Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

APRENDIZAJE ACTIVO EN SISTEMAS DE RECOMENDACIÓN

Autor: Adrián Pertejo Mangas
Tutor: Pablo Castells Azpilicueta

Junio 2018

APRENDIZAJE ACTIVO EN SISTEMAS DE RECOMENDACIÓN

Autor: Adrián Pertejo Mangas
Tutor: Pablo Castells Azpilicueta

Grupo de Recuperación de Información
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2018

Resumen

Resumen

En la actualidad, los Sistemas de Recomendación han adquirido una gran importancia. Grandes plataformas como Netflix o Amazon han contribuido a ello. Estos sistemas requieren de una cantidad masiva de datos para poder generar recomendaciones adecuadas a cada usuario, y es ahí donde entra en juego el Aprendizaje Activo, una rama del Aprendizaje Automático que se especializa en decidir qué opciones ofrecer al usuario, con el objetivo de optimizar el equilibrio entre el servicio prestado al usuario en un plazo inmediato, y la ganancia de información del sistema.

En este trabajo, se implementan estrategias de Aprendizaje Activo en el ámbito de la recomendación basada en filtrado colaborativo, y se evalúan simulando la interacción iterativa entre un sistema de recomendación y sus usuarios. Para ello, se han implementado diversas estrategias de recomendación con aprendizaje activo, y diferentes comportamientos que un usuario podría seguir a la hora de puntuar los objetos recomendados.

Para la mayor parte de las estrategias simples se ha utilizado el framework Ranksys, desarrollado por el grupo de investigación del tutor de este trabajo. Se han usado las estrategias simples más extendidas en el campo de la recomendación colaborativa: vecinos próximos, factorización de matrices, popularidad, etc.

Entre las estrategias implementadas se incluyen algoritmos adaptativos de recomendación, que seleccionan entre diferentes métodos dependiendo del estado temporal del sistema. También se implementan estrategias combinadas de recomendación, que combinan las puntuaciones de múltiples algoritmos simples de recomendación en un solo *ranking*.

Adicionalmente, se ha realizado un análisis de la eficacia de cada estrategia, probando diferentes configuraciones y tres conjuntos de datos. Finalmente, se ha sintetizado toda la información en las conclusiones del trabajo.

Palabras Clave

Aprendizaje Activo, Relevancia, Sistemas de Recomendación, Filtrado colaborativo

Abstract

Nowadays, recommender systems have reached a great importance. Huge platforms as Netflix or Amazon have contributed to it. These systems require a great amount of data in order to be able to generate appropiate recommendations to each user, and this is where Active Learning comes into play, a field of Machine Learning specialized in choosing the options the user should be shown, with the aim to optimize the balance between the service provided to the user and the information gain of the system.

In this job, several Active Learning strategies, delimited in the context of the collaborative filtering based recommendations, have been implemented, and they are evaluated simulating the iterative interaction existing between a recommender system and its users. To that end, several recommendation and active learning strategies have been implemented, as well as diverse possible user behaviours, simulating how they could react when a recommendation is given to them.

The framework Ranksys have been used for most of simple strategies, rather than use new and self-made implementations. This framework have been made by the investigation group of the tutor of this project, using the most extended ones in the field: k-nearest neighbours, matrix factorization, popularity, etc.

Among the implemented methods, several adaptative recommendation algorithms are included, which select among different strategies depending on the temporal state of the system. It is also implemented diverse combined recommendation techniques, which combine the scores of multiple simple recommendation algorithms in just one ranking.

Besides, an analysis of the effectiveness of each strategy have been made, trying different settings and three datasets. Finally, all the results have been synthesized in the conclusions of this work.

Key words

Active Learning, Relevance, Recommender Systems, Collaborative filtering

Agradecimientos

A mi familia y amigos, sobre todo a mi madre, por haberme permitido desarrollarme como persona y haber llegado hasta aquí.

Y en especial a Pablo, por haber trabajado y haberme ayudado tanto durante estos cuatro meses.

Índice general

Índice de Figuras

Índice de Tablas

1

Introducción

1.1. Motivación del proyecto

En el campo de la Inteligencia Artificial, (AI de *Artificial Intelligence* en inglés), una de las principales ramas es el Aprendizaje Automático, (ML, de *Machine Learning* en inglés). Esta disciplina busca desarrollar algoritmos capaces de generalizar comportamientos a partir de unos ejemplos previamente dados, también llamados patrones. En este paradigma, nos encontramos el principal problema que motiva este trabajo, la necesidad de obtener estos ejemplos que nos permitan aplicar los algoritmos de ML. La obtención de estos patrones iniciales conllevan a menudo un gran coste, ya sea computacional, económico, etc. Por ello, surge una nueva disciplina, en concreto, una subrama del Aprendizaje Automático, conocida como Aprendizaje Activo [16].

El Aprendizaje Activo es un caso especial de Aprendizaje Automático, donde se aplica un algoritmo capaz de, al mismo tiempo que resuelve el problema u objetivo principal (detectar objetos en imágenes [19], clasificar opiniones en Twitter [18], recomendación en general [7] [14]), obtener nueva información a partir de consultas realizadas al usuario, o a la fuente de información pertinente, con el objetivo de mejorar el modelo a medio y largo plazo. Es decir, tolera un pequeño sacrificio en la efectividad y precisión del modelo a corto plazo, a cambio de obtener nueva información relevante que permita al modelo realizar mejores predicciones en el futuro.

Aunque el paradigma del Aprendizaje Activo se puede aplicar a múltiples ramas del Aprendizaje Automático o al campo de la Minería de Información, este trabajo se centra en el Aprendizaje Activo aplicado a Sistemas de Recomendación de Filtrado Colaborativo. Estos sistemas tienen el objetivo de generar recomendaciones a unos usuarios, a partir de las preferencias de otros. Para ello, los algoritmos de filtrado colaborativo asumen que, si un usuario A tiene la misma opinión que un usuario B en un tema, es más probable que A y B compartan opinión en otro tema. En estos sistemas, el Aprendizaje Activo toma un papel importante, debido a la necesidad de obtener información relevante para asegurar la calidad del sistema, no solo para los usuarios que ya están en el sistema, sino también para los que lleguen como nuevos usuarios.

1.2. Objetivos y enfoque

El objetivo principal de este Trabajo Fin de Grado, es explorar mecanismos de evaluación y técnicas concretas de Aprendizaje Activo, aplicados a sistemas de recomendación. Para ello, se realizarán diversos experimentos en los que se simulará el comportamiento de un sistema real, y se observará la evolución del mismo, en términos de métricas de calidad y cantidad de información relevante descubierta y por descubrir. Este objetivo general se desarrolla en los siguientes objetivos más detallados:

- Explorar la definición de estrategias de Aprendizaje Activo en Sistemas de Recomendación:
  - Implementar diversos algoritmos de recomendación orientados al Aprendizaje Activo: popularidad por promedio de puntuación, recomendación por entropía, por varianza y técnicas combinadas.
  - Investigar la idea de *explore and exploit*, utilizando unos usuarios para explorar qué estrategias son más eficientes en un momento temporal concreto del sistema, para aplicar esas estrategias luego al resto de usuarios. Otra alternativa es alternar pasos de exploración y explotación sobre todos los usuarios. Estos algoritmos reciben el nombre de estrategias adaptativas.
- Probar y evaluar las técnicas ideadas en experimentos concretos. Para ello:
  - Definir una tarea experimental concisa y representativa del problema.
  - Poner a punto un sistema simple de simulación de un entorno de recomendación. El objetivo es que la plataforma de simulación permita tomar un conjunto de datos cualquiera de preferencias de usuario, simular ciclos de recomendación y feedback interactivo de usuario, que se incorpore a los datos de entrada de los algoritmos de recomendación y aprendizaje activo para continuar el ciclo, y evaluar la efectividad de diferentes algoritmos sobre esta base.
  - Definir y simular un modelo sencillo de comportamiento que tendría un usuario real ante las recomendaciones generadas por el sistema de recomendación. Se han simulado tres posibles comportamientos: que el usuario puntúe todas las recomendaciones, solo las que le gustan, o solo la que más le gusta. Esto se implementa con las estrategias de "feedback".
- Realizar estudios sobre la evolución del sistema siguiendo el ciclo recursivo, con diferentes particiones de entrenamiento-test, y diferentes conjuntos de datos:
  - Utilizando un algoritmo simple de recomendación, y asumiendo un estado estático del sistema (no llegan nuevos usuarios).
  - Utilizando un algoritmo combinado o adaptativo de recomendación, y asumiendo un estado estático del sistema (no llegan nuevos usuarios).
  - Utilizando un algoritmo con buen rendimiento, simular un estado en el que el sistema no tiene usuarios inicialmente, y llegan nuevos usuarios por lotes en cada iteración (con incrementos fijos en todas las iteraciones).

Todos estos experimentos se han realizado para tres conjuntos de datos: MovieLens-100k [8], CM100k [3] [2] y Yahoo [13] [12].

1.3. Organización de la memoria

En el capítulo 2 se presenta un estudio del estado del arte del Aprendizaje Activo y los Sistemas de Recomendación, con el objetivo de contextualizar este trabajo.

En el capítulo 3 se detalla la metodología utilizada en el desarrollo de algoritmos y sistemas utilizados para realizar los experimentos de este trabajo.

En el capítulo 4 se concretan las condiciones en las que se han realizado estos experimentos y se presentan los resultados de los mismos.

En el capítulo 5 cerramos este trabajo con las conclusiones extraídas de los resultados presentados en el capítulo anterior, y planteamos las líneas de un posible trabajo futuro.

Por último, se presenta la bibliografía utilizada en este trabajo.

2

Estado del arte

2.1. Introducción

En los últimos años, con el auge de plataformas a escala masiva como Netflix, Amazon, Spotify, etc. el campo de la recomendación ha alcanzado una importancia considerable. La necesidad de ayudar al usuario a encontrar información, sin que éste necesariamente la haya pedido de forma explícita, es el problema que busca resolver un Sistema de Recomendación. Esta necesidad viene motivada por las cantidades masivas de datos que se manejan actualmente en muchas plataformas, que abruman al usuario cuando éste accede a al servicio que le ofrece miles o millones de opciones a elegir. Esta necesidad se traduce a una demanda por recomendaciones personalizadas a cada usuario, teniendo en cuenta sus intereses y preferencias.

Sin embargo, en la tarea de realizar recomendaciones a un usuario, aparece un problema no trivial, y es la necesidad de reunir información suficiente del usuario para poder realizar recomendaciones adecuadas. Es decir, conocer esos intereses y preferencias del usuario, con el objetivo de poder personalizar la recomendación. Es ahí donde surge el Aprendizaje Activo, una rama del Aprendizaje Automático, que utiliza estrategias especiales para decidir que recomendaciones ofrecerle al usuario, con el objetivo de mejorar las recomendaciones que el sistema hará en un futuro, a cambio de un sacrificio controlado -y reducido- de la satisfacción inmediata del usuario.

2.2. Sistemas de Recomendación

Un Sistema de Recomendación se define como un sistema de filtrado de información que extrae lo más relevante de un conjunto masivo de datos, generado por las preferencias, implícitas y/o explícitas, del usuario [10]. Normalmente, estos sistemas manejan información de usuarios, objetos, (ó *items*), y puntuaciones que los usuarios otorgan a los objetos (ó *ratings*). Estos sistemas utilizan algoritmos y tecnologías muy diversos, pero históricamente se han agrupado en dos categorías [11].

- Sistemas basados en contenido. Utilizan únicamente información de los objetos conocidos por el sistema. Normalmente, se utilizan los datos de características, (ó *features*), como entrada a estos algoritmos.

- Sistemas basados en filtrado colaborativo. Utilizan funciones de similitud entre usuarios y objetos para realizar recomendaciones. Estas funciones de similitud se computan en función de las puntuaciones de los usuarios a los objetos, no de las características de los mismos. A un usuario concreto se le recomendarán objetos parecidos a los que le han gustado ya, u objetos que gustan a usuarios parecidos a él.

2.2.1. Matriz de utilidad (ó matriz de puntuaciones)

Como ya hemos comentado, los sistemas de recomendación manejan dos entidades: usuarios y objetos, y las puntuaciones de los primeros a los segundos. Son estas puntuaciones la principal fuente de información de un Sistema de Recomendación. Por lo tanto, debido al gran tamaño que esta información supone en un sistema real, (millones de usuarios, objetos y las puntuaciones entre ellos), se debe utilizar una estructura de datos eficiente. A menudo, en la literatura se maneja un modelo teórico: la Matriz de Utilidad [11] (ó *Utility Matrix*). Se trata de una matriz dispersa, donde las filas representan usuarios, y las columnas objetos, (o por el contrario, filas para objetos y columnas para usuarios), y que almacena las puntuaciones. Como la gran mayoría de estas puntuaciones son desconocidas, todas sus celdas correspondientes estarán vacías, es por ello que hablamos de una matriz. Esta puntuación se puede representar de diversas maneras, aunque típicamente se utiliza un valor numérico, (por ejemplo, un número entre 1 y 5), que representa la puntuación que el usuario le ha dado al objeto, indicando lo mucho o poco que le gusta.

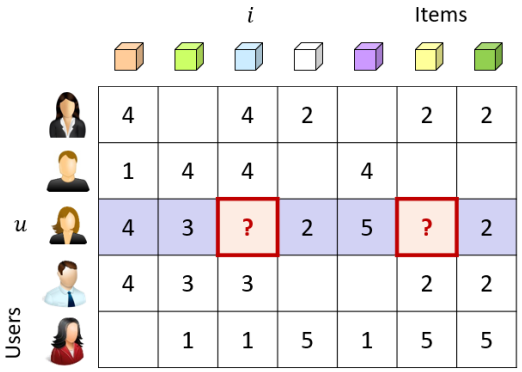

Figura 2.1: Ejemplo de una matriz de utilidad

El problema a resolver por un Sistema de Recomendación se puede formular como generar puntuaciones para aquellas celdas que están vacías. Para ello, se utilizan diversos algoritmos, que son brevemente explicados en la siguiente sección.

2.2.2. Algoritmos de recomendación

En esta sección vamos a comentar los principales algoritmos de recomendación, en especial aquellos que han sido utilizados en los experimentos realizados durante este trabajo. En la literatura [7] [14], se plantea una división de estos algoritmos según su:

- Personalización. Indica con qué grado la recomendación tiene en cuenta los intereses específicos de cada usuario. Un ejemplo de un algoritmo no personalizado es uno basado en la popularidad del objeto, ya que esta no depende de cada usuario. Un algoritmo personalizado podría ser uno basado en la similitud entre usuarios, ya que, obviamente, esas similitudes variarán de un usuario a otro.

- Hibridación. Indica si el algoritmo se compone de una sola estrategia de recomendación, o tiene en cuenta varias de ellas. Este proceso de combinar estrategias no es trivial, y por tanto se profundizará más en técnicas específicas de combinación de algoritmos de recomendación en la Sección de Sistema, diseño y desarrollo. Un algoritmo simple sería por tanto, popularidad, por ejemplo, y un algoritmo combinado sería uno que tuviera en cuenta tanto la popularidad del objeto como la similitud entre objetos o usuarios.

A parte de estas dos dimensiones, también se dividen los algoritmos según el tipo de sistema en el que son ejecutados:

- Basados en contenido. Estos algoritmos tienen en cuenta información de los objetos, como pueden ser las características del mismo. Se basan en la asunción de que si dos objetos tienen características parecidas, y un usuario U le ha gustado uno de ellos, habrá una alta probabilidad de que le guste también el otro objeto. Un ejemplo es un algoritmo basado en vecinos próximos (kNN, de *k-Nearest-Neighbours*), donde la similitud entre objetos depende de esas características.

- Basados en filtrado colaborativo. Utilizan como entrada las puntuaciones conocidas por el sistema. Un ejemplo puede ser de nuevo un algoritmo de vecinos próximos (kNN), aunque ahora la similitud entre objetos depende sólo de las puntuaciones que han recibido por los usuarios.

Cabe destacar que se han planteado muchas otras agrupaciones. Por ejemplo, en [1], se plantea que los algoritmos de filtrado colaborativo pueden a su vez dividirse en "basados en memoria", que utilizan todas las puntuaciones del sistema, y en "basados en modelos", que utilizan esas puntuaciones para generar un modelo y luego usar ese modelo para generar las recomendaciones.

El funcionamiento genérico de un algoritmo de recomendación es el siguiente:

- Se invoca al algoritmo, con las puntuaciones conocidas por el sistema, un usuario U objetivo y un número máximo de objetos a recomendar, como entradas del mismo.

- El algoritmo asigna a cada objeto conocido por el sistema una puntuación, (ó *score*). Es en el cálculo de esta puntuación donde cada algoritmo se diferencia del resto.

- El algoritmo ordena los objetos de mayor a menor según la puntuación otorgada a cada uno. Se limita la lista (ó *ranking*), con el número máximo de objetos a recomendar.

- Devuelve la lista.

En este trabajo se ha realizado una exploración principalmente en un sistema de recomendación basado en filtrado colaborativo. Es por ello que en la siguiente sección se van a presentar algoritmos basados en filtrado colaborativo, y no en contenido.

2.2.2.1. Algoritmos no personalizados

Estos algoritmos son los más simples, y a menudo son utilizados cuando no se conoce nada del usuario objetivo de la recomendación [7] [14].

- Aleatorios. Se presenta como el algoritmo más sencillo. Recomendar objetos aleatoriamente es una de las fórmulas que menos acierto puede conseguir. Sin embargo, puede ser, como

veremos, un ingrediente útil a la hora de explorar los gustos de los usuarios. Combinada con otras estrategias más complejas, la recomendación aleatoria puede ser efectiva para explorar puntuaciones desconocidas por el sistema.

- Basados en popularidad. Estos algoritmos recomiendan los objetos que mayor "popularidad" tienen. Esta popularidad puede ser calculada de diversas maneras: número de usuarios que han puntuado el objeto, puntuación promedio del objeto, entre otras.

2.2.2.2. Algoritmos personalizados

Estos algoritmos generan recomendaciones personalizadas a cada usuario concreto. Se cuentan por centenares, pero se van a nombrar solo las dos familias más importantes en sistemas basados en filtrado colaborativo.

2.2.2.2.1 Vecinos próximos (kNN)

Los algoritmos de esta familia [5] son muy populares, fáciles de entender y además son muy utilizados. Pueden ser orientados a usuarios o a objetos. Generan un vecindario con los N vecinos más próximos al objetivo es decir, que tienen una mayor similitud con el objetivo, ya sea un usuario o un objeto. Estos modelos calculan la predicción de la puntuación de la siguiente forma:

- Basado en usuario.

$$\hat{r}(u,i) = C \sum_{\substack{v \epsilon N_k(u) \\ r(v,i) \neq 0}} sim(u,v) r(v,i)$$

  Donde $N_k(u)$ es el vecindario de u, $\hat{r}(u,i)$ es la predicción de la puntuación del objeto i por el usuario u, y $r(v,i)$ es la puntuación conocida por el sistema que el usuario v ha dado al objeto i.

  C es una constante usada para normalizar. Si se desea que la puntuación predicha esté en el rango $[0,1]$, se puede usar $C = 1/(\sum_{\substack{v \epsilon N_k(u) \\ r(v,i) \neq 0}} |sim(u,v)|)$.

- Basado en objeto.

$$\hat{r}(u,i) = C \sum_{r(u,j) \neq 0} sim(i,j) r(u,j)$$

  donde $C = 1/(\sum_{r(u,j) \neq 0} |sim(i,j)|)$.

Para calcular la similitud entre usuarios y objetos, se han presentado en la literatura varias alternativas:

- Similitud coseno.

$$sim(u,v) = \sum_{\substack{i:r(u,i) \neq 0 \\ r(v,i) \neq 0}} r(u,i) r(v,i) / \sqrt{\sum_{i:r(u,i) \neq 0} r(u,i)^2 \sum_{i:r(v,i) \neq 0} r(v,i)^2}$$

- Correlación de Pearson.

$$sim(u,v) = \frac{\sum_{i \epsilon I_{uv}} (r_{u_i} - \bar{r}_u)(r_{v_i} - \bar{r}_v)}{\sqrt{\sum_{i \epsilon I_{uv}} (r_{u_i} - \bar{r}_u)^2 \sum_{i \epsilon I_{uv}} (r_{v_i} - \bar{r}_v)^2}}$$

2.2.2.2.2 Modelos de factores latentes

La principal alternativa a los modelos de vecinos próximos son los basados en factores latentes [7] [14] [9]. Un caso particular muy popular es la factorización de matrices. Este método busca, a partir de la matriz de utilidad del sistema, descubrir características latentes, llamadas factores, que están en concordancia con las puntuaciones obtenidas. Cada componente del vector indica el peso de ese factor latente en un usuario u objeto. Estas aproximaciones están dando los mejores resultados en los artículos que se publican últimamente.

2.2.2.3. Problema de arranque en frío

En los sistemas basados en filtrado colaborativo, surge un problema, conocido como arranque en frío [15], (ó *cold-start problem*), producido cuando se necesitan puntuaciones de un objeto que aún no ha sido puntuado por nadie. Es por ello, que a menudo se combinan sistemas de filtrado colaborativo con otros basados en contenido, para poder solucionar este problema, ya que los basados en contenido pueden detectar similitudes sin necesidad de que los usuarios hayan realizado puntuaciones previas. En [15] se proporcionan estadísticas sobre distintas configuraciones, combinando filtrado colaborativo y contenido, en el contexto de este problema, pero al no ser éste el objetivo de este trabajo, no se va a profundizar más en él.

2.2.3. Evaluación

En la literatura se han propuesto multitud de métricas para evaluar un Sistema de Recomendación. Una de las más importantes el Error Medio Absoluto (ó *Mean Absolute Error (MAE)*) [10]:

$$MAE = \frac{1}{N} \sum_{u,i} |\hat{r}(u,i) - r(u,i)|$$

donde $\hat{r}(u,i)$ es la puntuación predicha por el sistema del usuario u al objeto i, $r(u,i)$ es la puntuación real y N es el número total de puntuaciones.

Sin embargo, más recientemente se ha empezado a recomendar el uso de otro tipo de métricas, conocidas como "métricas de apoyo a la precisión de la decisión"[10], siendo las más utilizadas la Precisión y el Recall. Se suele usar la notación P@K, para indicar la Precisión de una recomendación de K objetos, o R@K, análogamente para el Recall:

$$P@K = \frac{|\text{Objetos relevantes en top K}|}{K}$$

$$R@K = \frac{|\text{Objetos relevantes en top K}|}{|\text{Objetos relevantes}|}$$

Aquí se incluye un nuevo concepto: la relevancia [17]. En el campo de la recuperación de información, se distingue entre datos relevantes y no relevantes. Típicamente, un dato relevante es aquel que satisface al usuario. En concreto, en lo referente a una recomendación, es relevante si ésta le gusta al usuario. Normalmente se define un umbral de relevancia, y se dice que toda recomendación con una puntuación mayor o igual a ese umbral es relevante, y menor es no relevante.

2.2.4. Novedad y Diversidad

A medida que avanzaba la investigación en el campo de la recomendación, fue ganando peso la idea de que una sola métrica de calidad, como pueden ser las medidas de precisión discutidas en el apartado anterior, no eran suficientes. En los años 2000, surgieron los conceptos de Novedad y Diversidad [4]. La Novedad se puede entender como "la diferencia entre la experiencia pasada y presente", mientras que la Diversidad se define como "las diferencias internas dentro de una misma experiencia". Estas métricas empezaron a utilizarse en los Sistemas de Recomendación con el objetivo de ampliar la perspectiva de la recomendación. Los intereses de los usuarios son a menudo inciertos, y el hecho de diversificar una recomendación ayuda al sistema a ganar más información sobre ellos. Como se plantea en [4], un ejemplo podría ser una película gustada por un usuario, que es del genero "Terror", "Fantasía" y "Misterio". El sistema no sabe por qué genero le ha gustado al usuario la película, y es por ello que, en una próxima recomendación, en vez de recomendar películas de uno de los géneros, se diversificará la recomendación entre todos los géneros, con el objetivo de que el usuario resuelva la duda al sistema.

2.3. Aprendizaje Activo

Todo sistema de Aprendizaje Automático, necesita unos datos de entrada, típicamente llamados datos de entrenamiento, para poder realizar un aprendizaje inicial a partir del cual podrá generalizar y adquirir la habilidad de clasificar nuevas entradas al sistema. Cuando estos datos son adquiridos sin esfuerzo, o se proporcionan al sistema a priori, se dice que el Aprendizaje es Pasivo.

Sin embargo, a menudo estos datos son difíciles de conseguir, y ahí es donde entran en juego las estrategias activas. Un Aprendizaje Activo es aquel que, a la hora de clasificar nuevas entradas, lo hace no solo con la intención de minimizar el error, sino que tiene en cuenta también el proceso de obtener nuevos datos, con la intención de mejorar el error a más largo plazo.

En los Sistemas de Recomendación basados en filtrado colaborativo, la información del sistema se basa en las puntuaciones que los usuarios otorgan a los objetos. Es por ello, que la tarea de seleccionar qué objetos se van a presentar al usuario, para que este los puntúe, es de vital importancia, y es ahí donde entra en juego el Aprendizaje Activo [7] [14].

El Aprendizaje Activo "ataja el problema de obtener datos de alta calidad y que mejor representa las preferencias del usuario, y mejora la calidad de la recomendación" [7] [14]. Proporciona estrategias específicas para elegir qué objetos se le presentan al usuario en forma de recomendación. En la siguiente sección se plantean algunas de las más importantes propuestas en la literatura.

2.3.1. Algoritmo general de Aprendizaje Activo

El Aprendizaje Activo se basa en realizar consultas, es decir, en pedir la clasificación de un patrón. Son estas clasificaciones las que aportan información al sistema, y por tanto, las que

se deben evaluar para utilizar aquellas que optimicen el rendimiento del sistema. En [7] [14], se plantea un algoritmo genérico de Aprendizaje Activo:

- Durante n iteraciones:

  Para cada posible consulta:

  Evaluar métrica de calidad de la consulta.

  Obtener la consulta que maximiza la métrica de calidad.

  Actualizar el modelo M con la consulta y la respuesta de la misma.

- Devolver el modelo M.

2.3.2. Estrategias de Aprendizaje Activo en filtrado colaborativo

A menudo, se utilizan estrategias propias de problemas de recomendación como si fueran propias de Aprendizaje Activo [7] [14], pues estas tienen más probabilidad de recibir respuesta por parte del usuario. Sin embargo, también hay algunos algoritmos que basan sus puntuaciones en heurísticas que no tienen sentido enmarcadas dentro de una tarea de recomendación, ya que el objetivo último de este es satisfacer la necesidad de información de un usuario. Estos algoritmos, dejan de lado esta necesidad del usuario, y buscan realizar una recomendación que mejore la información del sistema. Algunos de los más extendidos se plantean en esta sección.

- Varianza [7] [14]. Esta estrategia favorece a los objetos con las puntuaciones más dispersas, y por tanto, los que son más inciertos para el sistema.

$$Var(i) = \frac{1}{|U_i|} \sum_{u \epsilon U_i} (r_{u_i} - \bar{r}_i)^2$$

  U_i es el conjunto de usuarios que han puntuado el objeto i, y $\bar{r}_i$ es la puntuación media del objeto i.

- Entropía [7] [14]. Otra estrategia que favorece a los objetos más inciertos, midiendo la dispersión de las puntuaciones.

$$Ent(i) = \sum_{r \epsilon R} p(r_i = r) log(p(r_i = r))$$

  Donde R es el conjunto de las posibles puntuaciones del objeto i, y $p(r_i = r)$ es la probabilidad de que un usuario puntúe el objeto i con una puntuación r.

- Log(popularidad) * Entropía. Es un ejemplo de una estrategia combinada. Se aplica el logaritmo a la popularidad ya que esta última dominaría sobre la segunda, ya que hay unos pocos objetos con muchas puntuaciones, frente al gran número de objetos impopulares que hay en el sistema.

2.3.3. Explorar y Explotar

A menudo, en un sistema de Aprendizaje Activo, hay que buscar un equilibrio entre la búsqueda de información, y su utilización. Es aquí donde se introduce el concepto de "explorar y explotar"[20]. Constituye una aproximación a un algoritmo de exploración, donde durante T iteraciones hay que escoger entre K estrategias (llamadas *bandits* en [20]). Encuadrado en el

ámbito del Aprendizaje Activo en sistemas de filtrado colaborativo, encaja perfectamente. Un sistema de recomendación envía sugerencias a cada usuario una sola vez por iteración, y las estrategias de recomendación constituyen las *bandits*.

En este caso, el paso de explorar constituye la fase de evaluación de diferentes estrategias de recomendación sobre un conjunto de usuarios, escogiendo la que maximice la métrica objetivo, y el paso de explotación constituye la fase de aplicación de esa estrategia óptima al conjunto de usuarios restante. También cabe concebir la exploración y la explotación como la alternancia de dos pasos orientados a un aspecto y al otro, sobre todos los usuarios.

2.3.4. Evaluación del Aprendizaje Activo

A la hora de evaluar un sistema de Aprendizaje Activo, se puede seguir una evaluación *online*, u *offline* [7] [14].

Una evaluación *online* es aquella que implica el reclutamiento de usuarios ex profeso para un experimento que se realiza como tal una sola vez. Sin embargo, una evaluación *offline* utiliza un conjunto cerrado de datos obtenidos previamente por un conjunto de usuarios, con experimentos que pueden repetirse una y otra vez sobre los mismos datos, sin que intervenga la obtención de datos nuevos durante el experimento.

Una forma muy común de evaluar una estrategia de Aprendizaje Activo, ha sido hacerlo con lo que se llama una evaluación centrada en un solo usuario [7] [14]. Es decir, se le presentan unos objetos a un usuario, este los puntúa y estas puntuaciones que el sistema adquiere son evaluadas con unas métricas calculadas solo sobre ese mismo usuario. Esta evaluación se centra en la llegada de usuarios nuevos.

Otro proceso a tener en cuenta es la adquisición natural de puntuaciones. En este escenario, se considera que la presentación de recomendaciones al usuario no es la única manera de obtener feedback por su parte, sino que también puede ser el propio usuario el que, por su cuenta, puntúe objetos sin que sean parte de una recomendación.

La manera más completa de evaluar un sistema de Aprendizaje Activo suele tener en cuenta los dos procesos por los que se pueden adquirir nuevas puntuaciones: la presentación de recomendaciones a un usuario, y la adquisición natural de puntuaciones. Si bien es la opción más flexible, la ejecución de experimentos de evaluación online implica un coste elevado. Por este motivo el presente trabajo se basará en la evaluación offline como base para el trabajo experimental.

3

Sistema, diseño y desarrollo

El objetivo de este trabajo es realizar una investigación aplicando distintas estrategias de Aprendizaje Activo a un Sistema de Recomendación basado en filtrado colaborativo, y observar cuál ofrece una mejor eficacia a lo largo de un período de tiempo.

Se ha realizado un considerable trabajo de desarrollo, aunque se ha utilizado como base el framework Ranksys[1] para la implementación y evaluación de algoritmos y técnicas de recomendación.

3.1. Conjuntos de datos utilizados

Para realizar el estudio experimental, se han usado tres conjuntos de datos diferentes. Los tres ofrecen casos complementarios al resto, y es por ello por lo que se ha tomado la decisión de utilizarlos todos. Son los siguientes:

- MovieLens100k [8]. Los objetos de este conjunto de datos son películas. Como ejemplo de colección clásica, típicamente utilizada en investigación sobre Sistemas de Recomendación, no podía faltar en un estudio del tema. Aporta datos de entrenamiento y test con sesgos naturales derivados de la interacción espontánea con usuarios de un sistema de recomendación.

- CM100k [3] [2]. Los objetos de este conjunto de datos son canciones. Normalmente, en cualquier conjunto de datos, la distribución de puntuaciones sobre usuarios, y especialmente sobre objetos, está muy sesgada. En este conjunto de datos, todos los objetos y usuarios tienen aproximadamente el mismo número de puntuaciones, por lo que nos aporta una visión complementaria a MovieLens.

- Yahoo [13] [12]. Al igual que CM100k, las puntuaciones son del ámbito de la música. Este dataset tiene la peculiaridad de que, en el conjunto de entrenamiento, los datos están sesgados, como estarían en un sistema real. Sin embargo, el conjunto de datos de test está distribuido aleatoriamente sobre todos los objetos y usuarios.

[1] Saúl Vargas, Pablo Castells. Ranksys framework. http://ranksys.org, 2018

En todos los conjuntos de datos se han usado dos particiones diferentes: una de 80 % para entrenamiento, y 20 % para test, que representa un estado avanzado del sistema; y otra con un 20 % para entrenamiento y 80 % para test, que representa un estado más temprano del mismo.

3.2. Sistema de simulación de recomendación recursiva

Para poder ejecutar las estrategias de Aprendizaje Activo, necesitamos simular el comportamiento de un sistema real de recomendación. Para ello, se ha implementado un sistema que gestiona el conjunto de datos de recomendación de la siguiente manera:

1. El sistema parte de un conjunto de datos de entrenamiento, que contiene las preferencias de cada usuario. Estos datos representan la información que un sistema real habría adquirido a lo largo del tiempo.
2. A partir de estos datos, el sistema presenta una recomendación a cada usuario, generada mediante un algoritmo o estrategia determinada.
3. Los usuarios dan feedback, siguiendo una estrategia de feedback determinada, a partir de la recomendación realizada.
4. El sistema añade este feedback a sus datos de entrenamiento. Los objetos que forman parte del feedback no volverán a ser recomendados. A partir de estos nuevos datos de entrenamiento, se generan nuevas recomendaciones a cada usuario.
5. Se comprueba la condición de parada: el sistema ha llegado al máximo de iteraciones deseado o se ha agotado la información disponible relevante en el conjunto de test. Si se cumple, el sistema para.
6. Volver al paso 3.

Por lo tanto, cada vez que se ejecutan todos los pasos del ciclo recursivo decimos que se ha realizado una iteración del sistema.

3.2.0.1. Estrategias de feedback

Cuando a un usuario real se le presenta una recomendación, no siempre reacciona de la misma manera. Típicamente, no se recomienda un solo objeto a la vez, sino que en una misma recomendación se incluyen varios objetos. Dependiendo de la calidad de la recomendación, o de la predisposición del usuario a valorarla, éste puntuará en diferente medida los objetos recomendados. En este trabajo, se han implementado tres estrategias de feedback, que simulan tres posibles comportamientos de los usuarios:

- Feedback al top k completo de la recomendación (feedback completo). Con esta estrategia, se simula la situación en la que el usuario aporta feedback a todos los objetos de la recomendación (los k primeros del ránking que se presenta al usuario), indicando si le gustan o no.
- Feedback sólo a los objetos relevantes del top k de la recomendación (feedback relevante). Ahora simulamos que el usuario solo aporta feedback a los objetos que le gustan, es decir, que son relevantes, de la recomendación.

- Feedback sólo al primer objeto relevante del top k de la recomendación (feedback al primer relevante). En este caso, simulamos que el usuario solo aporta feedback al objeto que más le gusta de toda la recomendación, (el objeto que aparece más alto en el ránking de recomendación).

3.2.0.2. Métricas de calidad

Para poder observar la evaluación de cada estrategia de Aprendizaje Activo, necesitamos recurrir al uso de alguna métrica, que valore el estado del sistema en un punto concreto del tiempo. Como se desea observar cuantas iteraciones tarda cada estrategia en descubrir el máximo posible de información relevante que hay en el conjunto de datos de test, esta métrica debe ser directamente proporcional al número de patrones relevantes.

Se podría observar directamente el número de patrones relevantes descubiertos por el sistema como métrica, pero dado que el *Recall* es una métrica ampliamente utilizada en los Sistemas de Recomendación, y que es fácilmente computable a partir del número de patrones relevantes, se ha escogido como métrica representativa.

Por lo tanto, el *Recall* de nuestro sistema recursivo, en una iteración t es:

$$Recall(t) = \frac{\sum_{k=0}^{k=t} |\text{Relevancia descubierta en la iteración } k|}{|\text{Relevantes en test}|} \epsilon [0,1]$$

Es decir, acumulamos el número de objetos relevantes descubiertos hasta la iteración t, y dividimos por el número de objetos relevantes que hay en test.

La métrica de recall permite así al mismo tiempo medir el acierto del sistema en cada paso (perspectiva de explotación), al mismo tiempo que la proporción de información relevante acumulada descubierta por el sistema (perspectiva de exploración).

3.3. Estrategias simples

Para realizar el estudio experimental, se han utilizado múltiples estrategias simples de Aprendizaje Activo. Algunas ya estaban implementadas en el framework usado como base del trabajo, RankSys. En el desarrollo del presente trabajo se han utilizado en particular las estrategias de recomendación aleatoria, popularidad, vecinos próximos orientado a usuarios y objetos, y factorización de matrices.

En el caso del algoritmo de vecinos próximos, se ha usado una similitud coseno y un algoritmo kNN no normalizado [5], ya implementado en RankSys. Además, para el algoritmo de factorización de matrices se ha utilizado la variante propuesta en [9], que también está implementado en RankSys.

A parte de estas, también se han implementado estrategias adicionales, con el objetivo de profundizar en las técnicas de Aprendizaje Activo. En concreto:

- Popularidad por puntuación media. A diferencia de popularidad, que tan solo tiene en cuenta el número de puntuaciones de un objeto, esta estrategia realiza el promedio sobre sus puntuaciones.

$$\hat{r}(u,i) = \frac{1}{N} \sum_{r(u,i)\neq 0} r(u,i)$$

N es el número de puntuaciones disponibles para el usuario u y el objeto i.

- Varianza. Como se explicó en la sección anterior, la varianza es un claro indicador de incertidumbre en las puntuaciones de un objeto. Se utiliza como estrategia específica de Aprendizaje Activo. Es decir, se sacrifica la satisfacción del usuario a cambio de obtener información que ayude a disipar esa incertidumbre.

- Entropía. Al igual que la varianza, se utiliza como estrategia específica de Aprendizaje Activo, debido a que es otro indicador de incertidumbre.

3.4. Estrategias combinadas

A menudo, una estrategia simple aporta una visión insuficiente del sistema. Por ejemplo, una recomendación basada en popularidad se centra demasiado en un pequeño número de objetos, que tienen una gran cantidad de puntuaciones, pero deja de lado los intereses personalizados del usuario, que pueden ser aprovechados por estrategias basadas en vecinos próximos, o factorización de matrices. Es por ello, que realizando combinaciones con estrategias que aportan diferentes y/o complementarias visiones del sistema se consigue mejorar los resultados enormemente.

Para combinar varias estrategias simples se pueden ejecutar por separado, obteniendo un ránking para cada una, y luego fusionar esos ránkings normalizando las puntuaciones de cada uno. Sin embargo, existe otra manera de hacerlo, y es crear una estrategia que a la hora de calcular la puntuación realice algún tipo de combinación con las heurísticas utilizadas en las estrategias simples. Este es el caso de una estrategia que hemos comentado en la sección anterior: *Log(popularidad)*entropía* [7] [14]. Esta estrategia no calcula los ránkings de popularidad y entropía por separado, sino que calcula ambas heurísticas, y las combina aplicando un logaritmo en base dos al primer factor, y multiplicándolo por el segundo.

Ambas formas de realizar combinaciones son perfectamente válidas, siendo recomendable la segunda cuando se quiere realizar una combinación matemática específica, como es en el caso del ejemplo, motivada por las distribuciones de cada heurística. Este proceso de combinar puntuaciones requiere que todas estén en una escala equivalente, es decir, requiere que las puntuaciones sean normalizadas.

3.4.1. Normalización de puntuaciones

En cuanto a realizar la normalización de las puntuaciones, existen múltiples métodos para ello, por lo que vamos a realizar una explicación de las alternativas más importantes.

3.4.1.1. Normalización basada en valores de puntuación

Estos métodos combinan los ránkings normalizando las puntuaciones de cada uno, para que se puedan combinar directamente. Por lo tanto, la diferencia entre métodos de este tipo radica en la tarea de normalizar las puntuaciones.

Los dos métodos más importantes y extendidos son:

- Min-max. Normaliza en el rango $[0, 1]$:

$$\bar{s}(d) = \frac{s(d) - \min_{d' \epsilon R} s(d')}{\max_{d' \epsilon R}(d') - \min_{d' \epsilon R} s(d')}$$

- Z-score. Normaliza a una muestra con media nula, y desviación típica igual a uno. Se calcula de la siguiente manera:

$$\bar{s}(d) = \frac{s(d) - \mu}{\sigma}$$

  Donde μ es la media de las puntuaciones del ranking, y σ es la desviación típica de las mismas.

3.4.1.2. Normalización basada en posición

Estas estrategias no tienen en cuenta la puntuación, solo la posición del objeto correspondiente en el ránking. El método utilizado en la realización de este trabajo ha sido $Rank - Sim$. Este método asigna a cada objeto una puntuación en el rango $[0, 1]$ basada en la posición del objeto en el ránking original. La nueva puntuación se obtiene de la siguiente manera:

$$\bar{s}(i) = \frac{|R| - rank(i)}{|R| - 1}, \text{ si queremos que } \bar{s}(i) \epsilon [0, 1]$$

$$\bar{s}(i) = \frac{|R| - rank(i) + 1}{|R|}, \text{ si queremos que } \bar{s}(i) \epsilon [\frac{1}{|R|}, 1]$$

Donde $rank(i)$ es la posición del objeto i en el ranking, y $|R|$ es la longitud del ranking.

3.4.2. Combinación de puntuaciones

Una vez hemos normalizado las puntuaciones de los distintos rankings, tenemos que proceder a combinarlas. Las formas más simples de hacerlo son sumando o multiplicando las puntuaciones de un mismo objeto, obteniendo una nueva. Sin embargo, una forma más sofisticada de hacerlo es realizar una combinación lineal ponderada.

3.4.2.1. Combinación lineal ponderada

La principal novedad de este método es la inclusión de unos pesos para cada estrategia simple. Es decir, en vez de realizar una combinación lineal uniforme, ponderamos cada estrategia por el peso que queremos que tenga. Esta idea tiene mucho sentido cuando observamos que ciertas estrategias obtienen de media mejores resultados que otras, pero no queremos despreciar completamente al resto.

También es útil cuando queremos que las estrategias simples tengan mayor o menor importancia dependiendo del estado del sistema en el que nos encontremos, es decir, dependiendo de en qué punto en el tiempo nos encontremos. Por ejemplo, como veremos en el siguiente apartado, la estrategia basada en popularidad, (*pop* en adelante) funciona muy bien en las primeras iteraciones, pero se estanca muy rápidamente. Por lo tanto, podría interesarnos ponderar con un peso grande a *pop*, y reducir ese peso en sucesivas iteraciones.

Por lo tanto, recibiendo una lista de pesos, w, y una lista de rankings, $ranks$, cuyas puntuaciones han sido normalizadas, se obtiene la puntuación de un objeto i de la siguiente manera:

$$\bar{s}(i) = \sum_{k=0}^{N} w_k * ranks_k(i)$$

3.4.2.2. Intercalado con recomendaciones aleatorias

A menudo, y como veremos en la siguiente sección, las estrategias de recomendación simple tienen un acotado período de vida. Es decir, durante un número de iteraciones descubren nueva información, pero llega un punto en el que se estancan, y no consiguen descubrir nueva información relevante del conjunto de datos de test. Por ello, es a menudo recomendable intercalar con las estrategias de recomendación algunos objetos que han sido seleccionados aleatoriamente, con el objetivo de evitar este estancamiento.

Para ello, añadimos un último paso, una vez obtenido el ránking combinado con el procedimiento explicado hasta ahora, intercalamos en esa lista con una cierta probabilidad p objetos aleatorios. Es decir, recorremos el ránking combinado, y decidimos si incluir el objeto del ranking o uno aleatorio basándonos en p.

El algoritmo es:

1. Inicializamos una lista vacia l. Para cada objeto del ranking combinado:

   Obtenemos un número aleatorio $x \epsilon [0, 1]$:

   Si $p \leq p$, añadimos a l un objeto aleatorio.

   Si $p > p$, añadimos a l el objeto del ránking combinado.

2. Devolvemos l.

3.5. Estrategias adaptativas

Hasta ahora hemos hablado sobre combinaciones estáticas de estrategias de Aprendizaje Activo. Se ejecutan de la misma manera independientemente del estado del sistema, o de lo temprano o avanzado que nos encontremos en el ciclo recursivo. Sin embargo, en [7] [14], se plantea la idea de crear una combinación adaptativa de estrategias, que dependiendo del estado del sistema, ejecute unas estrategias específicas. En concreto, se plantea la idea de la *Combinación adaptativa con intercambios* [6].

3.5.1. Combinación adaptativa con intercambios

La idea de esta estrategia es utilizar un grupo aleatorio de usuarios para explorar unas estrategias determinadas, y utilizando una métrica, evaluar sus resultados para seleccionar la mejor de todas, que será posteriormente aplicada al resto de usuarios.

La métrica utilizado ha sido *Recall@k*, ya que como se ha comentado en apartados anteriores, se ha escogido como la métrica más representativa para la tarea de descubrir la información de test lo más rápido posible.

La formalización del proceso es el siguiente:

1. Recibe como entrada al algoritmo una lista de estrategias, l, un ratio de exploración, r, y unos datos de entrenamiento, d_t.

2. Se obtiene el número de grupos de exploración $m = |l|$.

   Cada grupo de exploración tiene n usuarios:

   $$n = \frac{r * |\text{usuarios en } d_t|}{m}$$

3. Se obtienen los m grupos de exploración seleccionando a n usuarios aleatorios para cada uno.

4. Por cada grupo de exploración, se obtienen las recomendaciones aplicando la estrategia correspondiente y utilizando los datos de entrenamiento d_t. Por cada recomendación, se calcula su *Recall@k*.

5. Se selecciona la estrategia con mayor *Recall@k* como la mejor estrategia en la iteración actual. Se aplica esta estrategia al resto de usuarios.

6. Se evalúa la condición de parada (un número máximo de iteraciones, un número máximo de relevancia descubierta, etc.). Si se cumple, para. Si no, volver al paso 3.

A un usuario no tiene sentido realizarle una recomendación varias veces. Si ya se le ha mostrado un objeto, y el usuario lo ha puntuado, no tiene sentido que el sistema se lo vuelva a recomendar. Es por ello, que a los grupos de exploración se les realiza la recomendación de la estrategia a explorar, pero no se les repite la recomendación con la estrategia que ha resultado la mejor, pues no tendría sentido volver a presentarles otra recomendación en una misma iteración. Es también por eso por lo que en cada iteración se escogen unos grupos de exploración aleatorios, para que no siempre sean los mismos usuarios los que reciben unas recomendaciones más imprecisas, como son las generadas por unas estrategias de tanteo.

3.5.2. Combinación adaptativa con intercalado

A raíz de la estrategia que acabamos de explicar, surge una nueva idea: ¿y si, en vez de aplicar la exploración a un pequeño número de usuarios, se la aplicásemos a todos? Podría resultar más informativo, ya que grupos pequeños de usuarios podrían no ser representativos de la comunidad entera, especialmente si estos grupos se escogen de manera aleatoria. Sin embargo, no podemos explorar eternamente, el concepto de *explorar y explotar*, consiste en explorar para poder luego explotar.

Reuniendo estas ideas, surge el concepto de combinación adaptativa con intercalado. Se basa en dedicar una iteración para explorar, obteniendo una estrategia ganadora, que será aplicada en la siguiente iteración. Una vez aplicada, volvemos a repetir el ciclo, explorando de nuevo, volviendo a aplicar la mejor estrategia, etc.

Formalizando el algoritmo:

1. Recibe como entrada al algoritmo una lista de estrategias a explorar l, unos datos de entrenamiento d_t.

2. Para cada iteración (desde 0 hasta el máximo de iteraciones deseadas):

   - Si la iteración es par (consideramos que 0 es par):

a) Obtenemos $m = |l|$ grupos de exploración y n usuarios por grupo de exploración:

$$n = \frac{|\text{usuarios en } d_t|}{|l|}$$

b) Se asignan usuarios aleatorios a cada grupo de exploración. A cada uno se le aplica una estrategia de l, y se calcula su *Recall@k*. Se selecciona la estrategia con mayor *Recall@k* como la mejor estrategia.

- Si la iteración es impar:
  - Se aplica la mejor estrategia obtenida en la iteración anterior a todos los usuarios.

3. Si se cumple la condición de parada (no queda información relevante en el conjunto de test, o hemos alcanzado la iteración máxima), parar. Si no, volver al paso 2.

3.6. Entrada progresiva de nuevos usuarios

Hasta ahora hemos descrito métodos de Aprendizaje Activo que comienzan con unos datos de entrenamiento que cuentan con información sobre todos los usuarios. Sin embargo, esto no representa la situación de un sistema real de recomendación, donde cada día llegan nuevos usuarios de los que no se conoce nada.

Normalmente, para solucionar el problema del arranque en frío, típico de los sistemas basados en filtrado colaborativo, durante el proceso de registro del usuario, se le pide que puntúe una serie de objetos, que actúan como datos de entrenamiento inicial para poder aplicarle estrategias de filtrado colaborativo, ya que sin tener puntuaciones del usuario no se le puede recomendar nada mediante estas estrategias.

Esto nos lleva a plantear un nuevo escenario de Aprendizaje Activo, que simule este proceso de llegada de nuevos usuarios. Ahora, en cada iteración del ciclo recursivo, vamos a simular que llegan nuevos usuarios. Además, el sistema no va a comenzar con datos sobre todos los usuarios, sino que va a comenzar sin datos de ningún usuario, simulando el arranque real de un sistema de recomendación. Hemos decidido nombrar a este proceso *entrada progresiva de nuevos usuarios*, o *descubrimiento iterativo de usuarios*.

Formalizando el algoritmo:

1. Recibe como entrada un número de usuarios a descubrir por iteración, n, y unos datos de entrenamiento y test iniciales, d_{train} y d_{test}, que pueden estar vacíos ($d_{train} = \emptyset, d_{test} = \emptyset$). Además, recibe unos datos de entrenamiento y test modelo, que contienen los datos que se van a descubrir iterativamente, d'_{train}, d'_{test}.

2. Para cada iteración:

   Si quedan usuarios por descubrir, se escogen mín$\{|\text{usuarios restantes}|, n\}$ usuarios aleatoriamente. Se seleccionan sus datos de entrenamiento y test modelo, $d'_{train_i}, d'_{test_i}$, y se añaden a d_{train}, d_{test}. Esto simula el proceso de registro en el que un usuario real puntuaría a unos objetos iniciales presentados por el sistema.

   Se generan recomendaciones, y se sigue el mismo proceso de retroalimentación indicado en el apartado del ciclo recursivo.

3. Comprobar la condición de parada (un número máximo de iteraciones, un número máximo de relevancia descubierta, etc.). Si se cumple, parar. Si no, volver al paso 2.

4

Experimentos y Resultados

En esta sección se describirán en detalle los experimentos realizados para probar el funcionamiento del sistema descrito en la sección anterior. Estos experimentos tienen el objetivo de evaluar el sistema, aplicando diversas técnicas de Aprendizaje Activo y observando cuál descubre más rápido la información desconocida por el sistema.

Primero, vamos a observar detalles específicos de cada conjunto de datos, (por ejemplo, cómo están distribuidas las puntuaciones de los objetos en cada uno), con el objetivo de entender mejor los resultados que se mostrarán a continuación. Tras esto, vamos a definir una tarea experimental clara y concisa. En el siguiente apartado visualizaremos los resultados de los experimentos, previamente dando una justificación a los parámetros que se han considerado adecuados para realizarlos. Tras ello mostraremos la evolución del recall en cada iteración del ciclo recursivo, y qué estrategias se escogen en cada iteración en los métodos adaptativos. Por último, discutiremos los resultados obtenidos y extraeremos conclusiones.

4.1. Detalles de los conjuntos de datos

Vamos a observar en primer lugar la distribución de los conjuntos de datos con más detalle, representando la cantidad de puntuaciones que tiene cada objeto tanto en la partición de entrenamiento como en la de test.

Para MovieLens y Yahoo, se han utilizado las particiones proporcionadas por los creadores del conjunto de datos. Utilizan aproximadamente el 80 % de los datos para entrenamiento y el 20 % para test. Sin embargo, para CM100k no se proporcionaba ninguna partición por el creador, así que se ha realizado una partición 80-20 aleatoria. Para todas las particiones 20-80 se han invertido el entrenamiento y el test de la 80-20. Todas las gráficas que se muestran a continuación se han realizado con una partición 80-20. Para la partición 20-80, bastaría con interpretar la curva de entrenamiento como si fuera la de test, y viceversa.

- MovieLens100k. En la figura 4.1 podemos observar como tanto los datos de entrenamiento como los de test están claramente sesgados. Este sesgo pretende representar el que tendría un conjunto de datos de un sistema real, ya que normalmente hay unos pocos objetos muy populares con muchas más puntuaciones que la gran mayoría del resto de objetos.

- CM100k. En este caso, la figura 4.1 representa una distribución más uniforme, tanto en entrenamiento como en test, como cabría esperar de una partición aleatoria.

- Yahoo. Por último, en la figura 4.1 se representa el caso especial de este conjunto de datos: el entrenamiento está claramente sesgado, al igual que MovieLens, pero el test es uniforme. Esto es interesante ya que al estar el conjunto de test sesgado, también lo está nuestra evaluación del sistema. De esta manera, en este conjunto de datos, podemos realizar una evaluación más imparcial.

Paralelamente a la distribución de cada dataset, es interesante también observar sus tamaños, en cuanto a número de usuarios, objetos y puntuaciones. En el cuadro 4.1 se aportan estos datos para cada uno de ellos.

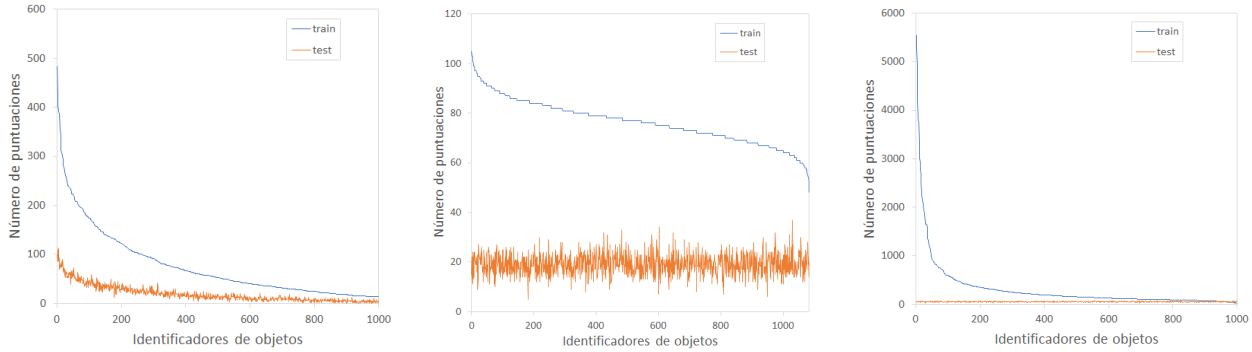

Figura 4.1: Distribución de los objetos en: a) MovieLens-100k; b) Cm100k c) Yahoo

	# usuarios	# objetos	# puntuaciones
MovieLens100k	1.000	1.700	100.000
CM100k	1.054	1.084	103.584
Yahoo	15.400	1.000	300.000

Cuadro 4.1: Número de usuarios, objetos y puntuaciones de cada dataset

4.2. Definición de la tarea

Para poder realizar un estudio de las diferentes estrategias de Aprendizaje Activo explicadas en la anterior sección, vamos a definir una tarea básica que aplicaremos en los experimentos. Esta tarea consiste en simular de manera simplificada, mediante evaluación *offline*, el proceso de recomendación continua que se produciría en un sistema real. Esto es, el sistema genera recomendaciones por lotes a cada usuario, estos usuarios devuelven información al sistema mediante el feedback que generan al puntuar esas recomendaciones, y el sistema incorpora esta información a la ya conocida para volver a generar recomendaciones. En el ámbito de este trabajo, es lo que se ha nombrado como "ciclo recursivo" en la Sección 3 de este documento.

El carácter temporal de este proceso hace que se planteen diferentes configuraciones temporales para simularlo: el sistema podría encontrarse en un estado de arranque en frío (fase temprana), o podría tener ya información suficiente sobre la gran mayoría de sus usuarios y por tanto disponer de una información poco dispersa (fase avanzada). Estas dos fases se representan con las dos particiones de los conjuntos de datos que se han probado en los experimentos: una partición 20-80 de entrenamiento-test para la fase temprana, o una 80-20 para la avanzada.

Además, podemos asumir que no llegan nuevos usuarios al sistema, es decir, que ya conoce una cantidad de puntuaciones mínimas sobre cada usuario disponible en el conjunto de datos, o, por el contrario, que dispone de información sobre un número limitado de usuarios y por tanto, llegan usuarios nuevos de los que el sistema no conoce nada. Esto se simula en el experimento de descubrimiento iterativo de usuarios.

Otro aspecto que varía la configuración del experimento es la estrategia de feedback con la que simulamos el comportamiento de los usuarios ante una recomendación. Distinguimos entre que el usuario puntúe solo lo que le gusta (feedback relevante) y puntúe lo que le gusta y lo que no (feedback completo). Dentro del feedback relevante, distinguimos entre que puntúe todos los objetos relevantes (puntúa los top n relevantes), o solo el más relevante (puntúa el top 1 relevante).

Por último, es importante aclarar una limitación importante de los experimentos *offline*. Con ellos, no podemos obtener información sobre puntuaciones que no hay en el conjunto de datos. Asumimos que esta falta de datos es equivalente a un dato no relevante. Una justificación para asumir esta equivalencia como razonable es que la probabilidad a priori de que un dato sea relevante (dado un usuario y un objeto al azar) es muy baja. Esta limitación es recurrente en todos los experimentos *offline* realizados en el campo de la recomendación.

4.3. Ajuste de parámetros de algoritmos de recomendación

Para evaluar las distintas estrategias de Aprendizaje Activo bajo las mismas condiciones, necesitamos definir varios parámetros que influyen en el rendimiento del sistema y aplicarlos en todos los experimentos.

El primero es la **longitud de la recomendación**. Es decir, cuantos objetos se le recomienda a un usuario concreto por iteración. A la hora de elegir este parámetro hemos tenido en cuenta que estamos simulando el comportamiento de un usuario real, y el incremento en el tiempo de ejecución que conllevaría incrementar la longitud de la recomendación. Hemos decidido utilizar **recomendaciones de diez objetos**, ya que nos parece razonable y aporta un buen equilibrio entre estos dos aspectos a tener en cuenta. Representa además un número típico de objetos que puede presentarse "por página" en un ránking de recomendación.

El siguiente es el **número máximo de iteraciones**. Es poco probable que una estrategia llegue a descubrir toda la información relevante que hay en test en un tiempo razonable. Por lo tanto, este parámetro va a conformar la única condición de parada del sistema. Hemos escogido **250 iteraciones como máximo** ya que hemos observado que es un número suficiente para poder observar casi todos los cambios de tendencia interesantes en la evolución del sistema, y el tiempo de ejecución no excede los límites razonables.

Otro parámetro importante es el **umbral de relevancia**. Es decir, qué puntuación marca la frontera entre lo relevante y lo no relevante. Tanto en MovieLens100k como en Yahoo, las puntuaciones siguen la codificación estándar en el rango $[1, 5]$, por lo que en ambos casos hemos decidido que el **umbral de relevancia sea la puntuación 4**. En CM100k, sin embargo, las puntuaciones están en el rango $[0, 4]$, por lo que el **umbral de relevancia es la puntuación 3**.

Por último, algunas estrategias también dependen de algunos parámetros clave en su rendimiento. Este es el ejemplo de los métodos orientados a vecinos próximos, que limitan el vecindario a los k vecinos más próximos. Tanto en Yahoo como en CM100k, el grupo de investigación del tutor aportó los parámetros óptimos: **en CM100k, no se restringe el vecindario a ningún k, tanto para orientado a usuarios como a objetos; en Yahoo, k=200 para orientado**

a usuarios y k=40 para objetos. Sin embargo, para MovieLens100k no tenían resultados claros, así que en este caso se ha realizado un pequeño experimento haciendo una búsqueda en rejilla con diferentes k para poder seleccionar una como óptima.

4.3.1. Ajuste de vecinos próximos en MovieLens-100k

El barrido se realiza para k=5, 10, 20, 50, 80, 100, 200, 500 en el basado en usuarios, y k=5, 10, 20, 50, 80, 100, 200, 500, además de otra opción en la que no se restringe el vecindario, es decir, **k=|número de objetos|**, en el basado en objetos. Esta última opción es la óptima en el basado en objetos, y **k=10 usuarios** lo es en el basado a usuarios, como se puede comprobar en la figura 4.2.

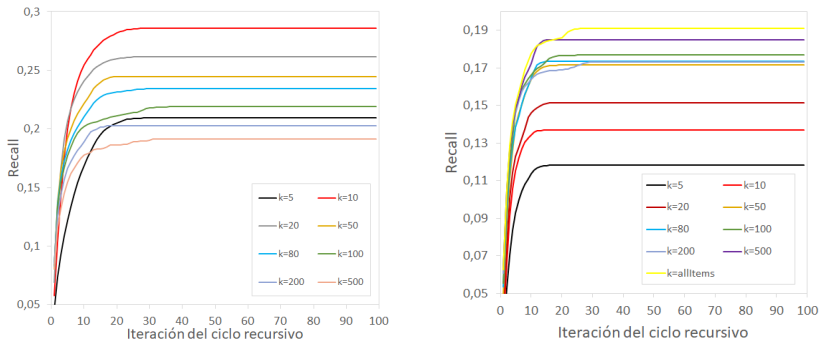

Figura 4.2: Búsqueda en rejilla de vecinos próximos en MovieLens100k basado en: a) usuarios; b) objetos

4.4. Algoritmos de Aprendizaje Activo

4.4.1. Rendimiento de estrategias de Aprendizaje Activo

Tras haber escogido unos parámetros adecuados, y haber visualizado la distribución de los diferentes conjuntos de datos, podemos realizar pruebas utilizando el ciclo recursivo. Se han realizado con las siguientes estrategias simples: aleatoria (**rnd**), popular basado en número de puntuaciones (**pop**), popular basado en puntuación media (**popAvg**), vecinos próximos orientado a usuarios (**ub**), y a objetos (**ib**), factorización de matrices (**hkv**), varianza (**var**) y entropía (**entr**).

Además, se han probado también estrategias combinadas: combinación del logaritmo de popularidad y entropía (**log-pop-entr**), combinación por rank-sim de ub, ib, hkv y entropía, (**comb-hkv-entr-ub-ib**); adaptativas: intercambio de hkv, entr y rnd, (**adapt-hkv-entr-rnd**), e intercalado con las mismas estrategias, (**switch-hkv-entr-rnd**). Por último, también se ha ejecutado una combinación por rank-sim de hkv, entr, intercalada con recomendaciones aleatorias, aplicando el descubrimiento iterativo de usuarios (**iter-hkv-entr**). De ahora en adelante, nos referiremos a cada estrategia por su acrónimo.

Ya que vamos a utilizar los acrónimos de cada estrategia en todo este apartado, en la tabla 4.2 puede visualizarse de manera clara el acrónimo correspondiente a cada estrategia. Para las estrategias combinadas, adaptativas y las que aplican el descubrimiento iterativo de usuarios, su acrónimo completo es el mostrado en la tabla junto a los acrónimos de cada estrategia simple que ejecutan, como se ha indicado en el párrafo anterior.

Estrategia	**Acrónimo**
Aleatoria	rnd
Popular por número de puntuaciones	pop
Popularidad por puntuación promedio	popAvg
Vecinos próximos orientado a usuarios	ub
Vecinos próximos orientado a objetos	ib
Factorización de matrices	hkv
Varianza	var
Entropía	entr
Combinación de popularidad y entropía	log-pop-entr
Combinación por rank-sim	comb
Adaptativa con intercambio	adapt
Adaptativa con intercalado	switch
Descubrimiento iterativo de usuarios	iter

Cuadro 4.2: Acrónimo para cada estrategia probada en los experimentos

Debido a la alta combinatoria experimental, (tres datasets, cada uno con dos particiones, cada una con tres posibles estrategias de feedback, con trece estrategias de Aprendizaje Activo a ejecutar, etc.), vamos a visualizar los resultados siguiendo un procedimiento específico, mostrándolos para los tres conjuntos de datos a la vez, para que sea más fácil realizar la comparativa: primero vamos a visualizar las estrategias simples basadas en algoritmos de recomendación, (pop, popAvg, rnd, ub, ib, hkv); después, vamos a incluir las estrategias de aprendizaje activo, (var, entr, log-pop-entr); por último, vamos a añadir las adaptativas y combinadas, (comb-hkv-entr-ub-ib, adapt-hkv-entr-rnd, switch-hkv-entr-rnd e iter-hkv-entr). En cada paso se irán descartando aquellas estrategias que tengan peores resultados, para poder visualizar claramente en cada figura las curvas de las estrategias.

Vamos a destacar dos aspectos de las curvas: lo lejos que llegan en el eje y, es decir, en recall, o en otras palabras, la cantidad máxima de información relevante que descubren (su recorrido); y, lo rápido que lo hacen, es decir, el número de iteraciones que tardan en llegar a descubrir ese máximo (su velocidad). Es importante tener en cuenta que no vale con descubrir toda la relevancia si se tarda demasiado en hacerlo. Por limitaciones de tiempo de este trabajo, no hemos tenido en cuenta la satisfacción ni la retención de usuarios, hemos asumido que siempre puntúan todas las recomendaciones que les realizamos, pero en un sistema real, si a un usuario se le recomiendan durante un largo período de tiempo objetos que no le gustan, probablemente abandonará el sistema. Es por ello, que aunque una estrategia no obtenga buenos resultados en términos de recorrido, si es muy eficiente en las primeras iteraciones puede ser interesante.

Se ha probado con las tres estrategias de feedback que se plantearon en la sección anterior, pero la opción de "feedback sólo al primer objeto relevante del top k de la recomendación" mostraba resultados equivalentes a los obtenidos con la opción de "feedback a los objetos relevantes de la recomendación". Es por ello que no se van a comentar resultados con esa estrategia de feedback. La estrategia que aporta feedback a todos los objetos del top k de la recomendación se va a nombrar "feedback completo", y la que aporta feedback sólo a los objetos relevantes se va a nombrar "feedback relevante".

Habiendo dejado claro como vamos a interpretar los resultados, vamos a realizar el análisis para cada una de las dos particiones utilizadas: 20-80, (fase temprana del sistema), y 80-20 (fase avanzada del sistema).

4.4.1.1. Fase temprana del sistema

La partición 20-80 representa un estado temprano del sistema. Cuenta con un arranque en frío, en el que los datos están muy dispersos. Vamos a visualizar los resultados aplicando las estrategias de feedback relevante y feedback completo.

4.4.1.1.1 Feedback relevante

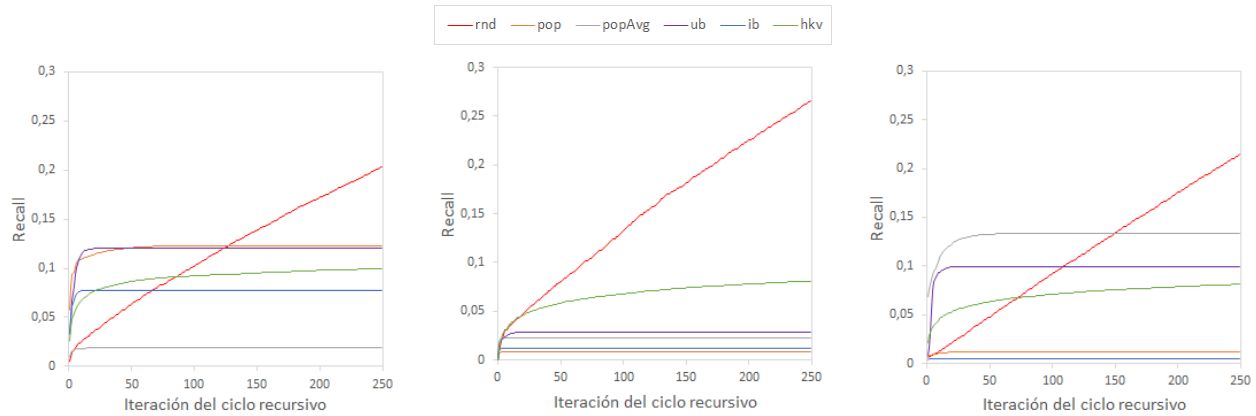

Figura 4.3: Algoritmos de recomendación aplicando feedback relevante en una partición 20-80 de a) MovieLens100k; b) CM100k; c) Yahoo

Comparando los resultados en la figura 4.3, podemos observar que popAvg funciona mal tanto en MovieLens100k como en CM100k, mientras que en Yahoo resulta ser el mejor, a parte de rnd. Esto se debe a que, como se comenta en [3], popAvg se ve muy perjudicado en datos sesgados, y como vimos en las distribuciones de los datasets, MovieLens100k tiene un sesgo importante tanto en entrenamiento como en test, mientras que en Yahoo, el test no está sesgado. Otro fenómeno destacable es que a largo plazo siempre gana rnd, lo que nos puede llevar a pensar que es la mejor estrategia de Aprendizaje Activo, lo cual es un error, pues como hemos comentado antes, rnd es muy lenta en las primeras 120-130 iteraciones. Lo que sí es interesante destacar es la gran capacidad exploratoria de rnd, lo cual nos llevará más adelante a incluir algunas recomendaciones aleatorias en estrategias más complejas.

Vamos a incluir ahora las estrategias de aprendizaje activo, quedándonos solo con rnd y la segunda estrategia ganadora en términos de recorrido alcanzado: pop en MovieLens100k, hkv en CM100k y popAvg en Yahoo.

Obtenemos unos resultados interesantes en la figura 4.4. En MovieLens100k, log-pop-entr ha ganado incluso a rnd, aunque en unas pocas iteraciones rnd volvería a superarle, pero log-pop-entr es extremadamente rápida en las primeras iteraciones, lo cual la convierte en la opción más eficiente. Como vimos en el paso anterior, pop funcionaba realmente bien en este dataset, y la combinación con entropía funciona aún mejor. Esto se debe a que combina la rapidez de descubrimiento de pop, y utiliza entr para evitar que se produzca el estancamiento una vez que se ha descubierto todo lo popular. Sin embargo, entr y var funcionan realmente mal por separado. En CM100k, entr resulta ser la que más recorrido tiene, sin tener en cuenta rnd, y var también supera a la mejor hasta ahora: hkv. Log-pop-entr funciona realmente mal, ya que pop no aportaba buenos resultados en el paso anterior. Por último, en Yahoo, ninguna llega a mejorar del todo a popAvg. Entr acaba siendo la mejor, pero a costa de un arranque más lento, lo que puede ser muy penalizado en cuanto a la retención de usuarios, por lo que seguimos considerando popAvg la mejor.

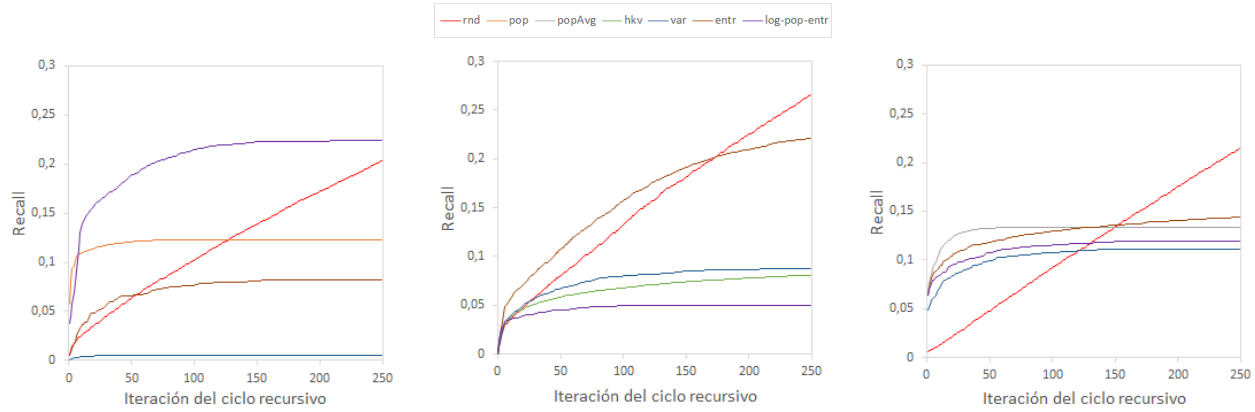

Figura 4.4: Algoritmos de Aprendizaje Activo aplicando feedback relevante en una partición 20-80 de a) MovieLens100k; b) CM100k; c) Yahoo

Finalmente, incluimos las estrategias combinadas y adaptativas. De nuevo nos quedamos con rnd y cada estrategia ganadora de este paso: log-pop-entr en MovieLens100k, entr en CM100k, y popAvg en Yahoo.

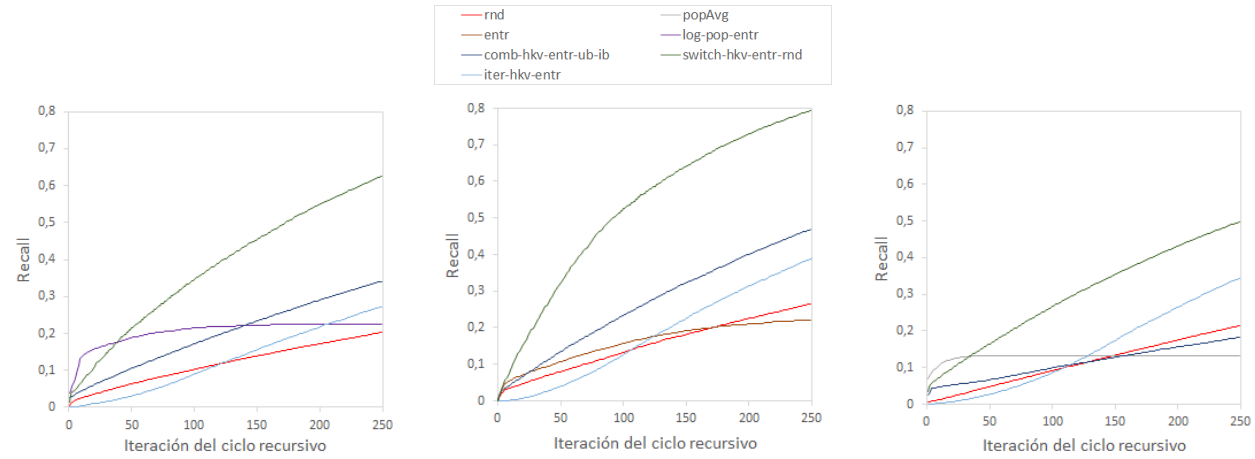

Figura 4.5: Algoritmos combinados y adaptativos aplicando feedback relevante en una partición 20-80 de a) MovieLens100k; b) CM100k; c) Yahoo

Cabe destacar que se ha eliminado la curva adapt-hkv-entr-rnd de la figura 4.5 debido a que siempre queda por debajo del resto, y empeora la claridad de los gráficos. En lo referente al resto, lo primero que salta a la vista es que ahora rnd no es la mejor estrategia en términos de recorrido. En los tres datasets, ahora es switch quien funciona mejor, seguido de comb en MovieLens100k y CM100k, e iter en Yahoo. El hecho de que gane switch es muy relevante, ya que es la mejor descubriendo relevancia, pero a la vez tiene en cuenta algoritmos personalizados, que buscan maximizar la satisfacción del usuario. Sin embargo, en MovieLens100k arranca lento, viéndose superada por log-pop-entr en las primeras 30-40 iteraciones, lo que nos lleva a pensar que tal vez podría aplicarse log-pop-entr durante esas iteraciones, y después aplicar el switch. En CM100k sin embargo también es prácticamente la mejor en las primeras iteraciones. En Yahoo se vuelve a ver superada, en este caso, por popAvg en las primeras 20-30 iteraciones.

4.4.1.2. Feedback completo

De nuevo, vamos a observar primero el comportamiento de los algoritmos simples de recomendación.

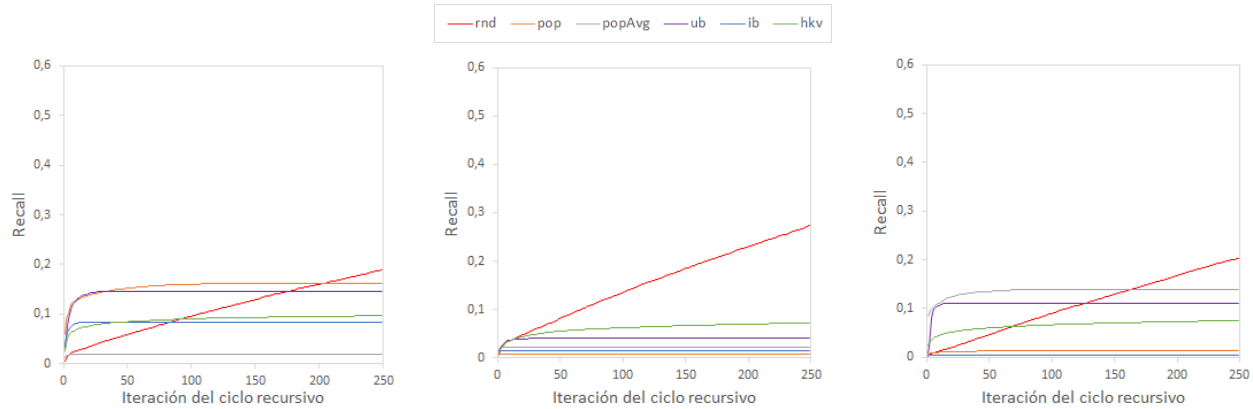

Figura 4.6: Algoritmos de recomendación aplicando feedback completo en una partición 20-80 de a) MovieLens100k; b) CM100k; c) Yahoo

En la figura 4.6, observamos que en MovieLens100k vuelve a ganar pop, en CM100k hkv y en Yahoo popAvg. Hasta ahora no apreciamos diferencias en el recorrido de cada una. Veamos que ocurre con las estrategias de Aprendizaje Activo.

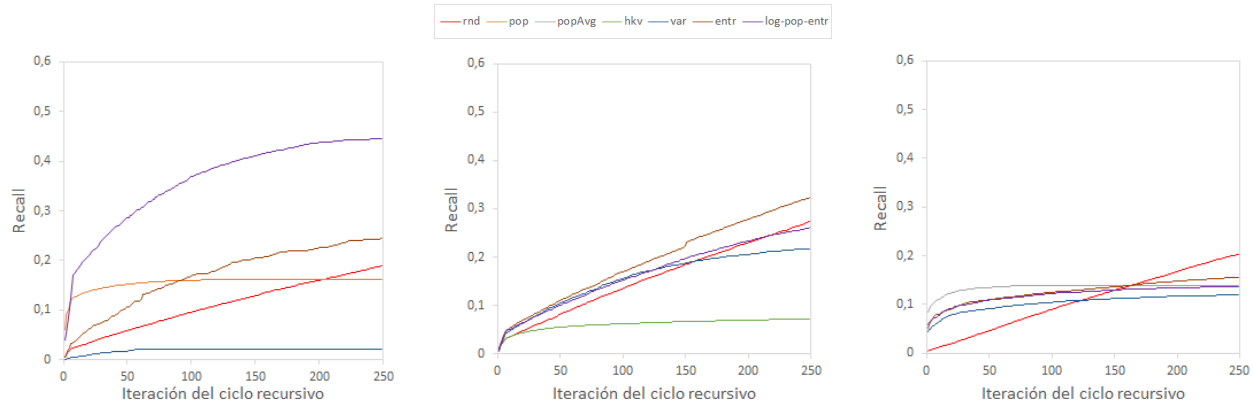

Figura 4.7: Algoritmos de Aprendizaje Activo aplicando feedback completo en una partición 20-80 de a) MovieLens100k; b) CM100k; c) Yahoo

De nuevo, no apreciamos diferencias con el caso de feedback relevante en el cuadro 4.7. Veamos por último las estrategias combinadas y adaptativas.

Ahora, sin embargo, sí observamos claras diferencias en la figura 4.8. De primeras, vemos que switch, a diferencia de con feedback relevante, no gana en ningún dataset. Es más, queda la última en todos. En MovieLens100k gana log-pop-entr, pero se aprecia que, si comb-hkv-entr-ub-ib e iter-hkv-entr siguieran con la misma tendencia, acabarían superando a log-pop-entr. En CM100k, acaba siendo iter-hkv-entr el que descubre más relevancia, seguido de cerca por comb-hkv-entr-ub-ib. De nuevo, iter-hkv-entr acaba ganando, pero sufre de un arranque muy lento, lo cual le hace menos eficaz. Por último, en Yahoo, acaba ganando comb-hkv-entr-ub-ib, que cuenta con un arranque aceptable, más lento que log-pop-entr en las primeras 20-30 iteraciones. Es por ello que lo más óptimo sería aplicar log-pop-entr hasta el punto de cruce con comb-hkv-entr-ub-ib, y a partir de ahí aplicar esta última.

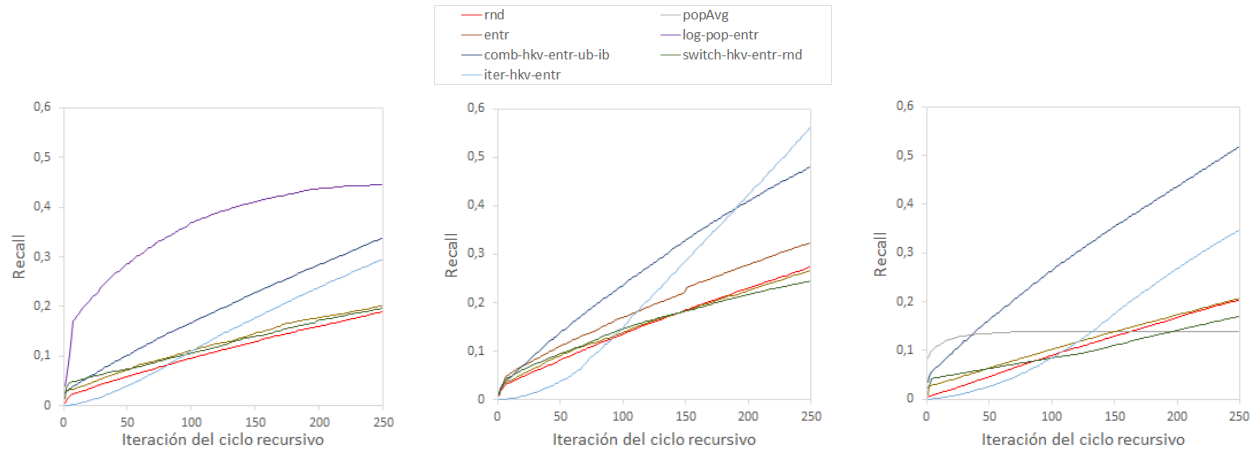

Figura 4.8: Algoritmos combinados y adaptativos aplicando feedback completo en una partición 20-80 de a) MovieLens100k; b) CM100k; c) Yahoo

Es interesante comprobar que ninguna estrategia es infalible, y aunque con feedback relevante parecía que switch era la mejor estrategia, hemos comprobado ahora que su rendimiento depende de las condiciones de cada experimento.

4.4.2. Fase avanzada del sistema

Mientras que en la anterior partición, estábamos evaluando un estado del sistema muy temprano, ahora, con una partición 80-20, vamos a evaluar un estado más avanzado del mismo. Los datos de entrenamiento ya no son tan dispersos como antes, por lo que cabría esperar que aquellas estrategias basadas en recomendar objetos con puntuaciones dispersas, funcionen peor. Además, tenemos menos recorrido, pues los datos de test ahora son de un calibre mucho menor que en el caso anterior.

4.4.2.1. Feedback relevante

Veamos el comportamiento de las estrategias básicas de recomendación en la figura 4.9. Ahora, en los tres datasets el que mejor funciona, a parte de rnd, es hkv. ¿Por qué ahora predomina hkv sobre pop o popAvg en MovieLens100k y Yahoo, respectivamente? Normalmente, hay pocos objetos con muchas puntuaciones. Es decir, hay pocos objetos que son realmente populares. Al haber decrecido tanto los datos de test, ahora quedan en test muy pocos objetos que sean realmente populares. Es por ello que ahora las estrategias basadas en popularidad no funcionan demasiado bien. Por su lado, hkv sigue descubriendo relevancia a un buen ritmo, al igual que en la partición anterior, aunque se ha producido otro cambio importante, y es que antes ub e ib quedaban por encima de hkv tanto en MovieLens100k como en Yahoo, pero ahora no.

Siguiendo con el análisis de las estrategias de Aprendizaje Activo, podemos observar los resultados en la figura 4.10. Como habíamos predicho, las técnicas de Aprendizaje Activo no funcionan nada bien en esta partición. Al no haber tanta incertidumbre en los datos de entrenamiento, no tiene sentido utilizar técnicas que buscan resolver esa incertidumbre. Quedan todas descartadas.

Pasamos ahora a las estrategias combinadas y adaptativas, en la figura 4.11. En MovieLens100k, hkv gana, aunque si siguiéramos ejecutando unas 50 iteraciones más, tanto comb-hkv-entr-ub-ib como iter-hkv-entr llegarían a superarle. Sin embargo, hkv es extremadamente

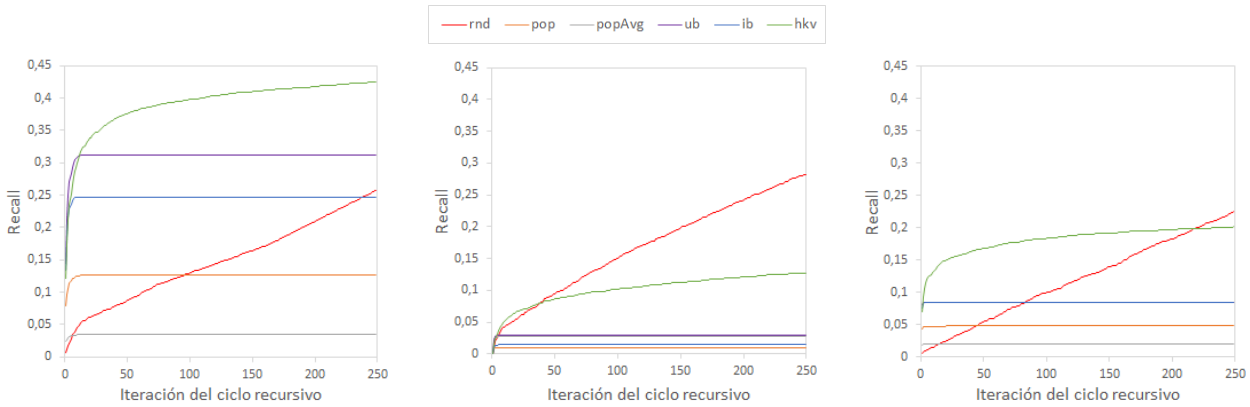

Figura 4.9: Algoritmos de recomendación aplicando feedback relevante en una partición 80-20 de a) MovieLens100k; b) CM100k; c) Yahoo

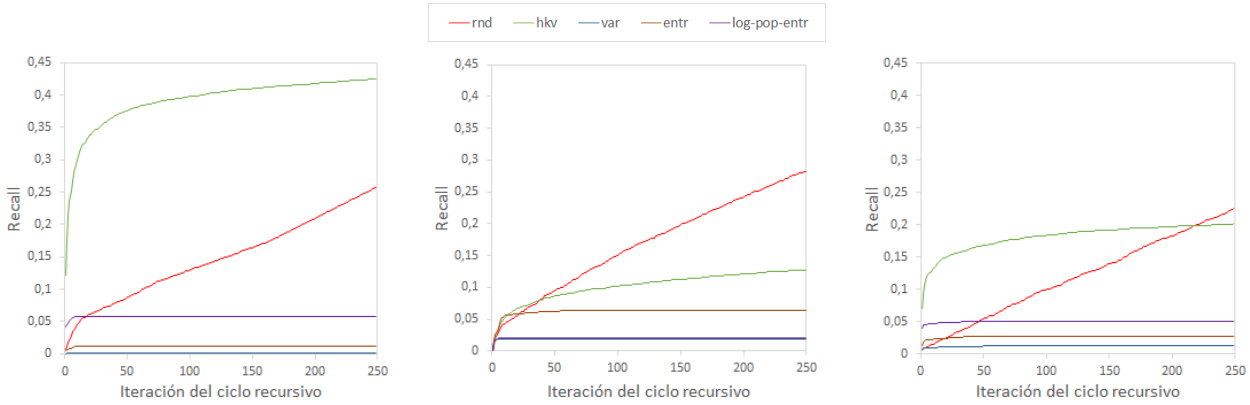

Figura 4.10: Algoritmos de Aprendizaje Activo aplicando feedback relevante en una partición 80-20 de a) MovieLens100k; b) CM100k; c) Yahoo

bueno en las primeras iteraciones, a diferencia de los adaptativos, lo que le hace el más atractivo. Además, es un algoritmo personalizado, y maximiza la satisfacción del usuario, por lo tanto parece la opción ideal. En CM100k, hkv es superado prácticamente desde el principio por comb-hkv-entr-ub-ib. En Yahoo pasa algo parecido, aunque hkv es mejor en la primeras 20-30 iteraciones.

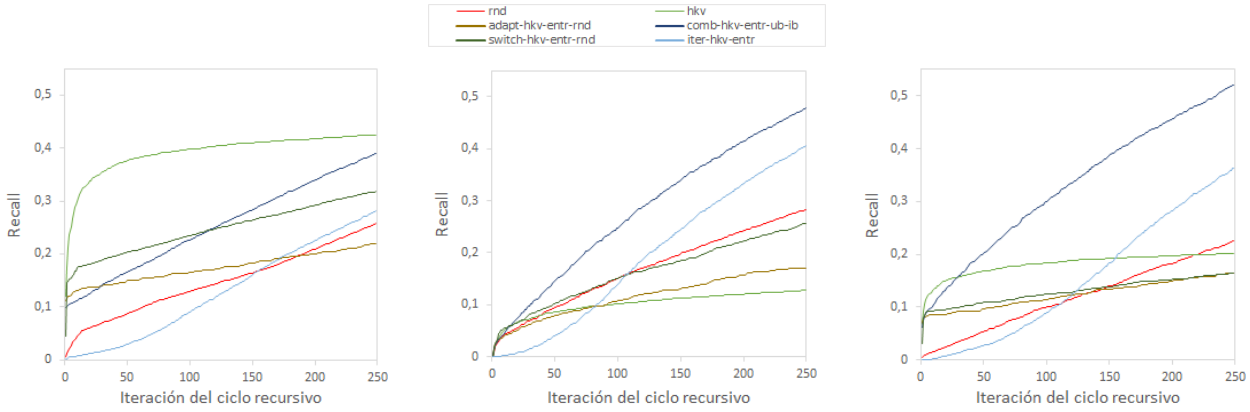

Figura 4.11: Algoritmos combinados y adaptativos aplicando feedback relevante con una partición 80-20 de a) MovieLens100k; b) CM100k; c) Yahoo

4.4.2.2. Feedback completo

Los resultados que obtenemos utilizando esta estrategia de feedback son exactamente iguales, en cuestión del orden en el que acaba cada estrategia, a los obtenidos utilizando feedback relevante. Es por ello que no vamos a presentar las gráficas aquí.

4.5. Estrategias adaptativas: selección de candidatos

En el apartado anterior hemos visto como las estrategias adaptativas son una buena opción a medio y largo plazo. Estas estrategias tienen una lista de candidatas, que son algoritmos simples de Aprendizaje Activo, y comprueban cuál tiene un mejor rendimiento en la iteración en la que se ejecuta. Por tanto, es interesante ver como evoluciona esa selección de candidatos, y observar como algunas son seleccionadas sobre todo a corto plazo, y otras a medio y largo plazo. Vamos a visualizar los resultados para la estrategia switch-hkv-entr-rnd, ya que adapt-hkv-entr-rnd no obtuvo buenos resultados en ninguno de los experimentos realizados.

4.5.1. Fase temprana del sistema

Comenzamos viendo cómo se seleccionan las candidatas en la partición 20-80 en la figura 4.12. En MovieLens100k, se ejecuta hkv en las primeras iteraciones, en concreto, en las dos primeras. Sin embargo, a partir de entonces se alternan las ejecuciones de rnd y entr, hasta la iteración 38. Después, se ejecuta siempre rnd. Como comentamos en el apartado anterior, rnd es una buena opción para obtener información, pero como hemos visto en las gráficas de Recall, tarda mucho en superar a los otros algoritmos. Es por ello, que ejecutando hkv y entr durante las primeras iteraciones, se mejora enormemente el resultado. En CM100k y Yahoo no se ejecuta nunca hkv, solo entr y rnd.

En la figura 4.13, se puede visualizar el rendimiento de hkv, entr y rnd en cada conjunto de datos. Ahí podemos comprobar como en MovieLens100k, hkv sube muy rápido al principio, pero luego se estanca, donde pasan a ejecutarse entr y rnd. Sin embargo, en CM100k, hkv es la última desde la primera iteración, y es por ello que no se selecciona nunca como ganadora. Algo parecido pasa en Yahoo: aunque hkv gana a rnd, esta es superada por entr, que es la que gana en las primeras iteraciones.

No vamos a visualizar más resultados, ya que son muy parecidos, y la única estrategia adaptativa que mostraba un rendimiento eficaz en el anterior apartado era switch-hkv-entr en el experimento de la partición 20-80 y feedback relevante.

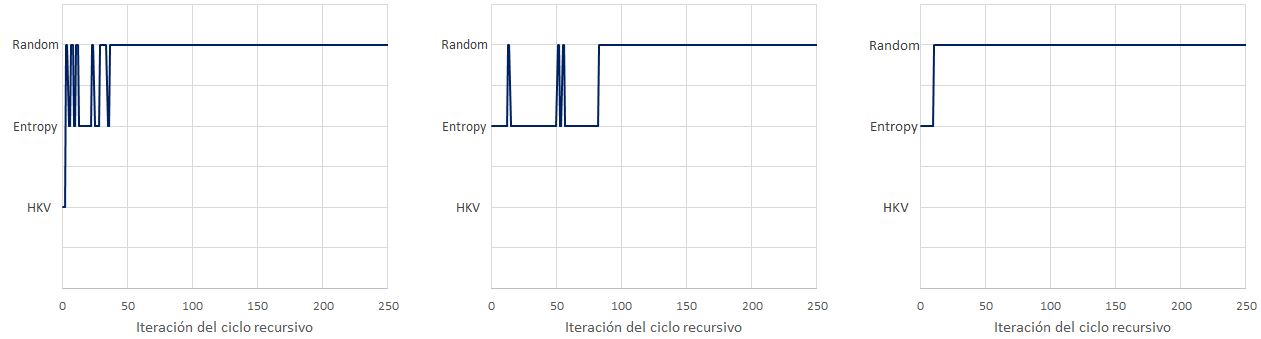

Figura 4.12: Selección de candidatas por switch-hkv-entr-rnd aplicando feedback relevante con una partición 20-80 de a) MovieLens100k; b) CM100k; c) Yahoo

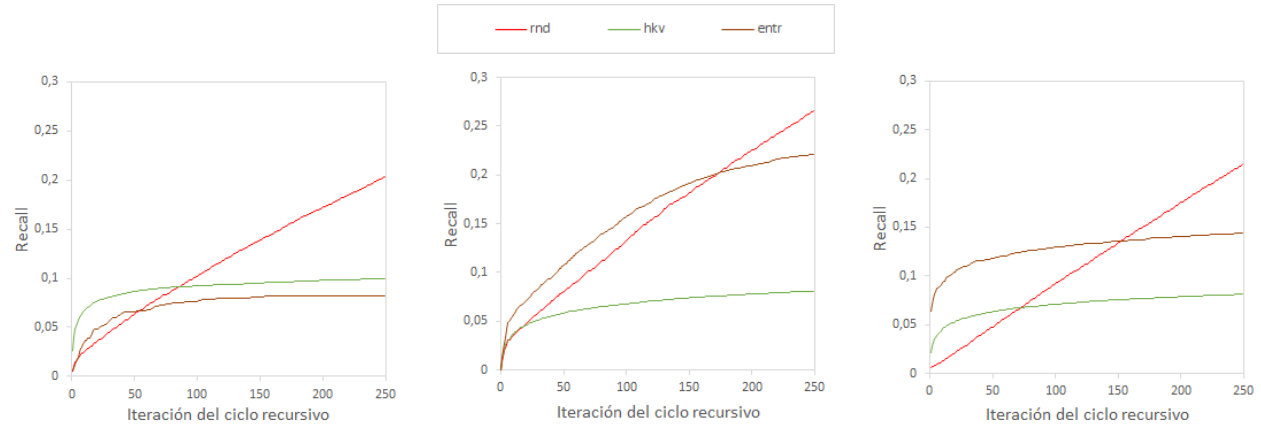

Figura 4.13: Rendimiento de hkv, entr y rnd aplicando feedback relevante con una partición 20-80 de a) MovieLens100k; b) CM100k; c) Yahoo

4.6. Discusión de los resultados

En los resultados expuestos en el apartado anterior hemos visto como el rendimiento de cada estrategia depende del conjunto de datos y del estado del sistema, y hemos realizado pruebas con un par de estrategias combinadas y adaptativas.

En las **tablas 4.3 y 4.4** se ofrece un resumen de los resultados de cada conjunto de datos y partición, dividiendo las 250 iteraciones mostradas en las gráficas del apartado anterior en tres fases: corto plazo (primeras 30 iteraciones), medio plazo (entre 30 y 150 iteraciones) y largo plazo (de 150 iteraciones en adelante).

En definitiva, las estrategias no personalizadas, como aquellas basadas en popularidad, son buenas en estados tempranos del sistema, y a corto plazo desde que empiezan a ejecutarse, pues una vez que han recomendado los pocos objetos con muchas puntuaciones, dejan de descubrir objetos relevantes.

Las estrategias basadas en la dispersión de puntuaciones, como varianza y entropía, son buenas cuando efectivamente las puntuaciones están dispersas. En estados avanzados del sistema, y a medio y largo plazo desde que empiezan a ejecutarse, no funcionan nada bien.

Las estrategias personalizadas funcionan realmente bien tanto en estados tempranos como en avanzados del sistema, aunque rápidamente se estancan y dejan de descubrir relevancia.

Ha quedado demostrado que se puede sacar mucho provecho de las estrategias combinadas y adaptativas. En todos los resultados que hemos mostrado, a largo plazo siempre quedaría en primer lugar una estrategia combinada o adaptativa. Además, gracias a la inclusión de recomendaciones aleatorias, se evita ese estancamiento en el descubrimiento de relevancia propio de las estrategias simples.

	MovieLens100K	CM100K	Yahoo
Partición 20-80	Mejor a corto plazo: **log-pop-entr** Mejor a medio y largo plazo: **switch**	Mejor a corto, medio y largo plazo: **switch**	Mejor a corto plazo: **popAvg** Mejor a medio y largo plazo: **switch**
Partición 80-20	Mejor a corto y medio plazo: **hkv** Mejor a largo plazo: **comb y switch**	Mejor a corto plazo: **switch** Mejor a medio y largo plazo: **comb**	Mejor a corto plazo: **hkv** Mejor a medio y largo plazo: **comb**

Cuadro 4.3: Resumen de los resultados para feedback relevante

	MovieLens100K	CM100K	Yahoo
Partición 20-80	Mejor a corto y medio plazo: **log-pop-entr** Mejor a largo plazo: **comb**	Mejor a corto plazo: **entr** Mejor a medio plazo: **comb** Mejor a largo plazo: **iter**	Mejor a corto plazo: **popAvg** Mejor a medio y largo plazo: **comb**
Partición 80-20	Mejor a corto y medio plazo: **hkv** Mejor a largo plazo: **comb y switch**	Mejor a corto plazo: **switch** Mejor a medio y largo plazo: **comb**	Mejor a corto plazo: **hkv** Mejor a medio y largo plazo: **comb**

Cuadro 4.4: Resumen de los resultados para feedback completo

5

Conclusiones y trabajo futuro

A lo largo de este trabajo se han puesto en práctica diversas estrategias de Aprendizaje Activo, en el ámbito de la recomendación basada en filtrado colaborativo, un tema que aún parece estar un poco incipiente y no se ha investigado tan en profundidad como su integración con el Machine Learning.

Se han realizado experimentos simulando un sistema real de recomendación, es decir, realizando una evaluación *offline*, implementando tres posibles reacciones de un usuario ante una recomendación: puntuar toda la recomendación, puntuar solo lo que le gusta, y puntuar solo el objeto que más le gusta.

Además, se han ejecutado estas estrategias con tres conjuntos de datos diferentes, debido a que aportan visiones complementarias. De cada uno de ellos, se ha utilizado una partición con un 80 % de los datos para entrenamiento y un 20 % para test, que representa un estado avanzado del sistema, y otra invertida, que representa un estado temprano del sistema.

En definitiva, en este trabajo se ha realizado una exploración del rendimiento de diferentes estrategias de Aprendizaje Activo en el ámbito de los Sistemas de Recomendación, bajo diferentes conjuntos de datos y configuraciones. Para evaluar cada una de ellas, se ha usado una métrica muy extendida: el Recall. En el futuro, se podrían realizar estudios evaluando con otras métricas, como la precisión. Además, la aplicación del Aprendizaje Activo se podría trasladar al campo de las redes sociales, que está muy relacionado con el de la recomendación.

Además, como cada estrategia combinada y adaptativa depende evidentemente de las estrategias simples que computan, en el futuro se podría ampliar el espacio experimental, utilizando en cada dataset aquellas estrategias simples que aporten mejores resultados, como candidatas en las combinadas y adaptativas. Adicionalmente, se podrían probar estrategias de *bandits* más sofisticadas.

Por otro lado, un reto para el futuro es ampliar la visión del trabajo a la satisfacción del usuario. Los resultados planteados en este trabajo responden a un experimento en el que se está asumiendo que el usuario puntúa todas las recomendaciones por igual, cuando en la realidad una recomendación personalizada es mucho más probable que sea puntuada por un usuario que una recomendación aleatoria.

Por último, se podría llevar a cabo experimentos de evaluación *online* con usuarios reales. La evaluación *offline* tiene limitaciones en el espectro de objetos que pueden ser candidatos de una

recomendación. Mientras que en un experimento *online*, cualquier objeto puede ser incluido en una recomendación, en uno *offline* solo aquellos que están incluidos en el conjunto de datos de test pueden ser incluidos en la recomendación. Esto limita sustancialmente la veracidad de los experimentos *offline*, y es por ello que deberían ser confirmados con pruebas realizadas teniendo en cuenta a usuarios reales.

Bibliografía

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.

[2] R. Cañamares and P. Castells. On the optimal non-personalized recommendation: From the prp to the discovery false negative principle. In *ACM SIGIR 2017 Workshop on Axiomatic Thinking for Information Retrieval and Related Tasks (ATIR 2017)*, Tokyo, Japan, August 2017.

[3] R. Cañamares and P. Castells. Should i follow the crowd? a probabilistic analysis of the effectiveness of popularity in recommender systems. In *41st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2018)*, Ann Arbor, Michigan, USA, July 2018.

[4] Pablo Castells, Neil J. Hurley, and Saul Vargas. *Novelty and Diversity in Recommender Systems*, pages 881–918. Springer US, Boston, MA, 2015.

[5] Christian Desrosiers and George Karypis. *A Comprehensive Survey of Neighborhood-based Recommendation Methods*, pages 107–144. Springer US, Boston, MA, 2011.

[6] Francesco Elahi, Mehdiand Ricci and Neil Rubens. Adapting to natural rating acquisition with combined active learning strategies. pages 254–263, 2012.

[7] Mehdi Elahi, Francesco Ricci, and Neil Rubens. A survey of active learning in collaborative filtering recommender systems. *Comput. Sci. Rev.*, 20(C):29–50, May 2016.

[8] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015.

[9] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[10] F.O. Isinkaye, Y.O. Folajimi, and B.A. Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3):261 – 273, 2015.

[11] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2nd edition, 2014.

[12] Benjamin M. Marlin and Richard S. Zemel. Collaborative prediction and ranking with non-random missing data. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 5–12, New York, NY, USA, 2009. ACM.

[13] Benjamin M. Marlin, Richard S. Zemel, Sam Roweis, and Malcolm Slaney. Collaborative filtering and the missing at random assumption. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, UAI'07, pages 267–275, Arlington, Virginia, United States, 2007. AUAI Press.

[14] Neil Rubens, Mehdi Elahi, Masashi Sugiyama, and Dain Kaplan. *Active Learning in Recommender Systems*, pages 809–846. Springer US, Boston, MA, 2015.

[15] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, pages 253–260, New York, NY, USA, 2002. ACM.

[16] Burr Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

[17] Guy Shani and Asela Gunawardana. *Evaluating Recommendation Systems*, pages 257–297. Springer US, Boston, MA, 2011.

[18] Jasmina Smailović, Miha Grčar, Nada Lavrač, and Martin Žnidaršič. Stream-based active learning for sentiment analysis in the financial domain. *Information Sciences*, 285:181 – 203, 2014. Processing and Mining Complex Data Streams.

[19] Ying Nian Wu, Zhangzhang Si, Haifeng Gong, and Song-Chun Zhu. Learning active basis model for object detection and recognition. *International Journal of Computer Vision*, 90(2):198–235, Nov 2010.

[20] Yisong Yue, Josef Broder, Robert Kleinberg, and Thorsten Joachims. The k-armed dueling bandits problem. *J. Comput. Syst. Sci.*, 78(5):1538–1556, September 2012.

A

Diagramas de clases del proyecto

En este anexo se incluye el diagrama de clases del proyecto Java desarrollado para la realización de los experimentos del trabajo. Para poder visualizarlo correctamente, se han eliminado las clases de menor importancia y se ha realizado una división en tres subsistemas.

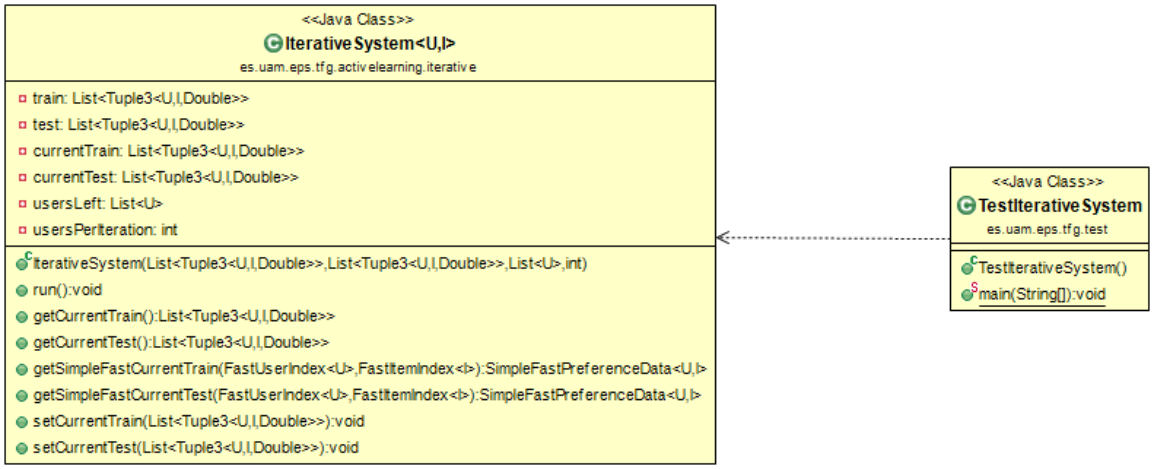

Figura A.1: Subsistema del descubrimiento iterativo de usuarios

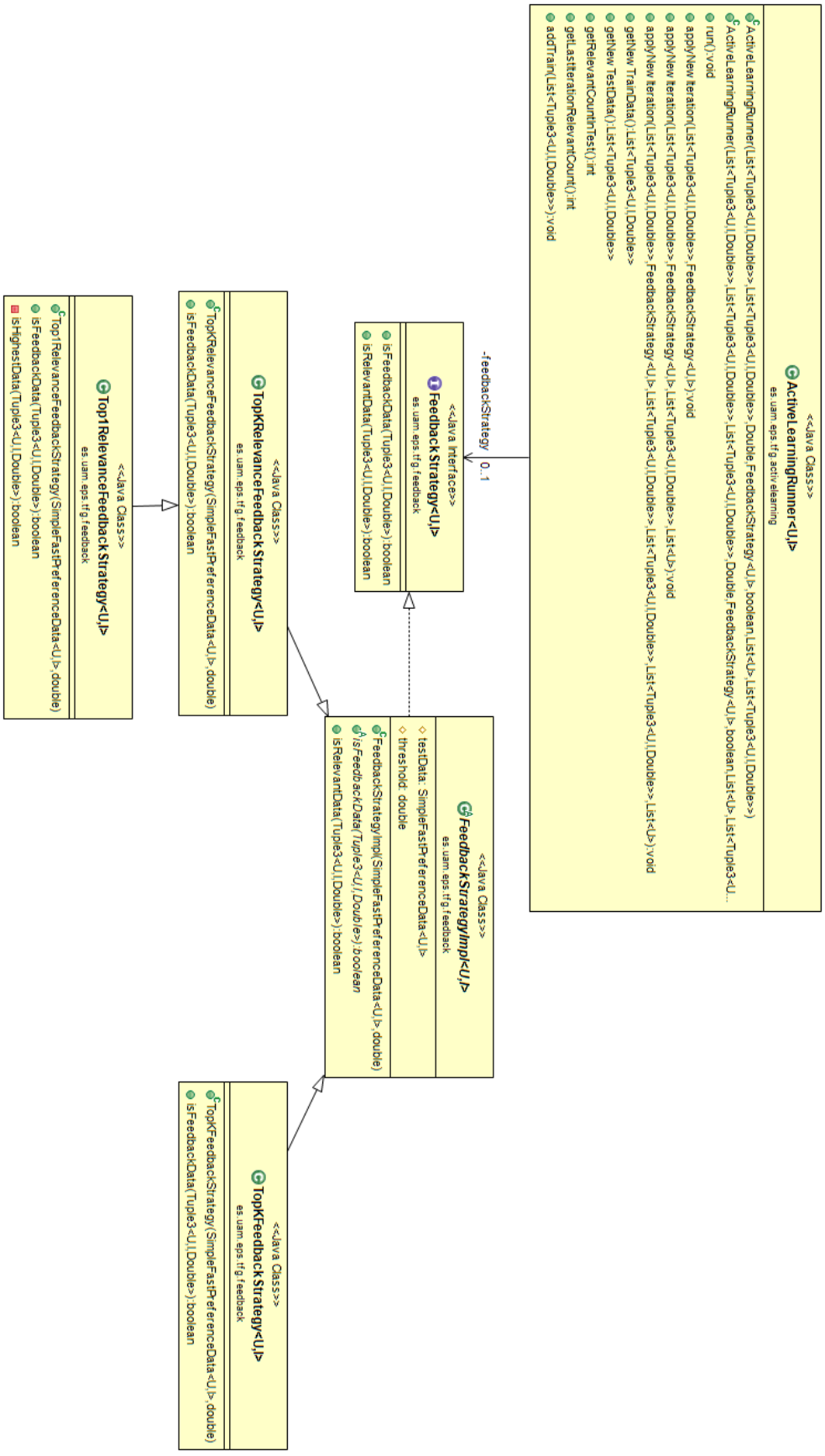

Figura A.2: Subsistema de la simulación de recomendación recursiva

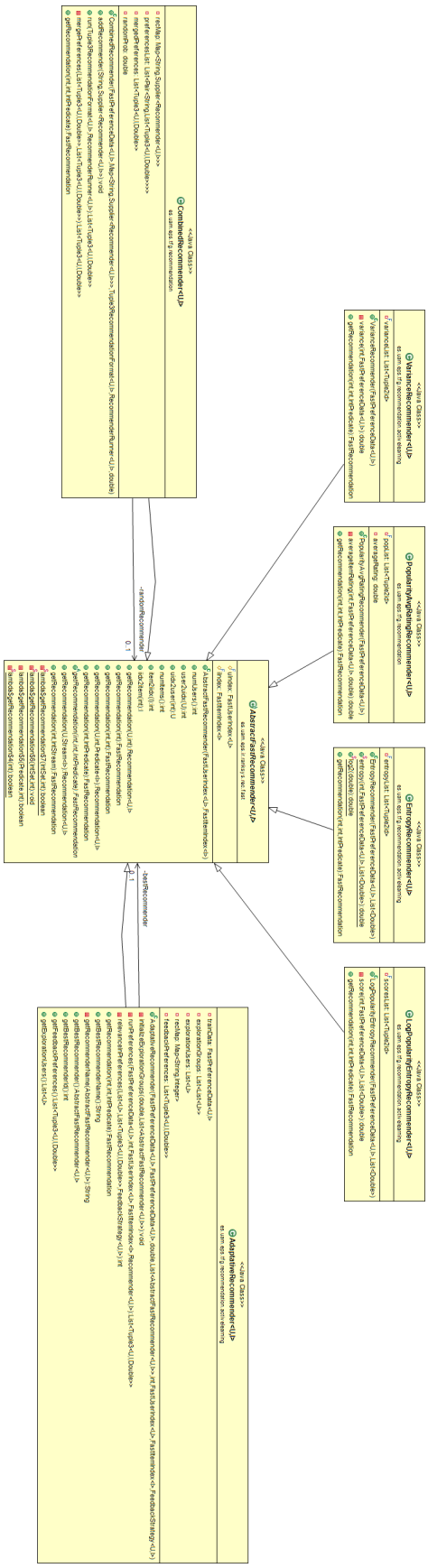

Figura A.3: Subsistema de recomendadores