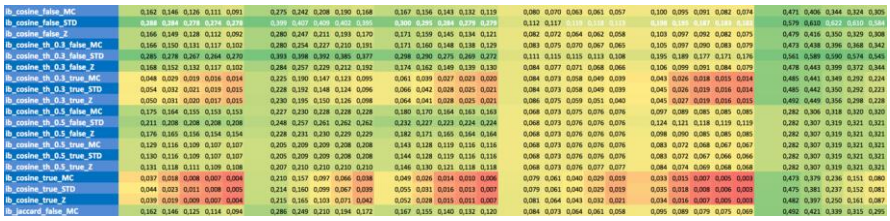


Tabla 7. Resultados de coseno basado en ítem

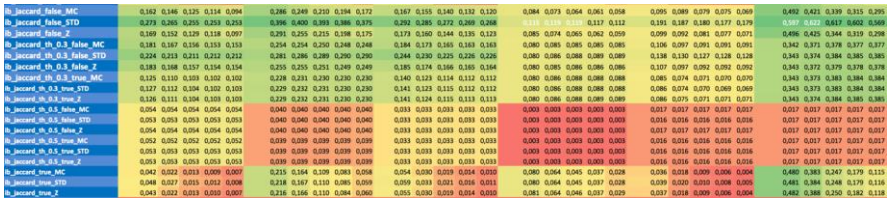


Tabla 8. Resultados de Jaccard basado en ítem

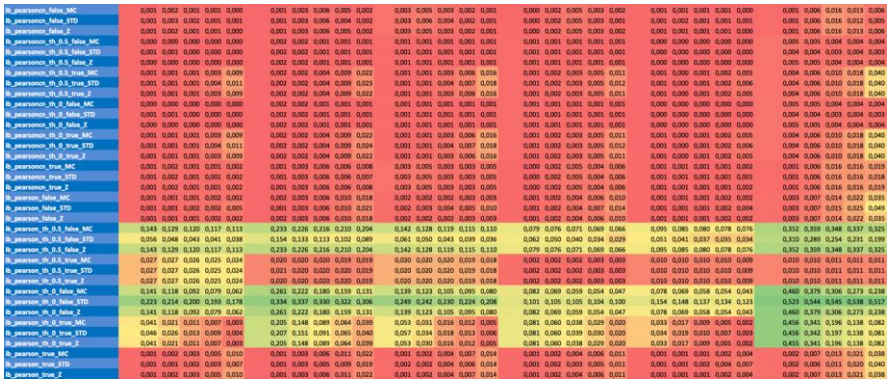


Tabla 9. Resultados de Pearson basado en ítem