

Trabajo fin de grado

Novedad y diversidad en recomendación con bandidos multi-brazo



Javier Aróstegui Martín

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR



Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

**Novedad y diversidad en recomendación con
bandidos multi-brazo**

**Autor: Javier Aróstegui Martín
Tutor: Pablo Castells Azpilicueta**

junio 2020

Todos los derechos reservados.

Queda prohibida, salvo excepción prevista en la Ley, cualquier forma de reproducción, distribución comunicación pública y transformación de esta obra sin contar con la autorización de los titulares de la propiedad intelectual.

La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual (*arts. 270 y sgts. del Código Penal*).

DERECHOS RESERVADOS

© 3 de Noviembre de 2017 por UNIVERSIDAD AUTÓNOMA DE MADRID
Francisco Tomás y Valiente, nº 1
Madrid, 28049
Spain

Javier Aróstegui Martín

Novedad y diversidad en recomendación con bandidos multi-brazo

Javier Aróstegui Martín

C\ Francisco Tomás y Valiente N° 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

RESUMEN

Hoy día, los sistemas de recomendación son un pilar clave en los servicios de consumo en línea. La dimensión cíclica de la recomendación ha empezado a tenerse explícitamente en cuenta hace poco más de una década. Estos sistemas permiten mejorar considerablemente la experiencia de los usuarios que lo usan y por tanto se traduce en usuarios más contentos y mayor consumo por su parte. Se trata de una situación en la que todas las partes salen ganando. Por ello y por otros motivos como el crecimiento de las plataformas de *streaming* en los últimos años, se ha producido en una inversión en investigación de métodos y algoritmos muy alta causando un gran crecimiento en esta área.

En este TFG se plantea el efecto de los algoritmos multi-brazo en dimensiones de novedad y diversidad, más allá del acierto de la recomendación en su sentido más estrecho. Para ello he implementado en este trabajo una serie de dichos algoritmos y unas métricas de acierto, novedad y diversidad con las que evalúo el rendimiento de los distintos algoritmos y sus hiperparámetros.

Una de las dificultades habituales en la investigación de métodos multi-brazo es el coste computacional. Ello motivará la investigación adicional de técnicas que reduzcan este coste sin comprometer la efectividad de los algoritmos.

Han sido implementados tres algoritmos multi-brazo, ϵ -greedy, *Upper Confidence Bound* y *Thompson Sampling*. Además, como *baseline* para la experimentación con aprendizaje por refuerzo se ha implementado un algoritmo de filtrado colaborativo basado en vecinos próximos (kNN) con el que tener una base realista con la que comparar el rendimiento. Los experimentos nos apuntan a que no existe ninguna aproximación perfecta para todos los problemas al mismo tiempo. Cada algoritmo tiene ventajas y desventajas en distintos aspectos. También cabe destacar la importancia del correcto ajuste de los hiperparámetros de los algoritmos ya que cada uno puede tener comportamientos totalmente diferentes dependiendo de esos valores.

PALABRAS CLAVE

Bandidos multi-brazo, diversidad, novedad

ABSTRACT

Today, recommender systems are a key pillar in online consumer services. The cyclical dimension of the recommendation has begun to be explicitly taken into account just over a decade ago. These systems make it possible to considerably improve the experience of the users who use it and therefore it translates into happier users and greater consumption on their part. It is a situation in which all parties win. For this reason and for other reasons such as the growth of textit streaming platforms in recent years, there has been a very high investment in research on methods and algorithms causing great growth in this area.

In this work, it is therefore proposed to investigate the effect of multi-armed algorithms on novelty and diversity dimensions, beyond the success of the recommendation in its narrowest sense. For this I have implemented in this work a series of said algorithms and some metrics of success, novelty and diversity with which I evaluate the performance of the different algorithms and their hyperparameters.

One of the common difficulties in investigating multi-armed methods is computational cost. This will motivate further investigation of techniques that reduce this cost without compromising the effectiveness of the algorithms.

three multi-arm algorithms have been implemented, textit textepsilon-greedy, textit Upper Confidence Bound and textit Thompson Sampling. In addition, a collaborative filtering algorithm based on nearest neighbors (kNN) has been implemented as textit baseline for experimentation with reinforcement learning with which to have a realistic basis against which to compare performance.

Experiments point out that there is no perfect approximation for all problems at the same time. Each algorithm has advantages and disadvantages in different ways. It is also worth noting the importance of the correct adjustment of the hyperparameters of the algorithms since each one can have totally different behaviors depending on those values.

KEYWORDS

Multi-armed bandits, novelty, diversity

ÍNDICE

1	Introducción	1
1.1	Motivación	1
1.2	Objetivos	2
1.3	Estructura del documento	2
2	Estado del arte	5
2.1	Sistemas de recomendación	5
2.1.1	Tarea de recomendación	6
2.1.2	Aproximaciones al problema	7
2.2	Aprendizaje por refuerzo	9
2.2.1	Formulación del problema	10
2.3	Aprendizaje por refuerzo en recomendación	12
3	Diseño y desarrollo	13
3.1	Implementación y planteamiento	13
3.2	Análisis de los datos	14
3.3	Algoritmos de bandidos multi-brazo	14
3.3.1	ϵ -greedy	14
3.3.2	Upper Confidence Bound	15
3.3.3	Thompson Sampling	16
3.3.4	User-based Knn	16
3.4	Métricas de novedad y diversidad	18
3.4.1	Novedad	18
3.4.2	Intra-List Diversity	18
3.4.3	Unexpectedness	19
3.4.4	Coeficiente de Gini	19
4	Experimentos y resultados	21
4.1	Búsqueda de hiperparámetros	21
4.1.1	ϵ -greedy	21
4.1.2	Upper Confidence Bound	22
4.1.3	Thompson Sampling	23
4.1.4	User-based Knn	24
4.2	Comparativa de algoritmos	26

5 Conclusiones y trabajo futuro	29
5.1 Conclusión	29
5.2 Trabajo futuro	30
Bibliografía	31

LISTAS

Lista de algoritmos

3.1	Pseudocódigo de ϵ -greedy	15
3.2	Pseudocódigo de UCB	15
3.3	Pseudocódigo de Thompson sampling	16
3.4	Pseudocódigo de user-based knn	17

Lista de ecuaciones

3.1	Ecuación para elegir ítem en UCB	15
3.2	Fórmula UCB	15
3.3	Ecuación para elegir ítem en Thompson Sampling	16
3.4	Fórmula similitud u-knn	16
3.5	Fórmula u-knn	17
3.6	Fórmula novedad	18
3.7	Fórmula ILD	19
3.8	Fórmula unexpectedness	19
3.9	Fórmula coeficiente de gini	19

Lista de figuras

2.1	Estructura de datos de <i>ratings</i>	7
2.2	Esquema centroides	8
2.3	Esquema user-based knn	9
2.4	Esquema del aprendizaje por refuerzo	10
3.1	Esquema funcionamiento u-knn	17
4.1	Resultados ϵ -Greedy	22
4.2	Resultados UCB	23
4.3	Resultados Thompson 1	24
4.4	Resultados Thompson 2	25
4.5	Resultados Thompson 3	25

4.6	Resultados u-knn	26
4.7	Comparativa de los resultados	28

Lista de tablas

4.1	Tabla de resultados de métricas	28
-----	---------------------------------------	----

INTRODUCCIÓN

1.1. Motivación

Los sistemas de recomendación son relativamente jóvenes. Es por ello que aún hay cabida para el desarrollo e investigación de nuevas técnicas y nuevas aproximaciones al problema. Los sistemas de recomendación actuales siguen en su mayoría una aproximación llamada *codiciosa*, es decir, optimizan la satisfacción del usuario en cada paso, a corto plazo. Se ha comprendido recientemente sin embargo que esta aproximación da lugar a soluciones sensiblemente subóptimas, y se han empezado a adaptar técnicas de los llamados *bandidos multi-brazo* que combinan explotación (del conocimiento actual sobre el usuario) y exploración (de más gustos del usuario). Una gran parte del conocimiento del sistema sobre el usuario se obtiene a través de la interacción de éste con las recomendaciones de aquél. Además de contentar al usuario, cada recomendación representa pues una oportunidad para obtener información que ayude al sistema a mejorar las siguientes recomendaciones, en una perspectiva cíclica. Las estrategias multi-brazo asumen un planteamiento en el que se comprende la dualidad de objetivo de la recomendación: la explotación de las preferencias del usuario descubiertas hasta el momento para complacer a éste, y la exploración de nuevos gustos para mejorar y ampliar el acierto y el valor aportado al usuario.

Los bandidos multi-brazo son una formulación particular en la perspectiva del *aprendizaje por refuerzo*, a su vez un área del aprendizaje automático. El planteamiento del aprendizaje por refuerzo intuitivamente se adapta mucho mejor al problema de recomendación que los sistemas de recomendación que se usan en el presente. Un sistema que actúa, observa la recompensa que se le proporciona por esa actuación, reacciona y se adapta consecuentemente tiene un mayor potencial que un sistema que simplemente evalúa la situación en cada momento y actúa en consecuencia. Es por esto por lo que en este trabajo se quiere observar el comportamiento de esta nueva aproximación al problema para así poder evaluar los distintos algoritmos y poder determinar cuáles pueden ser útiles en distintos contextos.

El trabajo se centra en una dimensión poco analizada en este campo hasta la fecha: la novedad y la diversidad de la recomendación en un marco de aprendizaje por refuerzo. La noción de explora-

ción que introducen los bandidos multi-brazo, está estrechamente conectada con el descubrimiento, la novedad y la diversidad, que tienen un sentido particularmente relevante desde la perspectiva del usuario. El estudio explícito de esta dimensión, y sus conexiones con las nociones de novedad y diversidad ampliamente estudiadas en el campo de la recomendación “avara”, es la que nos lleva a abordar el presente trabajo.

1.2. Objetivos

A lo largo de este trabajo se abordará la construcción de unos experimentos utilizando bandidos multi-brazo en recomendación. El objetivo de estos experimentos será ahondar en la conexión entre la componente exploradora de los bandidos multi-brazo y la dimensión de descubrimiento y variedad. Para ello se plantean los siguientes objetivos:

- Estudio y comprensión teórica del aprendizaje por refuerzo, los bandidos multi-brazo, sus algoritmos y las métricas que se van a utilizar.
- Implementación de un escenario genérico de pruebas para recomendación donde probar los algoritmos y medir las métricas.
- Implementación de los algoritmos ϵ -greedy, Upper Confidence Bound, Thompson Sampling y User-based knn para la realización de los experimentos.
- Implementación de las métricas de acierto, novedad y diversidad para evaluar los algoritmos en los experimentos.
- Evaluación del rendimiento y comparación de los distintos algoritmos y sus conclusiones.

1.3. Estructura del documento

La memoria está organizada de la siguiente manera:

Capítulo 1: Introducción. En este capítulo se describe cuál es la motivación que conduce a la realización de este trabajo y los objetivos que se persiguen conseguir.

Capítulo 2: Estado del arte. En este capítulo se explican las bases de los sistemas de recomendación actuales, el concepto de aprendizaje por refuerzo en general y su aplicación en los sistemas de recomendación en concreto.

Capítulo 3: Diseño y desarrollo. Aquí se muestra el procedimiento de implementación que se ha utilizado, se explican los algoritmos que se utilizan y las métricas que miden el rendimiento.

Capítulo 4: Experimentos y resultados. Esta sección se divide en 2 experimentos distintos. Primero una búsqueda de hiperparámetros para cada uno de los algoritmos que se prueban y más adelante el experimento que compara los resultados de estos algoritmos entre sí.

Capítulo 5: Conclusiones y trabajo futuro. En este último capítulo se expone de forma sin-

tetizada la conclusión del desarrollo y los experimentos realizados en este trabajo y se plantean posibles direcciones de trabajo futuro.

ESTADO DEL ARTE

Cuando un usuario común utiliza aplicaciones donde se utilizan algoritmos de recomendación puede no ser consciente de que su vista es personalizada según sus gustos, historial, etc. Esto es gracias a la investigación que se ha llevado a cabo los últimos años necesaria por el gran crecimiento de la información *online*. En esta sección se explican las bases de los sistemas de recomendación actuales, el concepto teórico del aprendizaje por refuerzo en general y su aplicación a los sistemas de recomendación en concreto.

2.1. Sistemas de recomendación

Los sistemas de recomendación son las técnicas y herramientas *software* que proporcionan sugerencias de ítems a usuarios [Castells et al., 2015]. Para hacer estas sugerencias el sistema debe estimar el valor esperado de los ítems con los cuales el usuario al que se le quiere recomendar no ha interactuado. Estas estimaciones se hacen utilizando diferentes técnicas y algoritmos que siguen distintos enfoques los cuales se explican más adelante.

Aunque existen diferencias, los sistemas de recomendación están muy relacionados con la recuperación de información ya que ambos están pensados para que a partir de un conjunto de datos se obtenga información relevante para el usuario. También guardan relación con la minería de datos ya que los ítems no valorados por el usuario se pueden equiparar a datos perdidos cuyo valor queremos estimar o predecir.

La recomendación ha cogido mayor relevancia en estos últimos años debido al crecimiento de la información en la web. Acotar la información a los usuarios es un objetivo muy deseable ya que mejora notablemente la experiencia de estos. Es por esto por lo que la recomendación se ha convertido en una de las tecnologías más populares en la web. Hoy día se utilizan tanto para generación de listas de reproducción de música y vídeo en servicios como *Netflix*, *Youtube* o *Spotify*, recomendación de compra de productos como *Amazon* o la recomendación de contenido en redes sociales como *Facebook* o *Twitter*.

Aunque este trabajo se centra en la parte de inteligencia artificial y algoritmia, el desarrollo de

los sistemas de recomendación es una tarea multidisciplinaria. Involucra el esfuerzo de expertos de campos como la interacción persona-ordenador, la minería de datos, estadística, marketing, sistemas de soporte a la toma de decisiones y el análisis del comportamiento del consumidor [Castells et al., 2015].

Cabe mencionar brevemente que los entornos en los que trabaja un sistema de recomendación deben tener ciertas características para que pueda hacer su tarea. Por ejemplo, es necesario que los usuarios tengan una cuenta o perfil para que el sistema pueda almacenar en la base de datos sus gustos y puntuaciones. Esto conlleva posibles problemas de privacidad de los usuarios. Por dar un ejemplo más grave, si la aplicación es una plataforma de citas *online*, el sistema va a querer acceder a la ubicación de los usuarios para poder hacer buenas recomendaciones y puede que no todos los usuarios se den cuenta de esto. En conclusión, la gente debe conocer para qué se utilizan los datos que proporcionan y de qué manera se tratan.

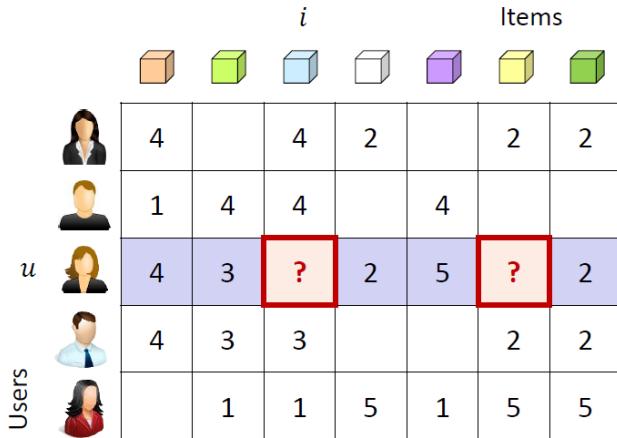
2.1.1. Tarea de recomendación

Un sistema de recomendación tiene que:

- **Observar** a los usuarios mientras evalúan ítems.
- **Detectar** patrones de comportamiento e identificar los gustos e intereses de los usuarios.
- **Predecir y recomendar** al usuario ítems que considera que al usuario le pueden interesar basado en el análisis de los comportamientos pasados.

Típicamente lo que se hace es generar un *ranking* de todos los ítems con los que el usuario aún no ha interactuado. Este *ranking* se puede generar siguiendo distintas estrategias que se explican más adelante. Si se quiere recomendar un solo ítem, se propone al usuario el top 1 de ese *ranking*. Normalmente se recomienda más de un ítem y por tanto se eligen los top k elementos del *ranking*, siendo k el número de ítems que se quieren recomendar.

Para que un sistema pueda cumplir estos objetivos es necesario guardar una serie de datos de los usuarios y evaluar estos datos de forma correcta. Por ejemplo, si la aplicación permite puntuar los ítems de 1 a 5 siendo 1 lo peor y 5 lo mejor, hay que guardar las puntuaciones que ha dado cada usuario a cada ítem y además determinar a partir de qué umbral se considera que al usuario le ha gustado ese ítem, por ejemplo 3. En el caso de sistemas de compras de productos esto puede determinar que el usuario termine comprando el ítem o no. La figura 2.1 muestra un esquema de una posible base de datos de ratings. Es una matriz de 2 dimensiones en la que se guardan las puntuaciones (celdas) de los usuarios (filas) para cada ítem (columnas). Los ítems que no han sido puntuados por el usuario dan lugar a celdas vacías y se podrían codificar como -1.



The diagram illustrates the data structure of a recommendation system. At the top, there is a legend labeled "Items" showing seven colored cubes: brown, green, blue, white, purple, yellow, and green. Below this, a matrix is shown with rows labeled "Users" (with icons for a woman, man, woman, man, and woman) and columns labeled "i" (with icons for the same seven items). The matrix contains numerical values representing user ratings (e.g., 1, 2, 3, 4, 5) or binary values (e.g., 0, 1). Two specific cells in the matrix are highlighted with red borders: the cell at row 3, column 3 (User 3 rating for Item 3) and the cell at row 3, column 6 (User 3 rating for Item 6). These highlighted cells contain question marks, indicating missing or unknown data.

		Items						
		i						
		brown	green	blue	white	purple	yellow	green
Users	User 1	4		4	2		2	2
	User 2	1	4	4		4		
	User 3	4	3	?	2	5	?	2
	User 4	4	3	3			2	2
	User 5		1	1	5	1	5	5

Figura 2.1: Esquema de la estructura de datos que maneja un sistema de recomendación.

2.1.2. Aproximaciones al problema

Existen dos aproximaciones claras al problema de la recomendación. La recomendación no personalizada y personalizada. En este apartado se van a explicar cada una de ellas, sus utilidades y algunos algoritmos que siguen cada una de las filosofías. Hay muchos algoritmos y muchas variantes de estos como para explicarlos todos así que se van a explicar los más utilizados y que se consideran más relevantes.

No personalizado

La recomendación no personalizada genera un *ranking* común entre todos los usuarios ya que no se fija en los datos de usuarios individuales si no que observa el conjunto total. Los sistemas no personalizados permiten generar recomendaciones a usuarios que aún no han interactuado con el sistema ya que no se tienen datos de ellos todavía. Un ejemplo de recomendación no personalizada es la media de la puntuación de cada ítem (número total de ventas en un sistema de compra *online*). El ítem con mayor puntuación media es el ítem que se recomienda a los usuarios nuevos. Esta estrategia puede tener distintas variaciones. Por ejemplo, se puede evaluar la puntuación media pero en un periodo de tiempo, el gestor del sistema puede querer darle más importancia a determinados ítems, etc. Por enumerar algunos algoritmos más también es común recomendar por *opinión experta* basada en las puntuaciones que ha dado alguien que se dedica profesionalmente al ámbito y *Most popular* la recomendación por *popularidad*, que recomienda las opciones más vendidas, visitadas, etc., aunque no tengan necesariamente la mejor valoración. Todas estas estrategias son muy triviales, si bien su uso es común por su sencillez de implementación despliegue y su fácil interpretación.

Personalizado

Para solucionar el problema de no tener datos de los usuarios nuevos se le puede pedir al usuario que evalúe unos cuantos ítems al crearse su perfil, entre otras opciones. Si se hace esto y con un poco de ayuda de los sistemas no personalizados, pueden empezar a actuar los sistemas de recomendación personalizados. La recomendación personalizada puede ser dividida en dos grandes grupos: la recomendación basada en contenido y el filtrado colaborativo. Existen otras aproximaciones como los algoritmos híbridos que mezclan dos o más algoritmos para compensar las debilidades de cada uno.

1.– La **recomendación basada en contenido** solamente se fija en cada usuario individual. Necesita clasificar los ítems en distintas categorías dependiendo de sus características. Si por ejemplo se trata de una aplicación de películas, se puede tener una base de datos de director, actores, géneros, etc. Este tipo de recomendación es capaz de recomendar ítems que sean nuevos o poco conocidos ya que no necesita que hayan sido puntuados. Sin embargo, tiene el problema de que recomienda siempre cosas parecidas a las que el usuario ha valorado y por tanto se pierde el efecto de exploración. Cada ítem se representa como un vector en el espacio de características y mediante una función de similitud, podemos medir cuán parecidos son dos ítems. En este trabajo en concreto se utiliza el índice de Jaccard para la similitud, pero existen otras como la similitud coseno. Aunque cualquier método de clasificación sería válido para esta tarea, se van a explicar dos métodos muy comunes que son:

- La recomendación **basada en centroides** adapta la clasificación Rocchio al problema [Figueroa et al., 2004]. El algoritmo de Rocchio es una forma de construir patrones de cada clase. Para categorizar nuevos usuarios, simplemente se estima la similitud entre él y cada uno de los patrones. En la adaptación que se realiza, los patrones se refieren al vector de características de cada ítem. Una vez tenemos un vector de usuario en el espacio de características de los ítems dependiendo de los ítems con los que haya interactuado, calcular la similitud del usuario con cualquier otro ítem es trivial utilizando la función coseno, por ejemplo. El ranking de ítems se obtiene ordenando de mayor a menor estas similitudes.

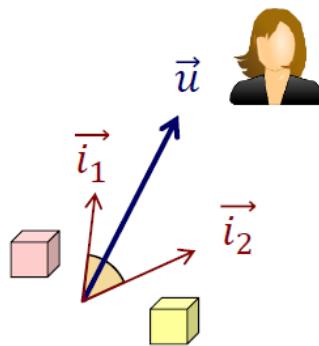


Figura 2.2: Esquema de un usuario en el espacio de características de los ítems

- El método de ***k* vecinos más próximos(knn)** identifica los ítems con las características más parecidas a los ítems que al usuario le han gustado. El *rating* esperado de cada ítem se calcula como la suma de similitudes de los *k* ítems más cercanos a ese ítem multiplicado por el *rating* que ha dado el usuario de esos *k* ítems.

2.– El **filtrado colaborativo** se basa en la observación del resto de usuarios para recomendar ítems. Se trata de fijarse con el resto de los usuarios parecidos al usuario al que se le quiere hacer la recomendación. Existen dos

ramas, los métodos basados en memoria (knn), que se explicará más adelante y se utiliza en los experimentos. Este se basa en la similitud entre las valoraciones de los usuarios. La segunda rama son los métodos basados en modelo que son muchos y muy variados: regresión, redes neuronales, factorización de matrices, entre otros. Esta aproximación en general da buenos resultados, pero presenta problemas cuando aún no se conocen datos de usuarios y de ítems. Se necesita que los usuarios hayan puntuado unos cuantos ítems y que los ítems hayan sido puntuados para que los resultados sean fiables. Si no, puede ocurrir que los usuarios no tengan suficientes vecinos o que estos no hayan visto nada nuevo que ellos no conozcan.

- **Knn** en filtrado colaborativo tiene dos aproximaciones distintas pero muy parecidas. La primera es knn colaborativo basado en ítem. Sigue la misma filosofía y estructura que el knn explicado en la recomendación basada en contenido salvo porque la similitud no se basa en las características de los ítems sino en sus *ratings*. La segunda aproximación es knn colaborativo basado en usuario. Esta aproximación es la que se utiliza en los experimentos. Consiste en coger los k usuarios más parecidos al usuario al que se le quiere hacer la recomendación y utilizar los *ratings* de los ítems que han puntuado esos usuarios similares y el usuario no. En la figura 2.3 se muestra un esquema.

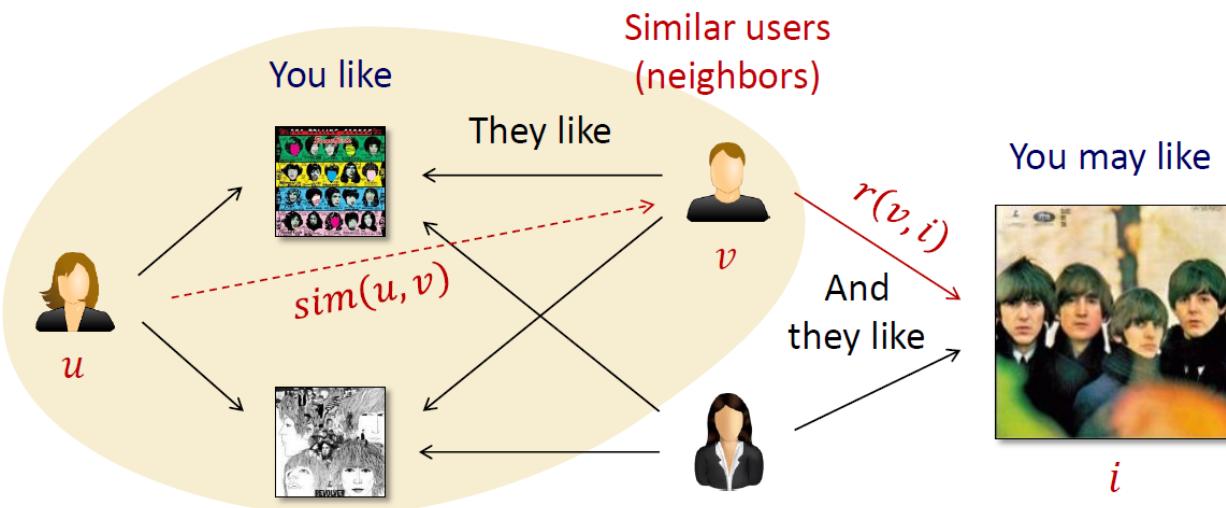


Figura 2.3: Esquema del funcionamiento de filtrado colaborativo knn con similitud entre usuarios.

2.2. Aprendizaje por refuerzo

El aprendizaje por refuerzo es un enfoque computacional al entendimiento del aprendizaje con objetivo y la toma de decisiones [Sutton and Barto, 2018]. Se inspira en el aprendizaje que desarrollan los seres vivos. Un ser vivo que no conoce su entorno comienza tomando decisiones y poco a poco va aprendiendo por prueba y error a tomar las que le generan recompensas positivas, aunque no deje de explorar opciones nuevas. Al agente no se le dice qué acciones debe tomar, sino que debe descubrir cuáles son las que generan la mejor recompensa a largo plazo. Por tanto, el problema se puede construir genéricamente como un **agente** que toma determinadas **acciones** en un **entorno** y obtiene una **recompensa** que utiliza para aprender. Se diferencia de otras herramientas de aprendizaje automático en su énfasis en dejar al agente interactuar con el entorno sin supervisión ni información

completa del entorno [Sutton and Barto, 2018].

2.2.1. Formulación del problema

A continuación, se van a explicar dos de las variantes de aprendizaje por refuerzo más extendidas: los procesos de decisión de Markov y los bandidos multibrazo. Estos últimos son la aproximación que se usa en la implementación de este trabajo.

Procesos de decisión de Markov finitos

Los procesos de decisión de Markov, MDP a partir de ahora, al igual que en la explicación general, se formula como un agente en un entorno haciendo determinadas acciones que le proporcionan una recompensa, pero en este caso el entorno se encuentra en un **estado** determinado. En la figura 2.4 se muestra un esquema del planteamiento. Por lo tanto, las acciones ahora no generan solamente una recompensa si no que hacen que el entorno cambie a un estado determinado y por tanto abre nuevas posibles recompensas. Por tanto, los MDP implican una recompensa retrasada a cambio de una recompensa inmediata.

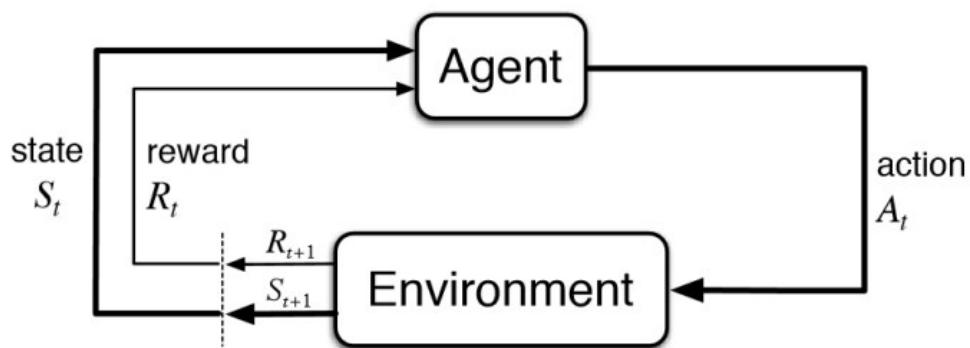


Figura 2.4: Esquema de MDP [Sutton and Barto, 2018].

Más concretamente, el agente y el entorno interactúan durante una secuencia de instantes en el tiempo $t = 0, 1, 2, \dots$. En cada uno de estos momentos el agente recibe una representación del estado en el que se encuentra el entorno S_t del total de estados S . Con esta información el agente elige una acción A_t del conjunto de acciones posibles en ese estado $A(S)$. En el siguiente momento de tiempo, como respuesta de su acción el agente recibe una recompensa R_{t+1} como un valor numérico real. El entorno cambia a un nuevo estado S_{t+1} y se repite el proceso. Por tanto, el entorno y el agente generan una secuencia que comienza de esta manera: $S_0, A_0, R_1, S_1, A_1, R_2, \dots$. Al ser finito, el número de estados, acciones y recompensas también es finito. Los estados y las recompensas siguen una distribución de probabilidad discreta que depende solamente del estado anterior y de la acción [Sutton and Barto, 2018].

Bandidos multibrazo

Los bandidos multibrazo son una simplificación del problema de aprendizaje por refuerzo y de los procesos de decisión de Markov ya que no involucra distintos estados. En este caso el agente se enfrenta a k distintas acciones que puede tomar. Después de tomar una de estas acciones recibe una recompensa numérica que sigue una distribución de probabilidad dependiendo de la acción tomada [Sutton and Barto, 2018].

El término es una analogía a una máquina tragaperras con muchas palancas en vez de una. Las palancas de la maquina son cada una de las posibles acciones que el bandido puede accionar y el bote es la recompensa. Mediante repetición el bandido debe maximizar las ganancias concentrando sus acciones en las mejores palancas. Para ello el bandido debe compensar la exploración (accionar palancas que no dan la recompensa óptima pero que proporcionan información a largo plazo) y la explotación (acciones que proporcionan recompensas altas a corto plazo). El objetivo del agente es maximizar la recompensa esperada en, por ejemplo, 100 acciones.

Cada una de las k acciones posibles que el agente puede tomar tiene una recompensa esperada llamada *valor* de esa acción. Si se conocen los valores exactos de cada acción, el problema se resuelve de manera trivial, solamente hay que elegir la acción con el valor máximo. Sin embargo, no se conocen estos valores y se deben ir ajustando. Para estimar estos valores se puede hacer la media de recompensas de cada acción:

$$Q_t(a) = \frac{\text{suma de recompensas de } a \text{ antes de } t}{\text{numero de veces que se ha realizado } a \text{ antes de } t}$$

Si nos encontramos en un arranque en frío (aún no se ha tomado ninguna acción y no hay información del entorno), como ocurrirá en los experimentos de este trabajo, el denominador es igual a 0 y por tanto hay que generar un valor por defecto, 0. A medida que se toman más acciones el denominador aumenta. Cuando el denominador tiende al infinito, por la ley de los números grandes, la estimación $Q_t(a)$ tiende al valor real de la acción.

Una vez tenemos las estimaciones de las recompensas esperadas para cada acción, hay que implementar algoritmos que se encarguen de seleccionar estas acciones de manera óptima. La aproximación más trivial es *greedy* (codicioso) que simplemente elige la acción con la recompensa esperada más alta en cada instante t . No obstante, esta estrategia no genera ningún tipo de exploración y por tanto da resultados no óptimos. A partir de aquí, existen muchos y diferentes algoritmos que afrontan este problema de diferentes maneras. En este trabajo se han implementado ϵ -*greedy* [Sutton and Barto, 2018], *Upper Confidence Bound* [Auer et al., 2002], *Thompson Sampling* [Chapelle and Li, 2011] y más adelante serán explicados.

2.3. Aprendizaje por refuerzo en recomendación

Como ya se ha dicho anteriormente, en este trabajo se va a usar la aproximación de bandidos multibrazo. Es necesario adaptar los elementos de bandidos multibrazo a un sistema de recomendación. Cada uno de los brazos del bandido puede ser cada uno de los ítems posibles por recomendar. El valor esperado por tanto puede ser el número de veces que el ítem ha sido recomendado y valorado positivamente entre el número de veces que el ítem ha sido recomendado. En un arranque en frío todos los ítems tienen un valor esperado igual a 0 y por tanto las primeras recomendaciones son aleatorias. Puede haber otras maneras de aproximar el problema, pero esta es la que se sigue en este trabajo.

DISEÑO Y DESARROLLO

En esta sección se va a explicar el procedimiento de desarrollo que se ha seguido, se va a describir la base de datos utilizada, se van a explicar los algoritmos que se van a utilizar en las pruebas con su explicación teórica y un pseudocódigo y por último se explican las métricas que se van a evaluar en los experimentos.

3.1. Implementación y planteamiento

El código implementado en este trabajo ha sido realizado en su totalidad por mi. El lenguaje de programación utilizado ha sido python 2.7. Parte del código se ha desarrollado en el laboratorio de investigación del tutor y parte en mi ordenador personal. Se han utilizado algunas implementaciones de referencia como [Castells and Sanz-Cruzado, 2019] basado en [Sanz-Cruzado et al., 2019] y [Galbraith, 2016] para la implementación de los algoritmos. Dado que una de las métricas que se va a medir es el tiempo de ejecución, ha sido necesario revisar el código para que sea lo más rápido y eficiente posible. Por ello en general los datos se guardan con redundancia para que el acceso sea directo y se pierda el mínimo tiempo. Por ejemplo, la estructura que guarda los ítems que aún no se le han recomendado a un usuario no es necesaria, pero nos permite hacer recomendaciones aleatorias mucho más rápido. Además, se utilizan técnicas de programación funcional por la misma razón. Realizar llamadas a funciones *get* y *set* de una clase tarda aproximadamente el doble de tiempo que acceder a la estructura directamente y por eso todas las clases son públicas y se accede de manera directa.

El planteamiento que se utiliza para las pruebas consiste en un proceso continuo en el que se recorren en orden aleatorio todos los usuarios de la base de datos 100 veces y se les recomienda un ítem (100 ítems recomendados por usuario). Se observa si el usuario ha valorado positivamente ese ítem en la base de datos: si su valoración es positiva se habrá acertado y si es negativa o inexistente, se habrá fallado. Por tanto, la base de datos se utiliza como herramienta de simulación de usuarios valorando ítems y para las etiquetas de esos ítems, nunca para entrenar los algoritmos. Las estrategias parten en todos los casos de un escenario virgen.

Al tratarse de una aproximación diferente a la estándar, la estructura general del sistema y sus estructuras de datos han tenido que ser adaptadas al problema. Por ejemplo, en un sistema de recomendación personalizado no se necesita guardar la "puntuación" (ratio de aciertos frente a número total de recomendaciones) de los ítems. Sin embargo para el sistema implementado si ha sido necesario guardar estos datos.

3.2. Análisis de los datos

La base de datos que se va a utilizar para los experimentos es MovieLens 1M [Movielens, 1997]. Se trata de 1 millón de puntuaciones dadas por 6000 usuarios a 4000 películas. Estas puntuaciones son un número del 1 al 5 siendo 5 la mejor nota y 1 la peor. Para este trabajo se ha considerado una puntuación igual o mayor a 3 positiva (al usuario le ha gustado la película) y en caso contrario negativa (al usuario no le ha gustado la película). La base de datos fue publicada en 2003 y existen diferentes variantes con más y menos datos.

Además de las puntuaciones de los usuarios la base de datos nos proporciona una serie de etiquetas para cada película. Se tratan de 14 géneros ya predefinidos. Estas etiquetas se utilizan en este trabajo para el cálculo de ciertas métricas de novedad y diversidad: *unexpectedness* e ILD, que se definirán en la sección 3.4.

3.3. Algoritmos de bandidos multi-brazo

En esta sección se van a explicar los algoritmos de recomendación implementados que se utilizarán más adelante para los experimentos. Se proporciona una explicación teórica de su funcionamiento además de su pseudocódigo.

3.3.1. ε -greedy

ε -greedy es una estrategia que tiene un valor ε fijado que determina cuando debe explorar y cuando explotar [Ek and Strigsson, 2015]. Con probabilidad ε , elige un brazo aleatorio, y con probabilidad $1 - \varepsilon$ elige el brazo con mejor puntuación. Como derivación de esta estrategia podemos sacar las estrategias *greedy* y *random* para los valores 0 y 1 respectivamente. Se puede ver su pseudocódigo en la figura 3.1.

```

input :  $\varepsilon$ , usuario
output: ítem recomendado para el usuario
1   X ← random ( [0, 1] );
2   if x less than  $\varepsilon$  then
3     | return randomChoice ( usuario )
4   end
5   else return itemPuntuacionMaxima ( usuario );

```

Algoritmo 3.1: Pseudocódigo de ε -greedy.

3.3.2. Upper Confidence Bound

Upper confidence bound, al contrario que ε -greedy, no explora con la misma probabilidad durante su tiempo de ejecución. El brazo accionado en cada ejecución depende de dos términos de una suma [Ek and Strigsson, 2015].

$$i_e = \operatorname{argmax}_i [s_e(i) + \gamma\sigma_e] \quad (3.1)$$

Donde $s_e(i)$ es el score del ítem i en la época e , γ el hiper parámetro del algoritmo y σ_e el término exploratorio en la época e . Las épocas se refieren a cada recomendación que se hace a cualquier usuario.

$$\sigma_e = \sqrt{\frac{\ln(e)}{\alpha(i) + \beta(i)}} \quad (3.2)$$

Donde $\alpha(i)$ es el número de veces que el algoritmo ha recomendado el ítem i satisfactoriamente a un usuario y $\beta(i)$ el número de veces en las que ha fallado en la recomendación para ese ítem. Es decir, el denominador es igual al número total de recomendaciones de un ítem y por tanto de esta manera el algoritmo prioriza en su término exploratorio ítems que hayan sido menos recomendados. Se puede ver su pseudocódigo en la figura 3.2.

```

input :  $\gamma$  y un usuario
output: ítem recomendado para el usuario
1   foreach ítem i in dataset do  $X \leftarrow \text{calculoFormula}(i)$ ;
2   return max ( X )

```

Algoritmo 3.2: Pseudocódigo de UCB.

3.3.3. Thompson Sampling

Thompson sampling es una estrategia que muestrea las acciones más prometedoras a la vez que descarta las que se creen que rendirán pobremente. Con los datos observados hasta el momento se muestrea una distribución de probabilidad para cada ítem. Cuando la recompensa de los ítems sigue una distribución Bernoulli –tomando valores 1 o 0–, es común utilizar para este muestreo una distribución Beta con parámetros $\alpha_e(i)$, $\beta_e(i)$ [Russo et al., 2018]. $\alpha_e(i)$ corresponde al número de veces que se ha acertado al recomendar el ítem i mientras que $\beta_e(i)$ se refiere al número de veces que el ítem ha sido recomendado y ha fallado. Por tanto, la probabilidad de acierto de un ítem es $\frac{\alpha_e(i)}{\alpha_e(i) + \beta_e(i)}$. Por tanto, la estrategia elige el ítem según la fórmula 3.3 [Ek and Strigsson, 2015].

$$i_e = \operatorname{argmax}_i [\operatorname{beta}(\alpha_e + \alpha_0, \beta_e + \beta_0)] \quad (3.3)$$

Cabe mencionar que a priori la actualización de las distribuciones de probabilidad se realiza para cada iteración (cada vez que se recomienda un ítem). Sin embargo, en este trabajo se van a explorar los resultados que nacen al realizar estas actualizaciones cada κ veces, siendo κ un número mayor que 1. Y por lo tanto κ es un nuevo hiperparámetro que hay que concretar. Se puede ver su pseudocódigo en la figura 3.3.

```

input :  $\alpha_e(0)$ ,  $\beta_e(0)$  y un usuario
output: item recomendado para el usuario
1 foreach item  $i$  in dataset do  $x \leftarrow \text{calculoFormula}(i)$ ;
2 return max (  $X$  )

```

Algoritmo 3.3: Pseudocódigo de Thompson sampling.

3.3.4. User-based Knn

Este algoritmo se ha escogido como punto de partida para comparar con el resto ya que se trata de un algoritmo totalmente codicioso y que sigue la filosofía de los sistemas actuales. En la Figura 3.1 se muestra un esquema intuitivo del planteamiento del algoritmo.

Cuando se le quiere recomendar un ítem a un usuario lo primero que se hace es observar a los k usuarios más similares, siendo k un hiper parámetro del algoritmo. La similitud entre usuarios se puede calcular de muchas maneras. En nuestro caso trabajamos con puntuaciones usuario-ítem binarizadas, por lo que utilizamos la similitud de Jaccard según la formula 3.4 siendo i_u los ítems puntuados positivamente por el usuario u e i_v por el usuario v .

$$\operatorname{sim}(u, v) = \frac{|i_u \cap i_v|}{|i_u| + |i_v| - |i_u \cap i_v|} \quad (3.4)$$

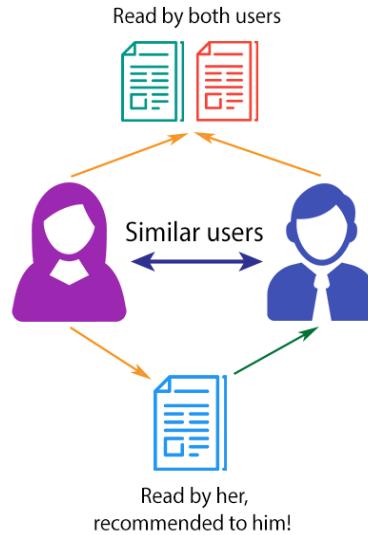


Figura 3.1: Esquema del funcionamiento de u-knn [Liao, 2018]

Una vez se conocen los k usuarios más similares, se recorren todos los ítems puntuados positivamente y se hace una suma acumulativa según la fórmula 3.5 para cada uno de esos ítems no puntuados por el usuario al que se le está realizando la recomendación. Se recomienda el ítem que termine con el valor más alto. La implementación que se ha realizado en este trabajo solo contempla las puntuaciones como positivas o negativas. Por tanto, $r_v(i)$ solamente puede tomar los valores 0 y 1 para puntuación negativa y puntuación positiva relativamente. Se puede ver su pseudocódigo en la figura 3.4.

$$r_u(i) = \sum_{v \in knn(u)} sim(u, v) \cdot r_v(i) \quad (3.5)$$

```

input : usuario  $u$ 
output: item recomendado para el usuario
1 for vecino  $v$  in getKVecinos( $u$ ) do
2   for item  $i$  in itemsPositivos( $v$ ) do
3     if getRating( $v, i$ ) > threshold then
4        $r(i)$  += calcularSimilitud( $u, v$ )
5     end
6   end
7 end
8 return max( $r$ )

```

Algoritmo 3.4: Pseudocódigo de user-based knn.

3.4. Métricas de novedad y diversidad

El principal objetivo de un algoritmo de recomendación es predecir los gustos de los usuarios. Sin embargo, se puede tener una visión más allá del simple acierto y por eso se buscan métricas que midan parámetros interesantes como la novedad y la diversidad.

La novedad se entiende como la diferencia entre experiencias pasadas y presentes del usuario mientras que la diversidad se refiere a las diferencias internas en las partes de una experiencia [Castells et al., 2015]. Un sistema muy novedoso tenderá a mostrarnos elementos que nunca se nos hayan mostrado. Por otro lado, un sistema muy diverso nos mostrará elementos diferentes entre ellos, por ejemplo, distintos géneros musicales. En el caso de la diversidad se necesitan conocer características internas de los elementos, en el caso de este trabajo, géneros de películas.

Con el propósito de medir y comparar la novedad y diversidad de los distintos algoritmos se utilizan una serie de métricas que se explican en esta sección. En la notación, i y j se usan para denotar ítems, R todos los ítems recomendados e I el conjunto total de ítems.

3.4.1. Novedad

La novedad nos muestra en qué medida se recomiendan ítems más o menos populares. Se calcula la media de $1 - \text{popularidad}(i)$ para todos los ítems recomendados para que la métrica sea mejor cuanto más alta sea su valor como se observa en 3.6.

$$\text{Novedad} = \frac{1}{|R|} \sum_{i \in R} \left(1 - \frac{|i^+|}{|i|}\right) \quad (3.6)$$

Donde $|i^+|$ es el número de usuarios a quienes les gusta el ítem i y $|i|$ el número total de usuarios cuyo gusto por i se conoce.

3.4.2. Intra-List Diversity

ILD se define como la media de las distancias de los ítems recomendados par a par. Para poder calcularlo hace falta definir una manera de medir distancias entre ítems [Castells et al., 2015]. Generalmente estas distancias se calculan en función a las características de los ítems. En el caso de este trabajo se utilizan los géneros proporcionados en la base de datos y la similitud Jaccard para resolver este problema, que es apropiada cuando se comparan características binarias.

Para que la métrica nos muestre el comportamiento de los algoritmos una vez ya han realizado un gran número de recomendaciones se utiliza ILD@k. Esto quiere decir que en vez de calcular la

distancia de todos los ítems par a par, solamente se calcula para los últimos k ítems recomendados. En el caso de este trabajo se va a utilizar ILD@10.

$$ILD = \frac{1}{|R|(|R|-1)} \sum_{i \in R} \sum_{j \in R} d(i, j) \quad (3.7)$$

3.4.3. Unexpectedness

Unexpectedness se define como la media de las distancias del último ítem recomendado en cada iteración con el resto de ítems recomendados 3.8. Intuitivamente nos indica cuan diferentes son las características de los ítems que se están recomendando respecto a todas las recomendaciones pasadas. Cuanto mayor sea la métrica mayor novedad y por tanto más valor se aporta al usuario [Castells et al., 2015].

De la misma manera y por la misma razón que en ILD, tomamos Unexpectedness@10 y la similitud Jaccard para el cálculo de las distancias.

$$Unexpectedness = \frac{1}{|R||I_u|} \sum_{j \in R} \sum_{i \in I_u} d(i, j) \quad (3.8)$$

Donde I_u son los ítems con los que u ha interaccionado.

3.4.4. Coeficiente de Gini

El coeficiente de Gini es una medida de desigualdad 3.9. En el caso de este trabajo en concreto, un coeficiente de Gini bajo nos indicaría que muchos ítems se recomiendan un poco mientras que un valor alto nos indicaría que muy pocos ítems se recomiendan mucho. Toma un valor entre 0 y 1 [Castells et al., 2015].

$$Gini = \frac{1}{|I|-1} \sum_{k=1}^{|I|} (2k - N - 1)p(i_k) \quad (3.9)$$

Donde $p(i_k)$ es la probabilidad de que se recomiende el k-ésimo ítem menos recomendado.

EXPERIMENTOS Y RESULTADOS

En esta sección se van a mostrar y describir los experimentos realizados con los algoritmos anteriormente explicados. Primero se va a hacer una búsqueda de hiperparámetros para cada uno de los algoritmos y luego se hace una comparativa entre ellos con esos hiperparámetros. Por tanto, el objetivo principal de esta sección es coger la configuración óptima de cada algoritmo para luego compararlos unos con otros.

4.1. Búsqueda de hiperparámetros

La búsqueda de parámetros se define como la práctica de ajustar los parámetros de los algoritmos cuyo valor se determina antes de que el proceso de entrenamiento comience. En este caso se refiere a las variables ε para ε -greedy, γ para UCB, α_0 , β_0 y κ para Thompson sampling y k para user-based knn. Se observan los resultados en dos gráficas: número total de aciertos y media de 1-popularidad (novedad).

Se ha utilizado como técnica de ajuste de hiperparámetros la búsqueda por rejilla que consiste en hacer una búsqueda no aleatoria en el espacio de posibles valores de las variables. En este caso se han escogido valores arbitrarios que se consideraban que podían tener un resultado relevante para los experimentos. Para escoger los valores de los atributos para cada algoritmo en la comparación final se escogerán aquellos valores que resulten en un acierto mayor.

Con la finalidad de evitar posibles confusiones se han particionado los datos de MovieLens 1M en dos divisiones. La primera división incluye el 20 % de los datos y se va a utilizar para optimizar los hiperparámetros de los algoritmos. En la segunda división se incluyen el 80 % restante de los datos y solamente se utilizará para la comparativa final de los algoritmos.

4.1.1. ε -greedy

La búsqueda de hiperparámetros para esta estrategia involucra al valor de ε el cual solamente puede estar contenido en el intervalo [0,1]. El comportamiento esperable de este algoritmo es trivial.

Cuanto mayor sea el valor de ϵ , más probable será que se recomiende un ítem aleatorio y por tanto más novedad y menos acierto.

Los resultados que se muestran en la Figura 4.1 nos revelan el comportamiento esperado. Por lo general, para valores más cercanos a 0, mayor acierto se consigue y menor novedad.

Sin embargo, hay resultados que no cumplen estrictamente la regla. Si observamos en la figura de aciertos el rendimiento del algoritmo para $\epsilon=0.1$ vemos que obtiene un resultado mejor que para $\epsilon=0.0$ y $\epsilon=0.01$. Esto puede ser debido a que el algoritmo se beneficie del factor exploratorio ya que, si habiendo un ítem mejor que el que cree ser el óptimo, pero no lo explora, no podrá optimizar los resultados. En la gráfica de novedad podemos observar un comportamiento equivalente y por el mismo motivo.

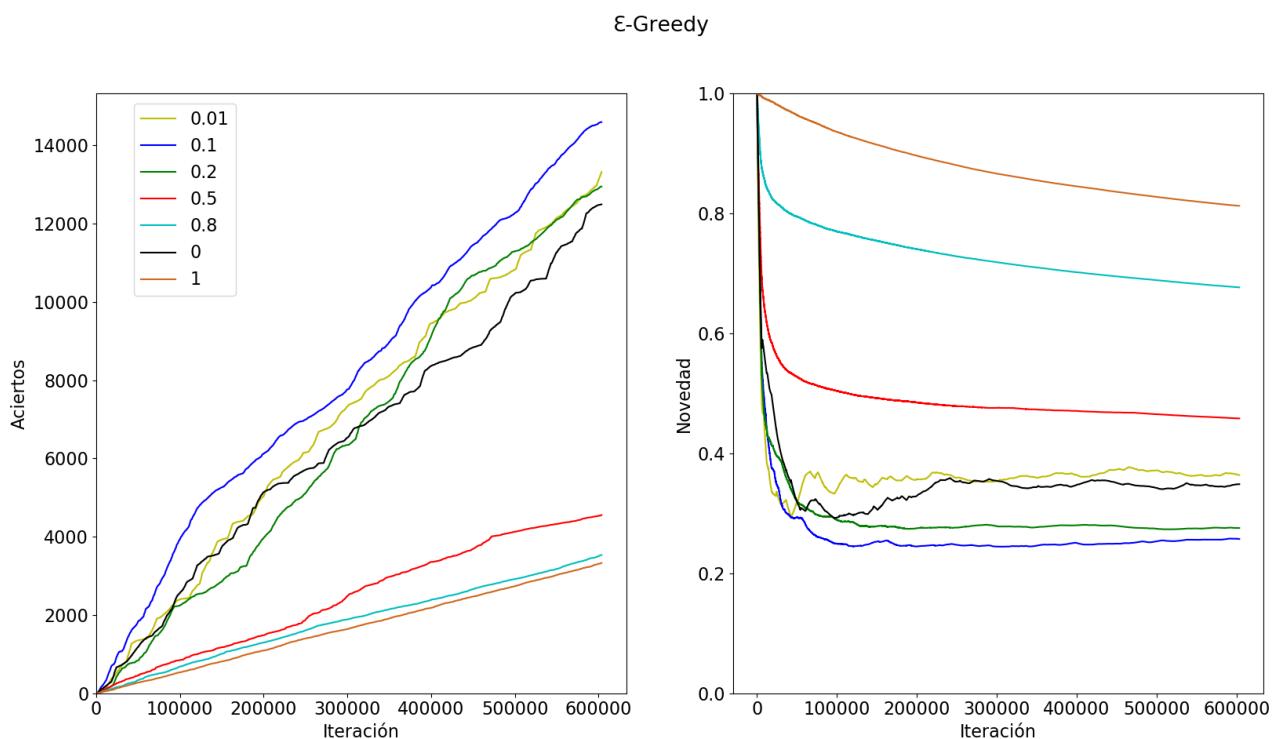


Figura 4.1: Comparativa de aciertos y novedad para distintos valores de ϵ .

4.1.2. Upper Confidence Bound

La búsqueda de hiper parámetros para esta estrategia involucra el valor de γ . γ puede tomar cualquier valor mayor o igual a 0 y representa cuanta importancia se le quiere dar al término exploratorio de la ecuación. Cuanto mayor es el valor de γ mayor importancia cobra el aspecto exploratorio.

Los resultados se muestran en la Figura 4.2. En el caso de UCB los resultados no son tan triviales. Teóricamente cuanto menor sea γ más acierto y menos novedad. Esto se puede observar claramente

en el caso de $\gamma=0.01$. Sin embargo, también hay resultados que presentan comportamientos no intuitivos, por ejemplo, $\gamma=10$ debería sacar un acierto menor a $\gamma=0.1$ y una novedad más alta. No obstante, los resultados muestran un comportamiento contrario, estando $\gamma=0.01$ mucho más cerca de los resultados de $\gamma=10$ que de $\gamma=0.1$. Esto puede ser debido a que si se le da suficiente tiempo, el haber dado más prioridad a la exploración comienza a merecer la pena aunque en los inicios no de tan buenos resultados como $\gamma=0.1$.

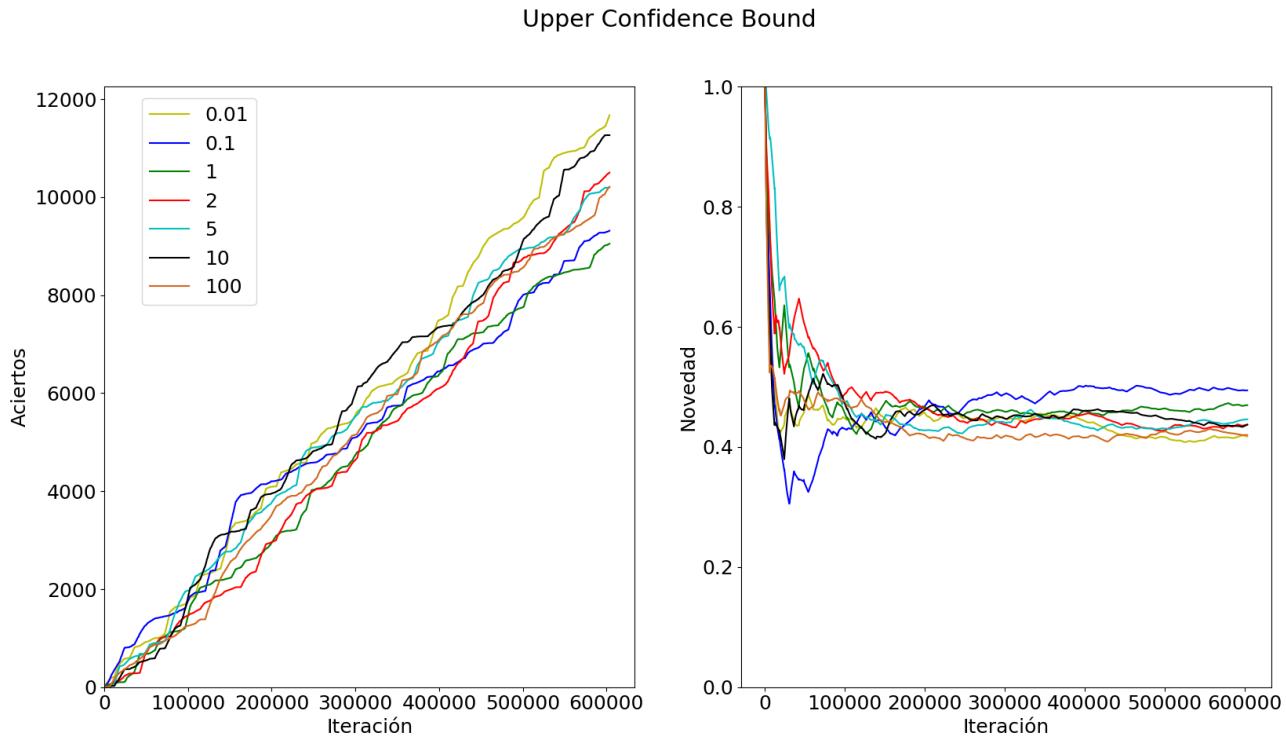


Figura 4.2: Comparativa de resultados para distintos valores γ .

4.1.3. Thompson Sampling

En el caso de Thompson sampling son 3 los parámetros que podemos modificar. Los dos primeros son α_0 y β_0 los cuales indican cuán optimista o pesimista van a ser las evaluaciones del algoritmo. κ indica cada cuantas iteraciones los valores de la distribución van a ser actualizados.

Los resultados que se muestran en la Figura 4.3 son los resultados que comparan distintos valores de α_0 y β_0 manteniendo fijo $\kappa=1$. Se quiere observar los resultados del algoritmo base actualizándose en cada iteración para luego comparar tiempos de ejecución, acierto y novedad con valores distintos de κ .

Cuando α_0 es mayor o igual que β_0 se observa que el algoritmo pasa a tener un comportamiento parecido a aleatorio. En el caso contrario, cuando β_0 es mayor o mucho mayor que α_0 el algoritmo

presenta resultados adecuados. Podemos ver que entre 1-20, 1-50, 1-100 y 1-200 no hay una diferencia enorme ni en término de aciertos ni en término de novedad. Sin embargo 1-100 es el que mayor aciertos nos da y es el que se va a utilizar para los próximos experimentos.

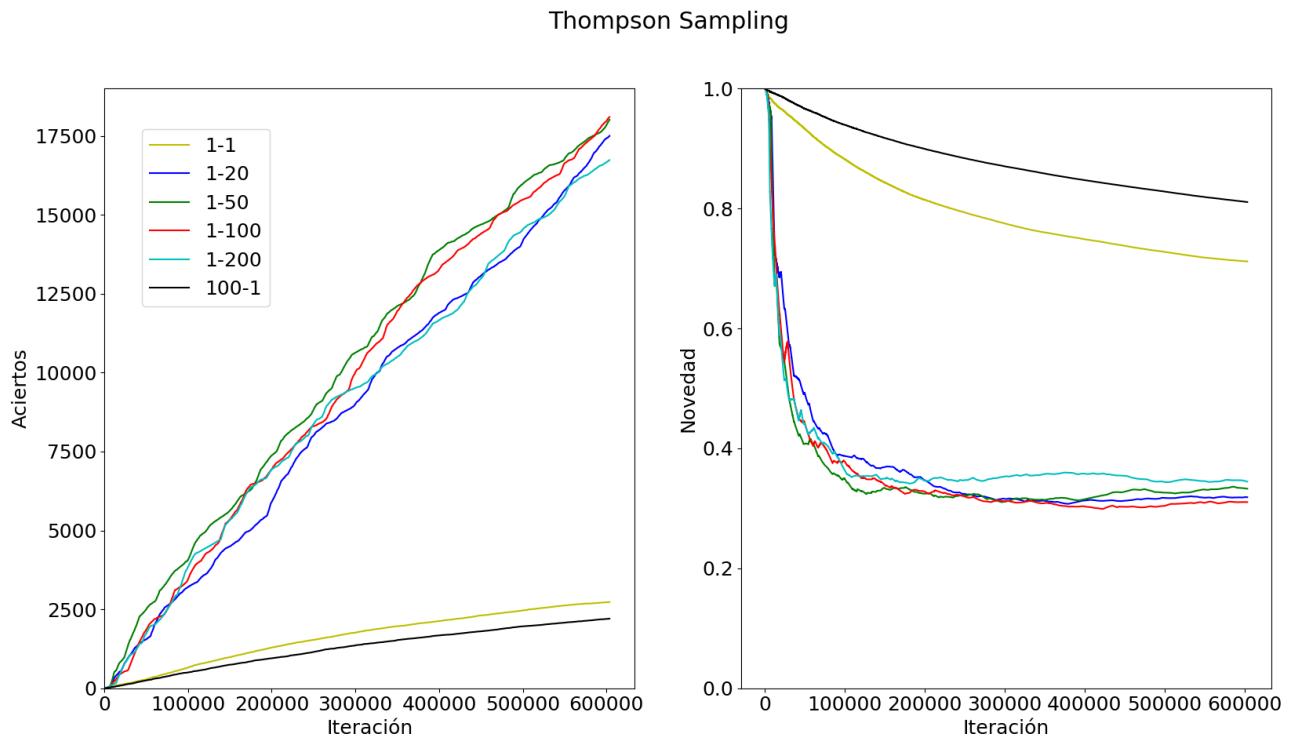


Figura 4.3: Comparativa de aciertos y novedad para distintos valores α_0 y β_0 .

Por otro lado, en la Figura 4.4 se muestran los resultados de modificar κ habiendo fijado los valores de α_0 y β_0 a 1-100. En este caso es esperable que cuanto mayor sea el valor de κ peores resultados se obtendrán puesto que la actualización de la información conocida se hace menos frecuentemente. Sin embargo, aunque la tendencia si es esa, podemos ver que con valores como $\kappa=2$ y $\kappa=5$ se consigue más acierto que con el caso original. No solamente se consiguen mejores aciertos si no que mayor novedad.

Estos resultados son sorprendentes ya que si observamos la Figura 4.5, en la cual se muestra el tiempo en segundos de cada una de estas ejecuciones, se puede observar como con $\kappa=2$ el algoritmo tarda aproximadamente la mitad y con $\kappa=5$ menos de la cuarta parte. Es decir, se obtienen mejores resultados de acierto y novedad en un tiempo de ejecución considerablemente menor.

4.1.4. User-based Knn

En user-based knn (u-knn) el único parámetro que se puede modificar es el número de vecinos que observamos. Intuitivamente podemos predecir que cuanto mayor sea k mejores resultados obten-

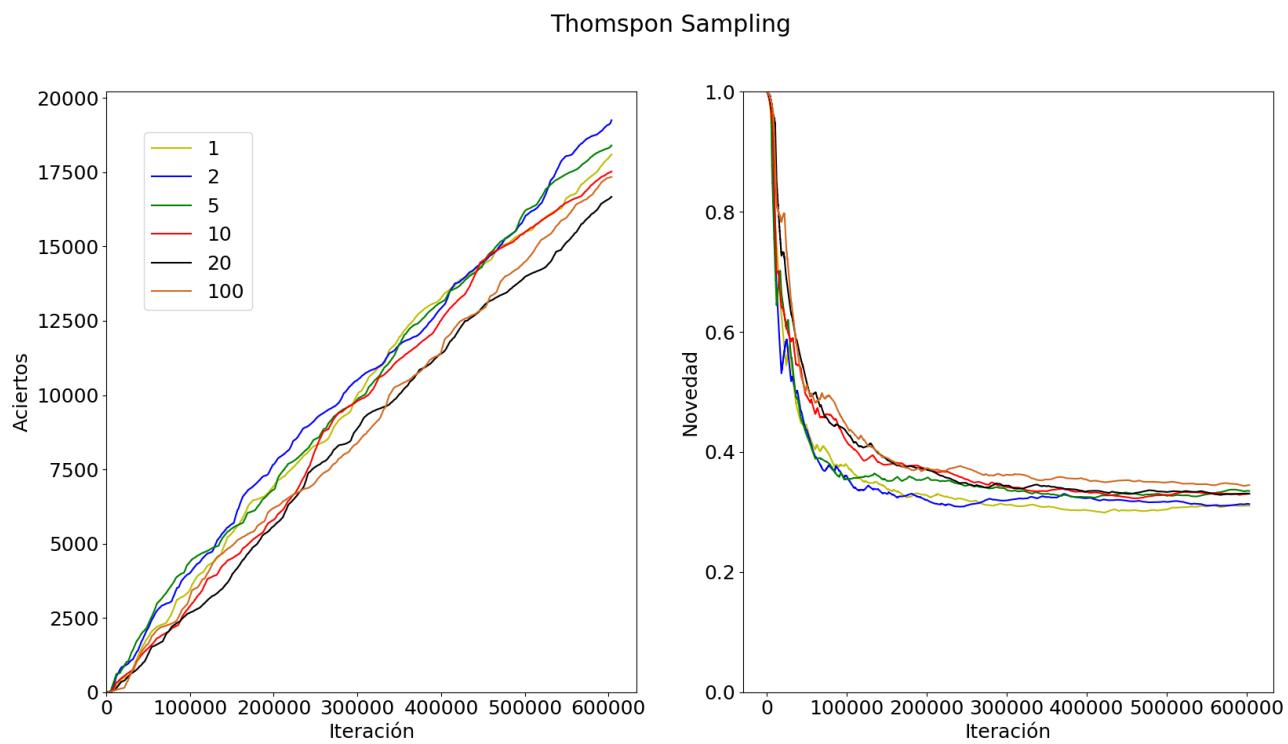


Figura 4.4: Comparativa de aciertos y novedad para distintos valores de κ .

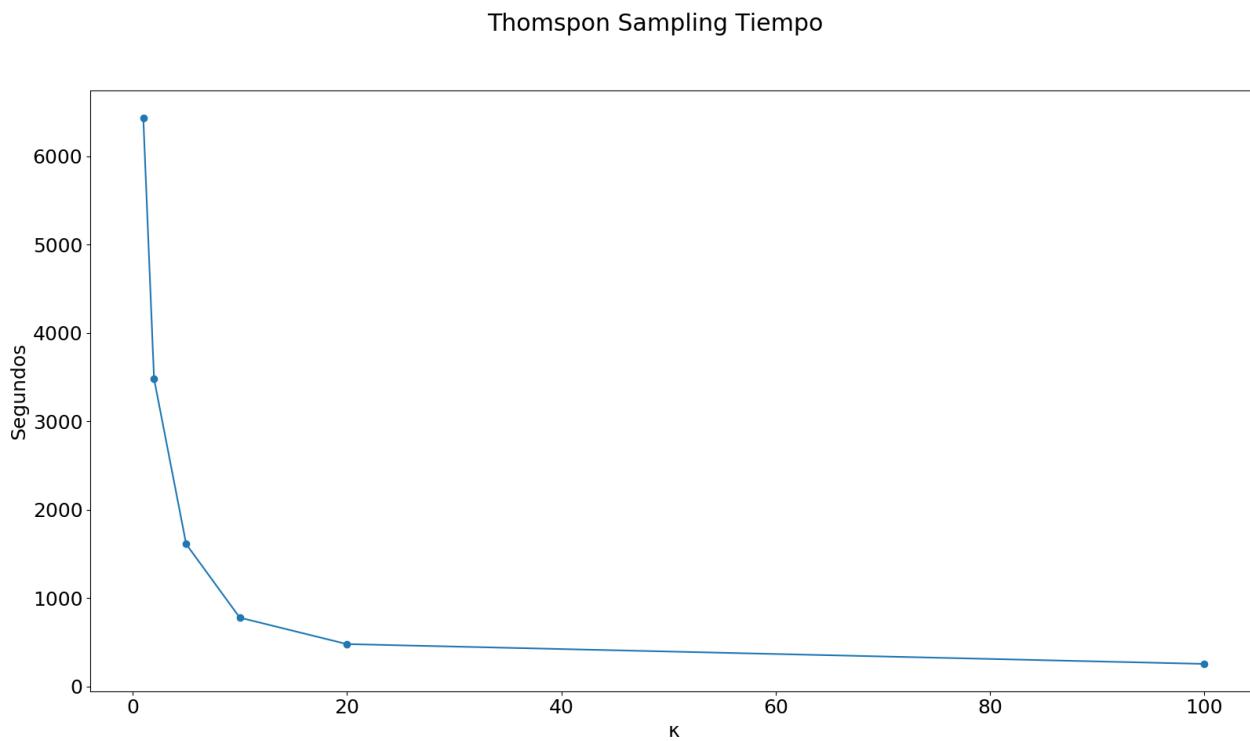


Figura 4.5: Comparativa de tiempos para distintos valores de κ .

dremos porque el sistema observa más información, aunque más costosa será la recomendación en términos de tiempo y computación.

Se puede observar en la Figura 4.6 que efectivamente los resultados confirman la intuición. Idealmente el algoritmo podría explorar todos los demás usuarios y encontrar el ítem idóneo basado en su propia fórmula para encontrarlo. Sin embargo, sería tremadamente costoso y por ello probaremos valores relativamente bajos. El mejor valor de los probados es 10 vecinos que se utilizará más adelante como comparación. Es un resultado muy cercano al conseguido con el algoritmo ε -Greedy para $\varepsilon=0$ ya que ambas son estrategias puramente codiciosas y por tanto $k=10$ es un valor interesante para probar.

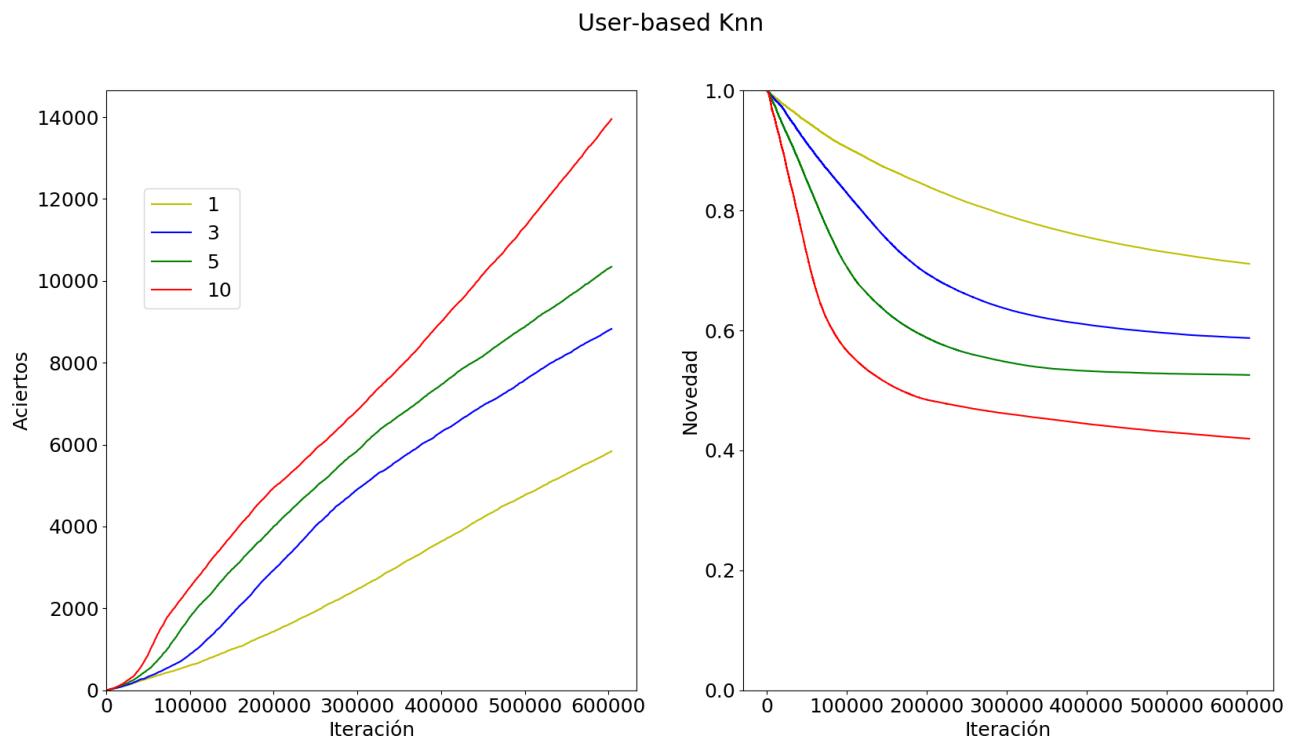


Figura 4.6: Comparativa de aciertos y novedad para distintos valores de k .

4.2. Comparativa de algoritmos

En esta sección se presentan los resultados de haber cogido los algoritmos anteriores con sus mejores hiper parámetros con la finalidad de poder evaluar comparativamente su comportamiento para distintas métricas. Para este experimento se ha utilizado la división del 80 % de los datos de MovieLens 1M anteriormente particionada. Al igual que en las pruebas anteriores se han iterado todos los usuarios 100 veces

Aparte de las gráficas de acierto y novedad que se han utilizado en los experimentos anteriores, para esta prueba se han añadido las nuevas métricas. Para las distancias entre ítems en Unex@10 e

ILD@10 se ha utilizado el índice de Jaccard.

En la Tabla 4.1 se muestran los datos resultantes del experimento. Las filas indican la métrica y las columnas el algoritmo. El algoritmo con el mejor valor viene coloreado del verde más oscuro y en negrita, a partir de ahí, cuánto más claro el color, peor es el algoritmo en esa métrica. ILD@10, Unex@10, Novedad y Acierto cuánto mayores son, mejor, mientras que Gini y tiempo peor.

Es lógico pensar que la recomendación aleatoria sea la que mejores valores tenga en métricas de novedad y diversidad. Realizar una recomendación aleatoria es trivial y por tanto tendría sentido pensar que también es la más rápida. Hasta cierto punto esta intuición se cumple ya que la recomendación aleatoria es mucho más rápida que las demás, el coeficiente de Gini es el más bajo por mucha diferencia y la novedad también muy alta. Sin embargo, tanto u-knn como *Thompson Sampling* superan ligeramente a la recomendación aleatoria en Unex@10 e ILD@10. Además, el acierto de la recomendación aleatoria es muy pequeño por lo que no es una estrategia que tenga sentido utilizar en el mundo real.

Por otro lado, el algoritmo con mayor acierto es *Thompson Sampling*. Es un algoritmo que además de acertar considerablemente más que el resto, presenta unos valores de Unex@10 e ILD@10 altos. No obstante, su novedad es muy baja y su coeficiente de Gini muy alto, es decir, recomienda muchas veces pocos ítems y muy pocas veces el resto de ítems, se centra en los más populares. Que ocurra este choque entre valores buenos de Unex@10 e ILD@10 y malos de novedad y Gini quiere decir que se están recomendando muy pocos ítems del total pero que estos ítems son poco similares entre ellos. Podría ser interesante utilizar esta estrategia si se prioriza el acierto.

Otras estrategias más equilibradas son greedy (ε -greedy para $\varepsilon=0$), u-knn y UCB. Greedy y u-knn tienen la misma filosofía de recomendación la cual es puramente codiciosa. Como es de esperar, ambas tienen un comportamiento muy parecido. Para esta prueba en concreto podríamos concluir que Greedy es mejor que u-knn ya que obteniendo un acierto ligeramente superior y un tiempo considerablemente menor no es perjudicado en las demás métricas respecto a u-knn. A pesar de esto, u-knn tiene aún margen de mejora aumentando el número de vecinos que explora. Por otra parte, UCB tiene un comportamiento ligeramente distinto relativo a los otros algoritmos. Mientras que greedy y u-knn obtienen unos resultados buenos en Unex@10 e ILD@10 y algo peores en Gini y novedad, UCB muestra lo contrario. Si greedy y u-knn están recomendando poca variedad de ítems, pero distintos entre sí (Unex@10 e ILD@10 altos y Gini y novedad bajos), UCB está recomendando ítems parecidos entre sí, pero con más variedad respecto al conjunto total de ítems (Unex@10 e ILD@10 bajos y Gini y novedad altos). Además, UCB obtiene un acierto levemente inferior.

ε -greedy, aunque teniendo buenos valores de acierto y tiempo de ejecución, se queda corto en métricas de diversidad. Quizás utilizando un ε más alto se podrían equilibrar algunas de las métricas a cambio de pérdida en el acierto. De todas formas, ε -greedy puede ser interesante en casos en los que el acierto y el tiempo de ejecución sean los factores a tener en cuenta.

Cabe decir que, al igual que se ha dicho antes para ε -greedy, los hiper parámetros de los algoritmos podrían ajustarse para priorizar algunas métricas en concreto sacrificando aciertos. Este estudio podría ser una ampliación de este trabajo para estudiar más en profundidad cada detalle de cada estrategia con el fin de priorizar algún comportamiento específico.

	EGreedy0.1	Greedy	KNN10	Random	Thompson1-100-2	UCB0.01
ILD@10	0.725	0.748	0.75	0.747	0.757	0.693
Unex@10	0.81	0.84	0.84	0.83	0.845	0.834
Gini	0.11	0.11	0.111	0.023	0.162	0.108
Novedad	0.222	0.379	0.372	0.718	0.292	0.404
Acierto	0.093	0.079	0.077	0.021	0.1234	0.073
Tiempo	530s	476s	3380s	40s	3897s	528s

Tabla 4.1: Tabla de métricas de acierto, tiempo de ejecución, novedad y diversidad para los distintos algoritmos.

Comparativa Algoritmos

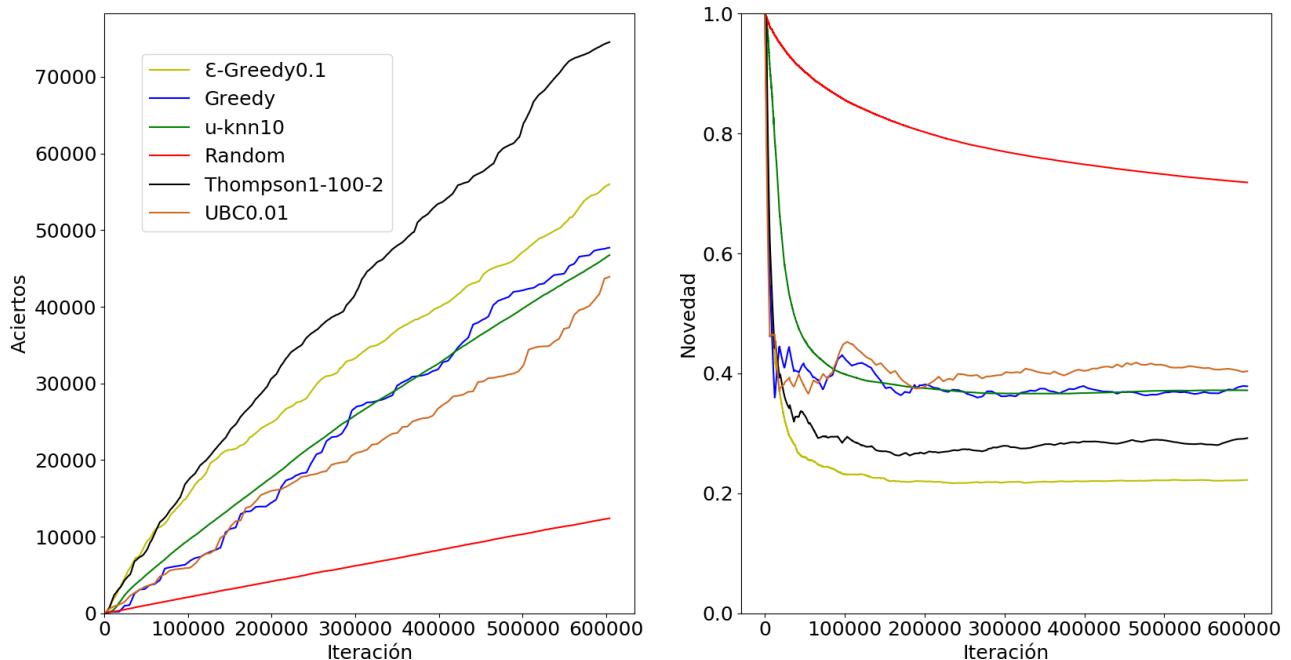


Figura 4.7: Comparativa de aciertos y novedad para los mejores hiperparámetros de todos los algoritmos anteriores.

CONCLUSIONES Y TRABAJO FUTURO

En esta sección se presentan las conclusiones que se han sacado en este trabajo junto con un esquema general de lo que se ha realizado. Además, se muestran las posibles ampliaciones y continuaciones que puede tener un trabajo que persiguiera continuar la investigación que realiza este.

5.1. Conclusión

En este trabajo de fin de grado se han implementado unos algoritmos de recomendación basados en bandidos multibraco y se han comparado sus rendimientos en distintas métricas de novedad y diversidad. Para realizar el experimento se ha utilizado una base de datos que simula el comportamiento de un sistema *online* en la que usuarios puntuán ítems. Estas pruebas nos permiten observar su rendimiento desde un sistema en estado virgen hasta un sistema ya entrenado.

Más concretamente, han sido implementados en el lenguaje de programación Python *ε-greedy*, *Upper Confidence Bound*, *Thompson Sampling* y *User-Based knn* además del entorno del sistema de recomendación. Las métricas que se han utilizado han sido: acierto, novedad, *intra-list diversity*, *unexpectedness*, coeficiente de Gini y tiempo de ejecución. Para realizar esta implementación se ha desarrollado un trabajo de investigación previo tanto teórico como de algoritmos ya implementados.

Se pueden dividir los experimentos en dos aspectos distintos. Primero se ha realizado la búsqueda de hiperparámetros. Las conclusiones que se sacan de estos experimentos son claras. En general, los algoritmos que se han probado pueden llegar a tener comportamientos totalmente distintos según su configuración de hiperparámetros. No es sorprendente, pero sí importante darse cuenta de esto para saber que antes de poder utilizar estos algoritmos hay que hacer una optimización de parámetros. Un algoritmo que dé buenos resultados en distintas métricas puede llegar a ser nefasto en muchas o incluso todas si los parámetros de entrada no tienen valores adecuados. No solamente un algoritmo puede pasar de tener un comportamiento satisfactorio a tener un comportamiento muy subóptimo si no que un mismo algoritmo puede adaptar comportamientos distintos. Es decir, un mismo algoritmo en determinadas situaciones puede servir para maximizar determinados aspectos de la recomendación como el acierto mientras que cambiando sus hiperparámetros puede, manteniendo un acierto

considerablemente alto, optar por ser más diverso según estos valores.

El segundo experimento que se ha realizado compara los algoritmos entre ellos. Las conclusiones que se pueden sacar de estos experimentos son variadas. Lo primero que nos muestran los resultados es que no hay ningún algoritmo que sea óptimo en todos los aspectos. Cada uno tiene sus ventajas y desventajas. Si queremos maximizar el valor de alguna de las métricas tiene que ser a costa de otras. Se puede concluir por tanto que las métricas que se han utilizado están de cierta manera relacionadas entre ellas. Sin embargo, sí hay algoritmos que destacan en alguna métrica. Por ejemplo *Thompson Sampling* obtiene muy buenos valores en acierto, ILD@10 y Unex@10 mientras que en coeficiente de Gini y novedad obtiene los peores. Por tanto, dependiendo de las preferencias del gestor del sistema pueden interesar determinados algoritmos o no.

No es posible tener un sistema de recomendación perfecto ya que a día de hoy los seres humanos son, en mucha medida impredecibles. No obstante, la investigación y el progreso en nuevas y mejores técnicas permite que estos sistemas puedan ser cada vez más útiles y por tanto es un ámbito en el que merece la pena ahondar.

5.2. Trabajo futuro

Existen bases de datos de mayor tamaño que la utilizada en este trabajo. Puede ser interesante ampliar los experimentos de este trabajo con un volumen de datos mayor. La base de datos de MovieLens 1M es frecuentemente utilizada en el ámbito de recomendación y por tanto para este trabajo es suficiente, pero se puede hacer un estudio más exhaustivo utilizando, por ejemplo, MovieLens 20M u otras más grandes. Esta ampliación en el tamaño de datos podría venir de la mano de un cambio de lenguaje de programación utilizado ya que Python no es especialmente rápido.

Además, existen otros algoritmos de aprendizaje por refuerzo (y modificaciones de los ya utilizados) que no se han implementado en este trabajo y que pueden ser interesantes de implementar y evaluar. De la misma manera las métricas que se han utilizado tienen variaciones y existen muchas otras métricas que dependiendo del problema que se proponga pueden tener interés.

BIBLIOGRAFÍA

- [Auer et al., 2002] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multi-armed bandit problem. *Machine learning*, 47(2):235–256.
- [Castells et al., 2015] Castells, P., Hurley, N. J., and Vargas, S. (2015). *Novelty and Diversity in Recommender Systems*, pages 881–918. Springer.
- [Castells and Sanz-Cruzado, 2019] Castells, P. and Sanz-Cruzado, J. (2019). [kNNBandit](#).
- [Chapelle and Li, 2011] Chapelle, O. and Li, L. (2011). An empirical evaluation of thompson sampling. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 2249–2257. Curran Associates, Inc.
- [Ek and Strigsson, 2015] Ek, F. and Strigsson, R. (2015). Recommender systems; contextual multi-armed bandit algorithms for the purpose of targete advertisemente within e-commerce.
- [Figuerola et al., 2004] Figuerola, C. G., Alonso, J. L., Zazo, A. F., and Rodríguez, E. (2004). Algunas técnicas de clasificación automática de documentos.
- [Galbraith, 2016] Galbraith, B. (2016). [Python library for Multi-Armed Bandits](#).
- [Liao, 2018] Liao, K. (2018). Prototyping a Recommender System Step by Step Part 1: KNN Item-Based Collaborative Filtering.
- [Movielens, 1997] Movielens (1997). [Movielens 1M](#).
- [Russo et al., 2018] Russo, D. J., Roy, B. V., Kazerouni, A., Osband, I., and Wen, Z. (2018). A tutorial on thompson sampling.
- [Sanz-Cruzado et al., 2019] Sanz-Cruzado, J., Castells, P., and López, E. (2019). A simple multi-armed nearest-neighbor bandit for interactive recommendation. In *13th ACM Conference on Recommender Systems (RecSys 2019)*, pages 358–362, Copenhagen, Denmark.
- [Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

