ISSN 1688-2806

Universidad de la República
Facultad de Ingeniería

Aplicación de métodos de clustering en sistemas de recomendaciones

Tesis presentada a la Facultad de Ingeniería de la Universidad de la República por

Diego Mazzuco

en cumplimiento parcial de los requerimientos para la obtención del título de Magister en Ciencia de Datos y Aprendizaje Automático.

Director de Tesis

Ignacio Ramirez ..... Universidad de la República

Tribunal

Matias Carrasco ..... Universidad de la República
Marcelo Fiori ..... Universidad de la República
Mario González ..... Universidad de la República
Ignacio Ramirez ..... Universidad de la República

Director Académico

Federico Lecumberry ..... Universidad de la República

Montevideo
lunes 19 mayo, 2025

Aplicación de métodos de clustering en sistemas de recomendaciones, Diego Mazzuco.

ISSN 1688-2806

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 90 páginas.
Compilada el lunes 19 mayo, 2025.
http://iie.fing.edu.uy/

Agradecimientos

Me gustaría aprovechar este espacio para expresar mi profundo agradecimiento a mi familia y amigos por su inquebrantable apoyo durante la realización de este trabajo. También quiero extender mi gratitud a todas las personas que han dedicado tiempo para leerlo y discutir ideas conmigo. Su participación y retroalimentación han sido invaluables en este proceso.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

Desde hace siglos, las empresas se han enfrentado a la difícil decisión de qué recomendar a sus clientes. Este desafío se ha intensificado en las últimas décadas con la llegada del internet, ya que los negocios ahora tienen acceso a una gran cantidad de información sobre los usuarios y desean aprovecharla para realizar las mejores recomendaciones posibles. Con el objetivo de aumentar los ingresos y atraer a más consumidores potenciales. Esta dinámica, tan común como compleja, plantea el desafío de determinar las mejores recomendaciones posibles.

Un ejemplo simple pero revelador de esta complejidad se encuentra en la elección de un producto, donde optar por uno implica necesariamente no elegir otros. En este contexto, la literatura ofrece diversas estrategias para abordar este dilema de elección. Sin embargo, este trabajo se enfocará en los algoritmos de Multi-armed bandit, proponiendo un nuevo algoritmo llamado CLinUCB, el cual utiliza el contexto del usuario y las opciones disponibles para ofrecer recomendaciones dinámicas que se alinean con el estado del arte en este campo.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Tabla de contenidos

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 1

Introducción

Realizar una recomendación entre una serie de opciones es un desafío no trivial que la literatura ha abordado desde diversos ángulos, ofreciendo múltiples soluciones. Una de ellas es el enfoque de los *Multi-Armed Bandits* (MAB) [1], inspirado en un problema ilustrativo donde un individuo en un casino enfrenta la incertidumbre de no saber cuál máquina tragamonedas (o 'One-Armed Bandit') ofrece la mejor recompensa, con el objetivo de maximizar sus ganancias de manera eficiente. Esta problemática se extrapola a diversos escenarios, como la recomendación de productos, vuelos, hoteles, restaurantes, publicidad y estrategias de marketing, o cualquier situación en la que se deba elegir entre diferentes acciones.

En esta tesis, realizaremos un análisis exhaustivo de los algoritmos de MAB, diseñados para equilibrar la exploración de nuevas opciones y la explotación del conocimiento adquirido. Por ejemplo, al elegir un restaurante, una estrategia podría ser visitar uno diferente cada semana para determinar cuál es el mejor. Sin embargo, esta estrategia podría hacer que se pierda la oportunidad de disfrutar nuevamente de una experiencia en un restaurante conocido y apreciado. Por otro lado, optar siempre por el mismo restaurante ignora la posibilidad de que otros puedan ofrecer una experiencia aún mejor. Este dilema, conocido como el paradigma de la exploración versus explotación, es fundamental en los algoritmos que hemos analizado en este trabajo.

1.1. Antecedentes

En la literatura, el término "Multi-Armed Bandit" [1] se refiere a cualquier problema en el que se busca maximizar una recompensa r a lo largo de un número determinado de iteraciones T. Para lograr esto, el algoritmo en cada iteración t selecciona una acción a de un conjunto de K opciones, cada una asociada a una función de densidad de probabilidad F_a para dicha recompensa. La recompensa se determina en el momento de la elección, y no se dispone de información previa sobre el valor esperado de la recompensa para cada acción a ni sobre sus distribuciones correspondientes.

Una característica central de estos algoritmos es que la exploración de acciones

a no óptimas conlleva un costo, lo que hace necesario desarrollar una estrategia para determinar qué opciones explorar y en qué medida hacerlo.

Varios algoritmos han abordado el problema del equilibrio entre exploración y explotación en la toma de decisiones. Ejemplos de ello son *Exploración Uniforme* y *Epsilon Decay* [1], los cuales definen desde el principio cuánto explorar para alcanzar una decisión óptima. Sin embargo, estos enfoques no consideran los resultados obtenidos durante el proceso, lo que llevó al desarrollo de métodos adaptativos como UCB [1]. Este último ajusta el nivel de exploración en función de las recompensas observadas, descartando aquellas opciones que generan bajas recompensas.

Posteriormente, surgieron los métodos contextuales, como LinUCB [2], que incorporan datos contextuales sobre las acciones. Este enfoque intenta identificar un vector para cada usuario que describa las acciones que toman en función de ciertos contextos, permitiendo así obtener mayores recompensas.

En varios escenarios, surgió la necesidad de agrupar estos vectores, lo que llevó al desarrollo de CLUB [3]. Este algoritmo asume que todos los vectores pertenecen a un grafo completamente conectado y, a medida que se obtienen recompensas, elimina las conexiones que superan un umbral específico. SCLUB [4], una mejora de CLUB, permite la fusión de clusters si estos se acercan lo suficiente.

Finalmente, dLinUCB [5] es una modificación de LinUCB que se adapta a cambios en el valor esperado de la recompensa, permitiendo que el algoritmo reaprenda cuál es la opción óptima en entornos dinámicos. La siguiente iteración, CodBand [6], combina la capacidad de adaptación cuando el valor esperado cambia con la posibilidad de hacer clustering, asumiendo que cada cluster describe una distribución y calculando la probabilidad de que un usuario pertenezca a un cluster, asignándolo al que tenga la mayor probabilidad.

1.2. Contribuciones

En este trabajo, resumimos los diferentes métodos de Multi-Armed Bandits (MABs) [1] y presentamos algunas demostraciones propias, como las cotas para los algoritmos *Greedy* y *Epsilon Decay*. Además, realizamos simulaciones para evaluar el rendimiento de los métodos discutidos. Proponemos un nuevo algoritmo, denominado ClinUCB (por Clustering y LinUCB), que obtuvo resultados comparables con los del estado del arte, como CLUB [3], SCLUB [4], dLinUCB [5] y CodBand [6]. Este algoritmo introduce mejoras, como la reducción del tiempo de cálculo de LinUCB mediante mínimos cuadrados recursivos y la incorporación de un parámetro de olvido a ClinUCB, lo que permite detectar cambios en el entorno, similar a lo que logran dLinUCB [5] y CodBand [6]. Finalmente, probamos estas afirmaciones con datos simulados y reales, demostrando que ClinUCB ofrece un rendimiento comparable al de otros algoritmos del estado del arte.

1.3. Organización del documento

Resumiendo el contenido de esta tesis, en el Capítulo 2 definimos de forma matemática qué son MAB, sus premisas y cómo se demuestra que un algoritmo es mejor que otro. Además, discutimos las diferentes validaciones existentes para estos algoritmos, comenzando por aquellos que no tienen información previa sobre las opciones o los usuarios.

En el capítulo 3 realizamos simulaciones en diferentes contextos para verificar todo lo afirmado y demostrado en los anteriores. En el capítulo 4 introduciremos el concepto de contexto y cómo esto cambia las bases y las limitaciones de MAB, mejorando las posibilidades de los mismos, pero requiriendo más información.

Luego, en el capítulo 5 propondremos un nuevo algoritmo para solucionar problemas de los anteriores, para luego en el capítulo 6 realizar simulaciones comparativas con otras variantes de LinUCB [2]. Finalmente, en el capítulo 7 se llevarán a cabo pruebas con datos reales utilizando un método que ha demostrado no introducir sesgos, seguido de conclusiones finales en el capítulo 8.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 2

Fundamentos

En este capítulo, presentaremos la definición matemática de varios modelos de Multi-Armed Bandits (MAB), comenzando por los más básicos y avanzando hacia aquellos que han demostrado alcanzar las mejores cotas teóricas dentro del marco que definimos en la siguiente sección.

2.1. Multi Armed Bandit

Formalizando el capítulo anterior, se establecerá el término MAB para referirse a cualquier modelo que tenga que elegir una acción a de una serie de opciones K. En cada cada iteración $t \in \{1, 2, \ldots, T\}$, existe una recompensa $r_t \in \{0, 1\}$, que es i.i.d., con una distribución Bernoulli F_a correspondiente a cada acción a, y el algoritmo realiza lo siguiente:

1. El algoritmo elige un acción a
2. Se selecciona una recompensa aleatoriamente de la distribución F_a
3. Dada esa recompensa, el algoritmo puede (o no) actualizar los parámetros correspondientes para la toma de decisiones futuras.

El objetivo del algoritmo previo es identificar la acción a^* que maximize la recompensa, o lo que es lo mismo, satisfacer la condición:

$$\mu(a^*) = \max_a\{\mu(a)\} = \max_a\{\mathbb{E}(F_a)\} \tag{2.1}$$

siendo $\mu(a)$ el valor esperado de la recompensa de F_a para una acción a particular. A continuación describiremos una serie de algoritmos, que van desde enfoques simples hasta soluciones más elaboradas, con el propósito de abordar esta meta.

2.1.1. Regret

A fin de debatir los algoritmos de manera efectiva, es necesario contar con una base de comparación, ya que decir que la recompensa promedio es 0.10 podría ser

extremadamente positivo o negativo. Por ejemplo, en el caso de la medicina, un medicamento que solo funciona el 10 % del tiempo sería un desastre, mientras que en el caso de acciones en el mercado, acertar el mismo porcentaje entre otras 1000 acciones sería considerado un buen algoritmo. Por eso comúnmente, se emplea lo que se denomina "regret", que se define como la diferencia entre la máxima recompensa potencial y la recompensa efectivamente obtenida:

$$R(T) = \mathbb{E}_{r_1, r_2, \ldots, r_T} \left[\sum_{t=1}^{T} r_t^* - r_t \right], \tag{2.2}$$

donde $R(T)$ es el regret luego de T rondas, r_t^* es la máxima recompensa promedio de cada ronda, r_t es la recompensa obtenida en la ronda t, y T es la cantidad total de rondas. Es relevante destacar que el regret **es una variable aleatoria** debido a su dependencia de la acción a, la cual contiene un componente aleatorio en la mayoría de los métodos que serán analizados. Por lo tanto, cuando se hable de la esperanza del regret, se estará haciendo referencia al valor esperado con respecto a las acciones tomadas:

$$\mathbb{E}[R(T)] = \mathbb{E}_{a_1, a_2, \ldots, a_T}[R(T)] = \mathbb{E}_{a_1, a_2, \ldots, a_T} \left[\mathbb{E}_{r_1, r_2, \ldots, r_T} \left[\sum_{t=1}^{T} r_t^* - r_t \right] \right]. \tag{2.3}$$

Otro aspecto crucial de estudio es la tendencia de $R(T)/T$ cuando T tiende a infinito. Los algoritmos para los cuales esta proporción tiende a cero se denominan de "zero regret". Esta propiedad será analizada en los distintos métodos para determinar si cumplen o no con esta propiedad.

2.2. Algoritmo Greedy

El método Greedy consiste en probar cada acción una sola vez y luego aplicar, hasta el final, la acción que obtuvo mayor recompensa. Para estudiar su comportamiento consideraremos un escenario de dos acciones posibles $a \in \{1, 2\}$ en donde la recompensa asociada a cada acción $a = i$ tiene una distribución de Bernoulli con parámetro p_i, $P(r_t = 1 | a_t = i) = p_i, \forall t = 1, \ldots, T$. En este escenario, la acción óptima es $a_t = 2, \forall t$. Independientemente de ello, el algoritmo utilizará las primeras dos rondas $t = 1, 2$ para explorar las dos opciones, y en las restantes $T - 2$ rondas utilizará la acción que resultó en una mayor recompensa, en caso de empatar se elegirá una acción al azar.

Teniendo lo anterior en consideración, calcularemos el regret esperado (2.3) para este algoritmo:

$$
\begin{aligned}
R(T) &= \mathbb{E}_{r_1,r_2,\ldots,r_T}\left[\sum_{t=1}^{T} r_t^* - r_t\right] \\
&= \mathbb{E}_{r_1,r_2,\ldots,r_T}\left[\underbrace{(r_1^* - r_1) + (r_2^* - r_2)}_{\text{exploracion}} + \underbrace{\sum_{t=3}^{T} r_t^* - r_t}_{\text{explotacion}}\right].
\end{aligned} \tag{2.4}
$$

Para completar el cálculo debe tenerse en cuenta que las acciones $a_3, \ldots, a_T$ están determinadas por el resultado de las primeras dos rondas: si $r_1 > r_2$ tendremos $a_3 = a_4 = \ldots = 1$ y por ende $\mathbb{E}[r_t] = p_1, \forall t = 3, \ldots, T$ y si $r_2 > r_1$, $a_3 = a_4 = \ldots = a = 2$, $\mathbb{E}[r_t] = p_2, \forall t = 3, \ldots, T$ en caso de empate se elegirá una de las dos opciones al azar. Aplicando la definición de regret y calculando la probabilidades de elegir cada una de las opciones el regret será:

$$
R(T) = p_2 - p_1 + (T-2)\left[p_2 - (p_2 P(a = a_2) + p_1 P(a = a_1))\right] \tag{2.5}
$$

$$
\begin{aligned}
P(a = a_1) &= (1 - p_2)p_1 + \frac{p'}{2} \\
P(a = a_2) &= p_2(1 - p_1) + \frac{p'}{2}
\end{aligned}
$$

$$
p' := p_1 p_2 + (1 - p_1)(1 - p_2) \tag{2.6}
$$

Donde $P(a = a_i)$ es la probabilidad de elegir la acción a_i y p' es la probabilidad de empate. Como se puede ver en el anexo A.1 se puede simplificar lo anterior en:

$$
R(T) = \Delta p + (T-2)\frac{\Delta p}{2}\left[1 - \Delta p\right] \tag{2.7}
$$

donde $\Delta p = p_2 - p_1$. Y si $T \gg 2$ se puede simplificar a:

$$
R(T) \approx (T-2)\frac{\Delta p}{2}\left[1 - \Delta p\right]. \tag{2.8}
$$

Es interesante analizar los mínimos y máximos de la función (2.8), ya que nos proporcionan cuando el regret es el mínimo posible, lo cual es el objetivo del algoritmo, así como su peor caso.

Observamos que la función anterior tiene dos raíces, lo que implica que también tiene dos mínimos, dado que por definición el regret es la diferencia entre la máxima recompensa esperada y la obtenida, por ende $R(T) \geq 0$. Estos mínimos ocurren cuando $p_2 = p_1$ y cuando $p_1 + 1 = p_2$, lo cual implica $p_2 = 1$ y $p_1 = 0$, ya que estamos tratando con probabilidades. Esto es coherente, ya que cuando la diferencia entre las probabilidades es mínima, la penalización por una elección incorrecta es prácticamente inexistente, y cuando la diferencia es muy grande, la probabilidad de elegir incorrectamente es muy baja.

En cuanto al máximo, derivando la función, se obtiene que ocurre cuando $\Delta p = 1/2$ o $p_2 = p_1 + 1/2$. Este es un máximo global, dado que la función es un polinomio de segundo grado con concavidad negativa.

Como se verá más adelante en el capítulo 3, este algoritmo puede resultar efectivo cuando la diferencia de probabilidades es cercana a 1. Sin embargo, este enfoque no garantiza "zero-regret", ya que existe la posibilidad de tomar decisiones erróneas. Además, si se examina el comportamiento cuando T tiende a infinito del regret, podemos observar que $R(T) = O(T)$. Esto es fácilmente verificable al notar que en (2.8), T se multiplica por una constante que depende de las probabilidades. Además, este orden puede no ser muy alentador, ya que representa el peor caso para MAB porque siempre se cumple $R(T) \leq T$ si la ganancia máxima es 1 por ronda.

2.3. Exploración uniforme

El método Greedy se puede mejorar si, en lugar de explorar cada acción una sola vez, se explora N veces, y por ende Greedy sería un caso particular de este caso con $N = 1$. Si asumimos que la recompensa de cada acción a sigue una distribución i.i.d. Bernoulli, entonces el valor acumulado después de N pruebas se aproximará a $N\mu(a)$, donde $\mu(a)$ es el valor esperado de la recompensa de la acción a. En otras palabras, si durante las primeras N rondas se explora la acción a, $\bar{\mu}(a) = \sum_{t=1}^{N} r_t/N$ y $\bar{\mu}(a) \approx \mu(a)$.

Entonces el algoritmo sería:

1. Fase de exploración: Se procede a probar las K opciones N veces.

2. Fase de explotación: Se elige la acción a para la cual se obtuvo una recompensa empírica más alta.

Este algoritmo lo llamaremos Exploración Uniforme aunque también se lo conoce como A/B/n testing [7]. Lo siguiente es determinar cuándo comenzar a explotar la mejor opción disponible. Para ello, podríamos establecer un umbral o radio, de modo que, si la muestra está lo suficientemente cerca del valor real, dejemos de explorar. Además, necesitamos calcular la probabilidad de que esto ocurra. Por lo tanto, buscamos una ecuación que siga una estructura similar a:

$$P(|\bar{\mu}(a) - \mu(a)| \leq \text{rad}) \geq 1 - \xi \tag{2.9}$$

donde:

$$\bar{\mu}(a) = \frac{1}{N} \sum_{t=1}^{N} r_t$$

Donde *rad* proviene de "radio" y denota que $\bar{\mu}$ está a una distancia de "radio" *rad* de la verdadera media μ. Esta estrategia implica la combinación de tres factores:

1. La cantidad de muestras utilizadas para la estimación, N.

2. El radio del intervalo de confianza, rad.

3. La probabilidad de que dicho intervalo de confianza sea válido, $1 - \xi$

Observamos que al incrementar la cantidad de muestras utilizadas, la probabilidad aumentará y/o el radio disminuirá, mientras que los dos últimos factores son inversos. Esto se debe a que al reducir el intervalo, la probabilidad del mismo se reduce. Afortunadamente, existe la desigualdad de Hoeffding, que proporciona una relación entre todos estos elementos:

$$P(|\bar{\mu}(a) - \mu(a)| \leq \text{rad}) \geq 1 - 1/\tau^4, \text{rad} = \sqrt{2\log(\tau)/N}, \tag{2.10}$$

donde τ es una constante que vincula el intervalo de confianza previamente mencionado con la probabilidad del mismo. En este caso, τ se mantendrá como una constante a definir hasta que se decida tomar un valor específico, que en este contexto será T. Esto se hace para simplificar los cálculos y mantener en claro el origen de esta constante. Cabe destacar que τ se puede ajustar a cualquier valor mientras se tenga en cuenta que una elección menor conduce a un intervalo más estrecho y, por lo tanto, a una probabilidad más baja de que el intervalo sea válido. Sin embargo, es relevante señalar que en otros enfoques se podría seleccionar τ en función de K y N. Lo crucial es que estos valores sean relativamente grandes para asegurar que la probabilidad sea cercana a 1. Por esta razón, T es una elección adecuada como valor para τ.

En el caso en que se cumpla $|\bar{\mu}(a) - \mu(a)| \leq \text{rad} \forall a$ (ecuación derivada de la ecuación (2.9)), se dirá que se encuentra en un "clean case" o "clean event". Si no se cumple, se considera un "bad case" o "bad event". Esta distinción es de suma importancia, ya que servirá como base para analizar este algoritmo y los futuros que se presentarán.

Ahora vamos a proceder a acotar el regret de este nuevo algoritmo. Supongamos que se cumple un "clean case", pero se selecciona una acción a que no es la óptima, esto ocurre si $\bar{\mu}(a) > \bar{\mu}(a^*)$. De la definición de "clean case":

$$\begin{aligned} \mu(a) + \text{rad} \geq \bar{\mu}(a) > \bar{\mu}(a^*) \geq \mu(a^*) - \text{rad} \\ \Rightarrow \mu(a^*) - \mu(a) \leq 2rad \end{aligned} \tag{2.11}$$

Calculando el regret para K opciones y suponiendo que T es mayor que K, esta suposición es común, ya que al menos se desea probar cada acción una vez. En esta situación, se puede afirmar que:

$$R(T) \leq (K-1)N + 2\text{rad}(T - KN) < KN + 2\sqrt{\frac{2\log(\tau)}{N}}T \tag{2.12}$$

Donde $(K-1)N$ surge de la etapa de exploración, ya que hay una decisión óptima y $K-1$ decisiones no óptimas que, en el peor de los casos, aportarán 1 al regret. En cuanto a $(T - KN)$ se debe a que se tienen T rondas en total, menos las KN rondas que al estar en el "clean case" la media no pueden estar a una

distancia mayor de $2 \cdot \text{rad}$ de la recompensa óptima. En el Anexo A.2 calculamos el N óptimo para minimizar la ecuación (2.12):

$$N = \left(\frac{T}{K}\right)^{2/3} (2\log(\tau))^{1/3} \tag{2.13}$$

y en el mismo anexo también calculamos el regret correspondiente a ese N:

$$R(T) < 3 \cdot 2^{1/3} T^{2/3} \left(K \log(\tau)\right)^{1/3} \tag{2.14}$$

Ahora, tomando $T = \tau$ se desea examinar el caso completo teniendo en cuenta los "bad events", es decir, cuando $|\bar{\mu}(a) - \mu(a)| > \text{rad}$ o sea que la recompensa estimada está a más de una distancia rad de la real. En este contexto y recordando que $R(T) < T$, se tendrá:

$$\begin{aligned}
\mathbb{E}_{a_1,\ldots,a_T}(R(T)) &= \mathbb{E}(R(T) \,|\, \text{clean ev.})\, P(\text{clean ev.}) + \mathbb{E}(R(T) \,|\, \text{bad ev.})\, P(\text{bad ev.}) \\
&\leq \mathbb{E}(R(T) \,|\, \text{clean ev.}) + T \cdot O(T^{-4}) \\
&\leq O(T^{2/3}(K \log(T))^{1/3})
\end{aligned} \tag{2.15}$$

Lo primero a notar es que efectivamente esto mejora la cota en comparación con la estrategia "Greedy". Sin embargo, este enfoque también está basado en una suposición de que la distribución de recompensas es i.i.d. Si por alguna razón las probabilidades cambian, el algoritmo seguirá explotando la acción seleccionada sin adaptarse.

Otro desafío es que para lograr un regret menor según este modelo, es necesario conocer de antemano la cantidad exacta de rondas T y someterse a un largo período de entrenamiento, lo cual puede no ser práctico. Además, este algoritmo no garantiza "zero-regret" ya que se puede tomar una acción subóptima que esté a menos de $rad = \sqrt{\frac{\log(\tau)}{N}}$. Como N es una constante, rad también lo será, siendo mayor a 0 y, por ende, generando regret incumpliendo el "zero-regret".

Otra nota interesante es que τ podría tomar cualquier valor mientras el segundo término en (2.10) tienda a 0. Por ejemplo, se podría tomar $\tau = \sqrt{T}$, dando:

$$R(T) < 3 \cdot 2^{1/3} T^{2/3} \left(K \log(\sqrt{T})\right)^{1/3} \tag{2.16}$$

Sin embargo el orden no cambia ya que $O(\log(T^c))) = O(c \log(T)) = O(\log(T))$, siendo c en este caso una constante.

2.4. Epsilon-decay

Como hemos observado, una de las principales limitaciones del algoritmo anterior es la necesidad de realizar una fase de exploración exhaustiva antes poder realizar una explotación de los datos y la necesidad de conocer el número de iteraciones T. Una solución a este problema es distribuir el proceso de entrenamiento

de manera que el regret se vuelva independiente de conocer dicho valor de antemano. Una forma de lograr esto es mediante el algoritmo "Epsilon Decay", cuyo procedimiento es el siguiente; en cada ronda t:

1. Se sortea una muestra binaria ζ, tal que $P(\zeta = 1) = \epsilon_t$.
2. Si $\zeta = 1$, se realiza una exploración y se elige al azar una acción de manera uniforme.
3. Si $\zeta = 0$, se realiza una explotación eligiendo la acción con la media estimada más alta.

Siguiendo el mismo procedimiento que en la ecuación (2.11), llegamos a que en caso de elegir una muestra incorrecta en un "clean case", esta cumplirá:

$$\mu(a^*) - \mu(a) \leq 2\text{rad} \tag{2.17}$$

Ahora en este caso, la cantidad de veces promedio que se exploró cada brazo será $\frac{t\epsilon_t}{K}$. Por ende queda:

$$\mu(a^*) - \mu(a) \leq 2\sqrt{2K\log(\tau)/t\epsilon_t} \tag{2.18}$$

Calcularemos el regret para Epsilon Decay. Empezando con que el regret para un iteración t particular está acotado por la probabilidad de explorar multiplicada por el máximo regret en ese caso, que es 1, más la probabilidad de explotar multiplicada por el máximo regret, que es $2 \cdot \text{rad}$, ya que no es posible separar las medias más allá del intervalo de confianza en un "clean case". El regret sería:

$$\begin{aligned} R(T) &= \sum_{t=1}^{T} \mathbb{E}_{a_t, r_t}(r_t^* - r_t) && (2.19)\\ &\leq \sum_{t=1}^{T} P(\text{explorar}) \cdot 1 + P(\text{explotar}) \cdot 2\text{rad} \\ &= \sum_{t=1}^{T} \epsilon_t + (1-\epsilon_t) \cdot 2\sqrt{\frac{2K\log(t)}{t\epsilon_t}} \\ &\leq \sum_{t=1}^{T} \epsilon_t + 2\sqrt{\frac{2K\log(t)}{t\epsilon_t}}. && (2.20) \end{aligned}$$

Utilizaremos la ecuación (2.20) como cota teórica para las simulaciones. Sin embargo, es necesario calcular el valor óptimo de ϵ_t que minimice el regret. Este cálculo se realizó en el Anexo A.3:

$$\epsilon_t = \left(\frac{2K\log(t)}{t}\right)^{1/3}. \tag{2.21}$$

Además, en el Anexo A.3 también se calculó otra cota, la cual, aunque mayor, es más simple ya que no requiere calcular cada valor ϵ_t ni realizar la sumatoria:

$$R(T) < 3(2K\log(T))^{1/3}T^{2/3}. \tag{2.22}$$

Como se puede observar, se obtiene la misma cota que en el caso anterior (2.14). Sin embargo, en este algoritmo, la adaptabilidad juega un papel crucial. Ante cambios en las circunstancias, el modelo puede ajustarse, a diferencia del enfoque de Exploración Uniforme. También tiene la ventaja de que no se necesita esperar una cantidad significativa de rondas antes de poder aprovechar la información recabada, lo cual lo hace más atractivo en un caso real. Nadie quiere estar recomendando opciones al azar durante meses antes de tomar una decisión. Señalamos que ya no es necesario conocer la cantidad de rondas T, ya que ϵ está definido según t y no depende de T. Esto es muy interesante, porque muchas veces es difícil estimar cuánto tiempo se va a estar utilizando el algoritmo. El escenario de "bad case" es idéntico al mencionado anteriormente.

En cuanto a la búsqueda de "zero regret", se puede considerar $\tau = T$ y calcular:

$$\mathbb{E}_{a_1,a_2,\dots a_t}(R(T))/T \leq \frac{3(2K\log(T))^{1/3}T^{2/3}}{T} \xrightarrow[T\to\infty]{} 0 \tag{2.23}$$

Como lo anterior tiende 0 cuando T tiende a infinito, se cumple el "zero regret". Notar que ϵ_t está determinado en (2.21) para un caso genérico. Dependiendo de los r_a, podría ser mejor alterar este valor, pero para ello sería necesario conocer de antemano cierta información sobre los r_t o adaptarse según los resultados obtenidos, lo cual es precisamente lo que harán los algoritmos adaptativos que veremos en la siguiente sección.

2.5. Exploración adaptativa

Hasta el momento, hemos trabajado con algoritmos que no tienen en cuenta los resultados para determinar si se va a realizar una exploración o explotación, lo cual es una limitante. Ahora exploraremos algoritmos que usen esta información para determinar si se va a explorar y que acción a elegir. Para ello se definen los siguientes términos:

- Una ronda se considerará de exploración si la información recopilada se utiliza para ajustar las decisiones futuras de la acción a.
- Un algoritmo se considera adaptativo si el conjunto de rondas a explorar no está predeterminado al comienzo del algoritmo, sino que depende de los resultados obtenidos en las rondas de exploración anteriores.

Destacamos que los algoritmos anteriormente definidos no encajan en esta definición, ya que el número de rondas tomadas no depende de los resultados de las acciones previas, sino de T.

2.5.1. Clean event en exploración adaptativa

En el caso de la exploración adaptativa, ya no es posible utilizar la desigualdad de Hoeffding para acotar $\bar{\mu}(a) - \mu(a)$, porque r_t deben ser i.i.d. y, al poder descar-

tar o disminuir cuanto se elije ciertas acciones, la distribución de la recompensa cambia. No obstante, en [1] se demuestra que se puede seguir usando Hoeffding, pero con una cota menor:

$$P(|\bar{\mu}(a) - \mu(a)| \leq \text{rad} \geq 1 - 1/t^2 \tag{2.24}$$

Lo cual nos permitirá proseguir de una forma parecida en las siguientes demostraciones. También definiremos:

$$\text{UCB}_t(a) := \bar{\mu}(a) + \text{rad} \tag{2.25}$$
$$\text{LCB}_t(a) := \bar{\mu}(a) - \text{rad} \tag{2.26}$$

Notar que $\mu(a) \in [\text{LCB}_t(a), \text{UCB}_t(a)]$ tiene una probabilidad de $1 - \frac{1}{t^2}$, lo cual proporciona un intervalo conocido en el que se encuentra la estimación de la recompensa y que le llamaremos intervalo de confianza.

2.6. Algoritmo de Eliminación

Teniendo un intervalo de confianza $[\text{LCB}_t(a), \text{UCB}_t(a)]$, es posible descartar cualquier acción a cuyo intervalo de confianza sea menor que $\text{LCB}_t(a)$ ya que si se tiene a_1 y a_2 / $\text{UCB}_t(a_1) < \text{LCB}_t(a_2)$, tendremos que $\mu(a_1) \leq \text{UCB}_t(a_1) < \text{LCB}_t(a_2) \leq \mu(a_2)$ con probabilidad $1 - \frac{1}{t^2}$. Por lo tanto, $\mu(a_1) < \mu(a_2)$ y se puede descartar la acción a_1. Usaremos lo anterior para crear un nuevo algoritmo que descarte las opciones a subóptimas:

- Setear todas las acciones a como activas
- En cada iteración:
    1. Explorar todas las acciones activas una vez
    2. Desactivar toda acción a' que $\text{UCB}_t(a') < \text{máx}_a \, \text{LCB}_t(a)$

Ahora calcularemos la cota del $R(T)$ del Algoritmo de Eliminación. Primero, expresaremos la diferencia entre una acción óptima $a^* : \mu(a^*) > \mu(a) \ \forall a$ y otra acción a cuyo intervalo de confianza se intersecta y por ende siga activa:

$$\Delta(a) = |\mu(a^*) - \mu(a)| \leq 2(\text{rad} + \text{rad}) = 4\text{rad} \tag{2.27}$$

Aplicando la definición rad (2.10) se obtiene:

$$\text{rad} = \sqrt{2\log(\tau)/|T_a|} \tag{2.28}$$

donde $T_a = \{t : a_t = a\}$ y expresaremos el regret de la acción a como $R(t, a) = \mathbb{E}_{\vec{r}_a}\left[\sum_{t \in T_a} r_t^* - r_t\right]$, donde $\vec{r}_a = (r_t : a_t = a)$. Entonces:

$$R(T,a) = \sum_{t \in T_a} \Delta(a) \tag{2.29}$$

$$\leq \sum_{t \in T_a} 4\left(\sqrt{2\log(\tau)/t}\right)$$

$$= 4\left(\sqrt{2\log(\tau)}\right) \sum_{t \in T_a} \left(\sqrt{1/t}\right) \tag{2.30}$$

El termino $\sum_{t \in T_a} \left(\sqrt{1/t}\right)$ se puede acotar por la integral de $\int_1^{T_a} \sqrt{1/t} < 2\sqrt{|T_a|}$:

$$R(T,a) \leq 4\left(\sqrt{2\log(\tau)}\right) \sum_{t \in T_a} \left(\sqrt{1/t}\right) \tag{2.31}$$

$$< 8\left(\sqrt{2\log(\tau)T_a}\right) \tag{2.32}$$

Sumando para todas las acciones:

$$R(T) = \sum_{a=1}^{K} R(T,a) \leq 8\sqrt{2\log(\tau)} \sum_{a=1}^{K} \sqrt{|T_a|} \tag{2.33}$$

Notar que $\sum_a |T_a| = T$, y que $\sqrt{x}$ es una función cóncava, entonces podemos encontrar la siguiente desigualdad aplicando la desigualdad de Jensen:

$$1/K \sum_{a=1}^{K} \sqrt{|T_a|} \leq \sqrt{\sum_{a=1}^{K} |T_a|/K} = \sqrt{T/K} \tag{2.34}$$

Uniendo los términos (2.33) y (2.34), obtenemos la cota del regret:

$$R(T) \leq 8\sqrt{2KT\log(\tau)} \tag{2.35}$$

La ecuación (2.35) agrega una dependencia con K, pero al mismo tiempo se reduce el orden en T. Teniendo en cuenta que por lo general $K \ll T$, esto es algo sumamente positivo, ya que garantiza un mejor regret.

Ahora demostraremos que la estrategia efectivamente alcanza el "zero-regret". Analizando (2.35) cuando T tiende infinito divido por T y tomando $\tau = T$ queda:

$$\frac{R(T)}{T} \leq 8\frac{\sqrt{KT\log(T)}}{T} \xrightarrow[T \to \infty]{} 0 \tag{2.36}$$

alcanzando el "zero regret". Otra observación es que este algoritmo posee cierta adaptabilidad para incorporar nuevas acciones entre las opciones disponibles. Estas nuevas acciones comenzarán con $|T_a| = 0$ y serán explorados inicialmente. Sin embargo, es necesario considerar que este enfoque asume que, una vez que una acción ha sido explorada, su rendimiento no cambia, lo cual podría ser una suposición demasiado rígida en ciertos escenarios.

2.7. Upper confidence Bound (UCB)

El algoritmo UCB es una alternativa al algoritmo anterior que tiene una cota del Regret menor. Esto se consigue tomando la acción para la cual $\mathrm{UCB}_t(a)$ sea el máximo en vez de usar el intervalo de confianza, dando el siguiente algoritmo:

- En cada iteración, se toma $a = \arg\max(\mathrm{UCB}_t(a))$
- Se actualizan los UCB_t

Ahora demostraremos que efectivamente este algoritmo tiene una cota menor que el anterior. En este caso para que una acción a sea elegida sobre el óptimo a^*:

$$\mathrm{UCB}(a) > \mathrm{UCB}(a^*) \tag{2.37}$$

$$\bar{\mu}(a) > \bar{\mu^*}(a) \tag{2.38}$$

De (2.25) se puede obtener que $\mu(a)+\mathrm{rad} > \bar{\mu}(a) > \mu(a)-\mathrm{rad}$ con probabilidad $1 - 1/T^2$, aplicándolo a (2.38):

$$\mu(a) + \mathrm{rad} > \bar{\mu}(a) > \bar{\mu^*}(a) > \mu(a^*) - \mathrm{rad} \tag{2.39}$$

$$\mu(a^*) - \mu(a) < 2rad \tag{2.40}$$

Este resultado es similar a la cota (2.27), pero la cota divida entre 2. Por lo tanto, podemos hacer el mismo análisis, pero con una cota dos veces menor:

$$R(T) \leq 4\sqrt{KT\log(\tau)} \tag{2.41}$$

Siendo un algoritmo bastante parecido, pero con un regret acotado a la mitad.

En cuanto al "bad event" es despreciable, ya que la contribución de $T \cdot O(T^{-2})$ es insignificante en comparación con el resto de los términos.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 3

Simulación de algoritmos no contextuales

En este capítulo, llevaremos a cabo evaluaciones exhaustivas de los métodos previamente descritos, seguidas de comparaciones detalladas y conclusiones relevantes. Evaluaremos mediante simulaciones, mientras que en el capítulo 8 haremos pruebas con datos reales. A lo largo de este proceso, también exploraremos los desafíos inherentes de ambas modalidades de evaluación, proporcionando una visión completa de los hallazgos.

Es esencial recordar que las simulaciones están basadas en suposiciones específicas que pueden no cumplirse fielmente en situaciones reales. En particular, las simulaciones se basarán en las siguientes premisas:

1. Las recompensas r son i.i.d. y siguen una distribución de probabilidades.

2. En las simulaciones, se conoce la mejor acción disponible en todo momento. Esto permite calcular el regret, recordar que se definió como $R(T) = \mathbb{E}_{r_1,r_2,\ldots,r_T}\left[\sum_{t=1}^{T} r_t^* - r_t\right]$, donde r_t^* es la recompensa óptima en la iteración t.

3. Para los métodos en este capítulo, se supone que la probabilidad de éxito depende únicamente de la acción elegida.

Estas suposiciones permiten llevar a cabo simulaciones controladas y evaluar el rendimiento de los métodos en condiciones ideales. No obstante, es fundamental reconocer que los datos reales pueden desviarse de estas premisas, lo que plantea desafíos adicionales en la aplicación práctica de estos métodos en situaciones del mundo real.

3.1. Greedy

Para verificar la deducción realizada en la ecuación (2.7), se llevará a cabo una simulación con 20 iteraciones repetidas 1000 veces. Se graficará el promedio, el

percentil 25 y el percentil 75, utilizando valores de $p_2 = 0.5$ y $p_1 = 0.25$. En la figura 3.1 presentamos los resultados.

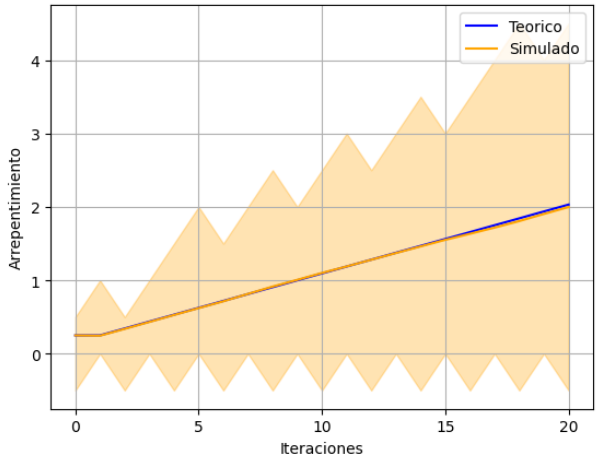

Figura 3.1: Regret de Greedy Simulado vs Teórico K=2, donde los mismos se diferencian en menos de un 10 % y la diferencia promedio es del 2.5 %.

Como se puede observar, las dos primeras iteraciones tienen un regret constante. Esto se debe a que en la primera iteración se selecciona la acción a_1, lo que resulta en un regret promedio de $p_2 - p_1 = 0.25$, y en la segunda iteración se selecciona la acción a_2, que es la óptima y no genera regret. Posteriormente, se explota la acción que proporcionó el mejor resultado y obtuvimos una diferencia promedio del 2.5 % entre las simulaciones y el valor teórico. En ciertos escenarios observamos como el percentil inferior es negativo. Esto ocurre porque utilizamos $r_t^* = 1 \cdot p_2$ para calcular el regret, que es efectivamente la media para la mejor acción. Sin embargo, en situaciones particulares, por aleatoriedad, cualquiera de las dos acciones podría generar un promedio mayor que la media esperada de p_2, lo que resultaría en un regret negativo en ese caso específico.

Otra deducción interesante es la ecuación (2.7), que demuestra que el regret de este algoritmo solo depende de la diferencia de probabilidades. Al graficar el regret total $R(T)$ en funcion de Δp dado un número de iteraciones T, obtuvimos los resultados de la figura 3.2.

En la misma se puede ver que, a medida que T crece, la gráfica converge hacia (2.8), ya que $\frac{\Delta p}{T} = \frac{p_2 - p_1}{T}$ se vuelve despreciable y para $T = 1000$ es prácticamente imperceptible y los valores teóricos y simulados son muy cercanos. Además, como $R(T) \neq 0 \ \forall \Delta p \in (0, 1)$ se demuestra que no se cumple el "zero-regret".

3.2. Exploración Uniforme

La estrategia de Exploración Uniforme añade una discusión que no se aborda en el enfoque Greedy, el cual se centra en cuánto explorar y cuánto explotar las posibles acciones a tomar. Se tomará como punto de partida el valor de N obtenido

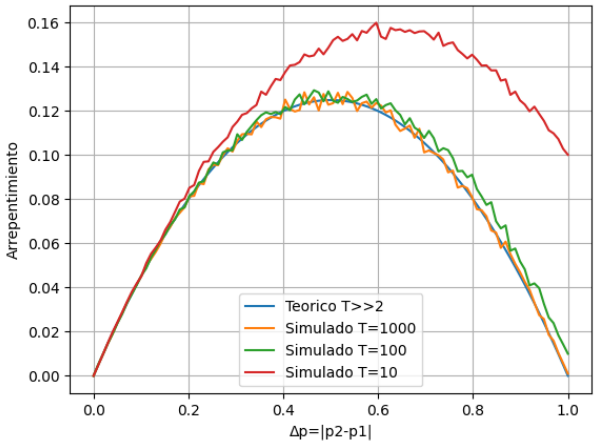

Figura 3.2: R(T)/T Simulado vs Teórico K=2 para distintos valores de $|p_2 - p_1|$. Observamos que el simulado converge al teórico al aumentar T.

en la ecuación (2.13), pero se explorará cómo diferentes parámetros afectan el regret en estas dos etapas diferenciadas y se analizarán las dependencias de cada una de las etapas, para ver la diferencia entre la cota teórica y el peor caso posible.

3.2.1. Primera etapa, exploración

Para la etapa de exploración se intentará hallar el peor caso. Esta etapa deriva del primer término en la ecuación (2.12). Cabe recordar que la Exploración Uniforme consiste en la exploración de las acciones para luego determinar cuál de ellas es la mejor para explotar. El regret de esta primer etapa depende de las diferencias de probabilidades con respecto a la mejor acción. Una forma más precisa de expresar el regret sería:

$$R(T) = \sum_{t=1}^{T} r_t^* - r_t = \sum_{t=1}^{T} p_t^* - p_t \tag{3.1}$$

Dado que cada acción se explora N veces, obtenemos:

$$R(T) = \sum_{t=1}^{T} r_t^* - r_t = N \sum_{i=1}^{K} (p^* - p_i) = NKp^* - N \sum_{i=1}^{K} p_i \tag{3.2}$$

Como T y K son entradas y que N ya está dado por el algoritmo en la ecuación (2.13), maximizar (3.2) implica que $p^* = 1$ y $p_i = 0$. Tomando $p^* = 1$ y $p_i = 0$, se obtiene:

$$R(T) = N(K-1)p^* \tag{3.3}$$

Aparece un término $K - 1$, ya que una de las opciones es la óptima.

Al graficar la ecuación para $K = 10$, con los correspondientes percentiles y las probabilidades mencionadas anteriormente, se obtiene la figura 3.3. La gráfica azul

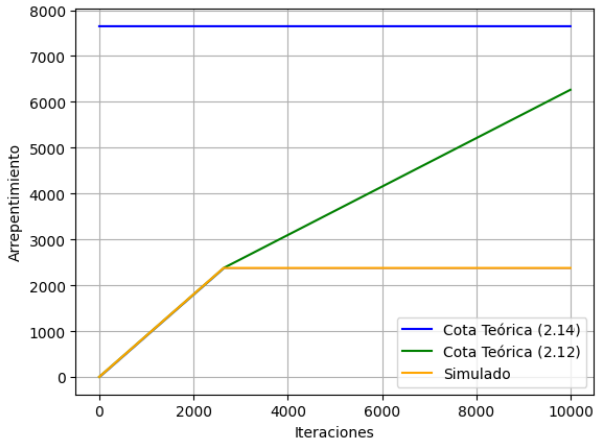

Figura 3.3: Regret de peor caso de exploración Simulado vs teórico K=10. Observamos que los valores simulados son menores que las cotas teóricas.

representa la cota obtenida en (2.14), la cual calcula el regret total. En cambio, la curva verde se obtiene de la ecuación (2.12), teniendo en cuenta que el segundo término se refiere a la explotación y es igual a 0 cuando $t < KN$.

Como se puede observar, la primera parte de la gráfica representa la fase de exploración, donde el valor teórico y el valor simulado son idénticos, debido a que el simulado es el peor caso, donde $r_t = 0 \ \forall r_t \neq r^*$. Sin embargo, en la fase de explotación no se acumula ningún regret, ya que la probabilidad de elegir una acción incorrecta es nula. En este fase se observa un comportamiento interesante: cuando el regret en la fase de exploración es alto, la probabilidad de elegir la acción correcta también aumenta, ya que las probabilidades se diferencian lo suficiente como para que el algoritmo detecte cual es la mejor acción.

Otro dato a destacar es que el regret en la fase de exploración depende tanto de N como de K. A mayor valor de estos parámetros, se explorará más y, por ende, se acumulará un mayor regret. Analizando un poco más K, en la ecuación (2.13) se observa que al aumentar K se disminuye N, pero como la cantidad de iteraciones en la exploración es de KN, el crecimiento en el regret del peor caso aumentará en un factor de $K^{1/3}$. Además, aumentará la probabilidad de equivocarse en la siguiente etapa ya que hay menos muestras para determinar la probabilidad de cada acción.

Una última nota a destacar es que en este caso particular, la gráfica es totalmente determinista, ya que al ser las probabilidades 1 o 0, no hay azar al elegir una acción. Esto produce que los percentiles no se puedan apreciar, ya que todas las simulaciones son idénticas.

3.2.2. Segunda etapa, explotación

Se puede deducir que el regret en la fase de explotación depende de la elección de una acción subóptima; en cualquier otro caso, será igual a cero por definición. Esta dependencia estará influenciada por varios factores. En primer lugar,

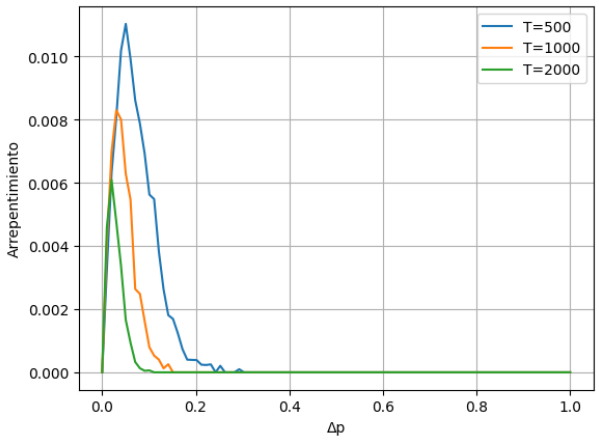

Figura 3.4: R(T)/T de explotación Simulado vs teórico K=10, donde observamos que disminuye el regret al aumentar las iteraciones

depende de T, ya que a medida que T aumenta, se dispone de más tiempo para explorar, lo que aumenta la probabilidad de elegir el a óptimo. Además, depende de K, ya que cuantas más opciones de elección haya, es más probable que se elija una acción subóptima. Por otro lado, el regret también está influenciado por las probabilidades, ya que si existe una gran diferencia entre las probabilidades y se elige una acción incorrecta, el regret será alto. Sin embargo, en este escenario, la probabilidad de tomar una decisión incorrecta será baja.

Para analizar todas estas dualidades, se llevará a cabo una simulación y se graficará el comportamiento del regret en la fase de explotación con diferentes números de iteraciones y acciones.

Para esta simulación, se considerará que la mejor probabilidad es $p_k^* = 1$, mientras que todas las demás probabilidades serán iguales, es decir, $p_1 = p_2 = \ldots = p_{k-1} = 1 - \Delta p$. Se graficará el regret ponderado en función de la cantidad de iteraciones, considerando este valor de Δp, para diferentes valores de K y T, corriendo el experimento 100 veces y tomando el promedio de los mismos.

La figura 3.4 muestra que, a medida que aumenta el número de iteraciones, el regret por ronda disminuye. Esto tiene sentido ya que un mayor valor de T implica un valor de N más alto, lo que a su vez aumenta la probabilidad de elegir correctamente a^*. Además, al aumentar la cantidad iteraciones T hace que sea más difícil elegir probabilidades con un Δp grande, lo que lleva a que el peor caso para valores grandes de T esté asociado con un valor pequeño de Δp.

Continuando con el siguiente caso, se observa algo similar pero a la inversa en relación con K, como se muestra en la figura 3.5.

A medida que K aumenta, el valor de N disminuye, como se puede apreciar en la ecuación (2.13). Esto significa que se explora menos cada acción y, al tener más acciones para elegir, la probabilidad de seleccionar una subóptima aumenta. Este comportamiento es inverso al efecto de aumentar T.
Es crucial tener en cuenta esta relación, ya que en ciertos escenarios, como el de Amazon, donde cada producto podría representar una acción, el valor de K

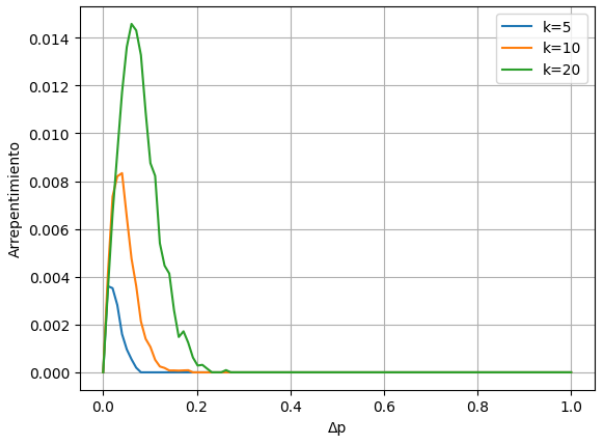

Figura 3.5: R(T)/T de explotación Simulado vs teórico T=1000, donde contemplamos que el regret aumenta al aumentar la cantidad de opciones.

podría llegar a ser extremadamente alto, en el orden de millones. Esto destaca la necesidad de tomar decisiones o utilizar técnicas para reducir el número de acciones en situaciones similares.

Por ejemplo, en lugar de que cada producto sea una acción, se podría usar clustering para agrupar los productos y disminuir el K.

3.3. Epsilon-decay

En el caso del Epsilon Decay, no se puede separar fácilmente la exploración de la explotación, ya que se intercalan de forma aleatoria. Por lo tanto, se limitará a verificar que las simulaciones cumplan con las cotas teóricas (2.20) y (2.22). Recordar que este método se caracteriza por ser un enfoque de exploración no adaptativa, pero que aún cumple con el "zero regret".

Para la figura 3.6 se graficó $R(T)/T$ simulado, con $p_1 = 1.0, p_i = 0.5\ \forall i \neq 1$ y las cotas teóricas son de las ecuaciones (2.20) y (2.22). Una diferencia fundamental con el caso anterior es que la derivada de $R(T)/T$ tiende a 0, como se puede ver en la figura ya mencionada. Observar que para Exploración Uniforme, aunque la probabilidad de elegir incorrectamente sea baja, si sucede alguna vez, el promedio del regret será mayor que 0 y, por ende, no se cumplirá el "zero regret".

Además, a diferencia de la Exploración Uniforme, nunca se utiliza el valor T de antemano en la ecuación (2.20), lo que significa que el regret es independiente de conocerlo. Esto es beneficioso en escenarios reales, ya que el regret tiende a cero sin necesidad de conocer con certeza cuántas iteraciones T estará disponible una acción a. Por ello, en la figura 3.6 se graficó la curva roja con un T menor y que tiene los mismos valores que la naranja, lo que confirma que el regret no depende de conocer el número de iteraciones con anticipación.

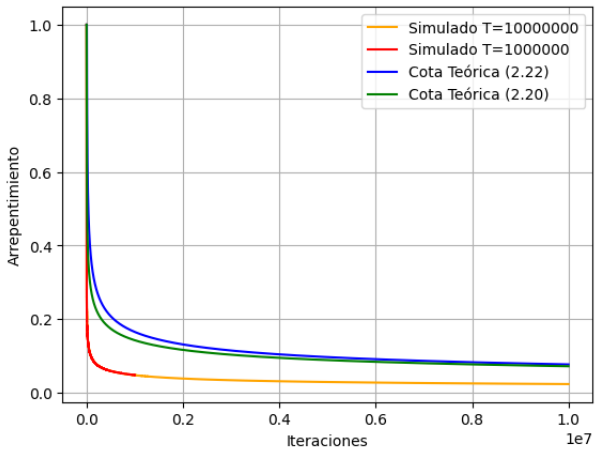

Figura 3.6: R(T)/T para epsilon greedy K=10, donde valor simulado cumple con la cotas teóricas.

3.4. Upper Confindence Bound (UCB)

Por último, se tiene la simulación de Upper Confidence Bound (UCB), que cuenta con una exploración adaptativa y, teóricamente, debería proporcionar un mejor regret.

Realizando el mismo procedimiento de la sección anterior, se graficó la función con su cota teórica en la figura 3.7.

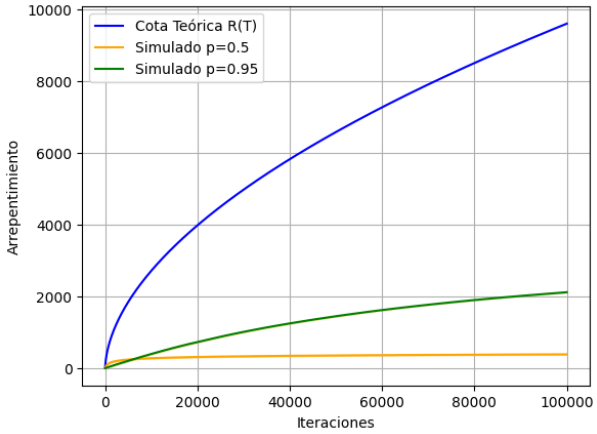

Figura 3.7: R(T) UCB, K=10, donde la cota es mayor que las simulaciones.

Se realizaron dos simulaciones: una similar a la anterior con $p_1 = 1.0, p_i = 0.5$ $\forall i \neq 1$, y una segunda con $p_1 = 1.0, p_i = 0.95$ $\forall i \neq 1$. Observamos que la cota dada por (2.41) se cumple en todos los puntos de la gráfica y que no depende de p_i, pero el resultado simulado sí lo hace. Esto se debe a que el algoritmo es adaptativo; si las recompensas son muy bajas para ciertas acciones, estas serán descartadas y no exploradas. En este experimento, notamos que el regret para la gráfica verde

comienza con una derivada menor porque el regret máximo es $p_1 - p_i = 0.05$. Sin embargo, debido a que esta diferencia es menor que en la otra simulación, el algoritmo requiere más iteraciones para descartar las otras opciones a. En la gráfica amarilla, el regret comienza con un valor mayor ya que algunas iteraciones tienen un regret promedio de 0.5, pero estas se descartan en menos iteraciones porque el intervalo de confianza decrece más rápidamente.

Realizando un estudio similar al realizado para Epsilon Decay, se obtiene el regret ponderado por ronda en la figura 3.8.

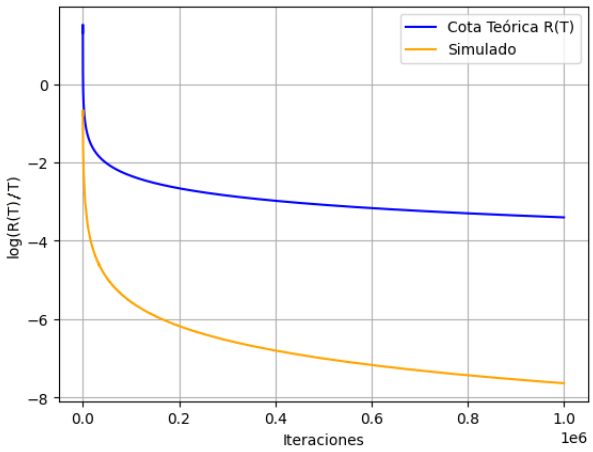

Figura 3.8: log(R(T)/T) UCB, K=10, donde comprobamos que R(T) tiende a 0 más rápidamente que epsilon decay y se cumple el "zero-regret".

Como se puede observar, UCB también alcanza zero-regret y su regret es independiente de conocer el valor de T, lo cual representa ventajas apreciables. Además, en comparación con los algoritmos anteriores, tiene la ventaja de una convergencia más rápida, ya que determina cuánto explorar basándose en resultados previos, lo que permite ahorrar iteraciones de exploración cuando se ha obtenido un resultado deseado.

En la siguiente subsección, se elegirá un caso para corroborar las ventajas en un escenario particular de los diferentes algoritmos.

3.5. Estudio comparativo

En esta sección comparamos todos los métodos mencionados hasta ahora, y se mostrarán tres escenarios diferentes. El primero es $p_1 = 1.0, p_i = 0.5\ \forall i \neq 1$, donde UCB y Exploración Uniforme tendrán un menor regret. El segundo es con $p_1 = 0.55, p_i = 0.5 \forall i \neq 1$, donde Epsilon Decay sera el mejor. Finalmente, el tercer escenario es $p_1 = 1.0, p_i = 0.01 \forall i \neq 1$, en el cual Greedy tendrá el menor regret. Observaremos y concluiremos por qué cada uno de los algoritmos minimiza el regret $R(T)$ en su respectivo escenario y sus razones. Comenzando con $p_1 = 1.0, p_i = 0.5 \forall i \neq 1$. Graficando las simulaciones con sus respectivos percentiles, se obtiene la figura 3.9.

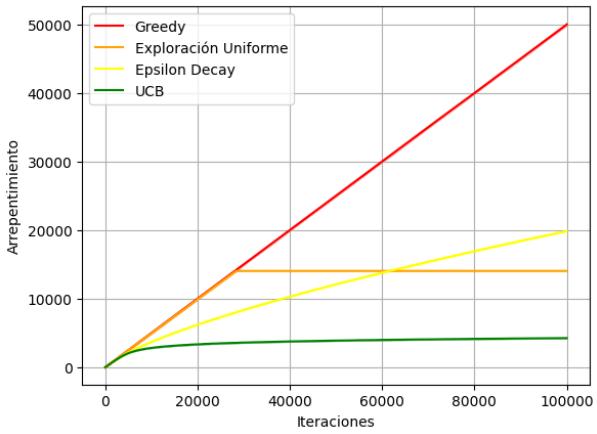

Figura 3.9: R(T) para todos los métodos con $p_1 = 1.0, p_i = 0.5 \forall i \neq 1$, K=100, donde UCB y Exploración Uniforme tienen el R(T) más bajo.

Como era de esperarse, el método más efectivo resultó ser UCB, gracias a su adaptabilidad de poder descartar las opciones con $p_i = 0.5$. En contraste, el peor desempeño se observó en el método de Greedy, ya que su crecimiento es de orden T y tiende a aumentar indefinidamente, a diferencia de los otros dos métodos, que después de explorar lo suficiente, logran mantener un bajo regret por ronda.

Es interesante destacar que el método de Exploración Uniforme superó a Epsilon Decay en la figura 3.9. Sin embargo, para lograr este resultado se necesita conocer el valor de T y otra desventaja es que este método no garantiza el "zero regret". Además, si el experimento hubiera terminado antes de las 50000 iteraciones, Epsilon Decay habría tenido un mejor desempeño, ya que al principio la Exploración Uniforme invierte muchas iteraciones intentando encontrar cuál es la mejor acción.

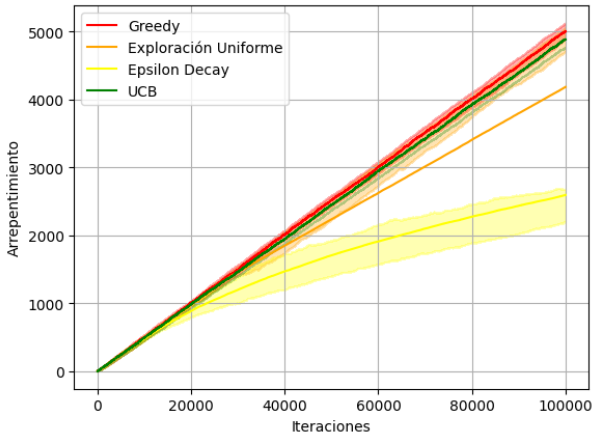

Figura 3.10: R(T) para todos los métodos con $p_1 = 0.55, p_i = 0.5 \forall i \neq 1$, K=100, donde Epsilon Decay tiene el menor regret

Para encontrar un caso que beneficie a Epsilon Decay, procedimos con el se-

gundo escenario con $p_1 = 0.55, p_i = 0.50 \forall i \neq 1$, y obtuvimos la figura 3.10. Lo primero a notar es que el regret total es menor, ya que en el peor de los casos $r^* - r_T = 0.05$, a diferencia del caso anterior donde era $r^* - r_T = 0.5$. Lo siguiente a observar es que Exploración Uniforme tuvo un desempeño mucho peor en este caso. Esto se debe a que la probabilidad de seleccionar la acción incorrecta es significativamente alta, dado que todas ellas son muy similares. En cuanto a UCB, al ser todas las opciones tan parecidas, los intervalos de confianza se intersectan. Por otro lado, Epsilon Decay fue muy bueno, ya que continuó recomendando la mejor acción mientras seguía explorando, lo cual lo hace particularmente efectivo cuando las opciones tienen una recompensa similar. Otro detalle es que hay una gran variabilidad en los resultados debido a que al ser las probabilidades tan parecidas, las diferentes simulaciones suelen arrojar resultados diversos. En cambio, en el caso $p_1 = 1$, donde la probabilidad de elegir incorrectamente es mucho menor, apenas hay variabilidad entre los experimentos.

Estas conclusiones son valiosas, ya que demuestran que hay escenarios en los cuales UCB, Epsilon Decay y Exploración Uniforme pueden ser muy deseables, dependiendo de las recompensas promedio de las distintas acciones. Incluso el método de Greedy puede superar a los otros si la diferencia de probabilidades es extremadamente alta, como se muestra en el último escenario 3.11 con $p_1 = 1.0, p_i = 0.01 \forall i \neq 1$. En este caso, la varianza de Greedy es muy grande, ya que aunque su mediana es la menor, a veces elige incorrectamente la acción y, por ende, da el peor resultado, lo que lo convierte en un método poco consistente.

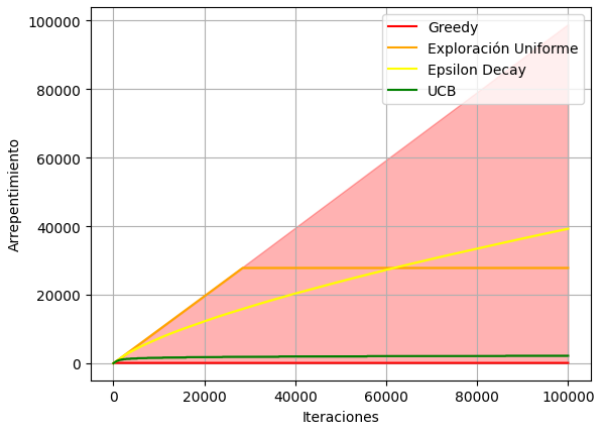

Figura 3.11: R(T) para todos los métodos con $p_1 = 1.0, p_i = 0.01 \forall i \neq 1$, K=100, donde el método con mediana de $R(T)$ más baja es Greedy pero con una variabilidad muy alta

Sin embargo, la figura 3.11 es un caso extremo y, en la mayoría de las situaciones, sería más adecuado utilizar UCB o Epsilon Decay con un ϵ más bajo, especialmente si se conoce que existe una diferencia significativa entre las probabilidades de las acciones.

Como conclusión, notamos que todos los algoritmos tuvieron un buen desempeño en uno de los tres escenarios. Esto sugiere que son opciones válidas, siempre y cuando se conozca con precisión en cuál escenario nos encontramos. Dicho esto, en

la práctica se suele preferir UCB, ya que suele ofrecer mejores resultados en casos más reales. Esto se debe a que es poco común que todos los opciones a tengan probabilidades similares, y en caso de que ocurra, las diferencias de las recompensas serán pequeñas y recomendar cualquier producto no produce un regret mayor que el 2rad visto en la ecuación (2.40). Finalmente, se puede observar en la figura 3.10 que el regret es diez veces menor que en el caso 3.9, lo que significa que incluso si UCB no fuera el mejor en ese caso específico, las recomendaciones no estarían muy alejadas de la mejor acción posible.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Algoritmos Contextuales

En este capítulo, cerraremos la discusión sobre los fundamentos clásicos de los MAB, introduciendo el concepto de *Multi armed bandit contextuales* (MABC), explorando su definición y las principales diferencias con respecto a los algoritmos no contextuales.

Hasta ahora, se han explorado los MAB con solo la información histórica de cada selección a_t y sus recompensas r_t. Estos algoritmos son poderosas herramientas para resolver problemas de toma de decisiones en situaciones de incertidumbre, donde se debe elegir entre varias opciones desconocidas para maximizar recompensas. Sin embargo, en muchas aplicaciones del mundo real, la información no se limita a conocer las opciones a sin ningún contexto de las mismas.

Supongamos una situación en la que no solo se tiene la información sobre las múltiples acciones, sino que además se cuenta con información adicional sobre el contexto que rodea cada elección. Esto puede ser información sobre el usuario, el entorno o los datos sobre la elección a tomar, y que puede ser crítica para tomar la mejor decisión. En este contexto, surge la necesidad de abordar los problemas de los *Multi-Armed Bandits Contextuales.*

Los MAB contextuales representan una expansión natural de los MAB clásicos. En lugar de tratar las opciones como entidades independientes, los MAB contextuales consideran que cada acción tiene asociada una cierta información contextual que influye en la recompensa esperada. Este enfoque más sofisticado permite que los algoritmos tomen decisiones más informadas, adaptándose a las condiciones cambiantes y aprovechando de mayor cantidad de datos. Esto tiene la desventaja de necesitar de esta información y que sea de calidad, ya que de lo contrario, no aporta ningún valor y se tiene un modelo más complejo sin ninguna ganancia.

Al pasar a una definición más formal, y ampliando la definición de *stochastic bandit* pero con contexto, queda de la siguiente manera:

1. El algoritmo observará un contexto x y elegirá un acción a.
2. Se observa una recompensa $r_{a,t}$ aleatoriamente de una distribución $F_{a,x}$.
3. Dada esa recompensa, el algoritmo podrá actualizar los parámetros correspondientes para la toma de decisiones futuras.

Las recompensas siguen siendo i.i.d., pero ahora r_t depende de a_t y x_t. Esto significa que, en lugar en encontrar el mejor $\mu(a)$ como en la ecuación (2.1), se busca alcanzar:

$$\mu(a^*|x) = \max_a(\mu(a|x)). \tag{4.1}$$

El regret acumulado se expresa ahora como:

$$R(T|x) = \mathbb{E}_{r_1,r_2,\ldots,r_T}\left[\sum_{t=1}^{T} r^*_{t,x} - r_{t,x}\right] \tag{4.2}$$

4.1. UCB contextual

Una forma muy sencilla de transformar cualquier algoritmo MAB en uno contextual es tener $M_1, \ldots, M_m$ modelos independientes para cada posible $x_1, \ldots, x_m$. Esto equivale a no asumir nada sobre el contexto y aceptar que pequeños cambios en el mismo pueden producir grandes variaciones en la recompensa esperada. El proceso se describe de la siguiente manera:

- Se observará contexto x, eligiendo el modelo M_x correspondiente.
- El modelo M_x elegirá una acción a y obtendrá una recompensa $r_{a,x}$ aleatoriamente de una distribución $F_{a,x}$.
- Dada esa recompensa, el modelo M_x puede (o no) actualizar los parámetros correspondientes para la toma de decisiones futuras.

Ahora, se analizará este enfoque para UCB. La primera parte del análisis de regret es idéntica, agregando la dependencia con x. Por lo tanto, usando la ecuación (2.29), pero teniendo un $T_{a,x}$ y $\Delta(a,x)$ independientes en cada contexto x para cada acción a, queda:

$$R(T,a,x) = \sum_{t \in T_{a,x}} \Delta(a,x) \leq 8\left(\sqrt{2T_{a,x}\log(\tau)}\right) \tag{4.3}$$

Sumando para todas las acciones y contextos:

$$R(T) = \sum_{a=1}^{K}\sum_{i=1}^{m} R(T,a,x_i) \leq 8\sqrt{2\log(\tau)}\sum_{a=1}^{K}\sum_{i=1}^{m}\sqrt{|T_{a,x}|} \tag{4.4}$$

Se puede aplicar la desigualdad de Jensen:

$$R(T) \leq 8\sqrt{2\log(\tau)}\sum_{a=1}^{K}\sum_{x}\sqrt{|T_{a,x}|} \tag{4.5}$$

$$\leq 8\sqrt{2\log(\tau)}\sqrt{\sum_{a=1}^{K}\sum_{x}|T_{a,x}|} \tag{4.6}$$

Y aplicando $\sum_{a=1}^{K}\sum_{x}|T_{a,x}| = TKm$:

$$R(T) \leq 8\sqrt{2\log(\tau)TKm} \tag{4.7}$$

Este enfoque acaba siendo una especie de algoritmo de fuerza bruta, ya que se calculan todas las posibles $\mu(a, x)$, lo cual, si se tienen suficientes rondas y m es bajo, puede converger. Sin embargo, este enfoque no aprovecha las posibles relaciones entre los contextos. En muchos casos, contextos similares tienden a producir resultados similares. Por ejemplo, un contexto podría representar el precio de un producto y, por lo tanto, tener un impacto significativo en la probabilidad de venta. Un pequeño cambio en el precio, en principio, no debería provocar un cambio drástico en la recompensa.

4.2. LinUCB

El caso anterior con UCB contextual presenta el desafío de obtener una cota precisa para el regret cuando hay una gran cantidad de contextos. En estos casos, a menudo es necesario hacer suposiciones sobre el comportamiento del vector de contexto x para reducir el regret teórico. Un enfoque que ha demostrado tener buenos resultados en la práctica es suponer que la recompensa depende linealmente del contexto más un ruido gaussiano, es decir, $\nu \sim N(0, \rho)$

$$r_{t,a} = x_{t,a}^T \theta_a + \nu \tag{4.8}$$

donde $\theta_a \in \mathbb{R}^d$ es un vector desconocido que se busca estimar para cada acción a. Este vector se denominará "vector de comportamiento", ya que describe cómo se modifica la recompensa esperada ante cambios en el contexto. En cuanto $x_{t,a} \in \mathbb{R}^d$ cabe recordar que es un vector i.i.d. que depende de la acción y la iteración. $x_{t,a}^T \theta_a$ sería equivalente a la nueva estimación de $\mu(a)$ que el algoritmo intenta calcular. La esperanza de lo anterior se expresa de la siguiente manera:

$$\mathbb{E}_{r_{t,a}}(r_{t,a}|x_{t,a}) = \mathbb{E}_{r_{t,a}}(x_{t,a}^T \theta_a + \nu|x_{t,a}) = \mathbb{E}_{r_{t,a}}(x_{t,a}^T \theta_a) + 0 = x_{t,a}^T \theta_a \tag{4.9}$$

En la anterior ecuación θ_a puede ser igual para todas las acciones, y en este caso se llamará LinUCB Global. Incluso en el paper original [2] se habla de otro modelo llamado híbrido porque combina LinUCB Global y LinUCB utilizando un parámetro β que es el mismo para todas las opciones a:

$$\mathbb{E}_{r_{t,a}}[r|x_{t,a}] = x_{t,a}^T \theta_a + x_{t,a}^T \beta \tag{4.10}$$

Sin embargo, como se puede leer en [2], esta variante no ha demostrado resultados significativamente mejores y, por ende, no se analizará en esta tesis.

Volviendo a la ecuación (4.9), lo primero a destacar es que se ha pasado de estimar $\mu(a) \in \mathbb{R}$ a querer estimar $\theta_a \in \mathbb{R}^d$. El enfoque utilizado en el paper original [2] usa regresión Ridge, donde E_a es una matriz que contiene los contextos x_a de entrenamiento donde la acción a fue la seleccionada, y ρ_a es un vector con las recompensas r cuando a fue seleccionada.

$$\hat{\theta} = (E_a^T E_a + \lambda_d)^{-1} E_a^T \rho_a \tag{4.11}$$

La ecuación anterior es la solución a una regresión Ridge, siendo λ_d el parámetro de la regularización, es decir, la matriz identidad multiplicada por la constante λ. Para el caso anterior [2], se puede demostrar que:

$$P\left(|x_{t,a}^T \hat{\theta} - \mathbb{E}_{r,a}[r_{t,a}|x_{t,a}]| \leq \alpha\sqrt{x_{t,a}^T (E_a^T E_a + \lambda_d)^{-1} x_{t,a}}\right) \geq 1 - \delta \tag{4.12}$$

$$\text{con } \alpha = 1 + \sqrt{\log(2/\delta)/2)} \tag{4.13}$$

donde la ecuación (4.12) acota la distancia entre el valor esperado por la recompensa $\mathbb{E}_{r,a}[r_{t,a}|x_{t,a}]$ y su estimación $x_{t,a}^T \hat{\theta}$ con una probabilidad $1 - \delta$. Además, se introdujo un nuevo hiper-parámetro α, que multiplica el intervalo de confianza, afectando su probabilidad, y por ende controla cuánto explorará o explotará el algoritmo. Con esto, se ha llegado a algo muy similar a UCB, una forma de estimar la recompensa esperada y un intervalo de confianza para esa recompensa.

LinUCB se puede expresar como en el Algoritmo 1, donde $D_{t,a}$ $b_{t,a}$ son estimaciones parciales de $E_a^T E_a + \lambda_d$ y $E_a^T \rho$ correspondientemente.

Algorithm 1 LinUCB

Inicialización:

- $D_{0,a} \leftarrow \lambda_d$
- $b_{0,a} \leftarrow 0_{d \times 1}$
- $t \leftarrow 0$

Y en cada iteración:

1. $\hat{\theta}_{t,a} \leftarrow D_{t,a}^{-1} b_{t,a}$
2. $p_{t,a} \leftarrow x_{t,a}^T \hat{\theta}_{t,a} + \alpha\sqrt{x_{t,a}^T D_{t,a}^{-1} x_{t,a}}$
3. Se elije la acción que maximiza $\arg\max_a p_{t,a}$, obteniendo la recompensa r_t
4. $D_{t+1,a} \leftarrow D_{t,a} + x_{t,a_t} x_{t,a_t}^T$
5. $b_{t+1,a} \leftarrow b_{t,a} + r_t x_{t,a_t}^T$
6. $t \leftarrow t + 1$

En caso de cumplir (4.12) se puede demostrar que LinUCB tiene una cota para el regret [2]:

$$R(T) \leq \tilde{O}(\sqrt{KTd}),$$

donde $\tilde{O}$ es similar a la notación *big O*, pero despreciando logaritmos. Claramente, esto es mejor que UCB contextual, ya que elimina la dependencia de la cantidad de contextos.

Volviendo a discutir el regret, parece ser similar a UCB no contextual, la diferencia radica en que el regret no tiene la misma definición en el caso contextual y para ello describiremos un ejemplo. Supongamos el siguiente caso: se tienen dos opciones, a_1 y a_2, con probabilidades de recompensa promedio $p_1 = p_2 = 0.5$. En un caso tradicional de MAB, el regret siempre sería igual a 0 porque es equivalente seleccionar cualquier acción. Pero si se tiene un contexto $x \in \{0, 1\}$ y tal que:

$$p_1 = \begin{cases} 0 & \text{si x=0} \\ 1 & \text{si x=1} \end{cases} \tag{4.14}$$

$$p_2 = \begin{cases} 1 & \text{si x=0} \\ 0 & \text{si x=1} \end{cases} \tag{4.15}$$

En este caso, el valor óptimo de r^* sería igual a 1, ya que en función del contexto se puede determinar correctamente cuál de las dos acciones será exitosa. Por lo tanto, $R(T)_{UCB}/T = 0.5$, dado que este algoritmo no tiene conocimiento del contexto. En cambio, UCB contextual o LinUCB podrán determinar cuál de las dos opciones es mejor y $R(T)/T$ tenderá a 0.

UCB contextual aún mantiene la dependencia con la cantidad de contextos. Por lo tanto, en un caso similar al anterior pero con muchos más contextos, LinUCB se comportaría mejor. Esto se debe a que, si se encuentra en una situación en la que la recompensa puede definirse de manera lineal en función del contexto, LinUCB aprenderá cuál es el mejor θ para cada acción. En contraste, UCB contextual tendría que aprender no solo por cada acción sino también por cada contexto, lo que lo hará converger más lento.

El modelo lineal de LinUCB es simple, pero ha demostrado ser efectiva en una variedad de escenarios [2] [5] [3] [6] [4]. Considerando un ejemplo ilustrativo, si una persona tiene un nivel socioeconómico bajo, es probable que no pueda permitirse una vivienda en el centro de la ciudad o en un barrio privado. Por otro lado, si la persona tiene un buen poder adquisitivo, es más probable que prefiera invertir más en una vivienda de mayor calidad. Este ejemplo, respaldado por experimentos y una selección adecuada de contextos, ilustra la capacidad del método LinUCB.

Todas estas premisas serán confirmadas mediante simulaciones en el capítulo 7 que compararán LinUCB, LinUCB Global y el método que se propone en esta tesis.

4.3. Conclusiones

Desde el principio de esta tesis se ha recorrido una variedad de métodos de MAB, desde los más simples hasta los más complejos, demostrando matemáticamente y a través de simulaciones cómo afectan al regret esperado. En el capítulo 7,

haremos simulaciones de algunos métodos contextuales. Hemos señalado las ventajas y desventajas de cada uno de ellos, así como los escenarios en los que pueden rendir mejor, lo cual generalmente se reduce a la verificación de sus premisas y si estas se cumplen en los contextos de aplicación. Por ejemplo, los métodos adaptativos suelen ser más efectivos cuando existe una gran diferencia en las recompensas entre las diferentes acciones, lo que permite descartar opciones de forma más eficiente. Por otro lado, si esta diferencia no es significativa, es posible que métodos como el de Epsilon Decay sean una mejor opción.

En cuanto a los métodos contextuales, LinUCB ha demostrado dar muy buenos resultados en la práctica, pero siempre que se pueda expresar de alguna manera la recompensa en función del contexto de manera lineal o similar. Es importante recordar que todas estas conclusiones se basan en las premisas de que tanto el contexto como la recompensa son i.i.d. Esta es una premisa bastante fuerte y, en caso de no cumplirse, otros métodos pueden dar un regret menor.

Capítulo 5

Algoritmo Propuesto

En este capítulo ofreceremos un nuevo enfoque al desafío que plantean los "multi-armed bandits" contextuales, basandonos en el enfoque de LinUCB y otros algoritmos del estado del arte en MAB [3] [4] [6] [5]. Además, se presentará una mejora en el coste computacional al algoritmo original, utilizando Recursive Least Squares (RLS).

5.1. LinUCB usando Mínimos Cuadrados Recursivos

Para el Algoritmo 1 es necesario calcular $\theta = D^{-1}b$, donde D es una matriz de dimensiones $d \times d$. El cálculo de la inversa de esta matriz, especialmente cuando d es grande, resulta muy costoso. Cabe destacar que en la literatura se encuentran valores de d igual a 10 [2] o 20 [6], pero este número depende de la cantidad de características utilizadas, por lo que podría ser aún mayor.

Al incorporar RLS [8] llegamos al Algoritmo 2:

Algorithm 2 Cálculo de θ usando RLS

Inicialización:

- $P_0 = I_{d\times d}/\lambda$
- $\theta_0 = \varphi$

for `t=1,...,T` **do**

$k_t = \frac{P_{t-1}x_t}{1+x_t^T P_{t-1}x_t}$

$P_t = P_{t-1} - k_t x_t^T P_{t-1}$

$e_t = r_t - x_t^T \theta_{t-1}$

$\theta_t = \theta_{t-1} - k_t e_t$

end for

donde $P_t = D_t^{-1}$, y $\varphi \sim N(0, \sigma), \varphi \in \mathrm{R}^d$ representa un ruido gaussiano, que puede ser tratado como cero, pero añadir un poco de aleatoriedad resulta beneficioso para las simulaciones, permitiendo observar el comportamiento del algoritmo ante pequeñas variaciones.

El Algoritmo 2 es más eficiente computacionalmente, ya que en cada ronda evita el cálculo de la inversa de la matriz D. Sin embargo, un problema que suele surgir es que P_t tiende a decrecer con el tiempo, lo que eventualmente puede provocar un *underflow* y una representación poco fiel de P_t (Lo mismo ocurre si se utilizan D y b). Para abordar este problema, se puede introducir una tasa de olvido en el algoritmo, denotada como $0 < \eta < 1$. Generalmente se utilizan valores cercanos a 1, por ejemplo 0.999. Al incorporar el nuevo hiperparámetro, se obtiene el Algoritmo 3.

Algorithm 3 Cálculo de θ usando RLS y factor de olvido

Inicialización:

- $P_0 = I_{d\times d}/\lambda$
- $\theta_0 = \varphi$

for `t=1,...,T` **do**

$k_t = \frac{\eta^{-1}P_{t-1}x_t}{1+\eta^{-1}x_t^T P_{t-1}x_t}$

$P_t = \eta^{-1}(P_{t-1} - k_t x_t^T P_{t-1})$

$e_t = r_t - x_t^T\theta_{t-1}$

$\theta_t = \theta_{t-1} - k_t e_t$

end for

Este enfoque no solo resuelve algunos problemas computacionales, sino que también elimina la suposición de que el regret es independiente del tiempo. Si el contexto cambia progresivamente, el algoritmo olvidará gradualmente la información anterior y se adaptará más rápidamente a las nuevas circunstancias, ofreciendo así una alternativa para abordar el mismo problema planteado en [5].

5.2. ClinUCB

En esta sección se explorarán diversas variantes de LinUCB que existen en la literatura, y se presentará el nuevo algoritmo propuesto en esta tesis, denominado ClinUCB. En los siguientes capítulos, se comparará este algoritmo con simulaciones de métodos contextuales clásicos, así como con una evaluación de rendimiento utilizando los datos propuestos en el paper original de LinUCB [2] contra los algoritmos del estado del arte.

En la literatura relacionada con LinUCB, se encuentran diversas variantes como CLUB [3]. Este algoritmo asume que todos los vectores pertenecen a un grafo completamente conectado y, a medida que se obtienen recompensas, elimina las conexiones que superan un umbral específico. SCLUB [4], una mejora de CLUB, permite la fusión de clústeres si estos se acercan lo suficiente. DLinUCB [5] es una modificación de LinUCB que se adapta a cambios en el comportamiento de las recompensas, permitiendo que el algoritmo reaprenda cuál es la opción óptima en entornos dinámicos. CodBand [6] combina la capacidad de adaptación a cambios temporales con la posibilidad de hacer clustering, asumiendo que cada clúster describe una distribución y calculando la probabilidad de que un usuario pertenezca

a un clúster, asignándolo al que tenga la mayor probabilidad. Todos los anteriores fueron analizados para implementar el algoritmo propuesto.

5.2.1. Vector de comportamiento

En la literatura se pueden encontrar varias definiciones con respecto al vector de comportamiento θ. En el paper original [2], se define un vector θ_a para cada opción a y $\hat{\theta}_{a,t}$ es la aproximación temporal de θ_a en tiempo t. Sin embargo, en otros trabajos y conjuntos de datos, se introduce el concepto de usuario $u \in U$. En cada iteración t, se considera un usuario u con su vector asociado $x_{t,u}$. Si además consideramos las acciones, $x_{t,a,u} = [x_{t,u}, x_{t,a}]$ es la concatenación de $x_{t,u}$ y $x_{t,a}$. Para reflejar esta dependencia, varios algoritmos [3] [6] [4] definen un vector de comportamiento diferente para cada usuario, θ_u y su aproximación temporal $\hat{\theta}_{u,t}$, en lugar de hacerlo por cada a. En consecuencia, la recompensa se expresa como:

$$\mathbb{E}_{r_t,a}[r_t|x_{t,a,u}] = x_{t,a,u}^T \theta_u \tag{5.1}$$

El vector de contexto, al depender del usuario u, describe las características que influyen en la decisión de ese usuario y en qué medida lo hacen. Estas características pueden incluir el género, la edad, la ubicación, el poder adquisitivo, entre otros. En otras palabras, este enfoque describe cuál es el contexto óptimo para cada usuario en lugar de describir el contexto óptimo para cada acción y cómo estos contextos afectan a la recompensa.

La elección entre usar un vector θ por cada usuario o por cada acción a dependerá de varios factores, como la cantidad de usuarios y acciones, así como de la información disponible sobre cada uno. Además, la decisión también estará influenciada por el comportamiento deseado del algoritmo. Recordar que a medida que la cantidad acciones K aumenta, el algoritmo se vuelve más versátil, pero al mismo tiempo, requiere más iteraciones en converger. Por lo tanto, si se busca una convergencia más rápida, posiblemente sea mejor utilizar el conjunto que tenga menor cantidad de elementos, lo cual suele ser las acciones. Si se desea un algoritmo más flexible y capaz de ofrecer respuestas más personalizadas a cada usuario, puede ser preferible implementarlo por usuario.

En todo caso independientemente de si se elige un vector de contexto por usuario o por acción a, el algoritmo intentará inferir las características de los acciones y usuarios a partir de estos contextos.

Por ejemplo, si hay 2 usuarios y 10 opciones, se obtiene un total de 20 posibles vectores $x_{a,u}$. Suponiendo que se distribuyen de manera uniforme, si se elige tener un vector θ por cada usuario, se tendrá dos θs, cada uno con 10 contextos. Viceversa tendremos 10 θs con 2 contextos cada uno. Esto puede afectar el aprendizaje y la generalización del algoritmo en función de la cantidad y la diversidad de datos disponibles.

En ciertos casos, el valor esperado de la recompensa será más dependiente con los usuarios, mientras que en otros estará más dependiente de las acciones. Esto puede influir en la efectividad del algoritmo y en la interpretación de los resultados. Por lo tanto, en el proceso de diseño y evaluación del algoritmo, es fundamental

realizar experimentos con ambas opciones y tener en cuenta el contexto específico de los datos.

En este trabajo, seguiremos la propuesta en CLUB [3], SCLUB [4] y CodBand [6]: utilizar un θ para cada usuario. No obstante, también se llevarán a cabo experimentos utilizando un θ por cada acción a para evaluar y comparar su rendimiento. Esto permitirá obtener una comprensión más completa de cómo se comporta el algoritmo en diferentes configuraciones y contextos de datos.

5.2.2. LinUCB Disjunto, Hibrido y Global

Como se ha mencionado en el paper original [2], se presentan dos variantes del algoritmo: el *Disjunto* y el *Híbrido*. En el *Disjunto*, como se describió anteriormente, se utiliza un parámetro θ_a para cada acción a. Por otro lado, el *Híbrido* introduce un nuevo parámetro β, que cumple la misma función pero se comparte entre todas las opciones. La recompensa esperada se define de la siguiente manera:

$$\mathbb{E}_{r_t,a}[r_t|x_{t,a}] = x_{t,a}^T\theta_a + x_{t,a}^T\beta \tag{5.2}$$

Existe una tercera variante que es un θ para todas las opciones posibles. Este algoritmo se conoce como LinUCB Global o LinUCB One o simplemente LinUCB.

Es interesante notar que LinUCB Global se destaca por su rápida convergencia, a pesar de sacrificar algo de versatilidad en comparación con sus contrapartes. Esta característica es particularmente efectiva cuando los valores de θ_a son muy similares entre sí, lo que resulta en un bajo error y una convergencia más rápida. Sin embargo, en ciertos escenarios, puede ser beneficioso transicionar gradualmente de *LinUCB Global* a *LinUCB Disjunto*. En las siguientes secciones se propondrá una forma de realizar esto.

5.2.3. Clustering

Ya sea que se elija utilizar un vector de comportamiento por acción a o por usuario u, es común que la cantidad de interacciones por acción (T/K) o por usuario (T/U) sea relativamente baja, a menudo solo cientos o miles en total. Considerando el caso de Amazon [9], la mayoría de los usuarios realizan menos de una compra por semana en Estados Unidos, lo que significa que no superan las cien compras al año. Por lo tanto, como se observará en las simulaciones, en estos casos, inicialmente, el rendimiento de LinUCB Global es superior a LinUCB, y a medida que aumenta el número de muestras, LinUCB mejora gradualmente. Esto sugiere que un enfoque que aproveche el comportamiento inicial de LinUCB Global y luego incremente su especificidad podría ser prometedor. Además, es plausible que los usuarios tengan preferencias generales que el algoritmo puede aprender.

Ciertos algoritmos como CLUB [3], SCLUB [4] y CodBand [6] han optado por estrategias que comienzan con un único clúster, similar al comportamiento de LinUCB Global, y luego incrementan la cantidad de clústers a medida que se obtiene más información y se puede determinar con mayor certeza las preferencias del usuario. Este enfoque permite adaptarse a la naturaleza de los datos y mejorar

la personalización del modelo con el tiempo. Esta aproximación es similar a cómo funcionan servicios como YouTube: cuando un usuario entra por primera vez, las recomendaciones suelen ser bastante genéricas y se basan en patrones de comportamiento de la población en general, considerando factores como la ubicación, la edad y otras características demográficas. A medida que el usuario interactúa más con la plataforma y consume contenido, el algoritmo comienza a personalizar las recomendaciones, teniendo en cuenta los gustos y preferencias individuales. Este proceso gradual de aprendizaje permite que las recomendaciones sean cada vez más personalizadas y relevantes para el usuario.

Para lograr esto, propusimos una serie de modelos $M_1, \ldots, M_m$, donde $M_j = \{c_1^j, \ldots, c_j^j\}$, siendo M_j uno de ellos con j particiones del conjunto de usuarios y M_t^* el modelo seleccionado en la ronda t. La diferencia en dichos modelos es la cantidad de clústers j, los cuales se determinan utilizando un clustering jerárquico aglomerativo con complete-linkage [10].

Para elegir el mejor M_t^*, se definira $R_{T,1}, \ldots, R_{T,m}$, $0 \leq R_{T,j} \leq 1$, que se calculan de la siguiente manera:

$$R_{T,j} = \sum_{t=1}^{T-1} \frac{r_t P_{t,a,c}(r_t = 1 | x_{t,a,u})}{T} = \sum_{t=1}^{T} \frac{r_t x_{t,a,u}^T \hat{\theta'}_{t,c}}{T} \tag{5.3}$$

donde $P_{t,a,c}(r_t = 1 | x_{t,a,u}) = \hat{\theta}_{t,c}^T x_{t,a,u}$ es una variable que como demostraremos tiende a una probabilidad, y representa una estimacion del valor esperado por el M_j en el cluster c_k^j del usuario u de que la recompensa sea 1, dado el contexto $x_{t,a,u}$ y la acción a. Ahora para demostrar que $P_{t,a,c}(r_t = 1 | x_{t,a,u})$ tiende una probabilidad, surge de que LinUCB asume lo siguiente:

$$E\left[r \middle| x\right] = E\left[\theta^T x + \nu\right] = P(r = 1 | x) \tag{5.4}$$

Y como el valor esperado de la recompensa es:

$$E\left[r \middle| x\right] = 0P(r = 0 | x) + 1P(r = 1 | x) = P(r = 1 | x) \tag{5.5}$$

Entonces el estimado $x_{t,a,u}^T \hat{\theta}_{t,c}$ tiende a una probabilidad.

Habiendo calculado $R_{T,j}$, se aplica el Criterio de Información de Bayes (BIC), que equilibra el ajuste del modelo y su complejidad, penalizando el número de parámetros. En este caso, al tener un θ por cada cluster y siendo el vector de dimensión d, el número total de parámetros es $j \cdot d$. Se selecciona el modelo M_j que minimiza:

$$j \leftarrow \arg\min_j \{-\ln(R_{T,j}) + j \cdot d \ln(t)/t\}$$

$$M_t^* \leftarrow M_j$$

Volviendo a la ecuación (5.4), si el modelo asigna un gran $P_{t,a,c}(r_t = 1 | x_{t,a,u})$ a la acción seleccionada en esa iteración, se recompensará si acierta; si no, se le castigará. Lo contrario aplica si el modelo asigna un bajo $P_{t,a,c}(r_t = 1 | x_{t,a,u})$; se

recompensará si acierta y se castigará si no, por lo tanto $R_{T,j}$ es una representación de una "recompensa promedio esperada" por cada Modelo.

Notar que en la ecuación (5.3) se usa θ_c, lo cual es el valor de comportamiento si todas las acciones a de los usuarios $u \in c$ se usaran para estimar un único θ_c. Por ende, surge la necesidad de calcular este nuevo parámetro y demostraremos que se puede calcular a partir de los datos de los usuarios, más específicamente de $D_{u,t} \in \mathbb{R}^{d\times d}, b_{u,t} \in \mathbb{R}^d / \theta = D_{u,t}^{-1} b_{u,t}$. Inicializando $D_{u,0} = I_{d\times d}\lambda$ y $b_{u,0} = 0_d$, siendo λ el parámetro de regularización de Ridge Regression:

$$D_{u,t_u} = \lambda I_{d\times d} + \sum_{i\in t_u} x_{i,a} x_{i,a}^T \tag{5.6}$$

$$b_{u,t_u} = \sum_{i\in t_u} x_{i,a} r_i \tag{5.7}$$

donde t_u son las iteraciones para el usuario u. Suponiendo que los usuarios en un mismo cluster c comparten el mismo θ_c, se cumpliran las ecuaciones (5.8) y (5.9).

$$D_{c,t} = \lambda I_{d\times d} + \sum_{u\in c} \sum_{i\in t_u} x_{i,a} x_{i,a}^T \tag{5.8}$$

$$b_{c,t} = \sum_{u\in c} \sum_{i\in t_u} x_{i,a} r_i \tag{5.9}$$

Observar que también se pueden usar las ecuaciones (5.6) y (5.7) en (5.8) y (5.9), obteniendo (5.10) y (5.11), lo cual es una forma de expresar D_c y b_c de un clúster a partir de D_u y b_u de los usuarios. Esto permite calcular θ_c sin necesidad de conocer todo el historial de cada usuario, sino los parámetros del clúster:

$$D_{c,t} = \lambda I_{d\times d} + \sum_{u\in C} (D_{u,t} - \lambda I_{d\times d}) \tag{5.10}$$

$$b_{c,t} = \sum_{u\in C} b_{u,t} \tag{5.11}$$

También es interesante ver qué sucede con $\text{UCB}_t = \alpha\sqrt{x_{t,a}^T D_{t,c}^{-1} x_{t,a}}$. Recordar que este es un valor que indica un intervalo de confianza que se va reduciendo al aumentar la cantidad de muestras y α es un hiperparámetro para ajustar este intervalo, [2] [11]. Por ende, al aplicar (5.10) y (5.11) sobre las muestras de todos los usuarios en un clúster c, se consigue un intervalo más pequeño y, por ende, una convergencia más rápida. Aplicando todo lo anterior se obtiene el Algoritmo 4.

5.2.4. Actualización de los vectores de comportamiento

Para la actualización de los θ, se utilizará el mismo enfoque que en el caso de CLUB [3], donde se emplea la regresión ridge de LinUCB para actualizar el θ_u del usuario que realizó la elección y nada más. En trabajos futuros, se podría explorar

Algorithm 4 ActualizarRecompensas:

Entradas:

$\hat{\theta}'_{t,1}, \ldots, \hat{\theta}'_{t,m}$ la aproximacion de los vectores comportamiento de los clústers en tiempo t

$R_{t,1}, \ldots, R_{t,m}$ recompensas promedio de los modelos en t

$x_{t,a}$ vector de comportamiento

r_t recompensa en t

Ejecución:

for j **in** $1, \ldots, m$ **do**

$p = \text{máx}\left(0, \hat{\theta}'_{t,j} x_{t,a}\right)$

$R_{t+1,j} = \left(R_{t,j} t + p r_t\right) / (t+1)$

end for

la posibilidad de compartir la recompensa entre los diferentes usuarios en el mismo clúster.

La actualización será de la siguiente forma, al igual que en LinUCB:

- $D_{t+1,u} = D_{t,u} + x_t x_t^T$
- $b_{t+1,u} = b_{t,u} + r_t x_t$

5.2.5. Algoritmo de clústers

Una de las ventajas del algoritmo propuesto respecto a otros como CLUB [3], SCLUB [4], o CodBand [6] es que no impone un método específico para realizar el clustering. En este caso, se definió una forma de elegir la cantidad óptima de clústers, pero el algoritmo de clustering en sí podría ser cualquiera que permita agrupar las θ en una cantidad de clúster específica. Esto proporciona flexibilidad para adaptarse a diferentes conjuntos de datos y necesidades específicas del problema.

Dicho esto, usamos complete linkage, ya que tiene varias ventajas. Por ejemplo, solo se calcula la distancia euclidiana entre todos los puntos una sola vez reusando estos valores para diferentes cantidades de clústers.

Otra ventaja es que, al agregar un clúster, este es una subdivisión de un clúster anterior, lo que hace que sea más fácil comparar.

En trabajos futuros, sería interesante no solo comparar diferentes cantidades de clústers, sino también utilizar el mismo algoritmo para comparar diferentes modelos de MAB. Esto abriría la posibilidad de explorar cómo diferentes enfoques de clustering afectan el rendimiento del algoritmo en términos de precisión y eficiencia computacional. Además, permitiría una evaluación más exhaustiva de las decisiones de diseño del algoritmo.

5.2.6. ClinUCB - Algoritmo

Resumiendo todo lo anterior, se procederá a definir el algoritmo. Se supondrá que se está trabajando en un entorno de aprendizaje por refuerzo donde en cada ronda $t \leq T \subseteq \mathbb{N}$, un agente deberá decidir una acción $a \in A$ de un conjunto de acciones posibles, conociendo una serie de contextos $X_t = \{x_{t,a_1}, \ldots, x_{t,a_K}\} \subseteq \mathbb{R}^d$ donde $||x_{t,a_t}|| = 1$. Después de realizar la acción a_t, se recibe una recompensa r_t con $r_t \in \{1, 0\}$. Considerando las mismas premisas que LinUCB, se supone que la esperanza de la recompensa puede describirse como:

$$\mathbb{E}_{r_t}[r_{t,a}|x_{t,a}] = x_{t,a}^T \theta_u \tag{5.12}$$

donde $\theta_u \in \mathbb{R}^d$ es un vector de comportamiento de un usuario u y $\hat{\theta}_{t,u}$ es su aproximación en la ronda t. Se supone que cada usuario u pertenece a un grupo de usuarios c, y si $u \in c \Rightarrow \theta_u = \theta'_c + \nu$ con $\nu \sim \mathcal{N}(0, \delta)$, lo que significa que cada θ_u de cada u difiere en un ruido ν del vector de comportamiento del clúster, quedando la esperanza expresada de la siguiente forma:

$$\mathbb{E}_{r_t}[r_{t,c}|x_{t,a}] = x_{t,a}^T \theta'_c \tag{5.13}$$

Usando todo lo anterior, se obtiene el Algoritmo 5. **Notar que los apóstrofes** se usan para indicar las variables **pertenecientes a un clúster** c y diferenciarlas de las de los usuarios. Por ejemplo, θ'_c es el vector comportamiento del clúster c a diferencia de θ_u que es el vector del usuario u, lo mismo con b', D' y las otras variables.

El método "ActualizarRecompensa" ya fue explicado y corresponde al Algoritmo 4. El Algoritmo 6 es el método jerárquico que también se ha mencionado, el cual asocia a cada usuario con un clúster usando complete linkage.

Con esto último, el algoritmo ClinUCB queda completamente definido. En la siguiente subsección se realizará una última mejora al coste computacional.

5.2.7. ClinUCB recursivo

Como se mencionó anteriormente, es posible aplicar RLS a LinUCB para mejorar el costo computacional. Al aplicar el mismo enfoque en CLinUCB, obtuvimos el siguiente Algoritmo 7.

Una mejora importante es la reintroducción del hiperparámetro η como término de olvido. Esto produce los mismos efectos que ya fueron mencionados, previniendo el underflow y mejorando la adaptación ante cambios dinámicos, como se mostrará en las simulaciones y pruebas reales. Esto acerca más el algoritmo a lo que serían algoritmos como dLinUCB [5] o CoDBand [6].

Esto concluye el capítulo con el nuevo algoritmo, el cual es capaz de adaptarse a lo largo de las iteraciones, de ajustarse a cambios dinámicos en la recompensa y de obtener patrones en los comportamientos de los usuarios, muy similar a otros algoritmos como [4] y [6], pero con sus propias ventajas, como veremos posteriormente.

Algorithm 5 ClinUCB

Entradas:

$x_{1,1}, \ldots, x_{1,K}, x_{2,1}, \ldots, x_{T,K}$: contextos para los T rounds y K acciones
$u_1, \ldots, u_T$: usuarios

Parámetros:

$\lambda \in \mathbb{R}^+$: parámetro de regularización de la regresión
$\alpha \in \mathbb{R}^+$: parámetro de exploración
τ: cada cuantas iteraciones se vuelven a calcular los clústers
σ ruido para inicializar $\hat{\theta}$: debe ser cercano a 0

Inicialización:

$b_{0,u} \leftarrow \vec{0}$
$D_{0,u} \leftarrow \lambda I_{d\times d}\ \forall u$
$\hat{\theta}_{0,u} \leftarrow \nu_u\ \forall u$ donde $\nu_u \sim N(0, \sigma)$
$\hat{\theta'}_{0,c} \leftarrow \vec{0}\ \forall c$
$R_{t,j} \leftarrow 1\ \forall j$
$M_1, \ldots, M_m = \text{ActualizarClústers}(M_1, \ldots, M_m, \hat{\theta}_{0,1}, \ldots, \hat{\theta}_{0,U})$
$M_1^* \leftarrow M_1$

Ejecución:

for $t = 1$ **to** T **do**
Se obtiene c tal que $u_t \in c$
$D'_{t,c} \leftarrow \lambda I + (\sum_{v\in c} D'_{t,v} - \lambda I)$
$b'_{t,c} \leftarrow \sum_{v\in c} b_{t,v}$
$\hat{\theta'}_{t,c} \leftarrow D'^{-1}_{t,c} b'_{t,c}$
$a_t \leftarrow \arg\max_{a\in A} \left(\hat{\theta}'_{t,c} x_{t,a} + \alpha\sqrt{x_{t,a}^T D'^{-1}_{t,c} x_{t,a}}\right)$
Se observa la recompensa r_t, elegida por el usuario u
Actualizar los valores para los usuarios:

- $D_{t+1,u} \leftarrow D_{t,u} + x_t x_t^T$
- $b_{t+1,u} \leftarrow b_{t,u} + r_t x_t$
- $\hat{\theta}_{t,v} = \hat{\theta}_{t-1,v}\ \forall v \neq u$

$\hat{\theta}'_{t,c'} = \hat{\theta}'_{t-1,c'}\ \forall c' \neq c$
$\text{ActualizarRecompensa}(\hat{\theta}'_{t,1}, \ldots, \hat{\theta}'_{t,m}, R_{t,1}, \ldots, R_{t,m}, r_t, x_{t,a})$
if $t\,\%\tau == 0$ **then**
$\hat{\theta}_{t,u} \leftarrow D_{t,u}^{-1} b_{t,u} \forall u$
$\text{ActualizarClústers}(M_1, \ldots, M_m, \theta_{t,1}, \ldots, \theta_{t,U})$
end if
$j^* \leftarrow \arg\min_j \{-\ln(R_{t,j}) + j \cdot d\ln(t)/t\}$
$M_{t+1}^* \leftarrow M_{j^*}$
end for

Algorithm 6 ActualizarClústers($M_1, \ldots, M_m, \hat{\theta}_{t,1}, \ldots, \hat{\theta}_{t,U}$): Vuelve a calcular los clústers dados θs actualizados, usando complete linkage

Entradas:

$M_1, \ldots, M_m$ Modelos con los clusers
$\hat{\theta}_{t,1}, \ldots, \hat{\theta}_{t,U}$: vectores comportamiento.

Algoritmo:

for $k = m, m-2, \ldots, 2$ **do**
$c, c' = \arg\max_{c,c' \in M_k} d(c, c')$
$M_{k-1} = M_k - c - c' + c \cup c'$
end for

Algorithm 7 ClinUCB Recursivo

Entradas:

$x_{1,1}, \ldots, x_{1,K}, x_{2,1}, \ldots, x_{T,K}$: contextos para los T rounds y K acciones
$u_1, \ldots, u_U$: usuarios

Parámetros:

$\lambda \in \mathbb{R}^+$: parámetro de regularización de la regresión
$\alpha \in \mathbb{R}^+$: parámetro de exploración
τ: cuantas iteraciones se vuelven a calcular los clústers

Inicialización:

$b_{0,u} \leftarrow \vec{0}$
$D_{0,u} \leftarrow \lambda I_{d\times d} \forall u$
$\hat{\theta}_{0,u} \leftarrow \nu_u \ \forall u$ donde $\nu_u \sim N(0, \delta)$,con $\delta \sim 0$
$R_j \leftarrow 1 \ \forall j$
$P'_{0,c} \leftarrow I_{d\times d}/\lambda$
$M_1, \ldots, M_m = \text{ActualizarClústers}(M_1, \ldots, M_m, \hat{\theta}_{0,1}, \ldots, \hat{\theta}_{0,U})$
Se inicializa θ's:

- $D'_{0,c} \leftarrow \lambda I + (\sum_{v\in c} D'_{0,v} - \lambda I)$
- $b'_{0,c} \leftarrow \sum_{v\in c} b_{t,v}$
- $\hat{\theta'}_{0,c} \leftarrow D'^{-1}_{0,c} b'_{0,c}$

$M^*_1 \leftarrow M_1$

Ejecución:

for $t = 1$ **to** T **do**
Se obtiene c tal que $u_t \in c$
$a_t = \arg\max_{a\in A} \left(\theta_c^T x_{t,a} + \alpha\sqrt{x_{t,a}^T P_{t,c} x_{t,a}}\right)$
Se observa la recompensa r_t, elegida por el usuario u
Actualizar los valores para el clúster c

- $k'_t = \eta^{-1} \frac{P'_{t-1,c} x_t}{1+\eta^{-1} x_t^T P'_{t-1,c} x_t}$
- $P'_{t,c} = \eta^{-1}(P'_{t-1,c} - k'_t x_t^T D'_{n-1})$
- $e'_t = r_t - x_t^T \hat{\theta}'_{c,t-1}$
- $\hat{\theta}'_{c,t} = \hat{\theta}'_{c,t-1} - k'_t e_t$

Actualizar los valores para el usuario

- $D_{t,u} = D_{t-1,u} + x_t x_t^T$
- $b_{t,u} = b_{t-1,u} + r_t x_t$
- $k_t = \eta^{-1} \frac{P_{t-1,c} x_t}{1+\eta^{-1} x_t^T P_{t-1,c} x_t}$
- $P_{t,u} = \eta^{-1}(P_{t-1,c} - k_t x_t^T D_{t,u})$
- $e_t = r_t - x_t^T \theta_{c,t-1}$
- $\theta_{u,t} = \theta_{u,t-1} - k_t e_t$

$\text{ActualizarRecompensa}(\bar{\theta}_{1,t}, \ldots, \bar{\theta}_{m,t}, R_1, \ldots, R_m, r_t, x_{t,a})$
if $t \,\%\tau == 0$ **then**
$\theta_u = D_{t,u}^{-1} b_{t,u} \forall u$
$\text{ActualizarClústersRecursivo}(M_1, \ldots, M_m, b_1, \ldots, b_U, D_1, \ldots, D_U)$
end if
$j^* = \arg\min_j \{-\ln(R_{t,j}) + j \cdot d\ln(t)/t\}$
$M^*_{t+1} = M_{j*}$
end for

Algorithm 8 ActualizarClústersRecursivo: Vuelve a calcular los clústers dado los θs actualizados

Entradas:

$M_1, \ldots, M_m$
$b_{t,1}, \ldots, b_{t,U}$
$D_{t,1}, \ldots, D_{t,U}$

Ejecución:

for $k = m, m-2, \ldots, 2$ **do**
  $c, c' = \arg\min_{c,c' \in M_k} d(c, c')$
  $M_{k-1} = M_k - c - c' + c \cup c'$
end for
for $c = 1, \ldots, m$ **do**
  $D'_{t,c} = (\lambda I + (\sum_{u \in c} D_{t,u} - \lambda I))$
  $P'_{t,c} = D'^{-1}_{t,c}$
  $b'_{t,c} = \sum_{v \in c} b_{t,v}$
  $\hat{\theta}'_c = D'^{-1}_{t,c} b'_{t,c}$
end for

Capítulo 6

Simulación MAB contextual

En este capítulo realizamos una serie de simulaciones para verificar la mejora en el costo computacional de usar mínimos cuadrados recursivos (RLS) en LinUCB, asi como las simulaciones para comprobar el regret de los diferentes modelos contextuales.

6.1. Simulaciones de performance

Se realizarán 3 casos de estudio: LinUCB Global, LinUCB y el nuevo algoritmo propuesto ClinUCB, cada uno comparando el método usando RLS y sin usar RLS.

En las simulaciones se graficara el tiempo medio con sus percentiles 25 y 75, con $T = 10000$ y se usarán $K = 10$ acciones con sus contextos y el mismo se compondrá de la concatenación de las características de las acciones y 100 usuarios. Esto es un caso común ya que uno suele tener más usuarios que acciones. En cada ronda se elegirá un usuario al azar y, por ende, el contexto cambiará en cada iteración.

Como esta mejora de rendimiento es afectada principalmente por la dimensión del contexto d, graficaremos como la misma afecta al tiempo de ejecución.

6.1.1. LinUCB Global

Empezando con LinUCB Global, este debería ser el algoritmo más rápido ya que calcula un solo θ en vez de uno por usuario.

En la figura 6.1 se puede observar que hay una clara mejora en tiempo al usar RLS. Si bien calcular formalmente el orden de complejidad de los métodos está fuera del alcance de este trabajo. Sin embargo, observamos que el uso de RLS reduce el tiempo en un 76 % promedio. Este cálculo se realizó para valores de $d = 1, 2, 4, \ldots, 512$.

6.1.2. LinUCB

En la figura 6.2 se observa que usando RLS una reducción del 64 % promedio. Comparando con la figura 6.1, se observa un tiempo un 10 % mayor para el que no

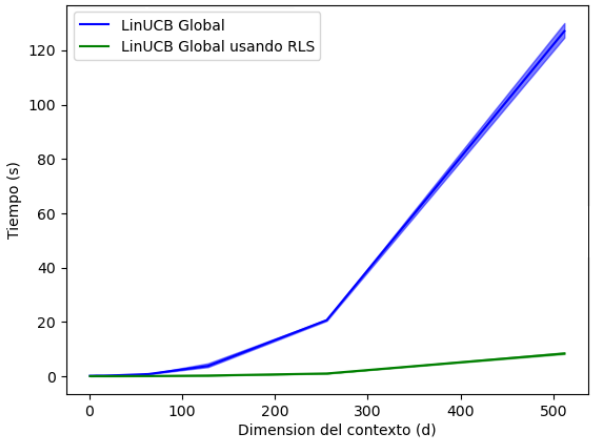

Figura 6.1: Tiempo de ejecución de LinUCB Global al variar la dimensión del contexto d, donde observamos que usar RLS en promedio 76 % mas veces más rápido.

usa RLS y casi el doble de tiempo para el que usa RLS. Esto se debe a que este método tiene un mayor número de parámetros que LinUCB Global.

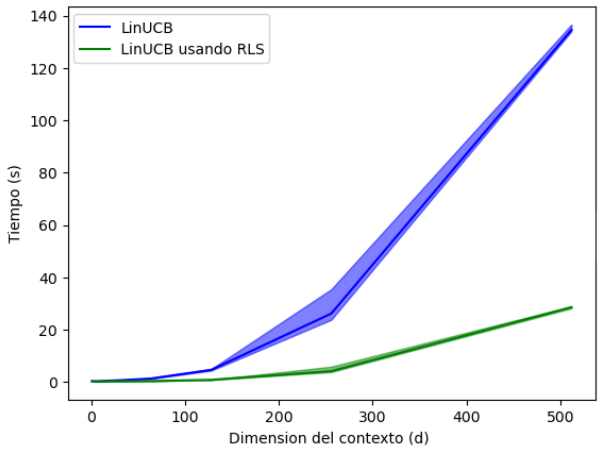

Figura 6.2: Tiempo de ejecución de LinUCB variando la dimensión del contexto d, siendo el LinUCB usando RLS un orden magnitud más rápido.

6.1.3. CLinUCB

En este último caso, se realizó clustering cada 500 iteraciones, tomando 3 clustering posibles. Si se observa la figura 6.3, se sigue viendo una mejora en el tiempo lo cual es interesante ya que el calculo de los θ en el clustering podría llevar más tiempo que el obtenido por la mejora. Esto dependerá cada cuántas iteraciones se haga clustering ya que a menos número de iteraciones se van a realizar más clustering y cálculos de los θs.

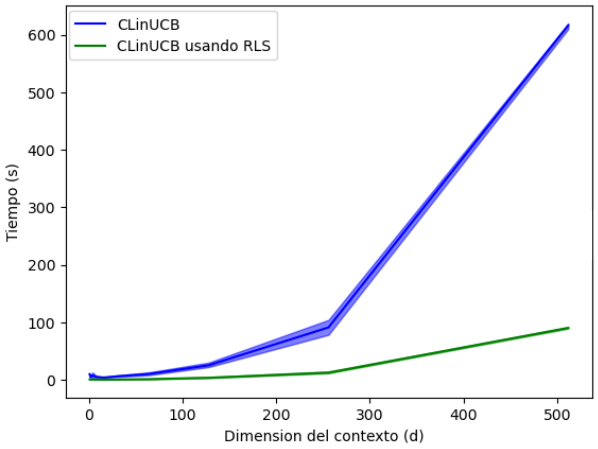

Figura 6.3: Tiempo de ejecución de CLinUCB variando la dimensión del contexto d.

6.2. Simulación LinUCB con clustering

En esta sección realizamos simulaciones comparando la recompensa total $\sum_t^T r_t$ de LinUCB, LinUCB Global y el algoritmo propuesto, CLinUCB. Para mantener la coherencia con el siguiente capítulo, se graficará la recompensa en vez del regret en función de la cantidad de iteraciones. Esto se debe a que en el próximo capítulo, al evaluar estos algoritmos con datos reales, no se conocerá la recompensa óptima, lo que hace imposible graficar el regret.

En las siguientes simulaciones se utilizarán 100 acciones a con vectores de contexto x al azar de norma 1. Este procedimiento, como se ha discutido, es estándar en la literatura. Cada simulacion contará con una cantidad de C clústers y con $1000/C$ usuarios por clúster. En cada iteración, el usuario será elegido al azar y el algoritmo podrá recomendar cualquiera de las 100 acciones a disponibles. Se utilizará $d = 2$, lo que facilitará el análisis y permitirá graficar los θ. Cada una de las simulaciones se realizará 100 veces y se graficarán las medias, y percentiles 25 y 75.

Las simulaciones comenzarán con un solo clúster, luego con dos, variando la posición de los centros de los clústers, y finalmente con cuatro clústers.

Cada uno de los clústers tendrá un $\|\theta_c\| \leq 1$. En cuanto a los θ_u, serán distribuidos uniformemente por cada clúster y tendrán el valor $\theta_u = \theta_c + \nu_c$, donde $\nu_c \sim \mathcal{N}(0, \sigma)$ y $\|\theta_c\| = 0.3$. Una vez que el algoritmo elija una acción, la probabilidad de que la recompensa sea 1 será $P(r = 1|x) = \text{máx}(\theta_u^T x, 0)$, por lo tanto, sin ruido se cumple que $P(r = 1|x) \leq 0.3$. Se eligió 0.3 ya que es un valor que se puede encontrar en datos reales [6] [5] y es lo suficientemente grande para no necesitar millones de iteraciones para converger a un valor, y como los θ_c son elegidos al azar, el mejor promedio obtenible es $r^* = 0.3$.

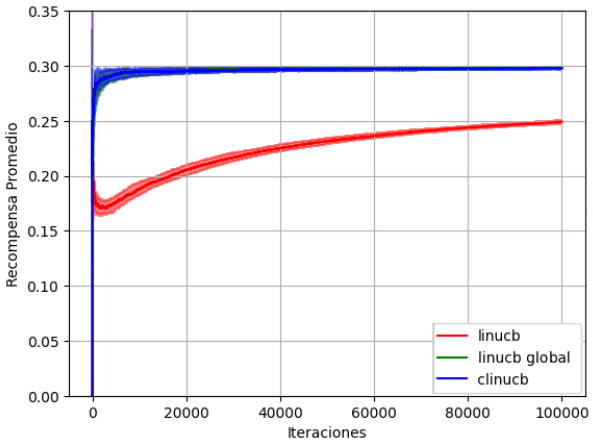

Figura 6.4: Gráfica de recompensa con un solo clúster, en la cual ClinUCB y LinUCB Global tienen la mejor recompensa y practicamente coinciden.

Figura 6.5: θs estimados y reales con un solo clúster.

6.2.1. Un clúster

Se comenzó simulando un solo clúster. Los resultados obtenidos se muestran en la figura 6.4. Como se puede apreciar, el algoritmo propuesto tuvo un mejor rendimiento y logró equipararse a LinUCB Global. Se puede verificar que se detectó el clúster al comparar los valores reales y estimados de θ mostrados en la figura 6.5. Tener en cuenta que la diferencia de la forma de los clúster no afecta al rendimiento del mismo ya que se toma el promedio del clúster para elegir la acción.

6.2.2. Dos clústers

Ahora, simularemos un peor caso del algoritmo cuando los clústers están en puntos opuestos al origen porque como veremos el algoritmo propuesto tiene peor rendimiento al detectar clústers que están distanciados entre sí. Consideraremos dos clústeres, uno en la posición (0.3,0) y el otro en la posición (-0.3,0). Se ob-

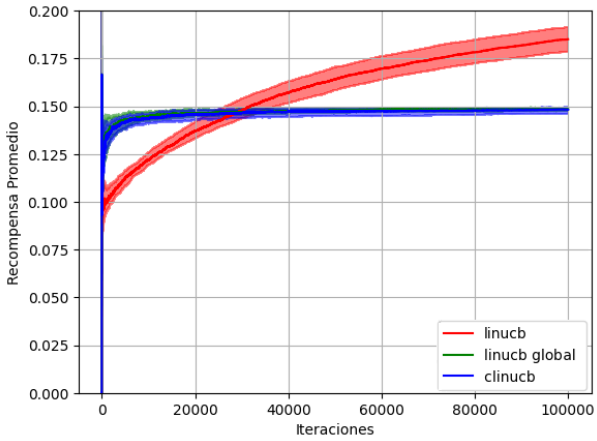

Figura 6.6: Gráfica de recompensa con dos clústers (0.3,0),(-0.3,0). LinUCB obtiene una recompensa un $24\,\%$ mayor porque ClinUCB no detectó el segundo clúster como se ve comparando la figura 6.7.

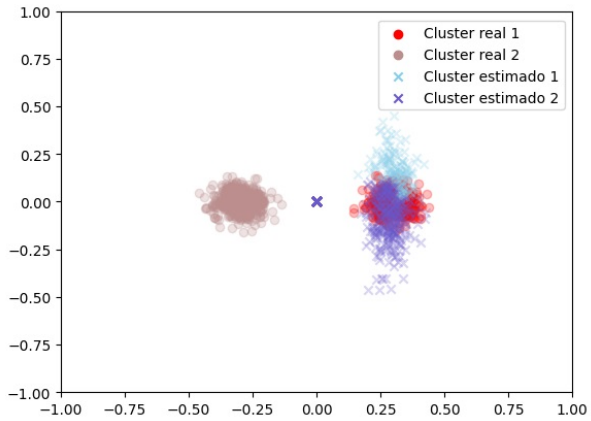

Figura 6.7: θ estimados con dos clústers, donde solo se detecto el clúster en $(0.3, 0)$ y no el clúster en $(-0.3, 0)$.

tuvieron los resultados que se muestran en la figura 6.6. Al principio, se logra el objetivo deseado, que es una rápida convergencia del algoritmo similar a LinUCB Global, pero luego LinUCB supera ampliamente a ClinUCB.

Observando la figura 6.7, concluimos que el algoritmo propuesto comienza recomendando opciones con vectores del clúster (0.3,0), lo que lleva a que los θs converjan rápidamente hacia ese punto. Sin embargo, no realiza ninguna recomendación para acciones en la dirección opuesta. Esto se debe a que un pequeño cambio en los θ no cambiará la dirección del vector hacia (-0.3,0) y, por lo tanto, no converge. Por esta razón, LinUCB, que converge individualmente para cada θ, tiene un mejor desempeño a largo plazo.

Una posible solución para abordar este problema es forzar al algoritmo a comenzar con múltiples clústers, esperando que alguno de ellos termine convergiendo hacia el segundo clúster. Sin embargo, en general, esta es una debilidad del algo-

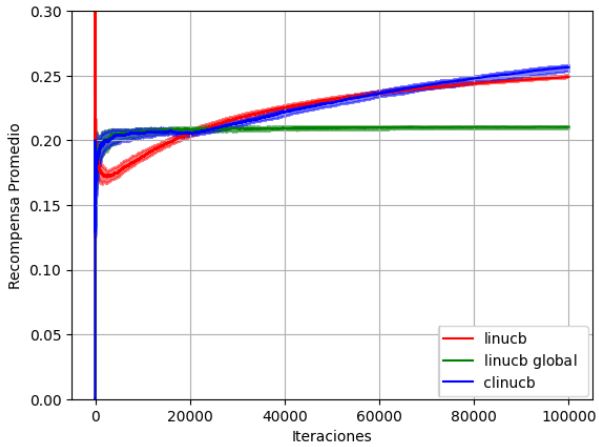

Figura 6.8: Gráfica de recompensa con dos clústers (0.3,0),(0,0.3), donde ClinUCB mejora con respecto al caso anterior porque los clústers están más cercanos entre si.

ritmo propuesto, especialmente cuando los clústers están muy separados y, sobre todo, si $\sum_{u \in U} \theta_u = 0$, lo que implica que no se tiene un factor compartido por todos los usuarios, lo cual debería ser un evento poco frecuente. Mejoras para evitar este comportamiento son posibles en trabajos futuros.

Un caso más favorable sería cuando se tienen dos clústers, uno ubicado en (0.3,0) y el otro en (0,0.3), lo que hace que $\sum_{u \in U} \frac{\theta_u}{K} = (0.3/2, 0.3/2)$. Esto permite que el algoritmo primero converja hacia $(0.3/2,\ 0.3/2)$ y luego se separe en dos clústers, logrando así el resultado deseado. La recompensa en este caso se gráfico en la figura 6.8.

Como se puede observar en la gráfica, inicialmente el algoritmo converge de manera más efectiva, simulando el desempeño de LinUCB Global. Luego pasa por una etapa de transición en la cual su rendimiento es ligeramente inferior al de LinUCB, y finalmente mejora.

La performance de CLinUCB y LinUCB se equipara recién después de alrededor de 20 mil iteraciones, todas las anteriores fueron, en promedio, peores. Además, inicialmente no se puede asegurar si se alcanzará esta convergencia; se necesitan al menos 20 muestras por usuario para que LinUCB supere a Global. Por lo tanto ClinUCB tiende a funcionar mejor en situaciones donde existen ciertas tendencias generales que se distribuyen en varios clústers, especialmente cuando estos clústers están más separados pero aún son detectables

En cuanto a los valores estimados de los θ, se puede observar en la figura 6.9 que el algoritmo detectó los dos clústers correctamente, lo que resultó en un mejor rendimiento en la mayoría de las iteraciones.

6.2.3. Cuatro clústers

Por último, simulamos con cuatro clústers ubicados a 20 y 60 grados del eje de las x en la esfera de radio 0.3, y con coordenadas de x positivas. Se obtiene la figura 6.10.

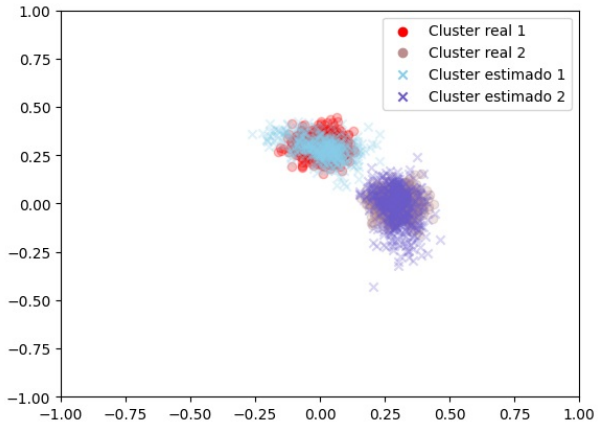

Figura 6.9: θs con dos clústers (0.3,0),(0,0.3) que se acercan más a los reales que en el caso de la figura 6.7.

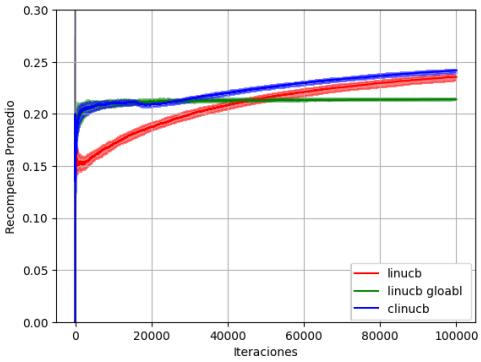

Figura 6.10: Gráfica de recompensa con cuatro clústers no equiespaciados, donde ClinUCB supera al resto de los algoritmos.

Los valores estimados se presentan en la figura 6.11. Como se puede observar,se detectaron los clústers exitosamente.

Una última observación es que existe una relación en la posición de los clústers y el rendimiento de ciertos algoritmos. Se ha observado que CLinUCB y LinUCB Global tienden a funcionar mejor cuando los clústers son cercanos entre si. Esta relación se verá nuevamente en la siguiente sección cuando se simule el CLinUCB con factor de olvido.

6.3. Simulaciones ClinUCB con olvido

Para las últimas simulaciones, se llevarán a cabo experimentos en los que se variará la posición de los valores θ_u en cada iteración para todos los usuarios. Este proceso de bucle por usuario consume bastante recursos computacionales. Por lo tanto, para poder llevarlo a cabo de manera continua, se ha decidido reducir el número de usuarios a 100.

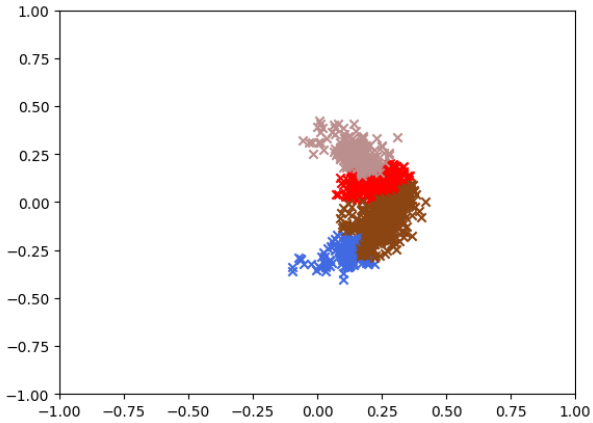

Figura 6.11: θs estimados con cuatro clústers, observamos que se detectaron los clústers correspondientes.

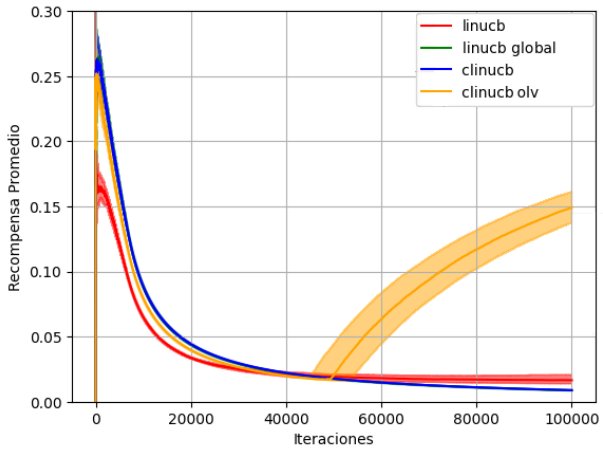

Figura 6.12: Gráfica de recompensa con un clúster variando θs. Observamos que ClinUCB con olvido toma entre 45000 a 55000 iteraciones en encontrar el nuevo clúster y partir de ahí obtiene una mejora en la recompensa.

6.3.1. Simulación con un clúster

Para comenzar, el primer escenario consistirá en un solo clúster con centro en (0,0.3), donde el centro del clúster cambiará en dirección $(0,-6 \cdot 10^{-5})$ en cada iteración hasta alcanzar la posición (0,-0.3).

Los resultados obtenidos son muy prometedores y se pueden apreciar en la figura 6.12. Como se observa, todos los algoritmos experimentan una disminución drástica en sus rendimientos, lo cual es lógico ya que al disminuir el valor de θ, la probabilidad asociada también disminuye, afectando así la recompensa esperada.

Sin embargo, el único algoritmo que logra adaptarse a la nueva tendencia es ClinUCB con olvido a partir de las 50000 iteraciones, lo que resulta en una mejora de rendimiento muy significativa.

Al observar los valores reales y estimados de θ en la figura 6.13, se puede

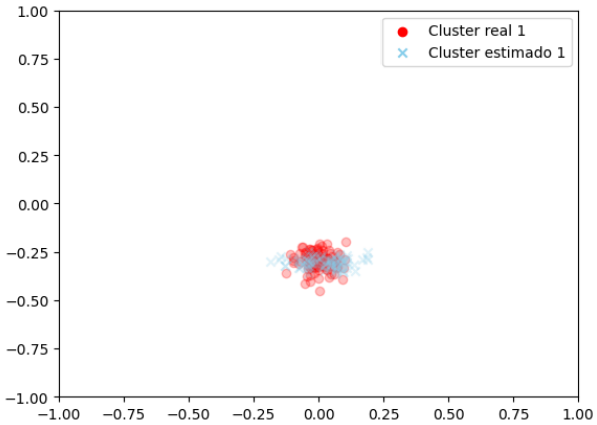

Figura 6.13: θs con un clúster variando θs, donde se comprueba que se convergió al nuevo clúster.

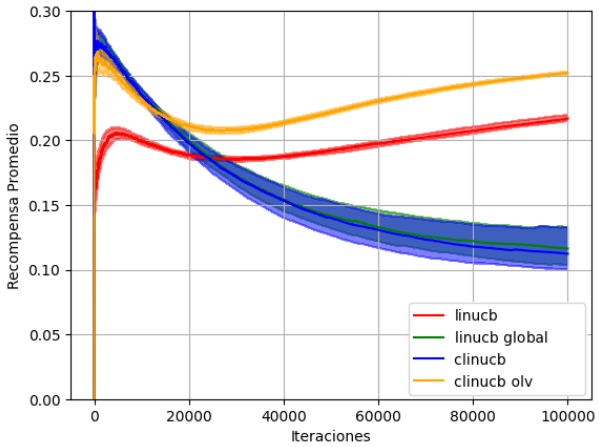

Figura 6.14: Gráfica de recompensa con un clúster variando θs hacia (0,0.3), donde ClinUCB con Olvido tiene una recompensa 25 % mayor que LinUCB pero que este logra detectar el nuevo clúster.

apreciar que se logró una convergencia exitosa hacia la nueva tendencia, algo que resulta difícil para el resto de los algoritmos.

Ahora, al probar un caso más favorable para los otros algoritmos, se llevará a cabo un procedimiento similar, pero esta vez se iniciará con un clúster desde (0,0.3) hasta (0.3,0). Dado que este cambio implica un ángulo menor en la variación de θ, se espera que LinUCB, LinUCB Global y CLinUCB tengan un mejor desempeño.

Como se puede observar en la figura 6.14, todos los algoritmos muestran un mejor desempeño. Esto se debe, en primer lugar, a que la probabilidad de la recompensa nunca se anula. Además, tanto LinUCB Global como ClinUCB gradualmente comienzan a recomendar opciones subóptimas pero más cercanas al óptimo.

Sin embargo, LinUCB tiene un rendimiento ligeramente superior debido a que su intervalo de confianza tiende a disminuir más lentamente que los otros algorit-

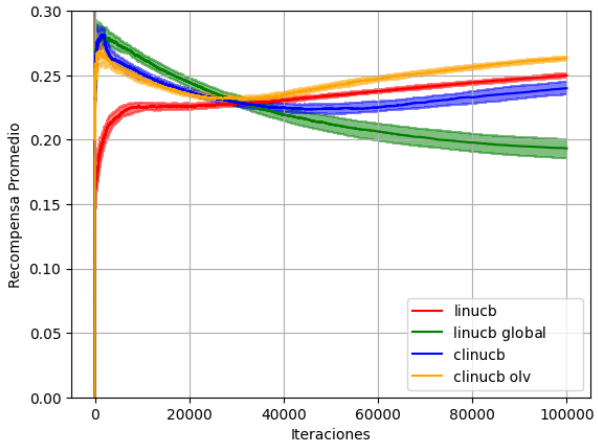

Figura 6.15: Gráfica de recompensa con dos clústers variando uno de ellos, donde observamos que todos los algoritmos mejoran sus recompensa con respecto al caso anterior

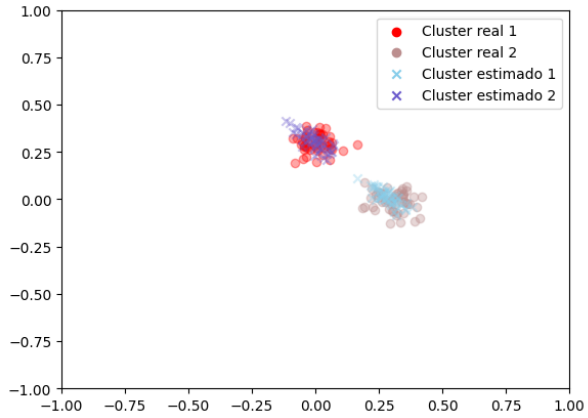

Figura 6.16: θs estimados con dos clústers variando θs

mos, lo que significa que no termina de converger cuando ocurre el cambio.

6.3.2. Simulación con dos clústers

Ahora se simulará que al principio todos los usuarios tienen cierta preferencia, pero que luego la mitad de ellos tiende al punto (0.3,0). El resultado se muestra en la figura 6.15.

Los resultados obtenidos por casi todos los algoritmos mejoraron, lo cual se debe al hecho de que el cambio en la tendencia es menor en este caso. Además, se observa que ClinUCB tuvo una mejor recompensa que LinUCB Global debido a que se tiene mas de un clúster.

Al observar las gráficas de los valores de θ 6.16, se puede apreciar que el algoritmo converge correctamente hacia el nuevo clúster.

Como última prueba, se tendrán dos clústers que comienzan en (0,0.3) juntos

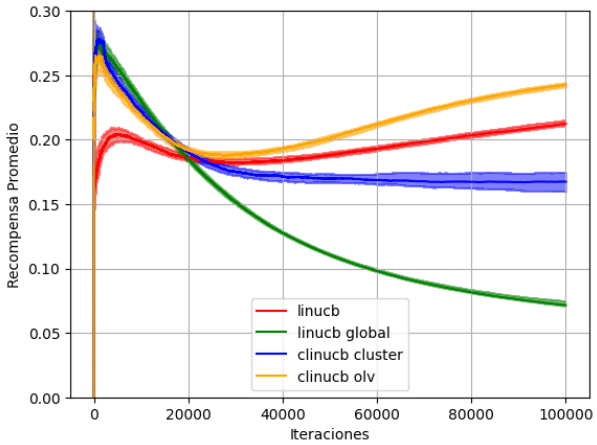

Figura 6.17: Gráfica de recompensa con dos clústers variando θ a puntos opuestos

y luego se separan a (-0.3,0) y (0.3,0), obteniéndose la figura 6.17.

Como se puede observar, los resultados son muy similares a los anteriores, con la única diferencia de que el desempeño de LinUCB Global es peor en este caso debido a que los clústers están más separados. Notar que este caso es similar al de la figura 6.6. Sin embargo, en este caso, ClinUCB con olvido mostró un mejor rendimiento que cualquier otro algoritmo en ambos casos. Esto posiblemente se deba a la capacidad de ClinUCB para detectar exitosamente los clústers. Los cambios graduales también pueden haber contribuido a su rendimiento, ya que le permiten converger incluso cuando los clústers terminan en posiciones opuestas. Como se observó previamente, ClinUCB a menudo tiene dificultades cuando un pequeño cambio en θ no resulta en una mejora del algoritmo, y se requieren cambios significativos para lograr la convergencia.

En conclusión, se puede observar que ClinUCB con olvido funcionó mejor que cualquiera de los otros algoritmos en estos experimentos. Esto es lógico, ya que es el único que considera la premisa de que los valores de θ_c (y por ende, la recompensa) pueden variar con el tiempo, lo cual es común en muchos escenarios de la vida real. Estas simulaciones demuestran que ClinUCB y ClinUCB con olvido tienen mucho que aportar en los escenarios adecuados, y representan una mejora con respecto a LinUCB y al enfoque global en estos casos.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 7

Evaluación con datos reales

En este capítulo, llevaremos cabo una evaluación del algoritmo propuesto utilizando datos reales. Para ello, realizaremos un preprocesamiento de los datos, seguido de una comparación con otros algoritmos del estado del arte.

7.1. Regret y Recompensa con datos reales

La discusión sobre los algoritmos de Multi-Armed Bandit se centra en la dificultad de conocer la recompensa ideal r^* en datos reales, ya que sería necesario conocer cuál es la acción con la máxima recompensa esperada para cada contexto en cada momento, lo cual es prácticamente imposible. Por lo tanto, en este capítulo se continuará utilizando la métrica de la recompensa promedio, que es más realista y práctica de medir.

Para evaluar la recompensa promedio, se puede considerar el enfoque de on-policy o off-policy [12]. Si se usara el enfoque on-policy, se podrían evaluar diferentes algoritmos en una plataforma online a lo largo de diferentes períodos de tiempo y observar cuál logra la mejor recompensa promedio. Sin embargo, este enfoque presenta ciertas limitaciones, como la disponibilidad de un entorno de pruebas en línea y el hecho de que solo se puede hacer una recomendación a la vez, lo que impide obtener información sobre cómo se habrían comportado los otros algoritmos.

El enfoque que se explorará a continuación es el uso de off-policy. Este método implica obtener un historial de diferentes recomendaciones realizadas en línea y las recompensas asociadas a esas recomendaciones, para luego utilizar este historial para entrenar otros algoritmos de forma offline. Sin embargo, uno de los principales desafíos radica en que generalmente solo se dispone de la respuesta del usuario para el ítem recomendado, y no para las otras opciones que el algoritmo pudo haber seleccionado.

Por ejemplo, consideramos la tabla 7.1, donde se presentan tres acciones y en cada ronda se recomienda una acción, obteniendo la recompensa asociada a esa elección. En el ejemplo de la tabla, tener en cuenta que el hecho de que una recompensa sea 1 no significa necesariamente que las otras recompensas sean cero.

a_1	a_2	a_3
0	Desconocido	Desconocido
Desconocido	1	Desconocido
Desconocido	Desconocido	1
Desconocido	0	Desconocido
⋮	⋮	⋮

Tabla 7.1: Ejemplo de datos obtenidos

Además, podría ocurrir que el usuario sea muy conformista y otorgue una recompensa positiva independientemente de la acción recomendada. En este caso, conocer la recompensa para una acción no garantiza nada sobre las recompensas de las otras opciones.

Una posibilidad para predecir estos valores desconocidos sería utilizar un simulador, pero esto implica añadir una capa adicional de complejidad, ya que se pasa de tener un problema a dos. El desafío consiste en conseguir un simulador de calidad que permita evaluar los diferentes algoritmos sin sesgos, para luego seleccionar el algoritmo que maximice la recompensa deseada.

Irónicamente, obtener un simulador de calidad suele ser más complejo que desarrollar un buen algoritmo. En el simulador, se busca estimar con precisión todas las posibles recompensas en los diferentes contextos para cada acción, mientras que en el segundo caso, se busca identificar cuál algoritmo tiene una mayor recompensa.

Otra estrategia es tomar todos los casos en los que la recompensa fue 1 y seleccionar otras p opciones adicionales aleatoriamente para presentar al algoritmo, quien debe elegir la acción correcta entre ellas. Esta técnica ha sido utilizada en varios estudios [3], [4], [5], [6], no obstante, no se puede garantizar que las pruebas sean fieles a la realidad.

En primer lugar, al descartar los datos en los que la acción no fue seleccionada, se están eliminando datos valiosos que podrían confirmar la idoneidad de una acción en un contexto dado. Además, pueden ocurrir casos que no son contemplados en este nuevo contexto de prueba. Por ejemplo, podría darse el caso de que varias opciones sean correctas o incluso que ninguna lo sea, mientras que en la simulación se asume que solo hay una acción correcta.

Destacamos que, en una simulación con estas características, la recompensa promedio al elegir al azar sería $\frac{1}{p}$ de forma constante. Sin embargo, en datos reales, esta recompensa promedio puede variar entre 0 y 1, e incluso puede fluctuar según lo sugieran los datos del conjunto de datos que se utilizará.

Existen varios métodos para comparar diferentes algoritmos en línea, como el que se explica en [13], que son muy interesantes ya que permiten verificar de manera sin sesgo si un algoritmo es mejor o peor que el actual. Sin embargo, dado que no se cuenta con un portal en línea para llevar a cabo estas pruebas y recopilar cientos de miles de registros, se optará por utilizar un enfoque off-policy pero sin sesgos, como se detallará a continuación.

7.1.1. Evaluación insesgada

Se seguirá la propuesta presentada en [14], donde se explica que si el contexto x consiste en muestras iid, al igual que las recompensas r, y se cuenta con un algoritmo de referencia ALG_{log} que selecciona muestras al azar, se puede evaluar otro algoritmo ALG deseado sin sesgo. Este enfoque se describe en el Algoritmo 9.

Algorithm 9 Off-policy sin sesgo para MAB

- Se observa el siguiente evento x
- Si dado ese evento la acción a'_i seleccionada por ALG_{log} es la misma que a_i para ALG, se ejecuta el algoritmo
- Sino se descarta

Como se puede observar, el algoritmo es muy sencillo, si hay una coincidencia entre el evento y el contexto, se utiliza ese evento; de lo contrario, se descarta. Sin embargo, este enfoque introduce el problema de que, en promedio, solo se utilizarán $\frac{T}{K}$ eventos, lo que significa que si K es muy grande, este algoritmo puede no ser práctico. En tales casos, sería necesario realizar K recomendaciones antes de poder utilizar una, lo que hace que el proceso de entrenamiento sea extremadamente costoso, ya que aumenta el tiempo de ejecución en un factor de K.

A pesar de este inconveniente, la evaluación sin sesgos que proporciona este enfoque es muy poderosa para comparar diferentes algoritmos, y representa un buen punto de partida para la comparación y evaluación de algoritmos.

7.2. Dataset

Teniendo en cuenta lo anterior, se necesita un conjunto de datos contextuales que tenga un gran número de muestras y que la relación $\frac{T}{K}$ sea lo suficientemente grande. Por suerte, el conjunto de datos del módulo de Yahoo [2] cumple con todos estos requisitos. Este conjunto de datos consiste en millones de recomendaciones recopiladas durante 10 días, en los cuales se presentan menos de 100 opciones que varían según el día y la hora.

Este conjunto de datos registra los *clicks* de los usuarios en artículos de noticias en el portal de Yahoo, específicamente en la pestaña "Destacados del módulo de Yahoo!", durante los primeros diez días de mayo de 2009 (ver imagen 7.1). Los artículos fueron seleccionados aleatoriamente de un conjunto de artículos producidos por periodistas de Yahoo. El conjunto de datos contiene registros de 45 millones de usuarios, mostrando si hicieron click o no en el artículo presentado. Consiste en 10 archivos, uno por cada día, con el siguiente formato en cada iteración:

$$1241160900, 109513, 0$$

por cada usuario:

$$0.000012, 0.000000, 0.000006, 0.000023, 0.999958, 1.000000$$

Figura 7.1: Portal de destacados de Yahoo donde se eligió la noticia principal para la Story

y por cada noticia:

$$109498, 0.306008, 0.000450, 0.077048, 0.230439, 0.386055, 1.000000$$

Lo cual contiene lo siguiente:

- Timestamp
- Id de la noticia mostrada
- 0 o 1 para mostrar si se cliqueo.
- Sección con las features del usuario, donde la última siempre vale 1
- Una serie de noticias que se usaron para seleccionar las features de los artículos y con un id y un valor de 1 al final.

El vector de características de la noticia suma 1 (sin contar el que tiene el valor 1). Esto se debe a que las características han sido normalizadas utilizando la norma 1 y han sido preprocesadas para preservar el anonimato. Además, es relevante mencionar que no se incluye el identificador de los usuarios en los datos. También se observa que tanto los usuarios como las acciones tienen la misma estructura en cuanto a características, aunque no está claro si existe alguna correlación más allá del número de parámetros.

7.2.1. Preprocesamiento

El procesamiento de los valores contextuales de las acciones o usuarios se realizó de la siguiente manera:

1. Se eliminaron los usuarios o acciones que no contenían todas las características. En todo el conjunto de datos, solo se encontraron dos casos de este tipo.

2. Se elimino el valor al final de los vectores ya que siempre vale 1.0, y no aporta información adicional.

3. Se guardó el identificador junto con el vector contextual asociado. En el caso de los usuarios, el identificador se generó como un hash de todo el contexto concatenado.

4. Para cada iteración t, se registró qué usuario fue y su correspondiente identificador, así como las posibles opciones de acción con sus identificadores.

Este enfoque elimina una gran cantidad de redundancia, con la única pérdida de dos acciones que presentaban información incompleta, así como sus iteraciones correspondientes.

7.2.2. Otros dataset

Encontrar un dataset con datos claros y utilizados en otros trabajos de investigación para una comparación precisa con otros algoritmos puede resultar complejo en machine learning. Por ello se han revisado otros conjuntos de datos utilizados en estudios como CLUB [3], SCLUB [4], dLinUCB [5], y CodBand [6]. Sin embargo, el único conjunto de datos que asegura pruebas sin sesgo es el de Yahoo, dado que ninguno de los otros estudios utilizó un algoritmo para recolectar datos de manera aleatoria, lo que podría introducir un sesgo dependiendo del algoritmo utilizado para la recolección de datos. Lo que emplean es el método previamente mencionado, en el cual filtran y retienen únicamente las recompensas positivas. Luego, introducen otras opciones de manera aleatoria, realizando modificaciones en diversas características de la experiencia, como, por ejemplo, la recompensa media.

Además, varios de estos estudios no proporcionan detalles claros sobre cómo se preprocesaron los datos, y tampoco se ofrece código relacionado, a excepción de dLinUCB [5] y CodBand [6]. Un ejemplo es el conjunto de datos MovieLens, donde se filtraron los usuarios que habían realizado 3000 reseñas de películas. Esta práctica puede ser cuestionable, ya que, incluso si un usuario calificara una película al día, le llevaría casi nueve años completar esta cantidad de reseñas. Si bien puede ser cierto que existan alrededor de 100 usuarios con estas características entre cientos de miles, claramente representan casos atípicos en comparación con el comportamiento general de los usuarios, lo que pone en duda la validez de este procesamiento previo.

7.2.3. Ajuste de parámetros

Para el ajuste de los hiperparámetros, utilizamos el primer día. En cuanto al parámetro de olvido η, realizamos un grid search entre 0.900 a 1.000 y el valor que produjo mayor recompensa fue 0.999. También realizamos un random search para los valores de α y λ utilizando el rango estándar de 0 a 1. Además, actualizamos los clústers cada 1000 iteraciones, ya que no se observó una mejora significativa al hacerlo con más frecuencia.

Se llevaron a cabo 100 experimentos con valores aleatorios. Cada experimento se ejecutó con valores de θ distribuidos al azar en una gaussiana con centro en 0 y varianza de 0.1. Los cinco mayores resultados se muestran en la gráfica (ver Figura 7.2). Para llevar a cabo una comparación similar a la realizada en el artículo de LinUCB y otros trabajos, se normalizó la recompensa entre la probabilidad de

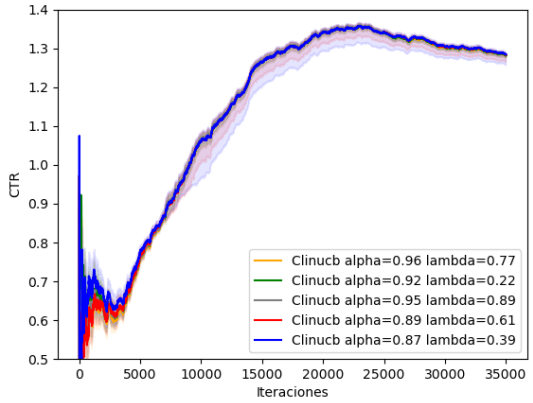

Figura 7.2: Recompensa Promedio del tuning, donde se muestran las 5 mejores gráficas obtenidas con una diferencia menor al 5 % entre cada una.

elegirla al azar. Este nuevo parámetro a menudo se conoce como *Click Through Rate* (CTR).

Como se puede observar en la gráfica 7.2, parece que el parámetro más influyente es α, el cual controla el equilibrio entre la exploración y la explotación de los algoritmos, así como el tamaño del intervalo de confianza. Por otro lado, λ parece tener menos impacto en el rendimiento. Por esta razón, utilizar valores seleccionados en un random search en lugar de una grid search predefinida resultó beneficioso, ya que permitió probar una variedad más amplia de valores para α sin tener que repetir el mismo valor varias veces al cambiar λ.

7.2.4. Prueba de Validación

Para la prueba con datos reales se usaron los parámetros óptimos encontrados ($\alpha = 0.96$ y $\lambda = 0.77$) y el mismo enfoque que se describió en [2].

Los resultados se muestran en la Figura 7.3. Se observa que ClinUCB supera a LinUCB gracias a su rápida convergencia en las primeras iteraciones, al igual que LinUCB Global, cuyo rendimiento también mejora con el tiempo. Sin embargo, lo más notable es que el mejor desempeño lo muestra ClinUCB con olvido. Esto posiblemente se deba a que las preferencias de los usuarios varían a lo largo del tiempo, lo que implica cambios en el conjunto de noticias relevantes y en la disposición de los usuarios a interactuar con ellas. Por ejemplo, es probable que los usuarios estén más interesados en noticias frescas que en aquellas que ya han sido mostradas.

En la figura 7.4, ClinUCB con olvido tiene un desempeño significativamente mejor con respecto a los otros algoritmos del artículo [5], especialmente durante el primer día. Esto puede atribuirse a que el ajuste de los parámetros se realizó ese día.

Sin embargo, varios estudios [3] [5] demuestran que, para este conjunto de datos, es mejor usar un θ por cada acción a y no por cada usuario u. Esto implica

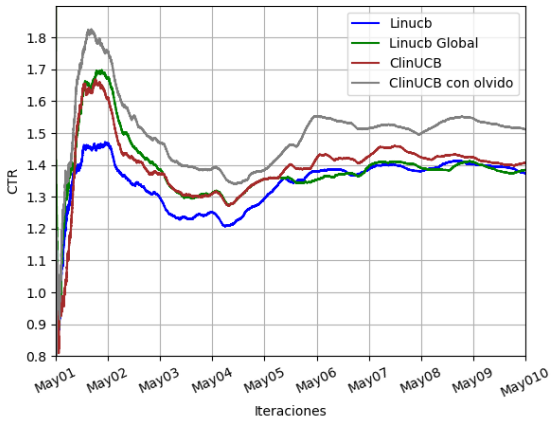

Figura 7.3: Recompensa Promedio del testing comparando ClinUCB con LinUCB y LinUCB Global, donde observamos que ClinUCB con olvido tiene un mejor CTR que los otros métodos.

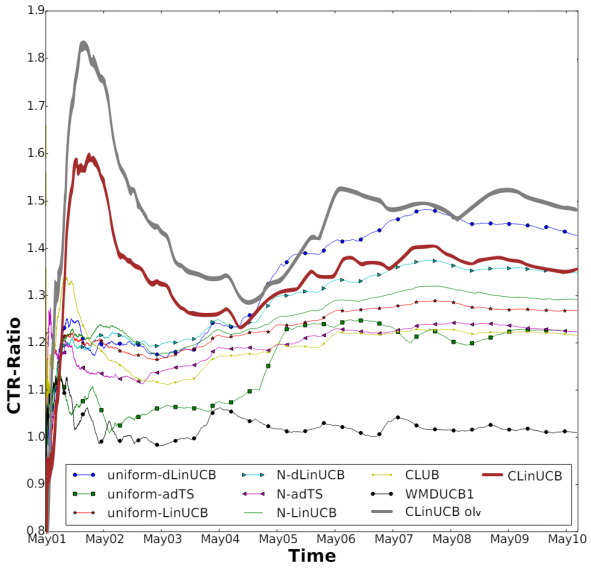

Figura 7.4: Recompensa Promedio del testing comparando con los resultados del artículo dLinUCB [5], donde se observa nuevamente que ClinUCB con Olvido tiene una mejor recompensa que los otros métodos del estado del arte.

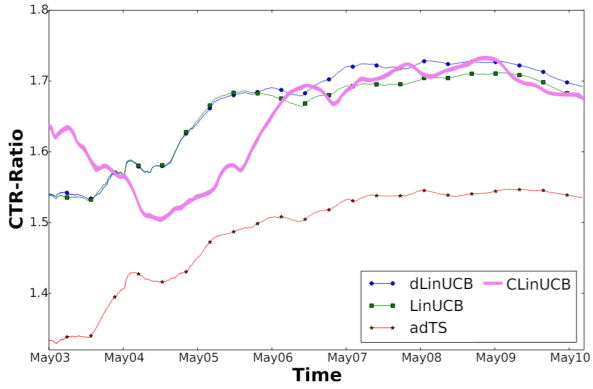

Figura 7.5: Recompensa promedio en testing, usando θ por cada accion en vez de usuario y comparándolo con los resultados obtenidos en el artículo [5]

que los clusters también deben agruparse por acción y no por usuario. Al hacer este cambio en el algoritmo propuesto, graficamos los resultados de los algoritmos a partir del tercer día, como se hizo en el artículo [5] y se obtuvieron los resultados que se muestran en la Figura 7.5. En esta figura, se puede observar que el rendimiento de ClinUCB con olvido es comparable al de dLinUCB, ambos utilizando información basada en las acciones. Esta similitud en el desempeño era esperada, ya que ambos algoritmos se basan en LinUCB . Sin embargo, en este caso, LinUCB tuvo un rendimiento similar a dLinUCB y ClinUCB. Es posible que el clustering por acción y la tasa de olvido no mejoren significativamente el desempeño debido a que la cantidad de acciones es baja (alrededor de 50 por día) lo que permite converger a los θ correspondientes a cada acción sin la necesidad de aplicar clustering.

Es muy satisfactorio comprobar que los resultados han sido muy buenos, ya que compiten con el estado del arte actual y demuestran la importancia de realizar clustering y poder adaptarse a cambios en los comportamientos de los usuarios.

Capítulo 8

Conclusiones y trabajos a futuro

En esta monografía, hemos logrado una exposición exitosa de los diferentes algoritmos básicos de Multi-Armed Bandit, detallando sus características y demostrando sus respectivas cotas para el regret y su relación con el "zero-regret". Hemos explorado las premisas subyacentes de cada algoritmo y hemos realizado simulaciones en diversos escenarios para comprender cómo diferentes parámetros afectan su desempeño.

Posteriormente, introdujimos el concepto de Multi-Armed Bandit Contextual, lo que amplía las posibilidades del algoritmo y mejora considerablemente la recompensa obtenida. Sin embargo, esta mejora conlleva el incremento en la cantidad de premisas y datos necesarios para su implementación. Esto ha sido corroborado mediante simulaciones en diversos escenarios, determinando en qué situaciones cada algoritmo tiene un mejor rendimiento.

Además, explicamos cómo utilizar el algoritmo RLS en LinUCB para mejorar su costo computacional, una técnica que no se ha utilizado en otros algoritmos similares. Las simulaciones muestran que esta mejora es significativa, posiblemente en un orden de magnitud, especialmente en dimensiones grandes.

Se han explorado también las variantes más clásicas de LinUCB, junto con la propuesta de un nuevo algoritmo, CLinUCB, el cual realiza clustering y los compara. Se ha demostrado que CLinUCB ofrece un rendimiento competitivo en comparación con el estado del arte actual, validando estos resultados con el dataset original utilizado para proponer LinUCB.

8.1. Trabajo futuro

8.1.1. Teoría

MAB es un tema interesante que se puede discutir desde muchos ángulos. A veces es difícil encontrar demostraciones detalladas de los algoritmos más simples, por ejemplo el regret de Greedy con $K = 2$ fue demostrado en esta tesis desde cero, y sería interesante extrapolar el caso para un numero genérico de K. En cuanto al algoritmo propuesto, se puede intuir que debería tener un regret similar al de

LinUCB debido a la demostración de CLUB [3]. No obstante, una demostración formal sería un añadido valioso para corroborar esta suposición.

8.1.2. Costo computacional

Sería importante analizar formalmente el orden de complejidad del algoritmo y ver cómo la recursividad mejora en cuanto a la dimensión del contexto d. Además, llevar el algoritmo al lenguaje C, utilizar caché y simularlo con estas optimizaciones proporcionaría mejoras significativas en el rendimiento del algoritmo. Estas optimizaciones serían necesarias para su implementación en un entorno comercial productivo.

8.1.3. ClinUCB

Hay muchos aspectos interesantes en este algoritmo, entre ellos la capacidad de utilizar diferentes algoritmos de clustering y observar cómo afectan al rendimiento del mismo. En ciertos escenarios, se ha observado que el algoritmo tiene dificultades para encontrar nuevos clústeres, especialmente cuando están muy separados del promedio general, lo cual lo hace un punto de mejora muy prometedor. A su vez se podría trabajar más en el tema de elección del mejor clustering. Por ejemplo, en vez de usar la cantidad óptima de clústeres, se podrían comparar diferentes modelos o diferentes métodos de clustering y obtener el mejor. Otra posibilidad sería ponderar los diferentes modelos y dar un resultado combinado.

Apéndice A

Anexo

En este anexo, se presentan los desarrollos matemáticos que sustentan las metodologías y algoritmos discutidos en el cuerpo principal del documento.

A.1. Simplificación de la fórmula de Regret del algoritmo Greedy

En esta sección simplificaremos la ecuación de regret de Greedy para obtener una versión más manejable. Recordemos que la ecuación de regret es la siguiente:

$$R(T) = p_2 - p_1 + (T-2)\left[p_2 - (p_2 P(a = a_2) + p_1 P(a = a_1))\right] \tag{A.1}$$

$$P(a = a_1) = (1 - p_2)p_1 + \frac{p'}{2}$$
$$P(a = a_2) = p_2(1 - p_1) + \frac{p'}{2}$$

$$p' := p_1 p_2 + (1 - p_1)(1 - p_2) \tag{A.2}$$

Primero desarrollaremos la ecuación (A.2):

$$p' = p_1 p_2 + (1 - p_1)(1 - p_2) = p_1 p_2 + 1 - p_1 - p_2 + p_1 p_2$$
$$\Rightarrow \frac{p'}{2} = \frac{1 - p_1 - p_2}{2} + p_1 p_2 \tag{A.3}$$

Volviendo a la ecuación (A.1) para simplificar expresaremos $R(T) - R(2)$, recordar que $R(2) = p_2 - p_1$

$$\begin{aligned} R(T) - R(2) &= \left[p_2 - \left[p_2\left(p_2(1 - p_1) + \frac{p'}{2}\right) + p_1\left(p_1(1 - p_2) + \frac{p'}{2}\right)\right]\right] \\ &= \left[p_2 - \left[p_2\left(p_2 - p_2 p_1 + \frac{p'}{2}\right) + p_1\left(p_1 - p_1 p_2 + \frac{p'}{2}\right)\right]\right] \end{aligned} \tag{A.4}$$

Usando el resultado de (A.3) en la anterior ecuación se tiene:

$$
\begin{aligned}
&= \left[p_2 - \left[p_2\left(p_2 + \frac{1-p_1-p_2}{2}\right) + p_1\left(p_1 + \frac{1-p_1-p_2}{2}\right)\right]\right] \\
&= \left[p_2 - \left[p_2\left(\frac{p_2+1-p_1}{2}\right) + p_1\left(\frac{p_1+1-p_2}{2}\right)\right]\right] \\
&= \left[p_2 - \left[(p_2-p_1)\left(\frac{p_2-p_1}{2}\right) + \frac{p_2+p_1}{2}\right]\right] \\
&= -(p_2-p_1)\frac{p_2-p_1}{2} + \frac{p_2-p_1}{2} \\
&= \left(\frac{p_2-p_1}{2}\right)\left[1-(p_2-p_1)\right]. \qquad \text{(A.5)}
\end{aligned}
$$

Despejando $R(T)$:

$$R(T) = p_2 - p_1(T-2)\frac{\Delta p}{2}\left[1-\Delta p\right] \qquad \text{(A.6)}$$

Definiendo $\Delta p = p_2 - p_1$, quedaría:

$$R(T) = \Delta p + (T-2)\frac{\Delta p}{2}\left[1-\Delta p\right] \qquad \text{(A.7)}$$

Y si suponemos que $T \gg 2$, la ecuación se simplifica en.

$$R(T) = (T-2)\frac{\Delta p}{2}\left[1-\Delta p\right] \qquad \text{(A.8)}$$

Este resultado da una fórmula más sencilla de usar y demuestra que si $p^* \approx p$, entonces $r^* - r \approx 0$ y en el caso en que $p^* \gg p$, la probabilidad de elegir la acción a sobre a^* es baja y $R(T) \approx 0$. Y además se puede ver que el máximo para este algoritmo en particular es cuando $\Delta p = \frac{1}{2}$.

A.2. Regret de Exploracion Uniforme

Se quiere hallar el N que minimice el regret, por eso derivaremos (2.12) que es convexa en los reales positivos y buscaremos el mínimo:

$$\frac{\delta(KN + 2T\sqrt{2\log(\tau)/N})}{\delta N} = K - N^{-3/2}\sqrt{2\log(\tau)}T \qquad \text{(A.9)}$$

Igualando a 0:

$$
\begin{aligned}
&K - N^{-3/2}\sqrt{2\log(\tau)}T = 0 \\
&\Rightarrow KN^{3/2} = (2\log(\tau))^{1/2}T \\
&\Rightarrow N = \left(\frac{T}{K}\right)^{2/3}(2\log(\tau))^{1/3} \qquad \text{(A.10)}
\end{aligned}
$$

Con el N óptimo dado por (A.10), lo aplicamos a (2.12), y obtuvimos la siguiente cota para el regret:

$$\begin{aligned} R(T) &< K\left(\frac{T}{K}\right)^{2/3}(2\log(\tau))^{1/3} + 2\sqrt{\frac{2\log(\tau)}{\left(\frac{T}{K}\right)^{2/3}(2\log(\tau))^{1/3}}T} \\ &< (2K)^{1/3}T^{2/3}\log(\tau)^{1/3} + 2\sqrt{\frac{(2K)^{2/3}\log(\tau)^{2/3}}{T^{2/3}}T} \\ &< 2^{1/3}T^{2/3}(K\log(\tau))^{1/3} + 2\sqrt{2}T^{2/3}(K\log(\tau))^{1/3} \\ &< 3\cdot 2^{1/3}T^{2/3}(K\log(\tau))^{1/3} \end{aligned} \tag{A.11}$$

Obteniendo la cota para el algoritmo de exploración uniforme que es lo que queríamos encontrar.

A.3. Regret Epsilon Decay

Calcularemos el regret para Epsilon Decay. Empezando con que el regret para una iteración t particular está acotado por la probabilidad de explorar multiplicada por el máximo regret en ese caso, que es 1, más la probabilidad de explotar multiplicada por el máximo regret, que es $2 \cdot \text{rad}$ ya que no es posible separar las medias más allá del intervalo de confianza en un "clean case". Entonce definimos $Reg(t)$ el regret en un ronda t, usando $\tau = t$:

$$\begin{aligned} Reg(t) = \mathbb{E}_{a_t,r_t}(r_t^* - r_t) &\leq P(\text{explorar})\cdot 1 + P(\text{explotar})\cdot 2\text{rad} \\ &= \epsilon_t + (1-\epsilon_t)\cdot 2\sqrt{\frac{2K\log(t)}{t\epsilon_t}} \\ &\leq \epsilon_t + 2\sqrt{\frac{2K\log(t)}{t\epsilon_t}} \end{aligned} \tag{A.12}$$

Hallaremos el mínimo $Reg(t)$, para de esa forma minimizar $R(T)$. Por eso derivaremos respecto a ϵ para encontrar el punto mínimo:

$$\frac{\delta\left(\epsilon + 2\sqrt{\frac{2K\log(t)}{t\epsilon}}\right)}{\delta\epsilon} = 0 \tag{A.13}$$

$$1 - \epsilon^{-3/2}\sqrt{\frac{2K\log(t))}{t}} = 0 \tag{A.14}$$

Despejando ϵ

$$\epsilon^{3/2} = \sqrt{\frac{2K\log(t))}{t}} \tag{A.15}$$

$$\epsilon = \left(\frac{2K\log(t)}{t}\right)^{1/3} \tag{A.16}$$

Sustituyendo (A.16) en (A.12):

$$\begin{aligned} Reg(t) &\leq \left(\frac{2K\log(t)}{t}\right)^{1/3} + 2\sqrt{\frac{2K\log(t)}{t\left(\frac{2K\log(t)}{t}\right)^{1/3}}} \\ &= \left(\frac{2K\log(t)}{t}\right)^{1/3} + 2\left(\frac{2K\log(t)}{t}\right)^{1/3} \\ &= 3\left(\frac{2K\log(t)}{t}\right)^{1/3} \end{aligned} \tag{A.17}$$

Ahora calculando el regret sumando en T:

$$\begin{aligned} R(T) &\leq \sum_{t=1}^{t=T} 3\left(\frac{2K\log(t)}{t}\right)^{1/3} \\ &\leq 3(2K\log(T))^{1/3}\sum_{t=1}^{t=T}\frac{1}{t^{1/3}} \\ &= 3(2K\log(T))^{1/3}(1+\sum_{t=2}^{t=T}\frac{1}{t^{1/3}}) \end{aligned} \tag{A.18}$$

Acotaremos la suma $\sum_{t=2}^{T}\frac{1}{t^{1/3}}$ por la integral $\int_1^T \frac{1}{t^{1/3}}$.

$$\begin{aligned} R(T) &\leq 3(2K\log(T))^{1/3}(1+\int_1^T t^{-1/3}) \\ &= 3(2K\log(T))^{1/3}(1+3/2(T^{2/3}-1)]) \\ &< 3(2K\log(T))^{1/3}T^{2/3} \end{aligned} \tag{A.19}$$

Referencias

[1] Aleksandrs Slivkins, "Introduction to multi-armed bandits," *ArXiv*, 2019. [Online]. Available: https://arxiv.org/abs/1904.07272

[2] John Langford y Robert E. Schapire Lihong Li, Wei Chu, "A contextual-bandit approach to personalized news article recommendation," *Nineteenth International Conference on World Wide Web (WWW 2010), Raleigh, NC, USA, 2010*, 2012. [Online]. Available: https://arxiv.org/abs/1003.0146

[3] Shuai Li y Giovanni Zappella Claudio Gentile, "Online clustering of bandits," *ArXiv*, 2014. [Online]. Available: https://arxiv.org/abs/1401.8257

[4] Wei Chen; Shuai Li y Kwong-Sak Leung Shuai Li, "Improved algorithm on online clustering of bandits," *ArXiv*, 2019. [Online]. Available: https://arxiv.org/abs/1902.09162

[5] Naveen Iyer y Hongning Wang Qingyun Wu, "Learning contextual bandits in a non-stationary environment," *ArXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1805.09365

[6] Chuanhao Li; Qingyun Wu y Hongning Wang, "When and whom to collaborate within a changing environment: A collaborative dynamic bandit solution," *ArXiv*, 2021. [Online]. Available: https://arxiv.org/abs/2104.07150

[7] Olivier Cappe´y Aurelien Garivier Emilie Kaufmann, "On the complexity of a/b testing," *ArXiv*, 2015. [Online]. Available: https://arxiv.org/pdf/1405.3224

[8] Haykin, *Adaptive Filter Theory, 4ta edicion, capitulo 9.* Pearson Education, 2002.

[9] Jack Flynn, "25+ amazon statistics [2023]: Facts about the largest u.s. e-commerce market," *Zippia*, 2024. [Online]. Available: https://www.zippia.com/advice/amazon-statistics/

[10] Prabhakar Raghavan y Hinrich Schütze Christopher D. Manning, "Introduction to information retrieval," *Stanford University*, 2008.

[11] Carlos Diuk y Michael L. Littman Thomas J. Walsh; Istv´an Szita, "Exploring compact reinforcement-learning representations with linear

regression," *ArXiv*, 2012. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1205/1205.2606.pdf

[12] Richard S. Sutton y Andrew G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[13] ACM RecSys, "Counterfactual learning and evaluation for recommender systems," in *RecSys 2021*, 2021.

[14] John Langford y Xuanhui Wang Lihong Li, Wei Chu, "Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms," *ArXiv*, 2012. [Online]. Available: https://arxiv.org/abs/1003.5956

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.
Compilado el lunes 19 mayo, 2025.
http://iie.fing.edu.uy/