



# Bandidos Contextuales: Fundamentos y Aplicaciones

Contextual Bandits: Foundations and Applications

Iván Hernández Roldán y Alejandro Magarzo Gonzalo

Dirigido por Miguel Palomino Tarjuelo

Doble Grado: Ingeniería Informática y Administración y Dirección de Empresas

Trabajo de Fin de Grado en Ingeniería Informática

Facultad de Informática, Universidad Complutense de Madrid

Curso académico 2022/2023

# Resumen

Como punto de partida, se abordan los fundamentos teóricos subyacentes a los bandidos multi-brazo, preparando así el terreno para la profundización en los bandidos contextuales. Los bandidos, como elemento fundamental en el aprendizaje por refuerzo, ofrecen una respuesta eficiente a los problemas básicos del dilema de la exploración frente a la explotación. Un problema de bandidos implica un juego secuencial entre un agente y un entorno, donde en cada ronda el agente tiene varias acciones a su disposición y debe elegir una para recibir la recompensa correspondiente como resultado. Basado en las recompensas anteriores, el agente deberá mejorar su toma de decisiones para obtener la máxima recompensa acumulada al final del juego, manteniendo un balance entre explorar acciones menos probadas y explotar la mejor acción según la información que posee.

Además, se explican los bandidos estocásticos y antagonistas como preludio para presentar varios algoritmos que serán de gran utilidad en una variante particular del modelo de bandidos: los bandidos contextuales. En este tipo de bandido, cada acción disponible está asociada a una distribución de probabilidad de recompensas, desconocida de antemano por el agente, de la cual se obtiene la recompensa correspondiente tras elegir una acción. Por lo tanto, el agente tratará de maximizar sus recompensas eligiendo los brazos que mayor recompensa media tengan en función del contexto.

A lo largo de este trabajo se presentan los algoritmos que resuelven los problemas de los bandidos planteados y se comparan sus rendimientos a través de la métrica del remordimiento. También, se tratan las diferencias entre los remordimientos de los algoritmos que se adaptan al contexto y los que no gracias a la exposición de un juego contextual. Tras abordar cada concepto teórico del ámbito de los bandidos contextuales, se expone una aplicación práctica en consonancia para estudiar el desempeño de los bandidos contextuales en diversos dominios. Las principales aportaciones prácticas de este trabajo se localizan dentro del sector financiero, concretamente en el departamento de la automatización de la inversión en el mercado de valores a través de los bots de comercio, y en el mundo digital, realizando un sistema recomendador de películas.

## Palabras clave

Bandidos multi-brazo. Exploración-explotación. Remordimiento. Bandidos estocásticos. Bandidos antagonistas. Bandidos contextuales. Clase política. Exp4. Bots de comercio. Sistema recomendador.

# Abstract

As a starting point, the theoretical foundations underlying multi-armed bandits are addressed, thereby laying the groundwork for a deep dive into contextual bandits. Bandits, as a fundamental element in reinforcement learning, provide an efficient response to the basic problems of the exploration-exploitation dilemma. A bandit problem involves a sequential game between an agent and an environment, where in each round the agent has several actions at his disposal and must choose one to receive the corresponding reward as a result. Based on past rewards, the agent should improve his decision-making to obtain the greatest accumulated reward at the end of the game, maintaining a balance between exploring lesser-tested actions and exploiting the best action according to the information he possesses.

In addition, stochastic and adversarial bandits are explained as a prelude to presenting various algorithms that will be extremely useful in a particular variant of the bandit model: the contextual bandits. In this type of bandit, each available action is associated with a probability distribution of rewards, unknown to the agent beforehand, from which the corresponding reward is obtained after choosing an action. Therefore, the agent will try to maximize his rewards by choosing the arms with the highest average reward based on the context.

Throughout this work, algorithms that solve the problems posed by bandits are presented and their performances are compared through the metric of regret. Also, the differences between the regrets of algorithms that adapt to the context and those that do not are discussed, thanks to the exposition of a contextual game. After addressing each theoretical concept in the field of contextual bandits, a practical application is presented to study the performance of contextual bandits in various domains. The main practical contributions of this work are located within the financial sector, specifically in the department of automating investment in the stock market through trading bots, and in the digital world, by implementing a movie recommendation system.

## Keywords

Multi-armed bandits. Exploration-exploitation. Regret. Stochastic bandits. Adversarial bandits. Contextual bandits. Policy class. Exp4. Trading bots. Recommendation system.

# Índice

<b>1. Introducción</b>	<b>6</b>
<b>2. Introduction</b>	<b>9</b>
<b>3. Problema de los Bandidos Multi-brazo</b>	<b>12</b>
3.1. Características y Diferenciaciones . . . . .	12
3.2. Tipos de Bandidos . . . . .	13
3.3. Aplicaciones de los Bandidos Contextuales . . . . .	13
<b>4. Bandidos Estocásticos</b>	<b>15</b>
4.1. Algoritmos Simples . . . . .	16
4.1.1. Algoritmo Exploración-Primero . . . . .	16
4.1.2. Algoritmo Épsilon-Avaricioso . . . . .	17
4.2. Algoritmos Avanzados: Exploración Adaptativa . . . . .	18
4.2.1. Algoritmo Eliminación Sucesiva . . . . .	18
4.2.2. Algoritmo UCB1 . . . . .	19
<b>5. Bandidos Antagonistas</b>	<b>21</b>
5.1. Primer Intento de Solución . . . . .	23
5.2. Algoritmos Efectivos . . . . .	24
5.2.1. Algoritmo Hedge . . . . .	24
5.2.2. Algoritmo Exp3 . . . . .	25
5.2.3. Algoritmo Exp4 . . . . .	27
<b>6. Bandidos Contextuales</b>	<b>29</b>
6.1. Bandidos Contextuales con Pocos Contextos . . . . .	30
6.1.1. Recomendador de Películas para Cuatro Grupos . . . . .	31
6.2. Bandidos Contextuales Lipschitzianos . . . . .	35
6.2.1. Motivación . . . . .	35
6.2.2. Implementación de Bandidos Contextuales Lipschitzianos del Mercado . . . . .	36
6.2.3. Demostración de Lipschitz para Bandidos Contextuales Lipschitzianos . . . . .	40
6.3. Bandidos Contextuales con Clase Política . . . . .	41
6.3.1. Algoritmo Exp4 con Políticas en lugar de Expertos . . . . .	42
<b>7. Bandidos Contextuales en Juegos Contextuales</b>	<b>44</b>
7.1. Introducción del Sistema . . . . .	44
7.2. Juego Contextual . . . . .	45
7.3. La Red . . . . .	45
7.4. Algoritmos . . . . .	46
7.4.1. Algoritmo Hedge . . . . .	46
7.4.2. Algoritmo GPMW . . . . .	49
7.4.3. Algoritmo cGPMW . . . . .	51
7.5. Conclusiones . . . . .	54
<b>8. Bots de Comercio</b>	<b>59</b>
8.1. Planteamiento . . . . .	59
8.2. Expertos . . . . .	61
8.2.1. Experto en Posiciones Cortas . . . . .	61
8.2.2. Experto en Posiciones Largas . . . . .	61
8.2.3. Experto en Posiciones Cambiantes . . . . .	62
8.2.4. Experto en Cruce de Medias Móviles . . . . .	62
8.2.5. Experto en RSI . . . . .	63
8.2.6. Experto en MACD y Bandas de Bollinger . . . . .	64
8.2.7. Experto en Todo . . . . .	64
8.3. Análisis de la Solución Planteada . . . . .	65
8.3.1. Consideraciones previas . . . . .	65
8.3.2. Análisis del Horizonte Temporal . . . . .	67
8.3.3. Análisis de la Tasa de Aprendizaje . . . . .	69
8.3.4. Análisis de la Tasa de Exploración . . . . .	72
8.3.5. Puesta en Práctica de la Solución . . . . .	72

<b>9. Sistema Recomendador de Películas</b>	<b>75</b>
9.1. Base de Datos . . . . .	75
9.2. Iniciación . . . . .	76
9.3. Funcionamiento . . . . .	77
9.4. Políticas . . . . .	78
9.4.1. Políticas Simples . . . . .	79
9.4.2. Filtro Colaborativo Basado en Vecindad . . . . .	79
9.4.3. Red Neuronal como Bandido Contextual . . . . .	82
9.5. Resultados . . . . .	86
9.5.1. Resultados Iniciales . . . . .	86
9.5.2. Resultados Finales . . . . .	87
9.5.3. Análisis Detallado . . . . .	90
<b>10. Aportaciones individuales</b>	<b>94</b>
<b>11. Conclusiones</b>	<b>96</b>
<b>12. Conclusions</b>	<b>96</b>
<b>13. Anexo</b>	<b>97</b>
13.1. Depuración de la Evolución de Pesos de los Bots con Probabilidades Individuales de Elección de Brazo de 100 % y 0 % . . . . .	97
13.2. Resultados Sistema Recomendador Películas . . . . .	98

## Notación

En este breve apartado se introducen algunos conceptos clave asociados a su símbolo. Estos símbolos serán utilizados en las secciones posteriores.

$K$	El número de brazos en el problema planteado.
$T$	El horizonte temporal o número de rondas del problema planteado.
$a_t$	Brazo elegido en la ronda $t$ .
$r_t$	Recompensa observada en la ronda $t$ tras elegir el brazo $a_t$ .
$D_a$	Distribución de probabilidad de las recompensas del brazo $a$ .
$IID$	Recompensas Independientes e Idénticamente Distribuidas. La recompensa para cada brazo es $IID$ cuando, en cada ronda, se toma una muestra de forma independiente a partir de la distribución $D_a$ . Es decir, es una distribución de recompensas independiente de la ronda y por tanto estática, pero diferente según el brazo.
$R(T)$	El remordimiento del algoritmo en la ronda $T$ . El remordimiento calcula la diferencia entre el mejor rendimiento posible del algoritmo durante dichas rondas y el rendimiento real.
$E[R(T)]$	El remordimiento esperado del algoritmo en la ronda $T$ .
$a^*$	El brazo óptimo.
$\mu(a)$	La recompensa media para un brazo $a$ .
$O()$	Orden que refleja la eficiencia del algoritmo.
$n_t(a)$	El número de veces que se selecciona el brazo $a$ hasta la ronda $t$ .
$r_t(a)$	El radio de confianza que representa la variabilidad de las recompensas obtenidas por el brazo $a$ en una ronda $t$ .
$UCB(a)$	Cota superior dado un brazo $a$ .
$LCB(a)$	Cota inferior dado un brazo $a$ .
$c_t(a)$	La tabla de costes que refleja el coste para un brazo $a$ en una ronda $t$ .
$\text{coste}(a)$	El coste total de un brazo $a$ , considerando todas las rondas que ha jugado.
$\hat{\mu}_a$	Predicción de recompensa para un brazo $a$ .
$Peso_a$	Importancia de un brazo para el modelo.
$w_a$	Peso o importancia de un brazo para el modelo.
$p_t$	Distribución de probabilidades de elegir un brazo.
$Probabilidad_a$	Posibilidades de elegir un brazo.
$l_a$	(En juegos contextuales) Pérdidas, o remordimiento, de un brazo en una ronda $t$ .
$L_a$	(En juegos contextuales) Pérdidas acumuladas de un brazo en una ronda $t$ .
$\epsilon$	Epsilon que representará la tasa de aprendizaje en Hedge.
$\gamma$	Gamma que representará la tasa de aprendizaje (juegos con contexto).
$x_t$	El contexto dada una ronda $t$ .
$UCB_i$	Una instancia del algoritmo $UCB$ , de los bandidos estocásticos.
$L$	La constante Lipschitz que se utiliza en los bandidos contextuales lipschitzianos.
$\gamma$	En el algoritmo $Exp4$ , esta variable refleja la tasa de exploración.
$r_{acum}(a x)$	La recompensa acumulada de un brazo dado un contexto $x$ .
$\pi$	Política, que se usa como experto, en el $Exp4$ .
$\Pi$	Conjunto de políticas presentado al $Exp4$ en su versión de bandidos contextuales con clase política.
$\hat{c}_t(e)$	Costes falsos que se calculan en el algoritmo $Exp4$ para trabajar con el algoritmo Hedge.
$\vec{p}_t$	Distribución de probabilidad de elegir una política $\pi$ . Se emplea en los algoritmos Hedge y $Exp4$ .

# 1. Introducción

## Origen de los Bandidos

El problema de los bandidos multi-brazo fue inicialmente estudiado por Thompson en 1933 cuando publicó un artículo que ahora ha pasado a la historia en la revista *Biometrika* [1]. En dicha revista, Thompson se interesó por las pruebas médicas que se realizaban en esa época, porque se consumían medicamentos sin conocer su efectividad. En estas pruebas, Thompson observó que los fármacos no eran buenos inicialmente ni se ajustaban sobre la marcha. Llegó a la conclusión de que se realizaban pruebas médicas “a ciegas”, surgiendo la necesidad de adaptación durante el ensayo para que el efecto del tratamiento fuera óptimo.

El nombre de los bandidos multi-brazo proviene de 1950s, cuando Frederick Mosteller y Robert Bush estudiaron el aprendizaje animal. Durante su estudio, realizaron pruebas a ratas en un laberinto con forma de T, es decir un camino recto que termina en una bifurcación. De esta forma, situaban a los animales ante un problema de decisión: izquierda o derecha. Solo en uno de los dos ramales había comida como recompensa, en el otro no había nada. Ante este escenario, el roedor elegía un camino sin saber qué iba a encontrar.

Con el objetivo de estudiar algo parecido con los humanos, se usaron unas máquinas tragaperras de dos brazos, también conocidas como “bandidos” de dos brazos porque robaban el dinero del jugador que las usaba. Para usar las máquinas tragaperras, en cada tirada o ronda el jugador elige un brazo, la palanca de la izquierda o la de la derecha. Se llama ronda a cada oportunidad que tiene el jugador para elegir un brazo. Al igual que con los roedores, el objetivo del jugador es conseguir la máxima recompensa posible teniendo en cuenta que al principio no sabe qué brazo devuelve las mejores recompensas. Se entiende que tirar de una palanca de la máquina tragaperras será elegir un brazo del bandido multi-brazo.

Los bandidos multi-brazo representan el problema que se da en una situación de aprendizaje similar a la de la máquina tragaperras: explorar una opción que puede parecer inferior a priori, según la experiencia previa del jugador, o continuar explotando la mejor alternativa o brazo. Por ejemplo, un jugador puede estar ante la disyuntiva de elegir entre un brazo con el cual ha conseguido mucho dinero en las rondas anteriores, o probar un brazo nuevo que aún no había usado. Por ello, encontrar el correcto equilibrio entre la exploración-explotación es el corazón de los problemas de los bandidos multi-brazo y la tarea de los algoritmos que se estudia a lo largo de este trabajo.

Estos algoritmos se encuadran dentro del aprendizaje automático en el ámbito del aprendizaje por refuerzo. El aprendizaje por refuerzo es un tipo de aprendizaje que tiene como objetivo maximizar las recompensas que consigue el algoritmo a través de sus acciones. Al principio el algoritmo no sabe qué acciones tomar, debe descubrir con el paso de las rondas qué acciones le proporcionan más recompensas.

La principal característica de los bandidos multi-brazo dentro del aprendizaje por refuerzo es que son un escenario modificado de aprendizaje por refuerzo cuyo aspecto diferencial es que el estado permanece constante. El estado es el entorno donde trabaja el algoritmo.

Para entender las ventajas de este escenario “seguro”, primero se explica el escenario opuesto, es decir una situación de aprendizaje por refuerzo completa donde las acciones del algoritmo modifiquen el estado del juego. Un ejemplo es el ajedrez. En este juego cada movimiento que hace un jugador (o algoritmo, en este caso) cambia el estado del tablero. Si el algoritmo decide mover un peón, esto cambia el estado del tablero y también las posibles acciones o brazos que tanto él como su oponente pueden tomar en el futuro. A su vez, estas decisiones también pueden influir en la estrategia del oponente, lo que a su vez cambia el entorno una vez más. Esta situación presenta varios desafíos:

- **Complejidad:** En el aprendizaje por refuerzo completo, el algoritmo necesita aprender una política completa que en función del estado elija la acción óptima, en lugar de solo aprender la recompensa esperada de cada acción. Esto puede hacer que el proceso de aprendizaje sea mucho más complejo y requiera más tiempo y recursos computacionales.
- **Mayor riesgo:** Dado que el estado del entorno puede cambiar con cada acción que se toma, existe un riesgo inherente de alterar el entorno de manera negativa durante el proceso de exploración. Esto puede dificultar el análisis del dilema de exploración-explotación y puede llevar a resultados indeseables si no se maneja adecuadamente.

- Ineficiencia: En los problemas de aprendizaje por refuerzo completo, la optimización puede ser menos directa y el aprendizaje más lento, ya que el algoritmo necesita aprender a estimar la recompensa de cada acción en cada posible estado. Esto puede requerir un número mucho mayor de interacciones con el entorno, lo que puede ser costoso en términos de tiempo y recursos.

Por tanto, al considerar que el estado no cambia, los bandidos multi-brazo evitan la mayor parte de la complejidad del aprendizaje por refuerzo completo y trabajan cómodamente uno de los conceptos más importantes del aprendizaje por refuerzo: el dilema exploración-explotación [7].

Los algoritmos de aprendizaje automático se pueden clasificar en dos áreas o tipos: aprendizaje offline y aprendizaje en línea. En este caso, los bandidos multi-brazo se encuentran dentro de la segunda categoría porque no se entrenan offline, sino que aprenden conforme van tomando decisiones.

En definitiva, los bandidos multi-brazo se han convertido en un marco simple pero muy útil para algoritmos que toman decisiones bajo incertidumbre a lo largo del tiempo.

## Objetivos y plan de trabajo

El propósito primordial de este trabajo reside en la investigación y comprensión del problema de los bandidos contextuales. Se hace hincapié en los detalles de los algoritmos que solucionan dicho problema tratando de explicar la lógica inherente a estos. Finalmente, este estudio propone la creación de aplicaciones prácticas con el objetivo de ilustrar los beneficios que los bandidos contextuales pueden aportar en una amplia variedad de dominios de aplicación. Por lo tanto, esta investigación tiene un enfoque eminentemente práctico sustentado en principios teóricos.

La estructura empleada en la memoria es la siguiente:

- Las secciones 1 y 2 conforman la introducción. Explican la motivación, los objetivos, el plan de desarrollo y los recursos utilizados. El contenido de la sección 1 está en español y el de la sección 2 en inglés, pero ambos son iguales.
- La sección 3 constituye la introducción de los bandidos multi-brazo, así como detalles a tener en cuenta.
- Las secciones 4 y 5 introducen dos tipos de bandidos esenciales para después trabajar con los bandidos contextuales.
- La sección 6 desarrolla al completo la teoría fundamental del trabajo, los bandidos contextuales, y expone las dos primeras aplicaciones prácticas.
- La sección 7 estudia un caso concreto donde se compara el rendimiento de algoritmos contextuales y no contextuales en un juego contextual.
- La sección 8 expone un aplicativo de los bandidos contextuales con una clase política, explicados en 6, dentro del mercado financiero.
- La sección 9 presenta un sistema recomendador como una aplicación de los bandidos contextuales con una clase política.
- Las secciones 11 y 12 forman la conclusión del trabajo. Ambas secciones tienen el mismo contenido, pero la sección 11 está en español y la sección 12 está en inglés.

## Recursos y Documentación

### Referencias bibliográficas principales

Este trabajo se ha basado fundamentalmente en el libro de Slivkins [2], donde se pueden encontrar las demostraciones de todos los resultados teóricos que presentamos. También se ha consultado cuando ha sido necesario los textos de Lattimore y Bubeck [3, 4].

## **Repository de implementaciones**

Las implementaciones de software realizadas durante el desarrollo de este trabajo quedan reflejadas en los repositorios personales de los autores [26, 27]. Los lenguajes de programación empleados son Python, principalmente, y C++. Python un lenguaje de programación de alto nivel muy versátil y fácil de usar. Debido a su menor complejidad de código y la presencia de bibliotecas avanzadas en computación científica y aprendizaje automático, Python es especialmente relevante en el campo de la inteligencia artificial, incluyendo el aprendizaje por refuerzo. Se han utilizado bibliotecas imprescindibles en el aprendizaje automático como numpy, pandas, scipy y tensorflow. Dichas bibliotecas permiten crear algoritmos y tratar adecuadamente los datos. Y también se han aprovechado otras bibliotecas para visualizar datos como matplotlib y seaborn.

## 2. Introduction

### Bandits origin

The multi-armed bandit problem was initially studied by Thompson in 1933 when he published a paper that has now gone down in history in the journal *Biometrika* [1]. In that journal, Thompson was interested in the medical tests that were being performed at the time, because drugs were being consumed without knowing their effectiveness. In these tests, Thompson observed that the drugs were not good initially and did not adjust as they went along. He concluded that medical trials were conducted “blindly”, with the need for adjustment during the course of the trial for the treatment effect to be optimal.

The name multi-armed bandits comes from the 1950s, when Frederick Mosteller and Robert Bush studied animal learning. During their study, they tested rats in a T-shaped maze, i.e. a straight path ending at a fork. In this way, they placed the animals before a decision problem: left or right. Only on one of the two branches was there food as a reward, on the other there was nothing. Faced with this scenario, the rodent chose a path without knowing what it would find.

In order to study something similar with humans, two-armed slot machines, also known as two-armed bandits because they stole money from the player using them, were used. To use the slot machines, in each spin or round the player chooses an arm, the lever on the left or the lever on the right. We call each opportunity the player has to choose an arm a round. As with the rodents, the player’s objective is to get the maximum possible reward, bearing in mind that at the beginning he does not know which arm returns the best rewards. We will understand that pulling a lever of the slot machine will be choosing an arm of the multi-armed bandit.

Multi-arm bandits represent the problem encountered in a slot machine-like learning situation: exploring an option that is presumed to be inferior, based on the player’s previous experience, or continuing to exploit the best alternative or arm. For example, a player may be faced with the dilemma of choosing between an arm with which he has made a lot of money in previous rounds, or trying a new arm that he has not yet used. Thus, finding the right balance between exploration-exploitation is at the heart of the multi-armed bandit problems and the task of the algorithms we will study throughout this paper.

These algorithms fall under the umbrella of machine learning in the field of reinforcement learning. Reinforcement learning is a type of learning that aims to maximize the rewards that the algorithm achieves through its actions. At first the algorithm does not know which actions to take; it must discover over rounds which actions provide it with the most rewards.

The main feature of multi-armed bandits within reinforcement learning is that they are a modified reinforcement learning scenario whose differential aspect is that the state remains constant. The state is the environment where the algorithm works.

To comprehend the benefits of the aforementioned “safe” scenario, it is crucial to first elaborate on the contrasting scenario. This involves a comprehensive reinforcement learning situation in which the actions executed by the algorithm alter the state of the game. An example is chess. In this game every move a player (or algorithm, in this case) makes changes the state of the board. If the algorithm decides to move a pawn, this changes the state of the board and also the possible actions or arms that both it and its opponent can take in the future. In turn, these decisions can also influence the opponent’s strategy, which in turn changes the environment once again. This situation presents several challenges:

- Complexity: In full reinforcement learning, the algorithm needs to learn a complete policy that based on the state chooses the optimal action, rather than just learning the expected reward of each action. This can make the learning process much more complex and require more time and computational resources.
- Increased risk: Since the state of the environment can change with each action taken, there is an inherent risk of altering the environment in a negative way during the exploration process. This can make the exploration-exploitation dilemma difficult to analyze and can lead to undesirable results if not handled properly.
- Inefficiency: In full reinforcement learning problems, optimization can be less straightforward and learning slower, as the algorithm needs to learn to estimate the reward of each action in each possible state. This

may require a much larger number of interactions with the environment, which can be costly in terms of time and resources.

Thus, by treating the state as unchanging, multi-arm bandits avoid most of the complexity of full reinforcement learning and comfortably work one of the most important concepts of reinforcement learning: the exploration-exploitation dilemma [7].

Machine learning algorithms can be classified into two areas or types: offline learning and online learning. In this case, multi-armed bandits fall into the second category because they are not trained offline, but learn as they make decisions.

In short, multi-armed bandits have become a simple but very useful framework for algorithms that make decisions under uncertainty over time.

## Objectives and roadmap

The primary purpose of this work lies in the investigation and understanding of the contextual bandit problem. Emphasis is placed on the details of the algorithms that solve such a problem by trying to explain the logic inherent in them. Finally, this study proposes the creation of practical applications with the aim of illustrating the benefits that contextual bandits can bring in a wide variety of application domains. Therefore, this research has an eminently practical approach supported by theoretical principles.

The structure used in the memory is as follows:

- Sections 1 and 2 make up the introduction. They explain the motivation, objectives, development plan and resources used. The content of Section 1 is in Spanish and that of Section 2 section in English, but both are the same.
- Section 3 introduces of the multi-armed bandits, as well as details to take into account.
- Sections 4 and 5 introduce two types of bandits essential to work with the contextual bandits.
- Section 6 fully develops the theory of the fundamental theme of the paper, contextual bandits, and exposes the first two practical applications.
- Section 7 studies a concrete case where the performance of contextual and non-contextual algorithms in a contextual game is compared.
- Section 8 presents an application of the contextual bandits with a political class, explained in Section 6, within the financial market.
- Section 9 presents a recommender system as an application of the contextual bandits with a policy class.
- Sections 11 and 12 correspond to the conclusion of the work. Both sections have the same content, but Section 11 is in Spanish and Section 12 is in English.

## Resources and Documentation

### Main bibliographic references

In the realization of this research, the primary source of reference has been the book by Slivkins [2], which contains the proofs for all the theoretical results presented. Additional references were sought from the texts by Lattimore and Bubeck [3, 4] when necessary.

### Repository of implementations

The software implementations made during the development of this work can be accessed in the authors' personal repositories [26, 27]. The programming languages used are Python, mainly, and C++. Python is a very versatile and easy-to-use high-level programming language. Due to its lower code complexity and the presence of advanced libraries in scientific computing and machine learning, Python is especially relevant in the field of

artificial intelligence, including reinforcement learning. Essential libraries in machine learning such as numpy, pandas, scipy and tensorflow have been used. These libraries allow algorithms to be created and data to be properly processed. Other libraries have also been used to visualize data, such as matplotlib and seaborn.

### 3. Problema de los Bandidos Multi-brazo

Partiendo del ejemplo de la máquina tragaperras, dada una situación en la que hay  $K$  brazos y  $T$  rondas, tal que las palancas de la máquina son los brazos y las fichas disponibles para jugar las rondas. El jugador, también conocido como agente, tendrá que decidir en cada ronda  $t \in T$  qué hacer con la ficha que tiene y jugar uno de los brazos  $a \in K$ . Por lo tanto, se encontrará ante el dilema exploración-explotación con muchos brazos.

Primero, el jugador elegirá un brazo y, al finalizar la ronda, obtendrá una recompensa asociada a dicho brazo. Las recompensas vienen determinadas por un entorno como respuesta a las acciones del jugador. En el caso de la máquina tragaperras, la recompensa será el dinero que la máquina devuelva al jugador según su decisión.

En los problemas de bandidos se trata de optimizar la recompensa a largo plazo. Para ello, el jugador aplicará un algoritmo que determina las instrucciones a seguir. Dicho algoritmo aprenderá en base a la información que el agente tenga de las rondas pasadas: los brazos que han sido jugados y las recompensas devueltas por estos brazos.

El algoritmo tratará de maximizar la recompensa acumulada, es decir, la suma de las recompensas de todas las rondas. Para argumentar si el algoritmo está haciendo un buen trabajo se hace uso del remordimiento. Esta medida indica la diferencia entre las máximas recompensas obtenibles y las recompensas realmente obtenidas por el algoritmo. Por tanto, el objetivo de los algoritmos de bandidos es minimizar el remordimiento.

#### 3.1. Características y Diferenciaciones

Una vez definido lo que es un problema de bandidos y los distintos componentes que se encuentran en estos, a continuación se comentan las distintas características de los bandidos y las diferencias que puede haber en función de estos aspectos.

En primer lugar, se distinguen tres tipos de respuestas de realimentación o retroalimentación disponibles para el algoritmo después de cada ronda:

- Retroalimentación de bandido, cuando el algoritmo observa la recompensa del brazo escogido y no del resto de brazos.
- Retroalimentación total, en este caso el algoritmo observa la recompensa de todos los brazos que podían haber sido escogidos.
- Retroalimentación parcial, si se refleja cierta información además de la recompensa del brazo elegido.

Pero debe aclararse que, estrictamente, la retroalimentación asociada a un problema de bandidos es la retroalimentación de bandido [3]. De todas formas, a lo largo de este trabajo se encuentran excepciones en las que estén presentes los otros dos tipos de retroalimentación. Se explicará cada situación en su debido momento.

También se encuentran distintas variaciones en cuanto al modelo de recompensas. Hay cuatro conceptos que deben ser explicados:

- Función de recompensas deterministas: En este caso, la recompensa es fija y conocida para cada brazo en cada ronda. Por ejemplo, si el algoritmo selecciona el brazo 1 en la primera ronda, siempre recibirá una recompensa de 3 ( $R(1) = 3$ ) y no otra.
- Función de recompensas aleatorias: Aquí, las recompensas se describen mediante una distribución de probabilidad. Por ejemplo, al seleccionar el brazo 1, la recompensa podría ser  $R(1) = 3$  con un 15% de probabilidad,  $R(1) = 2$  con un 55% de probabilidad y  $R(1) = 1$  con un 30% de probabilidad.
- Función de recompensas estáticas: La función de recompensas no cambia con el paso de las rondas. Por ejemplo, si se trata de una función de recompensas estática y determinista, el algoritmo siempre obtendrá una recompensa de 2 al seleccionar el brazo 1, independientemente de la ronda (en  $t = 1$ ,  $R(1) = 2$ ; en  $t = 2$ ,  $R(1) = 2$ ).
- Función de recompensas dinámicas: La función de recompensas cambia con el paso de las rondas. Por ejemplo, al seleccionar el brazo 1 en la ronda 1, el algoritmo podría obtener una recompensa de 4 (en  $t = 1$ ,  $R(1) = 4$ ), pero en la ronda 2, la recompensa podría ser 2 (en  $t = 2$ ,  $R(1) = 2$ ).

Las recompensas planteables en un problema de bandidos se clasifican en deterministas o aleatorias, y estáticas o dinámicas. Es decir, pueden darse recompensas deterministas y estáticas, deterministas y dinámicas, aleatorias y estáticas, y aleatorias y dinámicas. Un caso concreto son las recompensas IID (Independientes e Idénticamente Distribuidas). Las recompensas IID se dan cuando la recompensa para cada brazo es obtenida de una distribución de probabilidad que depende del brazo pero no de la ronda  $t$ . Es decir, existe una distribución de probabilidad de recompensas para cada brazo que es estática en todo el horizonte temporal  $T$ . Cualquier variable cuyo comportamiento esté descrito por una distribución de probabilidades es una variable aleatoria. Por tanto las recompensas IID son a la vez estáticas y aleatorias.

Para exemplificar el porqué una variable es aleatoria si existe una distribución de probabilidad que define su comportamiento se recurre a una situación en la que se tiene una bolsa con canicas de colores. Si se introduce la mano en la bolsa para sacar una canica, no se sabe de antemano de qué color será. Aunque se conozcan las probabilidades generales, el resultado específico de cada intento es incierto. Esta incertidumbre es la que hace que la variable (en este caso, el color de la canica) sea aleatoria.

Otra característica diferenciadora es el contexto, que solo está presente en el problema de los bandidos contextuales. El contexto es observado antes de elegir una acción por el algoritmo. Dicho contexto refleja unas propiedades del escenario y permite personalizar acciones. Varios ejemplos serían: en un periódico online la localización del usuario y su demografía además de variables como edad o sexo; respecto a un broker o una página de inversión, el contexto podría ser el estado actual de la economía.

Si los bandidos contextuales tienen en cuenta el contexto, no es suficiente con aprender si un brazo es bueno porque ese mismo brazo puede ser bueno para ciertos contextos pero malo para otros. Entonces, se trata de aprender la mejor política que asigne contextos a brazos. Este es el principal objetivo de este trabajo.

Por otro lado, se pueden tener acciones estructuradas, lo que significa que un algoritmo necesita tomar varias decisiones a la vez. En un sistema recomendador de artículos si se elige una categoría como brazo, luego se debe que un artículo, por lo que se estarán realizando acciones estructuradas.

Por último, cabe destacar la posibilidad de que aparezcan restricciones globales en el planteamiento de algún problema de bandidos. Esto sucede cuando el algoritmo está sujeto a limitaciones globales que afectan a la posibilidad de elección de ciertos brazos en determinadas rondas.

### 3.2. Tipos de Bandidos

En función de los aspectos definidos anteriormente se encuentran distintos tipos de bandidos multi-brazo que, dadas las circunstancias y teniendo en cuenta las especificaciones de los mismos, permiten adaptarse a diversas situaciones. Cada bandido tiene unas propiedades que lo hacen interesante y una serie de algoritmos apropiados.

Los bandidos que han sido estudiados en este TFG son los siguientes: los bandidos estocásticos, los bandidos antagonistas y los bandidos contextuales. Estos bandidos son definidos y explicados en las siguientes secciones de esta memoria, ofreciendo ejemplos o implementaciones para detallar sus características y por qué son interesantes.

Realmente, la motivación de este trabajo son los bandidos contextuales, pero explicar y estudiar los otros tipos de bandidos brinda un trasfondo para desarrollar complejos bandidos contextuales. Es por esto que estos son los primeros puntos en los que se profundiza en la memoria.

### 3.3. Aplicaciones de los Bandidos Contextuales

Los bandidos multi-brazo pueden ser una propuesta útil en problemas de decisión donde hay incertidumbre. Concretamente, los bandidos contextuales ofrecen aplicaciones que se pueden trasladar a la vida real y a problemas bastante significativos. De acuerdo a [5], estos son varios ejemplos de bandidos contextuales con los que se puede comprobar su versatilidad y utilidad:

## **Estudios Clínicos**

Una prueba clínica ayuda a los médicos e investigadores a evaluar cómo de bien funcionan los diferentes tipos de atención médica y si son seguros. Esta aplicación práctica es la que, realmente, dio lugar al nacimiento de los bandidos.

La recopilación de datos para evaluar la eficacia en ensayos con animales durante toda las etapas de una enfermedad puede ser difícil cuando se utilizan procedimientos convencionales de asignación aleatoria de tratamientos, ya que los tratamientos deficientes pueden causar el deterioro de la salud del sujeto. Por ejemplo, la warfarina es uno de los anticoagulantes más ampliamente usados en el mundo pero suministrar una dosis correcta es un reto ya que la dosis apropiada puede variar de forma significativa entre individuos debido a factores clínicos, demográficos y genéticos. Actualmente, se sigue una estrategia en la que los pacientes empiezan con 5 mg/día, que es adecuado para la mayoría de pacientes, y poco a poco se va ajustando la dosis durante las siguientes semanas mediante el seguimiento de los niveles de anticoagulantes. Sin embargo, una dosis inicial incorrecta puede ser muy perjudicial: un accidente cerebrovascular si la dosis es muy baja o una hemorragia interna si es demasiado alta.

Los autores de [28] pretenden diseñar una estrategia de asignación adaptativa para mejorar la eficiencia de la recopilación de datos mediante la asignación de más muestras para explorar tratamientos prometedores. De esta forma, se estaría sacrificando la explotación de los mejores tratamientos considerados hasta ese momento por la exploración de otros nuevos que puedan dar lugar a mejores resultados. Fundamentan esta aplicación como un problema de bandido contextual e introducen un algoritmo práctico para la exploración frente a la explotación en este marco.

Por tanto, resulta interesante el uso de este tipo de algoritmos en virtud de que permiten considerar las características, o el contexto, de cada sujeto que necesite tratamiento y adecuar así dicho tratamiento al propio sujeto. Esto se debe a que una dosis puede ser adecuada para un sujeto pero no para otro.

## **Sistema Recomendador**

Los sistemas recomendadores son frecuentemente usados con el objetivo de predecir las preferencias de los usuarios. Igualmente, también hacen frente al dilema exploración-explotación. Cuando hacen una recomendación pueden explotar su conocimiento sobre las elecciones realizadas con anterioridad donde el usuario tenía interés o, por el contrario, explorar nuevas alternativas.

En este caso, se puede considerar el contexto que rodea al usuario como el momento temporal, la localización, aspectos sociales, edad, género... De esta manera, estos sistemas de recomendación tienen amplias posibilidades ya que pueden ser implementados en periódicos digitales con el sistema de anuncio de noticias o páginas de *streaming* con las series o películas recomendadas.

En ambas situaciones, sería posible utilizar el contexto para intentar identificar los mejores brazos o acciones con el fin de obtener el mayor número de clics posibles. De esta forma, el algoritmo de un bandido contextual permitiría ofrecer un sistema recomendador personalizado, para cada usuario, que trate de optimizar las tareas de explorar y explotar. Después de que el algoritmo elija un brazo, el usuario haría clic o no en el anuncio, serie o película. Esto haría que la recompensa adoptase el valor 1 o 0, respectivamente, en función de si el usuario llegase a hacer clic.

## 4. Bandidos Estocásticos

El objetivo de este apartado es definir uno de los principales tipos de bandidos, así como detallar sus características para entender su esencia. Los estocásticos son el modelo básico de los bandidos multi-brazo. A lo largo de esta sección, se tratarán varios algoritmos disponibles para este tipo de bandidos y se compararán.

Se considera un problema en el que un algoritmo tiene  $K$  posibles brazos a elegir en cada una de las  $T$  rondas. Tras la elección de un brazo, el algoritmo obtiene una recompensa correspondiente al brazo elegido. El protocolo a seguir es el siguiente, donde  $[T] = \{1, 2, \dots, T\}$ :

### Problema Bandidos Estocásticos

Habiendo  $K$  brazos y  $T$  rondas,

En cada ronda  $t \in [T]$ :

1. El algoritmo elige un brazo  $a_t$ .
2. El algoritmo observa la recompensa  $r_t \in [0, 1]$  para el brazo elegido.

El escenario estocástico se genera bajo las siguientes tres suposiciones:

- El algoritmo únicamente observa la recompensa de la acción seleccionada. Esto se conoce como retroalimentación de bandido. Por tanto, el algoritmo no conoce las recompensas asociadas al resto de acciones que podría haber elegido.
- Las recompensas están acotadas al intervalo  $[0, 1]$ .
- La recompensa para cada brazo es IID. Para cada brazo  $a$  hay una distribución  $D_a$  llamada distribución de recompensas que es inicialmente desconocida por el algoritmo.

La motivación que hay detrás de acotar las recompensas en un intervalo entre 0 y 1 es permitir el cálculo del remordimiento en función del número de rondas  $T$  del problema. Esto será demostrado más adelante en la explicación del remordimiento del algoritmo Exploración-Primero, que es el primero de todos los que se comentan en este trabajo, con el ánimo de que quede claro para el resto de veces que aparezca la variable  $T$  en la fórmula de un remordimiento. Por este motivo, si las recompensas del problema vienen acotadas de la forma  $[m, n]$  tal que  $m < n$ , se recomienda realizar una traducción o reescala de las recompensas al intervalo  $[0, 1]$ .

Uno de los métodos más comunes para trasladar y reescalar recompensas al intervalo  $[0, 1]$  es la transformación lineal. La fórmula correspondiente sería:  $r' = \frac{r-a}{b-a}$ , donde  $r$  es la recompensa original y  $r'$  es la recompensa transformada en el intervalo  $[0, 1]$ .

No es necesario que las recompensas sigan una distribución específica o que cumplan alguna otra condición. Sin embargo, es muy importante tener en cuenta que la transformación lineal asume que la relación entre las recompensas se mantiene constante a lo largo del intervalo. Si este no fuera el caso, sería necesario una transformación diferente que se ajuste mejor a las características del problema.

Por otro lado, se define el vector de recompensas medias como  $\mu \in [0, 1]^K$ , donde  $\mu(a) = E[D_a]$  es la recompensa media del brazo  $a \in K$ . Se usa  $a^* := \arg \max_{a \in A} \mu(a)$  para definir el brazo óptimo, es decir, el que devuelve la recompensa media más alta.

Una forma aproximada de calcular el *remordimiento* es mediante la comparación de la suma de las recompensas medias de los brazos ya elegidos, respecto al *benchmark* establecido por escoger en todas las  $T$  rondas el brazo óptimo  $a^*$ . Formalmente, se define el remordimiento:

$$R(T) = \mu(a^*)T - \sum_{t=1}^T \mu(a_t)$$

Cabe destacar que  $a_t$ , el brazo tomado en cada ronda  $t$ , es una variable aleatoria si su elección depende de la aleatoriedad en las recompensas o la estrategia de selección de brazo del algoritmo. En consecuencia, mientras las recompensas sean aleatorias o el brazo sea escogido de forma aleatoria, el remordimiento  $R(T)$  también

es una variable aleatoria. Por tanto, cuando las recompensas sean IID, tal y como sucede en el escenario estocástico, el remordimiento es una variable aleatoria y se habla en términos de remordimiento esperado  $E[R(T)]$ .

A continuación, se detallan algunos de los algoritmos disponibles, empezando con los más simples para finalmente ofrecer soluciones más elaboradas y complejas.

## 4.1. Algoritmos Simples

### 4.1.1. Algoritmo Exploración-Primero

El primer algoritmo que se propone parte de la siguiente idea: explorar los brazos uniformemente durante  $N$  rondas y elegir el mejor brazo el resto de las rondas. Este algoritmo se conoce como *Exploración-Primero*.

---

#### Algoritmo 4.1 Exploración-Primero ( $N$ )

---

- 1: Fase de Exploración: Se prueba cada brazo  $N$  veces.
  - 2: Se selecciona el brazo  $a$  con la mejor media de recompensas.
  - 3: Fase de Explotación: El brazo  $a$  es utilizado el resto de rondas.
- 

**Teorema 4.1.** *El algoritmo Exploración-Primero consigue un remordimiento esperado  $E[R(T)] \leq T^{2/3} \times \mathcal{O}(K \log T)^{1/3}$  cuando  $N = (\frac{T}{K})^{\frac{2}{3}} \cdot \mathcal{O}(\log T)^{\frac{1}{3}}$ . Para diferentes valores de  $N$  no se asegura dicho remordimiento. Nótese que el valor de  $T$  es mayor que  $K$ , ya que es necesario explorar cada brazo al menos una vez.*

Después de presentar cada algoritmo se incluye un teorema demostrado en [2] que enuncia el remordimiento conseguido por dicho algoritmo y permite evaluar su rendimiento. Concretamente, cuanto menor es su remordimiento mejor es su rendimiento. Las fórmulas de los remordimientos de este trabajo establecen cotas superiores del “peor” rendimiento del algoritmo en cada problema y generalmente utilizan las variables  $T$  horizonte temporal y  $K$  brazos.

Con el objetivo de comprender la presencia de la variable  $T$  en la fórmula del remordimiento, se plantea un sencillo ejemplo que requiere que las recompensas estén previamente acotadas en el intervalo  $[0, 1]$ .

Existen dos cajas, una roja (caja A) y otra azul (caja B). Cada día, durante 10 días ( $T = 10$ ), se puede elegir una de las dos cajas para abrirla y recibir una recompensa. La caja A siempre tiene una recompensa de 1, mientras que la caja B siempre tiene una recompensa de 0. Si se elige siempre la caja A durante los 10 días, se recibe una recompensa total de 10 ( $1 * 10$ ). Por otro lado, si se elige siempre la caja B, la recompensa total será de 0 ( $0 * 10$ ). En el caso peor, eligiendo siempre la caja B, el límite superior del remordimiento viene dado por la variable  $T$  (10 en este ejemplo), ya que la diferencia entre la máxima recompensa (10) y la mínima recompensa (0) es igual a  $T$ . Es por ello que, formalmente, se define el remordimiento asociado al caso peor como  $R(T) = \mathcal{O}(T)$ .

Es importante entender que, en el modelo de bandidos multi-brazo, los casos cercanos al caso peor se interpretan como situaciones en las que el algoritmo no es capaz de aprender a obtener recompensas mayores con el paso de las rondas. Sin embargo, los algoritmos que se plantean a lo largo de este trabajo sí consiguen aprender a tomar las mejores decisiones con más frecuencia. Debido a este hecho objetivo, se podrá observar que los límites superiores de los remordimientos que ofrecen son significativamente menores que el límite superior en el caso peor ( $\mathcal{O}(T)$ ).

En la figura 1 se explica el aprendizaje del algoritmo Exploración-Primero de una forma gráfica. La función de color azul representa el remordimiento asociado al peor caso de todos en el que el algoritmo escoge siempre el peor brazo. Este caso específico simboliza la inexistencia de aprendizaje. Por otra parte, la función de color rojo representa el límite superior del remordimiento esperado asociado al algoritmo Exploración-Primero. Por ende, las líneas verticales representan el remordimiento que el algoritmo ha sido capaz de evitar o el nivel de aprendizaje del algoritmo tras 100 y 200 rondas, respectivamente.

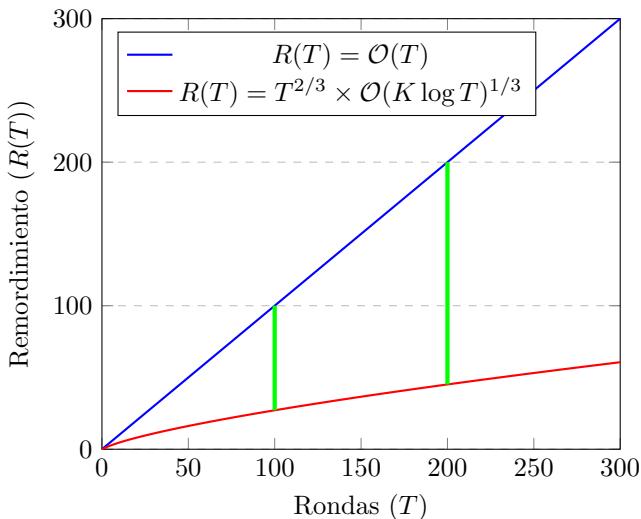


Figura 1: Gráfica de Remordimiento.

El algoritmo Exploración-Primero funciona en dos fases. Primero, explora cada brazo durante un número  $N$  predefinido de rondas. Despues, selecciona el brazo con la mejor recompensa promedio, observada durante la fase de exploración, hasta el final del horizonte temporal.

El problema de este algoritmo es que la exploración se realiza por completo al comienzo del proceso y no se distribuye a lo largo del tiempo. Esto significa que, una vez que se ha completado la fase de exploración, el algoritmo deja de explorar y se centra únicamente en la explotación del brazo que obtuvo el mejor rendimiento durante la fase de exploración. En muchos casos, sería más efectivo distribuir la exploración de manera más uniforme a lo largo del tiempo en lugar de realizarla toda al comienzo. Al distribuir la exploración de manera más uniforme, el algoritmo tendría más oportunidades de aprender sobre los brazos y actualizar sus estimaciones de recompensas medias a medida que avanza el tiempo. Esto podría ayudar a evitar que el algoritmo se atasque en una decisión subóptima basada en información inicial limitada y a aumentar el nivel de aprendizaje del algoritmo.

Por este motivo nace otro algoritmo simple: *Épsilon-Avaricioso*.

#### 4.1.2. Algoritmo Épsilon-Avaricioso

---

**Algoritmo 4.2** Épsilon-Avaricioso ( $\varepsilon_1, \varepsilon_2, \dots$ ).

---

```

1: for cada ronda  $t = 1, 2, \dots$  do
2:   Tirar una moneda con unas probabilidades de exploración  $\varepsilon_t$ 
3:   if exploración then
4:     explorar: elegir un brazo  $a$  uniformemente
5:   else
6:     explotar: elegir el brazo con la recompensa más alta hasta el momento
7:   end if
8: end for
```

---

**Teorema 4.2.** El algoritmo Épsilon-Avaricioso con probabilidades de exploración  $\varepsilon_t = t^{-1/3} \cdot (K \log t)^{1/3}$  alcanza un límite superior de remordimiento  $E[R(t)] \leq t^{2/3} \cdot \mathcal{O}(K \log t)^{1/3}$  para cada ronda  $t$ .

Se conoce como “avaricioso” ya que se fundamenta en elegir la mejor opción a corto plazo. Sin embargo, es un algoritmo que presenta una exploración de brazos distribuida uniformemente a lo largo de las rondas dado que en cualquier ronda, con  $\varepsilon_t > 0$ , hay probabilidades de explorar. De esta forma, se consigue solucionar el problema que presenta el algoritmo Exploración-Primero y, a la vez, como se consigue demostrar en [2], obtener un remordimiento idéntico cuando el número de rondas en las que se ha explorado es del orden de  $t^{2/3}$  con un horizonte temporal  $T = t$ . Esta cantidad de rondas exploradas se consigue, tal y como sugiere el teorema

anterior, con una probabilidad de éxito  $\varepsilon_t \sim t^{-1/3}$ . La notación “~” sugiere que una expresión es similar a otra.

En cada ronda  $t$ , el algoritmo explora con una probabilidad de  $\varepsilon_t$ . Es decir, la probabilidad de explorar  $\varepsilon_t$  se ajusta de acuerdo con el número de rondas que se han completado. La idea es que al principio se debe explorar en mayor medida para descubrir los brazos que dan las mejores recompensas y, a medida que se adquiere información adicional, se pueden explotar los brazos que ya se sabe que son buenos. Por lo tanto, la probabilidad de explorar se reduce a medida que aumenta el número de rondas.

Antes de continuar con la explicación de los algoritmos avanzados, se señala la relevancia de este algoritmo en una implementación que se llevará a cabo en la sección vertebral de este trabajo.

## 4.2. Algoritmos Avanzados: Exploración Adaptativa

Los algoritmos avanzados se han desarrollado a raíz de que los anteriores algoritmos, Épsilon-Avaricioso y Exploración-Primero, tienen el defecto de no planificar la exploración en función del historial de las recompensas observadas. Con el objetivo de ser más eficientes en términos de aprendizaje, es decir, necesitar menos rondas para adquirir el mismo nivel de aprendizaje, los algoritmos avanzados incorporan otro modelo de planificación conocido como *exploración adaptativa* que distribuye o adapta la exploración según las recompensas observadas.

Para comenzar, es necesario definir el concepto y significado de los *límites de confianza superior e inferior*, denominados como  $UCB_t(a)$  y  $LCB_t(a)$  respectivamente para un brazo  $a$  y una ronda  $t$ . Estos conceptos son la base en la que se cimientan los algoritmos avanzados que se comentan en este apartado.

El intervalo definido por  $[LCB_t(a), UCB_t(a)]$  es conocido como el *intervalo de confianza*. Matemáticamente, se definen:

$$UCB_t(a) = \mu_t(a) + r_t(a) \text{ y } LCB_t(a) = \mu_t(a) - r_t(a)$$

De esta forma, los *límites de confianza* se obtienen a partir de la recompensa media  $\mu_t(a)$  considerada para el brazo  $a$  en la ronda  $t$  y a partir del radio de confianza  $r_t(a)$ , que actúa como la variabilidad de la media de recompensas y se calcula en función de  $n_t(a)$ , donde  $n_t(a)$  representa el número de veces que se ha seleccionado el brazo  $a$  hasta la ronda  $t$ . Matemáticamente, el radio de confianza se calcula como sigue:

$$r_t(a) = \sqrt{2 \log T / n_t(a)}$$

El cálculo del radio sugiere que, como la variable  $n_t(a)$  se encuentra en el denominador de la expresión, cuanto menor sea el número de veces que se ha seleccionado el brazo  $a$  hasta la ronda  $t$ , mayor será el valor del radio de confianza para dicho brazo y dicha ronda. En resumen, al vector de recompensas medias se le suma o resta el radio de confianza para obtener el límite de confianza superior o inferior, respectivamente.

A partir del intervalo de confianza, que un brazo  $a$  sea elegido en la ronda  $t$  puede ser por dos razones: porque ha devuelto una recompensa media  $\mu_t(a)$  alta y/o porque el radio de confianza  $r_t(a)$  es alto debido a que el brazo  $a$  no se ha explorado mucho en comparación con el resto de brazos. Ambos motivos son alicientes para que el brazo sea elegido y, por tanto, la combinación empleada de  $\mu_t(a)$  y  $r_t(a)$  logra un equilibrio coherente entre la exploración y la explotación.

### 4.2.1. Algoritmo Eliminación Sucesiva

Un algoritmo que utiliza esta idea de límites superiores e inferiores a partir del vector de recompensas medias y el radio de confianza es el algoritmo Eliminación Sucesiva.

---

#### Algoritmo 4.3 Algoritmo Eliminación Sucesiva.

---

- 1: Inicialmente todos los brazos están “activos”;
  - 2: **for** cada fase  $f = 1, 2, \dots$  **do**
  - 3:     se prueban todos los brazos activos (por lo que cada fase puede contener múltiples rondas, tantas como brazos activos resten);
  - 4:     se desactiva un brazo  $a$  si existe brazo  $a'$  con  $UCB_t(a) < LCB_t(a')$ ;
  - 5: **end for**
-

**Teorema 4.3.** El algoritmo Eliminación Sucesiva tiene un remordimiento esperado  $E[R(T)] \leq \mathcal{O}(Kt \log T)^{1/2}$  para cada ronda  $t \leq T$ . Por tanto, en la última ronda del horizonte temporal  $T$ , el remordimiento esperado es  $E[R(T)] \leq \mathcal{O}(KT \log T)^{1/2}$ .

Para explicar cómo el algoritmo va descartando la explotación de los peores brazos sucesivamente, se incluye un corto ejemplo que supone 3 brazos y un escenario tras la fase 1 en el que se obtuvieron los siguientes UCB y LCB:

- Brazo A:  $UCB(A) = 0,5$ ,  $LCB(A) = 0,3$ .
- Brazo B:  $UCB(B) = 0,9$ ,  $LCB(B) = 0,6$ .
- Brazo C:  $UCB(C) = 0,8$ ,  $LCB(C) = 0,7$ .

En este caso,  $UCB(A) < LCB(B)$  y  $UCB(A) < LCB(C)$ , por lo que se desactiva el brazo A. Ahora, solo quedan los brazos B y C activos. El algoritmo continua con la siguiente fase y repite los pasos 2-4. Si en algún momento  $UCB(B) < LCB(C)$  o  $UCB(C) < LCB(B)$ , el algoritmo desactiva el brazo correspondiente y se queda con el brazo restante como el que tiene la mejor recompensa promedio estimada.

A continuación, en la figura 2, se analiza el remordimiento del algoritmo Eliminación Sucesiva en comparación con el de los algoritmos simples:

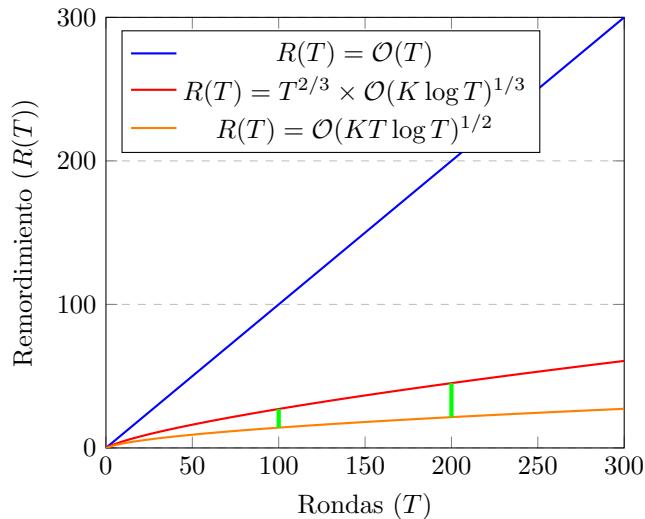


Figura 2: Gráfica de Remordimiento.

En términos de aprendizaje, la diferencia entre este primer algoritmo avanzado que se ha analizado y los algoritmos simples ya comentados viene representada por la longitud de las líneas verdes. En conclusión, se observa una ligera mejora o disminución del límite superior del remordimiento esperado al usar los conceptos UCB y LCB para solucionar un problema de bandidos estocásticos.

#### 4.2.2. Algoritmo UCB1

Otro enfoque para la exploración adaptativa en función del historial de las recompensas observadas es conocido como optimismo bajo incertidumbre. En esta variante, se asume que cada brazo es lo mejor que podría ser dadas las observaciones hasta el momento, y se elige el mejor brazo basándose en estas estimaciones optimistas. Esto da lugar al algoritmo conocido como UCB1.

**Teorema 4.4.** El algoritmo UCB1 se caracteriza por tener el mismo remordimiento que tiene el algoritmo Eliminación Sucesiva. Por lo tanto, su rendimiento también queda representado por la función de color naranja de la figura 2.

---

**Algoritmo 4.4 Algoritmo UCB1.**

---

- 1: Se prueba cada brazo una vez.
  - 2: En cada ronda  $t$ , se elige el  $\arg \max_{a \in [K]} \text{UCB}_t(a)$ , donde  $\text{UCB}_t(a) = \mu_t(a) + r_t(a)$ .
- 

Este algoritmo también podrá ser denominado como UCB a lo largo de este trabajo. Por otro lado, tal y como se ha mencionado anteriormente, un brazo  $a$  es elegido por haber obtenido una media de recompensas alta o por un *radio de confianza* alto. En otras palabras, la elección del brazo se basa bien en su explotación (dada su media de recompensas) o en su exploración (dada su radio de confianza). De esta manera, se consigue un punto óptimo entre la exploración y la explotación. Sin embargo, al sumar el *radio de confianza* se está aplicando un enfoque muy optimista y se supone que un brazo obtendrá la mejor recompensa que le es posible, y esto no siempre es cierto. Por otra parte, al partir de la base de que las acciones son prometedoras, UCB1 garantiza que no se descarten prematuramente acciones que podrían ser prometedoras en un futuro. Esto fomenta la búsqueda de nuevas soluciones y también permite una correcta adaptación al entorno.

Este algoritmo es uno de los más representativos dentro de los estocásticos, ya que realiza exploración adaptativa y muestra intuitivamente cómo un bandido multi-brazo trata de encontrar un equilibrio entre la exploración y la explotación. Además, su escalabilidad a distintas aplicaciones es otro factor importante que juega a favor de este algoritmo. Debido a su idiosincrasia, UCB1 es ampliamente usado y es incluso aplicable a los *bandidos contextuales*; es por ello que en las secciones posteriores se hará uso del algoritmo UCB1 en problemas más complicados que consideren el contexto.

## 5. Bandidos Antagonistas

Esta subsección está orientada a definir el problema de los bandidos antagonistas con el objetivo de identificar este escenario en las aplicaciones reales que se expondrán como trabajo principal de esta investigación. Será de gran utilidad conocer las especificaciones de dos algoritmos, Hedge y Exp4, ya que serán puestos en práctica en las aplicaciones que culminan este trabajo.

En los bandidos estocásticos, la función de recompensas es estática con respecto al paso de las rondas. En cambio, en el problema de bandidos antagonistas la función de recompensas es dinámica, en el sentido de que cambia con el paso de las rondas bajo la influencia de un adversario o antagonista.

Para ilustrar este nuevo escenario se plantea un problema en el que, dada una red de ciudades y carreteras, en cada ronda el objetivo es encontrar el camino, entendido como un brazo del bandido, más eficiente para un envío de paquetes con una ciudad origen y otra destino. Además, se supone que la longitud de todas las carreteras es igual para hacer más sencilla la explicación.

Se contempla una función de recompensas determinista que, en función de las ciudades de origen y destino, junto con la elección de ruta del algoritmo, asigna una recompensa de ‘1’ al camino más eficiente y ‘0’ a cualquier otra ruta. De todas formas, se podría hacer uso de una función de recompensas aleatorias sin que afectase a la comprensión del escenario de los bandidos antagonistas, tal y como se comentará más adelante.

Se toma la siguiente red para realizar esta explicación:

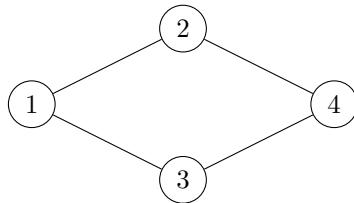


Figura 3: Red de ciudades y carreteras.

Supóngase que se pide al algoritmo encontrar el camino más rápido para ir desde la ciudad origen 1 a la ciudad destino 4. A partir de la figura 3 se distinguen dos únicos caminos posibles: el trazado 1-2-4, que se designará como “camino 1”, y el trazado 1-3-4, denominado “camino 2”.

La clave del problema, sin tener en cuenta las longitudes, se encuentra en la existencia de variables que el algoritmo no es capaz de controlar. La presencia de estas variables representa la presencia de un antagonista en el problema de bandidos. En este caso, la congestión de las carreteras es el antagonista. Esta variable desconocida para el algoritmo, que se supone que toma valores distintos en cada ronda, modifica el comportamiento de las recompensas de cada ronda sin que el algoritmo sea consciente. En consecuencia, ya no habrá un camino más eficiente que otro siempre, sino que la eficiencia de los caminos cambia en cada ronda según las congestiones. Por tanto, el algoritmo, cuyo objetivo es aprender el camino más eficiente dado un trayecto, observará en algún momento que el camino que ha considerado como más eficiente hasta ese momento dejará de serlo, y deberá considerar otro camino como el más eficiente. Como apunte adicional, la congestión es una medida que relaciona la ocupación de la carretera con su capacidad, pero esto ahora no es relevante porque se toma la congestión como un dato directo que no tiene que ser calculado.

Para comprobar que la congestión hace variar las recompensas que devuelven los caminos entre rondas, se plantea que en la ronda 1 la congestión de las carreteras 1-2, 1-3, 2-4 y 3-4 es 8, 3, 4 y 1 respectivamente, y que en la ronda 2 la congestión es 3, 6, 4 y 7, también respectivamente. Esta sucesión de congestiones se repite desde la ronda 3 en bucle.

De esta forma, la congestión total experimentada en el “camino” 1 tiene un valor de 12 en las rondas impares y de 7 en las rondas pares. En cuanto al “camino 2”, su congestión es de 4 en las rondas impares y de 13 en las pares. Por tanto, en las rondas pares las recompensas serán de ‘1’ para el camino 1 y de ‘0’ para el camino 2, mientras que en las rondas impares las recompensas serán de ‘1’ para el camino 2 y de ‘0’ para el camino 1. Es decir, como se ha querido demostrar desde el principio, este es un escenario en el que el problema planteado es de bandidos antagonistas porque las recompensas pueden variar entre rondas debido a la presencia de un

antagonista que el algoritmo no puede controlar.

La explicación más técnica del problema argumenta que las recompensas asociadas a cada camino no pueden modelizarse mediante una distribución estacionaria, por lo que sería necesario un conjunto más sofisticado de supuestos estadísticos. En general, puede ser difícil o imposible determinar los supuestos estadísticos correctos para un dominio dado, y algunos dominios pueden mostrar un alto grado de incertidumbre. Algunos ámbitos pueden presentar dependencias hasta el punto de que no resulte apropiado aplicar tales supuestos. De esta forma, en problemas en los que la recompensa se genere de una manera más compleja o pseudo-aleatoria y no se sepa claramente qué valor va a tomar, habrá que recurrir a considerar que la recompensa depende de factores ajenos desconocidos para el algoritmo, que representarán el papel de antagonista estudiado en esta sección [6].

A la hora de explicar los bandidos antagonistas, muchos autores como Aleksandrs Slivkins en [2] hacen hincapié en la idea de que en estos bandidos las recompensas varían como si estuviesen declaradas por un adversario con la intención maliciosa de fastidiar al algoritmo. Es común entre los autores que explican esta idea que comparan este tipo de problema con una casa de apuestas en la que las recompensas han sido definidas por un adversario, la casa de apuestas, con intención de que el jugador obtenga la mínima recompensa.

Esto puede llevar a confusiones al lector debido a una incorrecta generalización. Es decir, detrás de las recompensas variantes de los bandidos adversarios no tiene por qué haber una mala intencionalidad. Es cierto que es una traba para el algoritmo, pero puede no ser malintencionada. De hecho, en el ejemplo considerado anteriormente para explicar los bandidos antagonistas no existe ninguna intención de fastidiar detrás de la variabilidad de las recompensas, sino que está presente la congestión como una variable desconocida para el algoritmo que altera la función de recompensas, pero en ningún caso los valores de la congestión de las carreteras están sugestionados por algún adversario con mala intención.

Una vez explicado el concepto de recompensas dinámicas que difiere de las recompensas estáticas IID de los bandidos estocásticos, es importante saber que existe otra diferencia más entre las recompensas estocásticas y las antagonistas. Esta diferencia atiende a que las recompensas estocásticas IID, por norma general, son aleatorias debido a que están siempre descritas por una distribución de probabilidades, mientras que en las recompensas en un escenario antagonista no hay ninguna especificación, es decir pueden ser deterministas o aleatorias. Cabe destacar que las recompensas estocásticas pueden ser deterministas si su distribución es trivial y solo permite un valor. Las recompensas deterministas, por su propia definición, están descritas de forma determinada, sin aleatoriedad. Esto que se acaba de comentar tiene un efecto directo en el análisis del remordimiento de los problemas antagonistas porque obliga a distinguir entre si la función de recompensas es determinista o aleatoria.

Antes de explicar el remordimiento, este trabajo considera, al igual que los de otros autores, que la mejor forma de trabajar con bandidos antagonistas es con costes en lugar de con recompensas. Es decir, se entiende que lo que el algoritmo obtiene por la elección de un brazo es un coste. De esta forma, el objetivo del algoritmo es reducir el coste obtenido y, por tanto, se afronta la explicación del remordimiento desde el punto de vista de los costes obtenidos.

De todas formas, si el problema planteado se basa en la obtención de recompensas se puede realizar una conversión sencilla para que el problema se base en la obtención de costes en su lugar. Esta conversión se cimienta en la idea de que las versiones de pérdida y ganancia son simétricas, en el sentido de que se puede trasladar el análisis de una a otra mediante la equivalencia:

$$l_{i,t} = 1 - g_{i,t},$$

donde  $l_{i,t}$  es el coste asociado al brazo  $i$  en la ronda  $t$  y  $g_{i,t}$  es la recompensa asociada al brazo  $i$  en la ronda  $t$ . Por ejemplo, en el planteamiento de la red de ciudades y carreteras planteado anteriormente, si se asume que las recompensas están acotadas en el intervalo  $[0, 1]$  y la recompensa de elegir el “camino 1” en la ronda 1 es 0,3, el coste equivalente a dicha recompensa es 0,7.

El remordimiento asociado a un problema de bandidos antagonistas es un aspecto complejo de entender. Para un entendimiento riguroso y teniendo en cuenta que, a raíz de la existencia de recompensas dinámicas, no hay un solo brazo óptimo como en el caso de los bandidos estocásticos, sino más bien una secuencia óptima de brazos, el concepto de remordimiento controlaría la diferencia entre el coste que realmente ha obtenido y el coste que se habría acumulado si el algoritmo hubiera seguido en todas las rondas la estrategia marcada por esta secuencia óptima de brazos [8]. En este trabajo, al igual que en los del resto de autores, no se va a llegar tan lejos en el análisis y se va a realizar igual que en el entorno estocástico. Es decir, se va a considerar que sí existe un único brazo óptimo a lo largo de todas las rondas. Por tanto, el remordimiento expresa la diferencia

entre el coste obtenido por el algoritmo y el coste que podría haberse obtenido si el algoritmo hubiera jugado en todas las rondas el brazo óptimo. Se considera una “tabla de costes”  $c_t(a)$  que indica el coste para un brazo  $a \in K$  en la ronda  $t \in T$ :

- Si la función de costes es determinista, el coste total de cada brazo  $a$  es  $\text{coste}(a) = \sum_{t=1}^T c_t(a)$ . Intuitivamente, el brazo óptimo  $a^*$  es el brazo con el coste total más bajo. Formalmente,  $a^* := \arg \min_{a \in [K]} \text{coste}(a)$ . Por estas razones, el remordimiento con costes deterministas se expresa como:

$$R(T) = \sum_{t=1}^T c_t(a_t) - \min_{a \in [K]} \text{coste}(a)$$

- Si los costes son aleatorios, el brazo óptimo  $a^*$  es el brazo con el coste esperado total más bajo. Formalmente,  $a^* := \arg \min_{a \in [K]} E[\text{coste}(a)]$ , conociendo la distribución de probabilidades que describe el comportamiento de los costes de cada brazo. Por tanto, el remordimiento en este caso queda definido por:

$$R(T) = \sum_{t=1}^T c_t(a_t) - \min_{a \in [K]} E[\text{coste}(a)]$$

Tras haber expuesto estos aspectos fundamentales, se trata un punto clave: las variedades de antagonistas que pueden darse, entendiendo como antagonista aquellas variables desconocidas para el algoritmo que provocan la variación de la función de recompensas. Dependiendo de los conocimientos del adversario, se consideran dos tipos diferentes:

- Si el antagonista no conoce las elecciones de brazo del algoritmo se denomina indiferente. Por tanto, la variación que ejerce sobre las recompensas no depende del comportamiento pasado del algoritmo. Por ejemplo, el escenario planteado al inicio de esta subsección se caracteriza por la presencia de un antagonista indiferente porque el valor de las congestiones es independiente de las elecciones previas del algoritmo.
- En cambio, el antagonista adaptable o flexible, como su propio nombre indica, es un antagonista que es conocedor de las elecciones pasadas del algoritmo y puede variar tras cada ronda la configuración de la función de recompensas según los brazos escogidos por el algoritmo en las rondas anteriores. Por ejemplo, en un casino amañado, el propietario puede observar la forma en que apuesta un jugador para diseñar secuencias de ganancias maliciosas que vayan a contracorriente de las estrategias del jugador [4].

Independientemente de los ejemplos anteriores, la malintencionalidad no es un requisito obligatorio ni en el antagonista indiferente ni en el antagonista adaptable. Debe quedar claro que la presencia de esta característica no afecta al análisis del problema.

## 5.1. Primer Intento de Solución

Dicho esto, un primer enfoque para resolver un problema de bandidos antagonistas podría ser el empleo de un algoritmo determinista básico. A continuación, se muestra un ejemplo:

---

### Algoritmo 5.5 Algoritmo Seguir al Líder

---

- 1: Inicializar: Juegue cada brazo una vez para obtener una estimación inicial de los costes.
  - 2: **for** cada ronda  $t = 1, 2, \dots$  **do**
  - 3:     Seleccionar el brazo con el coste acumulado más bajo hasta el momento:  $a_t = \arg \min_a \sum_{s=1}^{t-1} c_s(a)$ .
  - 4:     Jugar el brazo seleccionado  $a_t$  y observar el coste incurrido.
  - 5:     Actualizar el coste acumulado del brazo  $a_t$ .
  - 6: **end for**
- 

Este algoritmo siempre selecciona el brazo con el coste acumulado más bajo en cada ronda. Sin embargo, este algoritmo no es eficaz para este problema. Esto se debe a que, como expone [2], incluso un antagonista indiferente puede fastidiar el algoritmo si conoce su estrategia antes de empezar, aunque no sea sabedor de sus elecciones durante las rondas. Podría aprovechar este conocimiento para manipular los costes de los brazos y

forzar al algoritmo a tomar decisiones subóptimas.

Para demostrarlo se plantea resolver un problema básico en el que solo hay dos brazos A y B con el algoritmo Seguir al Líder. Además, se supone que el antagonista presente en este problema es consciente de la estrategia que el algoritmo va a llevar a cabo. Al conocer la estrategia, el antagonista podría diseñar la siguiente función de costes:

En la primera ronda, el antagonista asigna un coste de 0.5 a A y un coste de 1 a B (coste acumulado de A < coste acumulado de B). El algoritmo, como no tiene referencias de los costes acumulados de cada brazo, elige de forma aleatoria el brazo A, por ejemplo. En este caso, el algoritmo ha escogido el brazo más eficiente.

En la segunda ronda, el algoritmo elige A porque tiene el coste acumulado más bajo. Sin embargo, el antagonista sabe que el algoritmo va a elegir el brazo A en la segunda ronda porque es consciente de que el brazo con menor coste acumulado tras la primera ronda es dicho brazo. Por ello, asignó otros valores a los costes de la ronda 2 de modo que el brazo A tuviese un coste asociado de 1 y el brazo B un coste asociado de 0. El algoritmo ha elegido el brazo menos eficiente esta vez.

En la tercera ronda, el algoritmo cambia de brazo de nuevo y opta por el brazo B ya que tiene menor coste acumulado ( $1.5 > 1$ ), pero como el antagonista lo podía anticipar, cambió la función de costes para que en la ronda 3 el brazo A tuviese coste 0 y B coste 1. De esta manera, el algoritmo ha vuelto a seleccionar de nuevo el brazo menos óptimo por segunda vez consecutiva.

Ya en la cuarta ronda, el algoritmo prefiere el brazo A porque ahora es el que tiene menos coste acumulado hasta el momento ( $1.5 < 2$ ), de nuevo el antagonista sabía que esto va a suceder y otorgó, para la ronda 4, al brazo A un coste de 0 y al brazo B un coste de 1. Esta ya es la tercera vez que el algoritmo pierde frente al antagonista.

Tras exponer el ejemplo anterior, se comprende, bajo la premisa de que el adversario descubre la estrategia de elección de brazos del algoritmo, que el adversario puede forzar al algoritmo a tomar decisiones equivocadas cada ronda. Concretamente, se observa que el algoritmo nunca será capaz de identificar cuál de los dos es el brazo que ofrece los costes óptimos.

Por tanto, a partir de la segunda ronda, aunque podría haber sido a partir de la primera ronda si la elección aleatoria del algoritmo hubiese sido el brazo peor en lugar del mejor, el algoritmo elige el brazo con mayor coste en todas las rondas posteriores ya que es engañado por el antagonista. Si se considera que cada ronda tiene una unidad de remordimiento porque en cada ronda el menor coste es 0 y el máximo es 1, al cabo de  $T$  rondas su remordimiento será de  $T$  unidades ya que sufre el máximo coste en todas las rondas. El remordimiento en esta situación se puede expresar de la siguiente forma:

$$R(T) = \mathcal{O}(T)$$

Este remordimiento, como ya se vió en la sección de los bandidos estocásticos 4, refleja la incapacidad de aprendizaje total por parte del algoritmo. Es decir, el peor o máximo remordimiento posible.

## 5.2. Algoritmos Efectivos

La idea clave para sortear esta dificultad es añadir aleatoriedad a la selección del brazo a jugar. De este modo, el algoritmo puede “sorprender” al adversario e impedir ser manipulado por el antagonista, ergo, no se alcanzarán remordimientos tan altos. Este efecto sorpresa basta para obtener un remordimiento esencialmente tan bajo como el remordimiento en el modelo estocástico.

### 5.2.1. Algoritmo Hedge

Por este motivo, se procede a explicar un algoritmo muy conocido en el escenario de bandidos antagonistas que incorpora esta aleatoriedad mencionada, el algoritmo Hedge 6 obtenido de [2].

En finanzas, el término “hedge” se refiere a una estrategia de inversión que se utiliza para reducir o eliminar el riesgo de pérdidas en una posición o cartera de inversión. La idea detrás del *hedging* es proteger una inversión contra posibles movimientos desfavorables en el mercado, lo que puede ayudar a minimizar las pérdidas en

---

**Algoritmo 5.6 Algoritmo Hedge ( $\epsilon$ )**

---

**Requiere:**  $\epsilon \in (0, \frac{1}{2})$

- 1: Se inicializan los pesos:  $w_1(a) = 1$  para cada brazo  $a$ .
  - 2: **for** cada ronda  $t = 1, 2, \dots$  **do**
  - 3:   Se calcula  $p_t(a) = \frac{w_t(a)}{\sum_{a'=1}^K w_t(a')}$  para cada brazo  $a$ .
  - 4:   Se muestrea un brazo  $a_t$  de la distribución  $p_t(\cdot)$ .
  - 5:   Se observa el coste  $c_t(a)$  para cada brazo  $a$ .
  - 6:   **for** cada brazo  $a$  **do**
  - 7:     Se actualiza el peso:  $w_{t+1}(a) = w_t(a) \cdot (1 - \epsilon)^{c_t(a)}$ .
  - 8:   **end for**
  - 9: **end for**
- 

caso de que el mercado se mueva en contra de la posición. De esta forma, se entiende la razón que hay detrás del nombre de este algoritmo, dado que incorpora aleatoriedad en su elección de brazos para cubrirse ante la posibilidad de que un antagonista esté presente en el problema.

Para comenzar el análisis del algoritmo, se debe observar en el paso 4 que Hedge utiliza una regla basada en la aleatoriedad para escoger un brazo. Cuando se toma una muestra de una distribución de probabilidad, cada elemento del conjunto tiene una cierta probabilidad de ser seleccionado. Esta probabilidad está asociada con el valor de la función de densidad de probabilidad (o función de masa de probabilidad en el caso discreto) en ese punto. El proceso de muestreo se realiza de manera aleatoria, lo que significa que el resultado de la selección no es seguro y puede variar en diferentes realizaciones del proceso porque depende del azar. De este modo, se consigue solucionar el problema que presentan los algoritmos deterministas al abordar problemas de bandidos antagonistas, y se consigue un remordimiento muy inferior a  $\mathcal{O}(T)$ .

**Teorema 5.1.** *El algoritmo Hedge consigue un límite de remordimiento esperado en  $\mathcal{O}(T \log K)^{1/2}$ . Es un remordimiento similar al conseguido con los algoritmos avanzados usados en problemas de bandidos estocásticos.*

Al principio, el algoritmo Hedge recibe un parámetro  $\epsilon$  que utiliza para ajustar la velocidad con la que el peso de cada brazo disminuye en cada ronda. Matemáticamente, cuanto más valor tenga  $\epsilon$  menor será  $w_{t+1}(a)$  con respecto de  $w_t(a)$ . Observando la actualización de los pesos de los brazos, se puede constatar que si  $\epsilon = 0$ , los pesos mantendrían el valor 1 en todas las rondas. Por este motivo, el algoritmo exige que  $\epsilon$  pertenezca al intervalo abierto con límite inferior en 0, es decir, que  $\epsilon$  no pueda tomar el valor 0.

Como indica el paso 5, Hedge funciona con *retroalimentación total*. Este concepto consiste en que el algoritmo obtiene los costes de todos los brazos a pesar de haber jugado solo uno. Es decir, el bandido conoce los costes de todos los brazos tras cada ronda. Se puede deducir que la retroalimentación total es una ventaja para el algoritmo ya que le aporta más información acerca de qué brazos desempeñan mejor la tarea. Precisamente, el algoritmo utiliza toda esa información para actualizar los pesos de cada brazo y, con estos, modificar las probabilidades de cada uno de ser elegido en la siguiente ronda. Así es como construye la distribución de probabilidad que utiliza para elegir brazo. Por tanto, aquellos brazos con menor peso con respecto al resto de brazos, han devuelto los costes más altos en las rondas anteriores y presentan una menor probabilidad de ser escogidos por el algoritmo.

Es por esto que la retroalimentación total solo puede estar presente en un algoritmo dedicado a resolver problemas de bandidos antagonistas. Si este tipo de retroalimentación estuviese presente en los bandidos estocásticos el algoritmo conocería los costes de todos los brazos sobre una distribución fijada de recompensas igual para todas las rondas y no tendría tanta necesidad de explorar. Atendiendo a esta explicación, el problema de bandidos estocásticos con retroalimentación total carece de sentido.

### 5.2.2. Algoritmo Exp3

A continuación, se presenta el algoritmo Exp3 (“Exponential-weight algorithm for Exploration and Exploitation”) 7 introducido por Freund and Schapire en [9], que servirá como base para comprender el algoritmo Exp4 que será analizado al final de esta sección. Exp3 es una variante del algoritmo Hedge.

---

**Algoritmo 5.7 Exp3 ( $K, T, \gamma$ )**


---

- 1: Se inicializan:  $w_i(1) = 1$  para  $i = 1, \dots, K$ .
- 2: **for**  $t = 1, 2, \dots, T$  **do**
- 3:    $p_t(i) = (1 - \gamma) \frac{w_t(i)}{\sum_{j=1}^K w_t(j)} + \frac{\gamma}{K}$  para  $i = 1, \dots, K$
- 4:   Se elige el brazo  $i_t$  con respecto a  $p_t(1), p_t(2), \dots, p_t(K)$
- 5:   Se recibe la recompensa  $x_{i_t} \in [0, 1]$
- 6:   **for**  $j = 1, 2, \dots, K$  **do**
- 7:      $\hat{x}_j = \begin{cases} \frac{x_j}{p_t(j)}, & \text{si } j = i_t, \\ 0, & \text{en otro caso.} \end{cases}$
- 8:      $w_j(t+1) = w_j(t) \exp(\frac{\gamma \hat{x}_j}{K})$
- 9:   **end for**
- 10: **end for**

---

Para comenzar el análisis es importante resaltar que comparte con Hedge la forma de seleccionar brazos aleatoriamente, lo que se puede comprobar en el paso 4. Por tanto, el algoritmo Exp3 también es eficaz ante un problema de bandidos antagonistas.

**Teorema 5.2.** *El algoritmo Exp3, según [6], produce un remordimiento esperado de  $\mathcal{O}(KT \ln K)^{1/2}$ . Para cualquier valor de  $K$ , este remordimiento es ciertamente superior al obtenido con Hedge.*

La figura 4 revela la diferencia entre los remordimientos de Exp3 (morado) y Hedge (amarillo) para un problema con  $K = 5$  brazos. También expone los remordimientos de obtenidos por los algoritmos simples (rojo) y avanzados (naranja).

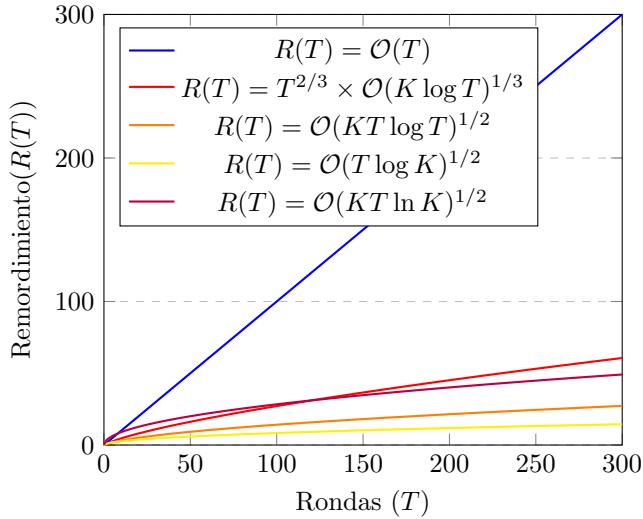


Figura 4: Gráfica de Remordimiento con  $K = 5$ .

Por otra parte, según el paso 5, el algoritmo solo contempla recompensas pertenecientes al intervalo cerrado entre 0 y 1. Por tanto, si las recompensas del problema están en el rango  $[a, b]$ ,  $a < b$ , entonces Exp3 se puede utilizar después de que las recompensas se han trasladado y reescalado al intervalo  $[0, 1]$ .

No debe ser motivo de preocupación observar que el algoritmo está en clave de recompensas, en vez de costes. Como se ha comentado con anterioridad, la conversión a costes es muy sencilla: costes = 1 – recompensas.

El factor de igualdad  $\gamma$  presente en este algoritmo funciona como un mecanismo para asegurar la explotación de todos los brazos creando una distribución de probabilidad que combina una distribución ponderada ( $\frac{w_t(i)}{\sum_{j=1}^K w_t(j)}$ ), actualizada exponencialmente cada ronda para cada brazo según las recompensas anteriores, con una distribución uniforme ( $\frac{\gamma}{K}$ ). Matemáticamente, cuanto más cercano sea el valor de  $\gamma$  a 1, es decir, si el factor de igualdad adopta un mayor valor, más parecida será la distribución de probabilidad a una distribución uniforme. En cambio, si  $\gamma$  toma valores cada vez más cercanos a 0, la distribución final se acercará más a una

distribución ponderada según los pesos. La presencia de la distribución uniforme es la que motiva al algoritmo a explorar los brazos con menor probabilidad de ser escogidos. Se puede configurar la intensidad del factor de igualdad para regular el nivel de exploración con respecto a la explotación. En las implementaciones asociadas a este trabajo, se comprobará qué valor del factor de igualdad es el más apropiado para favorecer el aprendizaje de los algoritmos ante las aplicaciones a las que se enfrentan.

Para concluir el análisis de Exp3, cabe destacar que también funciona mediante retroalimentación total. En este caso es una retroalimentación total simulada en la variable  $\hat{x}_j$ , de forma que estima una recompensa de ‘0’ para aquellos brazos que no hayan sido elegidos, consiguiendo que en el paso 8 el peso de dichos brazos no sea modificado porque  $\exp(\frac{\gamma^* 0}{K}) = 1$ . En el caso del brazo elegido, Exp3 recurre a un interesante método ( $\frac{x_j(t)}{p_j(t)}$ , si  $j = i_t$ ) para estimar su recompensa ponderándola con respecto a la probabilidad de escoger dicho brazo. Es decir, si un brazo tiene poca probabilidad de ser elegido, la recompensa que obtiene al ser elegido se calcula al alza para tenerlo en cuenta en su justa proporción. De forma inversa los brazos que tienen mucha probabilidad de ser elegidos por el algoritmo obtienen una recompensa infravalorada.

### 5.2.3. Algoritmo Exp4

Por último y más relevante, se presenta el algoritmo Exp4 8 utilizado en dos de las implementaciones más importantes que serán desarrolladas para ilustrar el problema del bandido contextual que motiva este trabajo de fin de grado.

---

#### Algoritmo 5.8 Algoritmo Exp4 para bandidos adversarios con asesoramiento de expertos $(E, \varepsilon, \gamma)$

---

**Requiere:** Conjunto  $E$  de expertos, parámetro  $\varepsilon \in (0, \frac{1}{2})$  para Hedge, parámetro de exploración  $\gamma \in [0, \frac{1}{2})$ .

- 1: **for** cada ronda  $t$  **do**
- 2:   Se llama a Hedge, recibir la distribución de probabilidad  $p_t$  sobre  $E$ .
- 3:   Se extrae un experto  $e_t$  de forma aleatoria a partir de  $p_t$ .
- 4:   Regla de selección: con probabilidad  $1 - \gamma$  seguir al experto  $e_t$ ; de lo contrario, elegir un brazo  $a_t$  uniformemente al azar.
- 5:   Se observa el coste  $c_t(a_t)$  del brazo elegido.
- 6:   Se definen los “costes falsos” para todos los expertos  $e$ :

$$\hat{c}_t(e) = \begin{cases} \frac{c_t(a_t)}{\Pr[a_t=a_{t,e}|\vec{p}_t]} & \text{si } a_t = a_{t,e}, \\ 0 & \text{en caso contrario.} \end{cases}$$

- 7:   Se devuelven los “costes falsos”  $\hat{c}(\cdot)$  a Hedge.
  - 8: **end for**
- 

Antes de comenzar, es interesante comentar la introducción a los expertos que realiza el algoritmo Exp4 (Exploración, Explotación, Exponencial, Expertos). A diferencia del Exp3, este algoritmo ya no escoge brazos directamente sino que elige expertos y son estos expertos los que se encargan de la elección del brazo. Se contempla tanto que el experto recomienda el mismo brazo durante todas las rondas como que el experto tenga recomendaciones distintas para cada ronda. Es importante aclarar que los expertos no aprenden con el paso de las rondas. Es decir, cada experto tiene su estrategia fija de elección de brazo.

El objetivo del algoritmo Exp4 es aprender cuál es el mejor experto dentro del conjunto de expertos que se le facilita para minimizar el coste obtenido. En este caso, el algoritmo se abstrae de la elección de brazo, no debe confundirse con los objetivos de algoritmos anteriores que persiguen aprender cuál es el brazo óptimo para explotarlo y recibir el mínimo coste posible.

En cuanto al resto de parámetros de entrada del algoritmo, la variable  $\epsilon$  se traslada directamente para que la use el algoritmo Hedge con la intención de moderar la velocidad con la que disminuye los pesos de los expertos tras cada ronda. Por otro lado, la variable  $\gamma$  es usada con el mismo objetivo que se usa en el algoritmo Exp3, es decir para establecer un equilibrio entre la explotación de los expertos que ha aprendido que funcionan y la exploración.

El proceso de elección de expertos también incluye la ayuda del algoritmo Hedge que se encarga de actualizar los pesos de cada experto y en consecuencia de renovar la distribución de probabilidad de elección de los mismos. Esta actualización se realiza en función de los “costes falsos” proporcionados el algoritmo Exp4, que simulan una retroalimentación total parecida a la del Exp3. Para calcular los “costes falsos” Exp4 hace uso de la variable  $\Pr[a_t = a_{t,\pi} | \vec{p}_t]$ . Esta probabilidad representa la probabilidad conjunta entre todos los expertos de recomendar el brazo  $a_t$ . Para calcularla, hay que sumar, de forma ponderada con respecto al peso de cada experto, las probabilidades individuales de cada experto de recomendación de dicho brazo.

Por ejemplo, supóngase un escenario con tres expertos, cada uno con una ponderación o peso asignado en función de su desempeño. Las probabilidades asignadas a cada experto de recomendar un brazo específico, por ejemplo,  $a_t$ , podrían ser 0,1, 0,2 y 0,3 respectivamente. A su vez, los pesos correspondientes a cada experto podrían ser 0,5, 0,3 y 0,2, también respectivamente..

Para calcular la probabilidad conjunta de que todos los expertos recomiendan el brazo  $a_t$ , se sumarían las probabilidades de recomendación de cada experto, ponderadas según su peso. Así, se tendría:

$$\Pr[a_t = a_{t,\pi} | \vec{p}_t] = 0,1 \times 0,5 + 0,2 \times 0,3 + 0,3 \times 0,2 = 0,23$$

En este supuesto, la probabilidad conjunta de que todos los expertos recomiendan el brazo  $a_t$  es de 0,23 o 23 %. Esto significa que existe una probabilidad del 23 % de que el brazo  $a_t$  sea recomendado por todos los expertos, teniendo en cuenta las probabilidades individuales de recomendación y los pesos de los expertos.

Por otra parte, aunque en este caso el algoritmo Hedge no se encargue de realizar la selección de brazo y se limite a realizar las tareas mencionadas anteriormente, es interesante resaltar que los “costes falsos” que recibe dependen de su propia actualización de la distribución de probabilidad. Esta dependencia se evidencia al observar que la distribución de probabilidad entra en el cálculo de los ‘costes falsos’ en el paso 6. Es como si para el algoritmo Hedge estuviese presente un antagonista adaptable muy peculiar, el mismo. Es decir, los costes que recibe dependen de sus propias actualizaciones en las rondas anteriores.

Para concluir esta sección, es determinante comprender el hecho de que en el ámbito de los bandidos el algoritmo Exp4 es un algoritmo “agnóstico”, aunque originalmente haya sido diseñado para el escenario antagonista. Cuando se dice que un algoritmo es “agnóstico” significa que es independiente de ciertos aspectos o características del problema que intenta resolver. Esto le permite funcionar de manera eficaz en una amplia gama de problemas, sin depender de la información específica del dominio. De hecho, será uno de los algoritmos empleados en la sección de los bandidos contextuales para solucionar un problema concreto de dicha sección.

El remordimiento asociado a este algoritmo depende de la configuración del problema al que se exponga.

## 6. Bandidos Contextuales

Este capítulo es el corazón de este trabajo. En esta sección se aborda en detalle el problema del bandido multi-brazo contextual. Para lograrlo, se distinguen diferentes aplicaciones de bandidos contextuales con el objetivo de exponer la casuística específica de algunas de las variantes que este problema puede presentar. Además, se ofrecen implementaciones respecto a cada apartado para asimilar los conceptos explicados y comprender de manera directa la lógica aplicada por estos algoritmos, así como evaluar su eficiencia.

Aprender los diferentes enfoques de los bandidos multi-brazo contextuales permitirá entender su funcionalidad y su utilidad en múltiples casos de uso. Sin embargo, para empezar, se pone el foco en definir lo que es un bandido multi-brazo contextual, así como sus principales rasgos, y, sobre todo, qué es lo que le diferencia de otro tipo de bandidos multi-brazo.

A continuación se ofrece una generalización del problema para apreciar sus dinámicas:

### Problema Bandidos Multi-brazo Contextuales

En cada ronda  $t \in [T]$ :

1. El algoritmo observa un contexto  $x_t$ .
2. El algoritmo elige un brazo  $a_t$ .
3. El algoritmo observa la recompensa  $r_t \in [0, 1]$  para el brazo elegido.

La idea fundamental en este tipo de bandidos es que la recompensa  $r_t$  de cada ronda  $t$  depende de dos cosas: del contexto  $x_t$  y del brazo  $a_t$ . Los bandidos contextuales pueden ser estocásticos o antagonistas, en términos de recompensas. Al igual que en ?? y a diferencia de en ??, donde se presentan estos bandidos bajo el modelo antagonista, en este trabajo se hace la suposición IID estocástica: la recompensa para cada brazo se obtiene independientemente de una distribución parametrizada por el par  $(x_t, a_t)$ , pero igual para todas las rondas  $t$ , es decir, no existe la presencia de un antagonista capaz de alterar la distribución de recompensas entre rondas. Por este motivo, en los bandidos contextuales se habla en términos de recompensa, y no de coste. La recompensa esperada para un brazo  $a$  dado un contexto  $x$  se denota  $\mu(a|x)$ .

Un ejemplo sencillo y breve que ilustra cómo las probabilidades de las distintas recompensas varían para cada brazo según el contexto podría ser el siguiente:

Supóngase un bandido multi-brazo contextual con 3 brazos, los cuales representan tres estrategias de inversión diferentes (A, B y C). El contexto, en este caso, podría ser el estado actual del mercado financiero, que se puede clasificar en dos categorías: mercado alcista (contexto 1) y mercado bajista (contexto 2).

En un mercado alcista (contexto 1), las probabilidades de obtener recompensas más altas podrían distribuirse de la siguiente manera para los brazos A, B y C: 70 %, 60 % y 40 %, respectivamente. Por otro lado, en un mercado bajista (contexto 2), las probabilidades podrían cambiar a: 30 %, 50 % y 80 %.

Este ejemplo demuestra que las probabilidades de obtener mayores recompensas varían para cada brazo según el contexto, y que un mismo brazo puede tener diferentes probabilidades de éxito dependiendo del contexto en el que se encuentre. En este tipo de problema, al algoritmo se le presenta un contexto en cada ronda y su objetivo es aprender a elegir el brazo que devuelve las mejores recompensas en el contexto dado.

Por otra parte, a pesar de que se ha mencionado que las recompensas serán consideradas estocásticas, en esta sección, el término antagonista va a ser empleado con otra acepción con el objetivo de explicar una casuística de los bandidos contextuales. Se recuerda el concepto de antagonista conocido hasta el momento: aquellas variables desconocidas para el algoritmo que pueden provocar la variación de la función de recompensas entre rondas. Se puede generalizar dicho concepto para después poder acogerlo en esta sección. Entonces, de forma general, un antagonista son aquellas variables desconocidas para el algoritmo que pueden provocar la variación de un elemento del problema de una ronda a otra. Ahora, puede resultar mucho más sencillo comprender el significado específico de antagonista en este escenario: aquellas variables desconocidas para el algoritmo son capaces de provocar la variación de los contextos tras cada ronda.

Una vez se ha acogido el concepto de antagonista para esta sección, es interesante identificar que el tipo de antagonista que va a estar presente detrás de los problemas que se plantean a continuación es un antagonista indiferente, dado que la variación de contextos no depende de las elecciones de brazo previas del algoritmo. Además, el contexto que recibe el algoritmo al comenzar una ronda no va a ser escogido con ninguna intención maliciosa.

Por ejemplo, el perfil de un usuario puede ser considerado el contexto de un problema de bandidos contextuales. Este usuario llega cada ronda, debido a variables desconocidas para el algoritmo (antagonista), y la clave de los algoritmos que se explica a continuación reside en que tienen la capacidad de personalizar la experiencia del usuario. Por este motivo, los bandidos contextuales son utilizados, entre otros escenarios, en sistemas de recomendación de artículos, anuncios o productos para mostrar al usuario. Por tanto, los brazos del problema son las distintas alternativas (artículos, anuncios o productos) que el algoritmo puede recomendar.

Las recompensas en los sistemas recomendadores se pueden definir a partir de los “clics” o a partir de las valoraciones que otorgan los usuarios. Es fácil entender, porque es común, que para distintos usuarios una misma recomendación puede dar lugar a diferentes recompensas debido a que cada perfil de usuario es distinto. Por ejemplo, hay usuarios que tenderán más a hacer clic en ciertos anuncios de política mientras que otros nunca harán clic en un anuncio de política.

La motivación de los bandidos contextuales es aprender cómo un contexto afecta a la recompensa y así ser capaz de mejorar su toma de decisiones o elección de brazos para adaptarse a contextos variados a lo largo de las rondas. Es decir, en el ejemplo anterior, el algoritmo aprendería el contexto de aquella persona que pinche en política con la intención de mostrar anuncios de política (el mismo brazo) a usuarios que presenten el mismo contexto que el mencionado, maximizando así su recompensa (clics). De este modo, en los bandidos contextuales, el algoritmo que aprende es el que es capaz de diferenciar entre contextos y aprovecharlo para optimizar su rendimiento.

## 6.1. Bandidos Contextuales con Pocos Contextos

Tras detallar la esencia de estos bandidos, se tratan varias aplicaciones como las comentadas anteriormente. En primer lugar, se explica un caso introductorio para entender cómo desarrollar bandidos multi-brazo contextuales en aplicaciones donde hay un bajo número de contextos.

Este algoritmo propone una implementación intuitiva pero también eficaz para ver en una primera instancia cómo se desenvuelven. Para ello, primero el algoritmo es expuesto, y posteriormente, se ofrece una implementación propia para analizar exhaustivamente este caso particular. Esta será la primera implementación expuesta en este trabajo.

En una situación donde hay un pequeño número de contextos, un enfoque adecuado podría ser aplicar el UCB1. Hay que recordar que dicho algoritmo corresponde a un bandido estocástico, expuesto en la sección 4. La idea radica en crear instancias de este algoritmo para cada contexto en particular. De este modo, un algoritmo en concreto, que es una instancia del UCB1, tratará cada contexto independientemente. Aplicando esta metodología se puede utilizar dicho algoritmo para cada posible contexto y sacar el máximo partido de cada uno de ellos. Por el contrario, si se usara un único UCB1 trataría todos los datos como si fueran del mismo contexto, mientras que este bandido contextual mejora el rendimiento en cada contexto mediante el uso de las copias o instancias. A continuación, se muestra la estructura básica de dicho algoritmo:

---

### **Algoritmo 6.9** Bandido contextual para pocos contextos.

---

- 1: Se inicializan: Para cada ronda  $x$ , se crea una instancia  $ALG_x$  del algoritmo  $ALG = \text{UCB1}$ .
  - 2: **for** cada ronda  $t = 1, 2, \dots$  **do**
  - 3:     Se invoca al algoritmo  $ALG_x$  según el contexto observado en la ronda  $x = x_t$ .
  - 4:     Se juega el brazo  $a_t$  elegido por el algoritmo  $ALG_x$ , y se devuelve la recompensa  $r_t$  al algoritmo  $ALG_x$ .
  - 5: **end for**
- 

**Teorema 6.1.** *El Algoritmo 6.9 tiene un remordimiento:  $E[R(T)] \leq O(KT|X|\log T)^{1/2}$  para cada ronda  $t \leq T$ , y siendo  $X$  el conjunto de contextos y  $|X|$  el número de contextos. Además para cada una de las instancias*

$ALG_x$ , se alcanaría un remordimiento de la siguiente magnitud:  $E[R(T)] \leq O(KT \log T)^{1/2}$ . Por lo que el remordimiento para todos los contextos sería la suma del remordimiento alcanzado por todas las instancias.

### 6.1.1. Recomendador de Películas para Cuatro Grupos

Con el objetivo de entender este algoritmo, se ha desarrollado una implementación basada en esta premisa de usar pocos contextos e instancias para cada uno de ellos. Concretamente, se trata de un sistema recomendador para un reducido número de contextos, 4, y cada uno es considerado como un grupo de personas. En el grupo  $x_1$  hay hombres de más de 40 años, en el grupo  $x_2$  hay mujeres de más de 40 años, en el grupo  $x_3$  hay hombres de menos de 40 años y en el grupo  $x_4$  hay mujeres de menos de 40 años. Con esta simplificación y un número tan reducido de grupos se puede ilustrar correctamente este caso.

Los brazos del sistema recomendador son películas, y son siempre las mismas 5 películas,  $K = 5$ . También cabe señalar que hay  $T$  rondas, siendo un parámetro arbitrario de acuerdo al experimento, y en cada ronda  $t$  se recomienda una película a un usuario, por lo que cada una de las rondas representa un usuario con una información contextual, esto es, un usuario que pertenece a un grupo. Por ejemplo, en la ronda uno se tiene que recomendar una película a un hombre mayor de 40 años y en la ronda 2 a una mujer de menos de 40, y así sucesivamente.

Por último, hay que señalar que las recompensas son obtenidas aleatoriamente para cada película, brazo, y para cada contexto, grupo. Las recompensas pueden tener tres valores [0, 0.6, 1]: siendo 0 si una película no le gusta nada al usuario, 0.6 si le gusta un poco y 1 si le gusta mucho.

Las recompensas se calculan aleatoriamente ya que no hay datos reales. Se utiliza una función creada por los autores que calcula las probabilidades aleatorias para cada grupo de usuarios y para cada brazo, y dentro de este contexto se generan tres probabilidades que suman 1 y representan lo que le puede gustar la película. Realmente las probabilidades de que a un grupo de usuarios le guste una película u otra se calculan de forma aleatoria ya que lo interesante de este apartado es tener diferentes gustos para las películas de cada grupo de usuario. Haciéndolo de esta manera, se asegura que al grupo de usuarios 1 es más probable que les guste una película A, mientras que al grupo de usuarios 2 habrá más posibilidades de que les guste una película D. Es necesario partir de esta premisa de que cada contexto, grupo de usuarios, tiene diferentes gustos, o lo que es lo mismo, distintas funciones de probabilidades, ya que al hacerlo es posible estudiar cómo de bueno es este algoritmo recomendando películas para cada grupo de usuarios con distintas preferencias. Sin embargo, sí que puede haber dos grupos de usuarios que tengan una distribución de recompensas parecidas, tal y como podría pasar si se analizan los gustos de películas de dos grupos en la vida real.

Posteriormente, durante la simulación, se extrae una probabilidad aleatoria para un brazo elegido en un contexto observado y usando su distribución de probabilidades, calculada anteriormente, se obtiene una recompensa dentro del rango mencionado que es elegida aleatoriamente pero en la distribución de probabilidades del brazo en el contexto.

Para ilustrar mejor cómo es la función de recompensas se expone un ejemplo. Dado el contexto 2, mujeres de más de 40 años, y el brazo 1, la película A, hay una distribución de probabilidades generada aleatoriamente que podría ser de la siguiente forma: 25 % a que no le gusta la película, 35 % a que le gusta un poco y 40 % a que le gusta mucho. Si en la ronda  $t$  aparece un usuario de este contexto y se elige el brazo 1, se cogerá uno de los tres posibles valores aleatoriamente de la recompensa conforme a esta distribución, pero al simular múltiples rondas, a pesar del factor de aleatoriedad, a la larga se obtendrá el 25 % una recompensa de 0, el 35 % una recompensa de 0.6 y el 40 % una recompensa de 1. Es importante considerar que la función de recompensas permanecerá constante. Por ejemplo, esto significa que dicha función de recompensas para mujeres de más de 40 años será igual para las películas A que se les muestren, sus gustos por dicha película serán iguales para todas las rondas, y habrá un 40 % de probabilidades de que le gusten mucho la película A en la ronda 1 y en la ronda  $T - 1$ .

Aplicando el algoritmo de bandidos multi-brazo contextuales para este caso de uso, se han definido cuatro instancias del algoritmo UCB1 de manera que cada instancia trata un grupo. Utilizando esta técnica es posible conseguir que la instancia 1 se especialice en encontrar cuál es el mejor brazo (película) para el grupo de varones mayores de 40 años mientras que la instancia 2 buscará dicho brazo para mujeres de más de 40 años. Esto permite que cada instancia explore y explote de manera diferente las películas según las preferencias de estos usuarios.

Por ejemplo, la instancia uno del algoritmo UCB1 que trata a varones mayores de 40 años podría explotar

mucho desde las primeras rondas la película C si observa que le da muy buenos resultados en comparación con el resto de películas. Por otro lado, la instancia que trate a las mujeres de menos de 40 años podría centrarse más en la exploración si hay tres películas: A, B y D que le reportan recompensas similares. De este modo, es apreciable que cada instancia adecuará su exploración-explotación de manera única para adquisearse a la función de probabilidades del grupo de usuarios que trata.

Tras definir el fundamento de estos algoritmos, se contemplarán cuáles fueron los pasos a seguir para implementar este ejemplo. En primer lugar, se crea una base de datos sintética, aleatoriamente ordenada de 10000 personas, es decir,  $T = 10000$  rondas. Las distribuciones en cada grupo son elegidas aleatoriamente, pudiendo haber un gran porcentaje de un grupo y un pequeño porcentaje de otros para ver cómo se desenvuelve el bandido con tamaños de grupos distintos. Cabe mencionar que, tal y como está siendo explicada esta sección, en este caso las palabras grupos y contextos son sinónimos, al igual que películas y brazos.

Después de haber obtenido los usuarios con su contexto, ha sido implementado el algoritmo UCB1 [de acuerdo al esquema del algoritmo 2.5]. El algoritmo consiste esencialmente en dos funciones fundamentales. Primero tiene que seleccionar un brazo usando el enfoque optimista, según el valor UCB, que se calcula con la recompensa media, y representa la explotación, y con el radio de confianza, que representa la exploración. Así el algoritmo busca el balance para cada grupo de personas de explorar o explotar. También el algoritmo debe actualizar la recompensa media para cada brazo, y se hace esta operación tras obtener una recompensa para el brazo obtenido. Esto se actualiza para que en las próximas rondas se tengan en cuenta los valores actualizados para la selección de brazo.

Una vez hechos todos estos pasos (la creación de usuarios, el algoritmo UCB1 y la función de recompensas), ha sido realizada una simulación para observar la actuación del algoritmo contextual. Como lo que se pretende es entender cómo de bueno es su rendimiento, se comparan sus recompensas y remordimiento con un algoritmo UCB1. Por lo tanto, por un lado está el algoritmo contextual que son cuatro instancias del UCB1, una para cada contexto, y por otro lado se encuentra el algoritmo UCB1 que trata todos los datos por igual y no distingue contextos. Este segundo algoritmo recomendará por igual a personas del grupo 1 y del grupo 4 por lo que lo que realmente se estudiará aquí es cuál es el valor de considerar la información contextual. Pues bien, se podrá observar en las siguientes gráficas que muestran el rendimiento de ambos bandidos durante una simulación de 10000 usuarios. Además debe tenerse en cuenta que el bandido UCB1 puede usarse como una representación del rendimiento de un bandido estocástico frente a un bandido contextual, y así se podrá apreciar cómo de importante es el contexto para dos algoritmos que tienen la misma estructura.

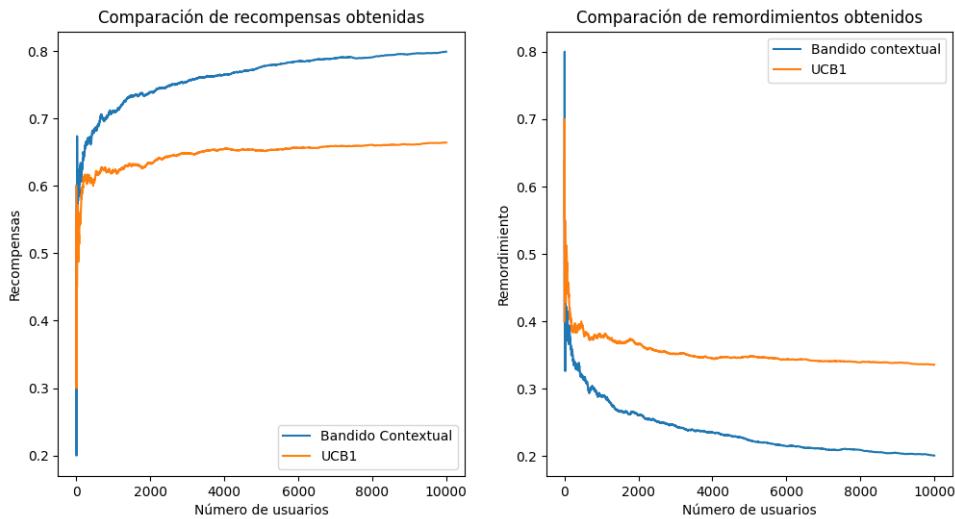


Figura 5: Recompensas medias y remordimientos medios.

En ambas gráficas se puede observar la evolución de las recompensas obtenidas para ambos algoritmos: el bandido contextual y el UCB1 que es un bandido estocástico. Se puede apreciar como aumenta considerablemente la recompensa en estos bandidos que consideran el contexto y produce una distancia significativa entre la recompensa y los remordimientos tipos. Además, se muestra como a lo largo de las rondas los dos bandidos van mejorando su rendimiento ya que aprenden, pero los bandidos contextuales aprenden más rápido al principio y por el hecho de tener en cuenta el contexto alcanzan mejores recompensas y, consecuentemente, menor

remordimiento.

Gracias a estos resultados, por primera vez aparece la utilidad del contexto para mejorar el rendimiento de los bandidos multi-brazo. Y además, es destacable que este es un enfoque intuitivo para ilustrar las ventajas de primera mano de los bandidos multi-brazo contextuales. Como conclusión, los resultados son más que satisfactorios y presentan un horizonte prometedor para el desarrollo de bandidos contextuales en posteriores aplicaciones prácticas.

A pesar de esto, se podría pensar que la distribución de la población, es decir, el número de personas de cada uno de los cuatro grupos puede afectar significativamente a las recompensas obtenidas, ya que el número de usuarios que pertenece a un grupo es obtenido aleatoriamente. Por este motivo, la misma simulación ha sido realizada para cien distribuciones o bases de datos distintas, habiendo en cada una de estas 10000 personas repartidas aleatoriamente entre los 4 grupos. Al usar este enfoque, se podrá ver si realmente los bandidos contextuales son mejores para distintas distribuciones de los usuarios, y si son eficientes en distribuciones con distintos porcentajes de usuarios de un grupo. En este análisis, se observará el rendimiento de este algoritmo para distribuciones en las que podrá haber pocos miembros de grupo de usuarios. Así, se estudiará si usar un bandido contextual puede ser una opción interesante aunque pudiese haber pocos usuarios de un cierto grupo y, en este caso, el aprendizaje de la instancia correspondiente se vería perjudicado. Por ello, es significativa esta cuestión. Los resultados son proporcionados en las siguientes gráficas.

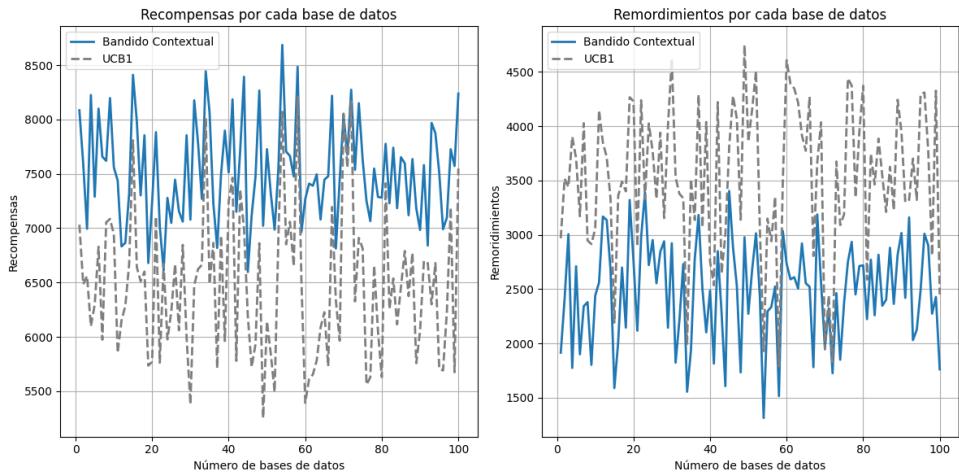


Figura 6: Resultados para distintas distribuciones de usuarios.

Cabe mencionar que estas últimas gráficas muestran las recompensas acumuladas para cada base de datos, estando una base de datos compuesta por 10000 usuarios y siendo la recompensa máxima 10000, el número de usuarios por la recompensa máxima, que es 1.

Se puede afirmar con total firmeza que los resultados son prometedores para distintas distribuciones, y para una cantidad suficiente de datos, la distribución de los contextos no es ningún problema para los bandidos contextuales. Generalmente el bandido contextual es más eficaz que un bandido estocástico, tal y como reflejan los datos de recompensas y remordimientos.

Por otra parte, el último experimento desarrollado en esta sección ha sido para ver en qué medida influyen el número de contextos para evaluar el rendimiento del bandido. Debido a esto, se supone que en vez de cuatro grupos de personas o cuatro contextos, hay diversos números crecientes de contextos para comprobar si este bandido contextual, enfocado a pocos contextos, seguiría obteniendo mejores recompensas para muchos contextos respecto a un bandido estocástico como el UCB1.

Esta última parte de la sección arrojará resultados determinantes respecto a la comparación entre bandidos contextuales y bandidos estocásticos y, en definitiva, concluirá por presentar las ventajas de estos bandidos en una primera instancia.

En la figura 7 están representados los resultados de un bandido contextual frente a un bandido estocástico para diferentes tamaños del contexto. Los brazos y la forma de obtener la recompensa no han variado respecto al inicio de la implementación, pero para utilizar varios tamaños de contextos se han tenido que realizar algunas

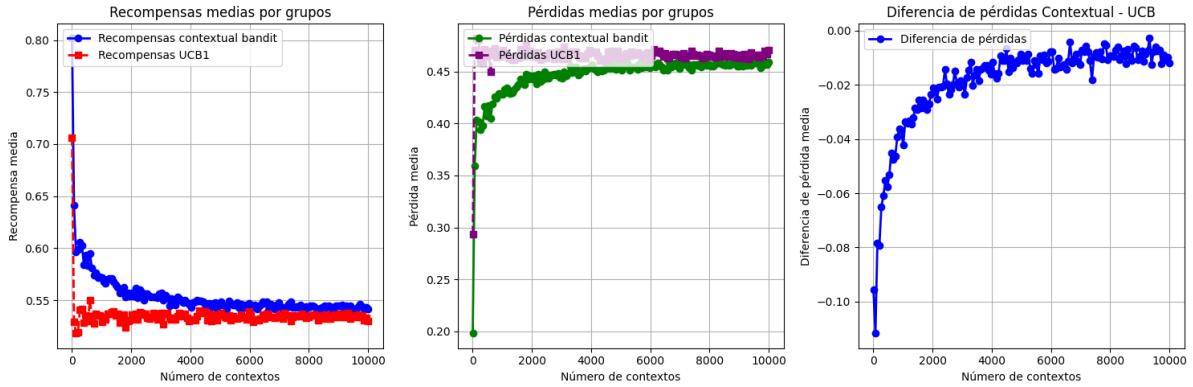


Figura 7: Resultados al aumentar el número de contextos.

modificaciones. Para ello, se han creado al igual que antes distribuciones de 10000 usuarios y 150 simulaciones, habiendo para cada una de estas distribuciones un número de contextos distintos.

Esto significa que para cada una de las 150 simulaciones se ha probado con un tamaño distinto del contexto, y para cada tamaño se han recomendado películas a 10000 usuarios. De esta manera, se observarán las recompensas medias si hubiera 4 grupos de usuarios y se les sugieren películas a todos ellos. También se analizan las recompensas medias si hay 10 grupos de usuarios, 100 grupos, 500 grupos... Así hasta los 10000.. Realmente ha sido realizado el mismo proceso de recomendación a 10000 usuarios pero 150 veces, y en cada una de estas veces los usuarios se dividen en menos o más grupos. De este modo, se puede ver cómo influye el tamaño del grupo.

Esta simulación se realizará 150 veces y en cada una de las iteraciones se usa un número de contextos distintos. Para ver realmente el efecto del número de contextos se plantea la siguiente lógica: se comienza con cuatro distintos contextos al igual que antes, y va aumentando el número de contextos hasta llegar al tamaño de la distribución, es decir, 10000, un contexto distinto para cada usuario. Concretamente, los 150 tamaños de contextos distintos se encuentran uniformemente divididos desde 4 hasta  $T$ , que es 10000.

Los resultados son esclarecedores y confirman las evidencias que se están exponiendo a lo largo de este apartado. Es señalable que el rendimiento de ambos bandidos es mejor para pocos contextos, y conforme aumenta el número de distintos contextos, ambos bandidos obtienen peor rendimiento, cosa que se ve reflejada en las recompensas medias y en las pérdidas, remordimientos medios. Además, hay una gráfica adicional para ver la diferencia entre remordimientos y con su ayuda se entiende perfectamente que el bandido contextual siempre alcanza menos remordimiento que el estocástico, ergo el rendimiento de un bandido contextual es considerablemente superior. Por ello, si se comparan estrictamente el UCB con un bandido contextual que instancie UCBs para cada contexto, es observable que el rendimiento de las instancias para cada contexto será superior.

No obstante, debe tenerse en cuenta que si se aumentan el grupo de usuarios hasta tener un grupo por cada usuario, es decir, 10000 contextos, cada instancia del UCB1 tratará exclusivamente a un usuario y por lo tanto, no aprenderá. Por este motivo, se observa en el gráfico que para muchos grupos de usuarios, y muchas instancias del UCB1, el algoritmo no aprende ya que tiene pocos usuarios a los que recomendar. Además, cabe señalar que las mejores recompensas se dan al principio de la gráfica, cuando hay pocos grupos de usuarios y sí que hay un espacio notable entre las recompensas del bandido contextual y el bandido estocástico. Y, a partir de aproximadamente los 2000 grupos de usuarios, 5 usuarios por grupo ( $K$  usuarios), las recompensas son extremadamente malas porque ninguno de los algoritmos aprenden pero el contextual sigue obteniendo una insignificante mejor recompensa media.

Debido a esto, los mejores resultados son obtenidos usando un número pequeño de grupos de usuarios, es decir, reduciendo los contextos. Y también estas recompensas se mejoran aprovechando los bandidos contextuales. Es conveniente recordar que en las primeras figuras de esta sección alcanzaban recompensas de entre 0,7 y 0,8.

Y para concluir, se puede afirmar que en efecto este tipo de bandidos contextuales rinde de manera sobresaliente en entornos con pocos contextos. Además, sirve de perfecta muestra para exemplificar lo interesante que puede ser desarrollar bandidos contextuales. Considerando el contexto es posible desarrollar algoritmos más óptimos y también estudiar problemas más complejos gracias a los bandidos multi-brazo contextuales.

## 6.2. Bandidos Contextuales Lipschitzianos

### 6.2.1. Motivación

En esta sección se considera una variante de los bandidos contextuales, los lipschitzianos o de tipo Lipschitz. Se tratará la motivación detrás de este tipo en concreto de bandidos contextuales y cuáles son sus principales características.

A continuación, se procederá a explorar en profundidad los bandidos contextuales con la condición de Lipschitz. Este marco permite manejar bandidos contextuales con un gran número de contextos. La condición de Lipschitz es el eje de esta sección. Además, en lugar de tener contextos discretos para elegir (como un número finito de opciones), hay contextos ilimitados en un espacio continuo. Es esencial reconocer que existen infinitos contextos, y estos se encuentran distribuidos de forma continua.

Dada la naturaleza continua del espacio de contextos, los problemas que involucran un número infinito de contextos representan una gran complejidad, volviéndose inmanejables en muchos casos. No obstante, el algoritmo de bandido contextual lipschitziano puede capitalizar esta circunstancia.

Se presume que es posible ubicar cualquier contexto en un intervalo  $[0, 1]$ . A este proceso se le denominará “mapear”, que refiere a situar un contexto dentro de un intervalo determinado. Se puede postular que, de acuerdo a la condición Lipschitz, las recompensas pueden ser interpretadas en relación a los contextos mediante la utilización de una variable  $L$ , conocida como constante de Lipschitz, que es reconocida por el algoritmo. A continuación, se presenta la condición de Lipschitz:

**Definición 6.1.** *Un bandido es Lipschitziano si cumple con la condición:  $|\mu(a|x) - \mu(a|x')| \leq L \cdot |x - x'|$  para cualquier brazo  $a, a' \in A$  y contextos  $x, x' \in X$ . La condición de Lipschitz implica que la diferencia entre las recompensas medias de un brazo para dos contextos diferentes es menor o igual a la diferencia entre los contextos multiplicada por una constante  $L$ .*

Si se cumple esta condición en un problema, será posible discretizar uniformemente el espacio contextual. La discretización es un enfoque que permitirá dividir el espacio contextual. Considerando que existe un espacio contextual donde se pueden representar todos los contextos posibles, se usará la discretización para dividir este espacio en distintas secciones. De esta manera, se trata el contexto observado en cada ronda como si perteneciera a una sección del espacio de contextos y cada uno de estos espacios discretizados será tratado por un algoritmo distinto.

Para explicar este proceso de discretización se puede usar un ejemplo muy intuitivo y representativo. Se puede suponer que se quiere estudiar la situación de unos estudiantes en función de su nota académica, que puede ser del uno al diez. Mediante el proceso de discretización, el espacio de posibles contextos será dividido en: sobresaliente, notable, bien, suficiente o insuficiente. Usando este enfoque, cada uno de los cinco espacios discretizados será tratado con un algoritmo distinto. Por ello, los estudiantes que tengan entre un siete y un ocho serán analizados por un algoritmo y los alumnos que tengan menos de un cinco serán estudiados por otro algoritmo ya que pertenecen a otro espacio discretizado. Utilizando esta aproximación, habrá algoritmos especializados para cada tipo de estudiante, y esto podría permitir a los algoritmos rendir mejor según el contexto.

Las ventajas que ofrece este sistema es que para un gran número de contextos, estos se pueden representar en un intervalo arbitrario, mapear. En cada una de las secciones habrá un algoritmo especializado en los contextos que haya en dicha sección. Además, la metodología será parecida al apartado anterior de pocos contextos 6.1. Habrá una instancia de un algoritmo UCB para cada espacio discretizado. En el ejemplo anterior, un algoritmo se encargaría de los alumnos de sobresaliente, otro de los de notable, otro de los de bien, uno distinto de los de suficiente y un último de los de insuficiente.

Para discretizar se usa la siguiente función:

$$f_S(x) = \min(\arg \min_{x' \in S} |x - x'|).$$

Con esta forma de mapear, cada punto es puesto en una región del espacio discretizado de acuerdo a esta función conocida como la *función de mapeo*. Gracias este mapeo, cada contexto observado en una ronda  $t$ , será ubicado en uno de los contextos discretizados, y para este contexto discretizado, existirá una instancia del algoritmo *UCBs* que operará exclusivamente en dicha zona discretizada.

Con el objetivo de entender correctamente los bandidos contextuales lipschitzianos, hay que exponer una implementación propia, y se profundiza en esta materia. Antes de poder ahondar en la implementación, es imprescindible demostrar que se cumple la condición de Lipschitz, y el siguiente apartado se centrará en esta cuestión.

En último lugar, se muestra el remordimiento alcanzado para este tipo de bandidos contextuales. El remordimiento obtenido es similar al que alcanzan los bandidos estocásticos, pero este bandido contextual lipschitziano permite abordar problemas complejos que, sin usar la condición de Lipschitz, no serían trazables por un bandido multi-brazo. El remordimiento es el siguiente:

**Teorema 6.2.**  $E[R(T)] \leq T^{2/3} \times O(LK \log T)^{1/3}$ . Este remordimiento es según el número de  $T$  rondas, el número de  $K$  brazos y la constante Lipschitz  $L$ .

### 6.2.2. Implementación de Bandidos Contextuales Lipschitzianos del Mercado

A continuación, se presenta el desarrollo de la implementación de bandidos contextuales de tipo Lipschitz de acuerdo a las directrices expuestas en el libro de Slivkins. Se procederá a detallar el objetivo de esta implementación, la metodología para desarrollar esta práctica y las conclusiones obtenidas tras realizar este trabajo.

En primer lugar, se define cuál es el fundamento de esta práctica. Se trata de un sistema recomendador que utiliza un *bandido multi-brazo contextual lipschitziano*. Este sistema recomendador cuenta con distintas carteras, que son los brazos, cada cartera está formada por un porcentaje distinto de acciones y bonos, de manera que hay carteras que tienen un gran porcentaje de bonos mientras que otras tienen casi todo acciones.

Se parte de la base de que en el siguiente apartado se analiza y demuestra la *condición de Lipschitz*. Esto aparece en la subsección 6.2.3 donde se demuestra que se cumple dicha condición. Dicha sección es fundamental para que se pueda plantear el problema ya que debe darse esta condición para desarrollar el algoritmo adecuadamente considerando el contexto.

En este problema, se considerará el contexto del crecimiento del mercado y la tasa de interés. Se aplica una lógica sencilla e intuitiva para entender el funcionamiento de este tipo concreto de bandidos. El mercado será reducido a dos variables macroeconómicas: el crecimiento del mercado y la tasa de interés, y se jugará con dichas variables para explicar este modelo. En un mercado real está claro que afectan innumerables variables al precio de activos financieros como bonos o acciones. Sin embargo, al prescindir de otras variables, es posible centrarse en un modelo simulado dependiente únicamente del crecimiento del mercado y la tasa de interés para poner en práctica el objetivo de este capítulo: la discretización para desarrollar bandidos contextuales tipo Lipschitz.

Respecto a las características del mercado financiero simulado, hay dos variables que son la tasa de interés y el crecimiento del mercado, y existen dos tipos de activos: los bonos y las acciones. Se considerará que los bonos son activos financieros que ofrecen poca rentabilidad. Se puede ganar poco dinero con ellos, pero su principal ventaja es que ante un decrecimiento del mercado no pierden tanto valor como las acciones. Debido a esto, es más interesante comprar bonos si el mercado está cayendo, frente a acciones. Además, su rentabilidad está ligada a la tasa de interés que fijan los bancos centrales. Una vez entendido esto, es posible comprender que ante una buena tasa de interés y un decrecimiento del mercado es bastante mejor opción comprar bonos que comprar acciones ya que a pesar de que no reporten tantos beneficios, son opciones más seguras que pueden ser más beneficiosas en estas situaciones, o contextos.

Por otro lado, las acciones dependen mucho de la situación del mercado. En un mercado en expansión, o en crecimiento, se puede suponer que las acciones van a subir mucho y ofrecen rentabilidades muy altas. Al contrario que los bonos, que ofrecen ganancias mínimas, las acciones proporcionan beneficios muy buenos. Lógicamente, si el mercado cae, las acciones perderán mucho valor porque al igual que dependen mucho para ganar, también bajan mucho si el mercado cae, por eso se dice que fluctúan mucho. Por ello, ante un crecimiento al alza del mercado es más interesante comprar acciones ya que un bono no otorgará apenas beneficios. Pero con un mercado en decadencia, las acciones pueden hacer perder mucho dinero y los bonos son valores más seguros.

Una vez explicado esto, ya está clara la idea de que con una buena tasa de interés son convenientes los bonos pero que si el mercado crece las acciones son muy tentadoras. Surge entonces la pregunta de cuál es la mejor estrategia para elegir una cartera con más bonos o más acciones según el contexto: mercado y tasa de interés. Estarán presentes los siguientes tipos de cartera a elegir para que el algoritmo pueda jugar con las carteras que mejor le parezcan según las condiciones del mercado.

- Brazo 1: 0 % de acciones y 100 % de bonos.
- Brazo 2: 25 % de acciones y 75 % de bonos.
- Brazo 3: 50 % de acciones y 50 % de bonos.
- Brazo 4: 75 % de acciones y 25 % de bonos.
- Brazo 5: 100 % de acciones y 0 % de bonos.

Estas cinco carteras van a permitir buscar la opción óptima conforme al contexto que haya. Como ya se ha explicado, el crecimiento del mercado afecta muy positivamente a las acciones. Por eso, con un crecimiento alto del mercado se elegirá siempre el brazo 5, ya que los otros brazos a pesar de que ganen dinero tendrán un remordimiento muy alto. Igualmente, ante un mercado en decrecimiento y una tasa de interés alto, el brazo uno reportará grandes recompensas y el resto de brazos tendrán un remordimiento significativo. Ahora bien, la cuestión que se debe plantear es qué brazo usar para todos los demás contextos, en los que no haya una tasa de interés excesivamente alto ni un crecimiento de mercado exagerado. Pues ese es el corazón de este problema, la elección de la mejor cartera respecto a un contexto. Una vez resaltada esta idea, debe ser abordada la función de recompensas.

El mecanismo de recompensas es el siguiente: se calcula una rentabilidad esperada para las acciones y otra para los bonos en función de las condiciones del mercado, contexto. Posteriormente, se obtiene la rentabilidad de la cartera, brazo, en función del porcentaje de bonos y de acciones, que suman uno. Es decir, con el brazo 4, se multiplica la rentabilidad de las acciones por el 75 % y la rentabilidad de los bonos por el 25 %. Así, se puede saber cuál es la rentabilidad total resultante de la distribución de bonos-acciones. Por último, se normalizará y creará una distribución normalizada usando una varianza asociada a la cartera, que se calcula de acuerdo al peso de las acciones y bonos en dicha cartera. Luego se obtendrá el valor medio usando dicha desviación y se coge un valor aleatoriamente. Dicho valor medio será normalizado entre 0 y 1 para ver la rentabilidad esperada respecto al brazo usado en un contexto dado.

Las recompensas se representan del siguiente modo: las recompensas están normalizadas entre 0 y 1. Se parte de un valor de 0.5 como recompensa. De este modo, una recompensa de 0.5 significa que ni se ha ganado ni perdido dinero. Si hay una recompensa menor de 0.5 reflejará que se ha perdido dinero con la cartera. Este suceso se podrá repetir en numerosas ocasiones ya que hay situaciones en las que es muy fácil perder dinero. Por el contrario, si se da una recompensa mayor de 0.5, se habría ganado una rentabilidad con la cartera elegida. Si se obtiene una recompensa de 1, la inversión se habrá multiplicado por dos, mientras que si se llega a 0, se habría perdido la inversión completamente.

Una vez definido el contexto y los brazos, hay que explicar el funcionamiento de este bandido contextual. Este sistema recomendador elige una cartera para cada inversor, y lo hace para  $T$  inversores, es decir, durante  $T$  rondas. Cada inversor tiene un contexto diferente, generado aleatoriamente, un mercado que varía desde un 10 % a un -10 % y una tasa de interés de 0 hasta 8 %. Estos  $T$  inversores se generan aleatoriamente entre estos rangos y en este caso, se utilizan 10000 inversores para ver el desarrollo real del bandido contextual recomendando carteras y obteniendo recompensas para distintos inversores.

Gracias a que se cumple la condición de Lipschitz, se pueden discretizar los contextos en un espacio contextual, o mapa, para este bandido contextual, donde las coordenadas  $y$  son el crecimiento del mercado y las  $x$  la tasa de interés. Habrá un espacio contextual dividido en cuadrículas, donde hay diez filas y diez columnas en función de estas dos variables. Usando esta discretización se reparte el espacio de contextos en 100 contextos discretizados y para cada uno hay una instancia del algoritmo UCB1, o también llamado UCB, de manera que dicha instancia se encarga de explorar y explotar una única región del espacio contextual. De esta manera, se utiliza la discretización de una forma muy visual ya que es posible imaginarse un mapa dividido en cien porciones; también sirve para poder plasmar un gráfico con los resultados para cada región.

Es fundamental señalar cómo aprenderá cada una de las instancias del algoritmo UCB. Hay que tener en cuenta que cada una de las instancias únicamente se utiliza en uno de los 100 contextos discretizados, pero no se usa en ningún otro contexto discretizado. En cada ronda una instancia decide qué brazo, o cartera, recomendar y sabe la recompensa que ha obtenido para ese brazo. Consecuentemente, cada instancia del UCB solo aprenderá cuando sea llamada. Una instancia es llamada cuando se de un contexto en la ronda  $t$  que encaje en su contexto discretizado, esta instancia del UCB hará una recomendación para este contexto y aprenderá de la recompensa obtenida, pero el resto de instancias no sabrán ni el brazo que ha elegido ni su recompensa. Por ello, debe haber el suficiente número de rondas para que cada instancia aprenda de sus acciones y elabore su

estrategia de exploración-explotación de acuerdo a lo que ha aprendido con sus decisiones anteriores.

Con el uso de estos bandidos lipschitzianos se estudia un caso de uso distinto y se saca partido a discretizar con un número alto de contextos, 100 en este caso. Además, será posible representar los posibles contextos en un mapa, donde cada inversor será tratado por una instancia de UCB. Dicha instancia conoce a la perfección los valores en los que se encuentra este inversor, tanto interés como mercado y así conoce la exploración y explotación en esa zona entre brazos, y aprende a explotar el brazo con el respectivo porcentaje de acciones y bonos necesarios. Por esto, este sistema recomendador aplica esta condición para aprovechar los contextos.

Al realizar esta implementación, se observa que el algoritmo ofrece muy buenos rendimientos y aprende a usar el brazo necesario para cada contexto mediante las instancias de UCB (que hay una para cada contexto discretizado). Además, se puede ver que los bonos son menos variables en la fórmula calculada y por ello, en las zonas de interés alto se ven muy buenos rendimientos. Este bandido contextual rinde bastante bien según los resultados obtenidos.

Con la discretización y el uso de bandidos contextuales lipschitzianos, el algoritmo puede explotar cada zona del contexto discretizado con una instancia distinta, que aprenderá más rápidamente a operar en un contexto determinado, con unos valores de tasa de interés y crecimiento del mercado concretos. Estas instancias sabrán cuál es el mejor brazo a aplicar en su propia región y esto permite optimizar un problema gracias a la discretización.

Tras haber expuesto los pilares de este sistema, se mostrarán los resultados obtenidos, en dos gráficas que son fruto de la implementación realizada en Python. Como se puede apreciar, los resultados son positivos y el algoritmo es capaz de no perder dinero mediante la elección del mejor brazo según el contexto, y también mejora con el tiempo y se contempla esta curva de aprendizaje y su mejoría. Ahora, se observarán los resultados obtenidos por medio de unos gráficos que ilustran cómo se desenvuelve el bandido.

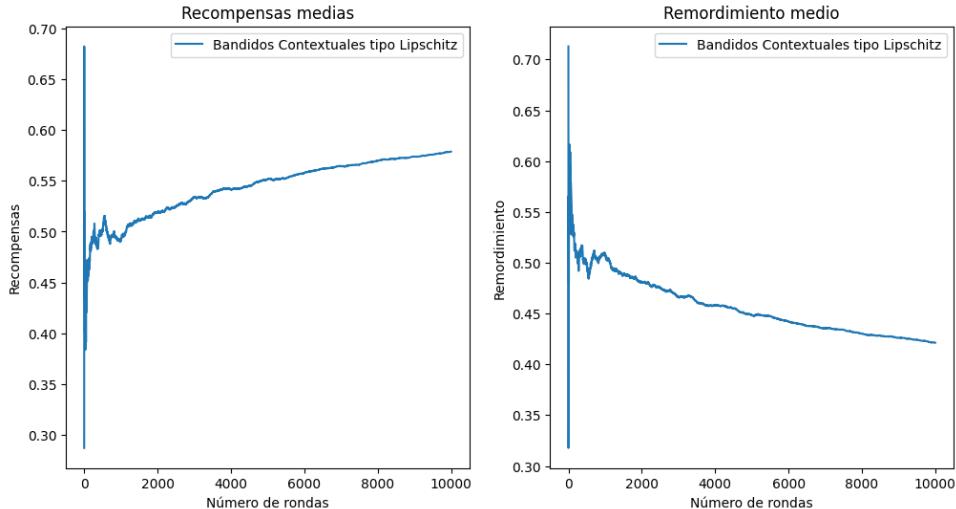


Figura 8: Recompensas medias y remordimientos medios con bandidos contextuales lipschitzianos.

En estos gráficos aparecen las recompensas promedias obtenidas a lo largo de 10000 rondas, y, en contraposición, también se ve el remordimiento que es lo que se deja de ganar. Cada una de las rondas representa a un inversor, y se observa un contexto diferente. Tal y como se ha detallado anteriormente, las recompensas reflejan que el algoritmo es capaz de mantener el dinero si la recompensa es igual a 0.5. Por otro lado, si la recompensa es superior a 0.5, el algoritmo está siendo capaz de obtener una rentabilidad eligiendo una buena cartera. En este caso, se ve que el algoritmo empieza en las primeras rondas con una variación muy alta de recompensas y ya a partir de la ronda 2000, es capaz de elegir los mejores brazos para siempre obtener una rentabilidad (positiva) de la cartera elegida. De hecho, solo al principio obtiene una recompensa media negativa y perdería dinero invertido. Esto significa que conforme va aprendiendo, es capaz de discernir qué cartera usar según el contexto para obtener beneficios.

La gráfica 8 muestra para una determinada ronda  $t$ , la recompensa media para esta ronda. Así se puede ver cómo varía la recompensa media obtenida. También se expone el remordimiento medio, que permite observar

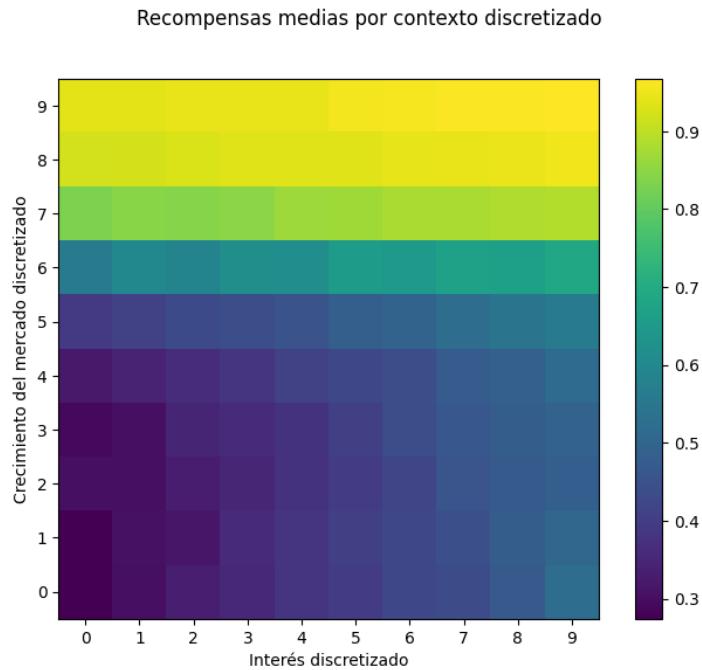


Figura 9: Mapa de calor para cada contexto discretizado.

si el algoritmo va obteniendo mejores recompensas conforme pasa el tiempo.

Se puede apreciar la mejora continua en el algoritmo y que aprende a lo largo de las rondas, y siempre proporciona recompensas positivas, esto es mayores de 0.5 a partir de la ronda 2000. Se observa lo mismo en el remordimiento que va disminuyendo conforme avanza la partida. Por lo tanto, es posible deducir que el algoritmo que ha sido desarrollado aprende a usar adecuadamente sus brazos y es capaz de mantener siempre la rentabilidad por encima del 0.5 para no perder dinero, y mejora con el tiempo. Esto es muy importante, ya que el algoritmo opera en situaciones muy negativas como un decrecimiento del mercado de  $-7\%$ , y en esta situación lo más probable es que con la mayoría de brazos el bandido pierda dinero, de hecho, así se observa en las primeras rondas donde hay rentabilidades de 0.3, lo que supone que en esas rondas el algoritmo pierde aproximadamente la mitad de su inversión. Esto contrasta con su evolución y su aprendizaje que queda reflejado en las curvas de recompensas y remordimiento, el bandido es capaz de rendir en cualquier contexto por muy negativo que sea, y ofrece recompensas positivas, por lo que su actuación es brillante.

Esta segunda imagen 9 es muy interesante, ya que se observan las recompensas medias obtenidas en cada contexto discretizado. Además, permite al lector ver una imagen visual de cómo se representa el espacio contextual según el crecimiento del mercado y el interés. En este mapa se ven las 100 secciones, o contextos discretizados, cada uno de estos es tratado por un algoritmo  $UCB_S$  en función de la tasa de interés y el crecimiento del mercado.

Los resultados muestran lo ya expuesto a lo largo de esta sección. Con un crecimiento del mercado muy positivo, se pueden alcanzar recompensas muy altas debido a la volatilidad de las acciones, que, hay que recordar que pueden subir mucho de precio. Por el contrario, ante una situación de una alta tasa de interés, las recompensas simplemente se mantienen o se gana un pequeño porcentaje. Este gráfico ilustra muy bien la lógica de cómo funciona el mercado financiero y arroja luz acerca de las recompensas obtenidas por el algoritmo.

El uso de un mapa de calor es una alternativa idónea en este caso porque es posible entender este modelo de manera intuitiva. Además, la discretización de los contextos aparece representada en este mapa de calor con sus recompensas asociadas y se puede intuir cómo de positivo va a ser el mercado, o cuántas probabilidades habrá de obtener una recompensa positiva (ganancias) según la información contextual.

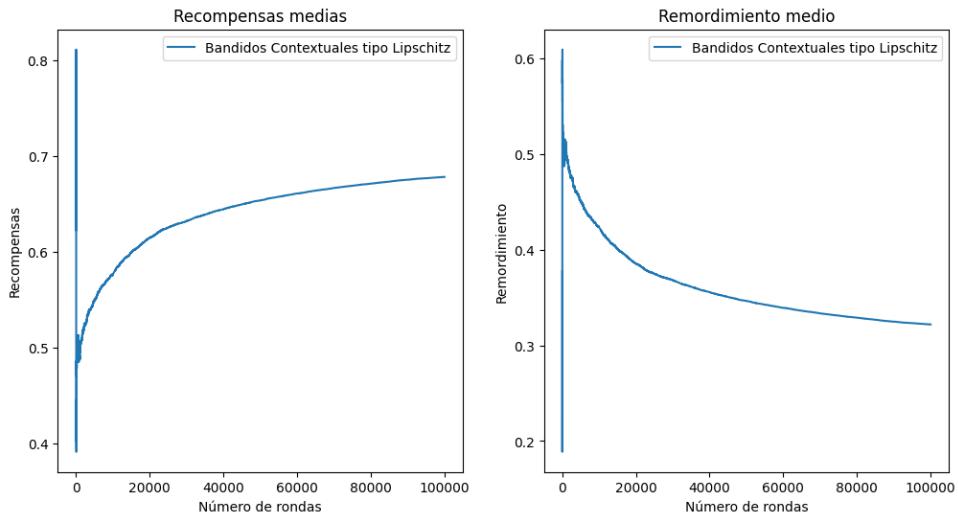


Figura 10: Evolución de las recompensas y remordimientos promedios si  $T$  es aumentado.

Por último, se pone el foco en la evolución de las recompensas con más rondas  $T$  para analizar mejor este algoritmo y su desempeño en el mercado.

Se puede ver en la figura 10 que se estanca el crecimiento de las recompensas promedias. A pesar de ello, sí que es posible apreciar que el crecimiento de las recompensas medias y también el decrecimiento del remordimiento promedio, mejora conforme van pasando las rondas, y confirma este aprendizaje por parte de el algoritmo de *bandidos contextuales tipo Lipschitz*.

Tras explicar todo esto, se puede concluir acertadamente que el algoritmo aprende a usar carteras con menos o más acciones en función del contexto con el objetivo de maximizar su recompensa cumpliendo en todo momento con la condición de Lipschitz. Cabe señalar que los resultados son muy interesantes porque permiten ver cómo para un gran número de contextos, gracias al cumplimiento de la condición de Lipschitz se puede conseguir un bandido contextual correcto que puede discretizar contextos. Esta implementación ha cumplido correctamente su función de exponer un problema que puede ser resuelto con los bandidos contextuales lipschitzianos y su proceso de discretización.

### 6.2.3. Demostración de Lipschitz para Bandidos Contextuales Lipschitzianos

A continuación, se presenta una explicación de la demostración de que la función de recompensa cumple con la condición de Lipschitz, utilizando el crecimiento del mercado y la tasa de interés como las variables del contexto. Se considerará que la variable  $x$  representa la tasa de interés y la variable  $y$  el crecimiento del mercado.

Debe recordarse que en este problema se da un contexto en cada ronda, que se compone de las variables  $x$  e  $y$ . Hay una recompensa que representa la rentabilidad de una cartera. Como ya se ha profundizado en cada detalle de esta implementación, ahora lo importante es demostrar que se cumple la condición de Lipschitz en este caso. En el caso de uso del mercado, se presupone que las recompensas vienen determinadas por la tasa de interés y el crecimiento del mercado. En este escenario de mercado, se asume que las recompensas están intrínsecamente ligadas a la tasa de interés y al crecimiento del mercado. Para simplificar la explicación, se postula que ambas variables, ponderadas por dos coeficientes distintos ( $a$  y  $c$ ), que podrían representar la rentabilidad de una cartera. Dicha función de recompensas es la siguiente:

$$R(x, y) = a * x + c * y,$$

En este caso, se utiliza la expresión  $R(x, y)$  para denotar las recompensas. De este modo queda latente que las recompensas vienen determinadas por estas dos variables pertenecientes al contexto. Para demostrar que esta función cumple con la condición de Lipschitz, debe demostrarse que existe una constante  $L$  tal que:

$$|R(x_i, y_i) - R(x_j, y_j)| \leq L * \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2},$$

para todos los contextos  $(x_i, y_i)$  y  $(x_j, y_j)$  donde  $x$  representa el crecimiento del mercado e  $y$  la tasa de interés. Analizando la diferencia entre las recompensas de los contextos se obtiene lo siguiente:

$$|R(x_i, y_i) - R(x_j, y_j)| = |a * (x_i - x_j) + c * (y_i - y_j)|$$

Aplicando la desigualdad de Cauchy-Schwarz de acuerdo a [10]:

$$|a * (x_i - x_j) + c * (y_i - y_j)| \leq \sqrt{(a^2 + c^2) * ((x_i - x_j)^2 + (y_i - y_j)^2)}$$

Así, se puede encontrar una constante  $L$  que cumpla con la condición de Lipschitz:

$$L = \sqrt{a^2 + c^2}$$

Finalmente, se observa lo siguiente:

$$|a * (x_i - x_j) + c * (y_i - y_j)| \leq L * \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Esta desigualdad se cumple para todos los contextos  $(x_i, y_i)$  y  $(x_j, y_j)$ , lo que demuestra que la función de recompensa cumple la condición de Lipschitz con constante  $L = \sqrt{a^2 + c^2}$ .

Cabe señalar que ha sido utilizada la desigualdad de Cauchy-Schwarz que establece que, para dos vectores  $\vec{u}$  y  $\vec{v}$  en un espacio vectorial con producto interno, se cumple que:  $(\vec{u} \cdot \vec{v})^2 \leq (\vec{u} \cdot \vec{u}) * (\vec{v} \cdot \vec{v})$ .

Esta desigualdad es vital para que se pueda demostrar la condición de Lipschitz. La demostración de la desigualdad de Cauchy-Schwarz se puede encontrar en el libro [10].

### 6.3. Bandidos Contextuales con Clase Política

Tras haber desarrollado ya múltiples tipos de bandidos contextuales, en este apartado se profundiza en unos bandidos muy especiales: los bandidos contextuales con *clase política*. Este es uno de los apartados más trascendentales del proyecto debido a la dificultad de los algoritmos empleados y a la complejidad de las implementaciones que se han llevado a cabo.

Antes de comenzar se define qué es una política. Una política es un asociación de contextos a acciones, es decir, es una estrategia que elige brazos en función del contexto con el objetivo de maximizar la recompensa. En este problema de bandidos contextuales hay varias políticas pertenecientes a una clase o conjunto de políticas  $\Pi$  dado, y se trata, de nuevo, de encontrar un correcto equilibrio entre la exploración y la explotación de las políticas con el objetivo de maximizar las recompensas. Es importante entender que el algoritmo solo podrá elegir entre las distintas políticas, siendo estas las que, según el contexto, seleccionen el brazo. Una ventaja considerable de este enfoque es que permite conectar el problema de los bandidos multi-brazo con otros algoritmos más complejos de aprendizaje automático, y esto proporciona el marco adecuado para desarrollar diferentes algoritmos que puedan poner a prueba sistemas muy elaborados.

Si se asume que cada contexto  $x$  es obtenido independientemente de una distribución de contextos  $X$ , para una política  $\pi$  determinada, la recompensa esperada se puede definir de la siguiente manera:

$$\mu(\pi) = E_{x \in X}[\mu(\pi(x))|x]$$

Esta recompensa estimada, también conocida como el *valor de la política*, permite comparar una política con otras, dando lugar a la política óptima que devuelve la recompensa media más alta, definida como:

$$\pi^* := \arg \max_{\pi \in \Pi} \mu(\pi)$$

Gracias al uso de esta medida, dado un conjunto de políticas  $\Pi$ , se puede definir el remordimiento de un algoritmo que trate de solucionar este problema como:

$$R_\Pi(T) = \mu(\pi^*)T - \sum_{t=1}^T \mu(\pi_t)$$

Es decir, el remordimiento alcanzado en el horizonte temporal  $T$  se calcula mediante la diferencia entre el acumulado de la recompensa media de la política óptima y el acumulado de la recompensa media obtenida por la política escogida por el algoritmo en cada ronda. Por tanto, si el algoritmo elige siempre la mejor política perteneciente a la clase  $\Pi$ , el remordimiento alcanzado será 0, ya que la diferencia entre las recompensas de la mejor política y las recompensas obtenidas por el algoritmo será 0. Por lo tanto, para esta sección es fundamental encontrar un algoritmo que sea capaz de identificar la mejor política, dentro de un conjunto de políticas dado, de manera que pueda minimizar el remordimiento y, en consecuencia, mejorar el rendimiento general de la solución.

### 6.3.1. Algoritmo Exp4 con Políticas en lugar de Expertos

La solución perfecta para este problema es el algoritmo Exp4, que ya ha sido explicado anteriormente en el apartado de bandidos antagonistas 5. Para adaptar este algoritmo al planteamiento de esta sección, se tomará en cuenta que las políticas juegan el mismo papel que los expertos presentes en la versión original. Consecuentemente, se considera que todas las políticas son como expertos en el sentido de que no aprenden. Es importante recordar que el algoritmo Exp4 elige entre las políticas dadas, y son estas políticas las encargadas de seleccionar los brazos según el contexto de cada ronda. De esta forma, tal y como se ha detallado anteriormente, se pueden desarrollar políticas muy interesantes, también fuera del ámbito de los bandidos multi-brazo, que se basen en algoritmos de aprendizaje automático u otra índole, favoreciendo un desarrollo más exhaustivo de la implementación.

Se define formalmente la adecuación del algoritmo Exp4 a la metodología de los bandidos contextuales con clase política:

---

#### Algoritmo 6.10 Algoritmo Exp4 con asesoramiento de políticas como expertos $(\Pi, \varepsilon, \gamma)$

---

**Requiere:** Conjunto  $\Pi$  de políticas, parámetro  $\varepsilon \in (0, \frac{1}{2})$  para Hedge, parámetro de exploración  $\gamma \in [0, \frac{1}{2})$ .

- 1: **for** cada ronda  $t$  **do**
- 2:    Se llama a Hedge para recibir la distribución de probabilidad  $p_t$  sobre  $\Pi$ .
- 3:    Se extrae una política  $\pi_t$  de  $p_t$ .
- 4:    Regla de selección: con probabilidad  $1 - \gamma$  seguir la política  $\pi_t$ ; de lo contrario, elegir un brazo  $a_t$  uniformemente al azar.
- 5:    Se observa el coste  $c_t(a_t)$  del brazo elegido.
- 6:    Se definen los “costes falsos” para todos las políticas  $\pi$ :

$$\hat{c}_t(e) = \begin{cases} \frac{c_t(a_t)}{\Pr[a_t=a_{t,\pi}|\vec{p}_t]} & \text{si } a_t = a_{t,\pi}, \\ 0 & \text{en caso contrario.} \end{cases}$$

- 7:    Se devuelven los “costes falsos”  $\hat{c}(\cdot)$  a Hedge.
  - 8: **end for**
- 

Hay que recordar que Exp4 debe actualizar los pesos de las políticas conforme avanzan las rondas de acuerdo a las recompensas observadas, por lo que delega este trabajo al algoritmo Hedge. Dicho algoritmo premiará a los expertos que mejor recompensas obtengan aumentando sus pesos para que sean elegidos con mayor probabilidad, de manera que al final de la ejecución de la solución lo más probable es que se elijan las mejores políticas, desestimando las peores.

Para este algoritmo son cruciales dos hiperparámetros: la tasa de aprendizaje y la tasa de exploración. La elección de sus valores dependerá de las características de la solución implementada para cada problema. En

la sección 8 se analizará, para cada tasa, qué valor es más conveniente aplicar con el objetivo de maximizar el rendimiento de la implementación. A continuación, se profundiza en estos hiperparámetros:

- Tasa de aprendizaje  $\varepsilon$ : permite establecer la velocidad a la que el algoritmo Exp4 aprende cuáles son las mejores políticas. Por lo tanto, una tasa de aprendizaje alta provocará que se castiguen mucho los costes fingidos altos y perjudicará gravemente a las políticas que en las primeras rondas son malas pero a la larga son las mejores. Esto se debe a que, como al principio han devuelto malas recompensas, Hedge les asigna rápidamente un peso tan bajo que disminuirá casi por completo sus probabilidades de ser escogidas, y el algoritmo habrá perdido la oportunidad de comprobar su potencial.

Por otro lado, una tasa de aprendizaje baja puede ser una mala opción también. Esto es debido a que una tasa de aprendizaje baja hará que el algoritmo Exp4 necesite muchas rondas en discernir cuáles son las mejores políticas. Durante estas rondas de aprendizaje, el algoritmo elegirá indistintamente entre políticas buenas y malas, por lo que el remordimiento será muy alto durante esta parte de la ejecución hasta que Exp4 consiga aprender cuáles son las mejores políticas.

Este hiperparámetro es uno de los aspectos más importantes a regular en el algoritmo ya que el remordimiento depende en gran medida de su ajuste.

- Tasa de exploración  $\gamma$ : este hiperparámetro será usado para explorar aleatoriamente sin tener en cuenta la recomendación de las políticas. Si se establece este parámetro a 0, nunca se explora aleatoriamente y siempre se sigue la recomendación de las políticas.

En cambio, si hay una tasa de exploración elevada se elegirá con una alta probabilidad un brazo aleatoriamente, en vez de elegir los brazos recomendados por las políticas. La tasa de exploración sirve de gran utilidad cuando existen ciertas políticas que se centran en explotar ciertos brazos, dejando de lado otros. Una gran tasa de exploración garantiza que Exp4 explorará y examinará la totalidad de los brazos disponibles. Pero también puede implicar que explote menos de lo que debería el conocimiento adquirido.

En cuanto al algoritmo Hedge, cabe destacar que para este escenario su funcionalidad cambia con respecto a lo estudiado en la sección de bandidos antagonistas:

---

#### **Algoritmo 6.11** Algoritmo Hedge para políticas $(\pi, \varepsilon, \hat{c}_t)$

---

**Requiere:**  $\varepsilon \in (0, \frac{1}{2})$ , la primera vez que se llama a Hedge los pesos son  $w_1(\pi) = 1$  para cada política  $\Pi$ .

- 1: **for** cada política  $\pi$  **do**
  - 2:    Se actualiza el peso:  $w_{t+1}(\pi) = w_t(\pi) \cdot (1 - \varepsilon)^{\hat{c}_t(\pi)}$ .
  - 3: **end for**
  - 4: Se calcula  $p_t(\pi) = \frac{w_t(\pi)}{\sum_{\pi'=1}^K w_t(\pi')}$  para cada política  $\Pi$ .
  - 5: Se devuelve  $\vec{p}_t$
- 

Como se puede observar, el planteamiento del algoritmo Hedge en este caso no incluye la elección de políticas sino que se limita exclusivamente a la actualización de los pesos y, consecuentemente, a la actualización de la distribución de probabilidad de elección de cada política.

En las secciones 8 y 9 se plantean un par de problemas prácticos que ejemplifican el problema de los bandidos contextuales con clase política estudiado en esta sección.

## 7. Bandidos Contextuales en Juegos Contextuales

En esta sección se presenta una aplicación práctica de los bandidos contextuales. Concretamente, se expone el desarrollo de los bandidos contextuales en un juego con características especiales. Este juego es un caso de bandidos contextuales basado en el artículo [12]. Este escenario corresponde a un juego contextual, cuya explicación se encuentra detallada en la subsección 7.2.

Gracias al uso de los bandidos multi-brazo contextuales, es posible aprovechar la repetición de escenarios o contextos a lo largo de las rondas del juego para desarrollar algoritmos que aprendan de manera eficaz y obtengan mejores resultados que los que no consideren el contexto.

Se plantean dos objetivos iniciales. El primero es desarrollar una adaptación en C++ que imite la implementación del juego en Python. Dicha implementación está realizada por los autores del artículo y que dejaron reflejada en [18]. Con este primer objetivo se pretende un acercamiento con las implementaciones de los algoritmos en un caso real, y así después poder llevar a cabo implementaciones propias al final de este trabajo.

El segundo y principal objetivo es estudiar en profundidad el comportamiento de estos algoritmos en la práctica para poder explicarlos razonadamente. Esto será de gran utilidad ya que se podrá observar las diferencias entre los rendimientos de los algoritmos que tienen en cuenta el contexto y los que no lo hacen. Por lo tanto, este capítulo constituye un preludio antes de que el trabajo se centre en otras aplicaciones desarrolladas por los propios autores. Es decir, las siguientes secciones: 8 y 9.

Las conclusiones detalladas en la sección 7.5 proporcionan una discusión acerca de los resultados obtenidos a partir de la adaptación que se ha realizado, así como una evaluación de hasta qué punto se han alcanzado los dos objetivos propuestos.

### 7.1. Introducción del Sistema

En primer lugar, hay que exponer minuciosamente cómo es el sistema en el que actúan los algoritmos y sus características fundamentales. Se trata de un sistema de enrutamiento, compuesto por carreteras y nodos; específicamente, hay 76 carreteras y 24 nodos. En esta red, hay un número arbitrario de agentes, que en este caso son  $N = 528$ .

La topología de la red está en este enlace [13]. Se ha recreado un grafo con Python que muestra la topología de la red considerando la posición de los nodos y sus carreteras:

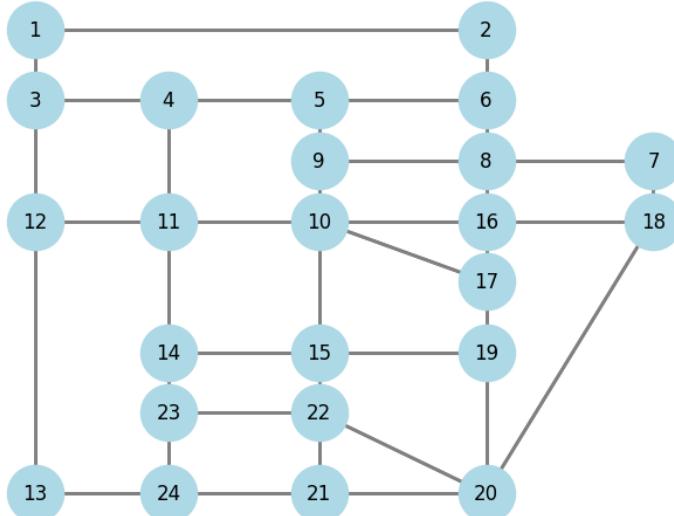


Figura 11: Ilustración de la red.

En esta red, cada uno de los 528 agentes tiene como objetivo mandar un número determinado de unidades,

$d_i$ , desde un nodo origen hasta un nodo destino. Por eso, un agente deberá pasar por distintas carreteras para llegar al nodo destino formando así un camino. Un agente tiene 5 posibles caminos,  $K$  brazos, para alcanzar su destino. De los cinco caminos, habrá algunos con los que tardará más y otros menos, dependiendo de la distancia y de la ocupación de las carreteras que necesita utilizar. Los agentes, o jugadores, se encuentran distribuidos en todos los nodos y en cada ronda  $t$  tienen que mandar un número preestablecido y constante de paquetes a su nodo destino, y deberán elegir entre los cinco caminos que tienen disponibles. Esta es la decisión que deberán tomar cada ronda e influirá en sus recompensas obtenidas. El tiempo que tarda el agente en transmitir sus paquetes será la recompensa. Por lo tanto, el agente pretenderá elegir los caminos rápidos para maximizar su recompensa. Los envíos de paquetes están relacionados con las recepciones, esto quiere decir que si un agente manda 5 paquetes desde el nodo 1 al nodo 17, en el nodo 17 hay otro agente que enviará 5 paquetes al nodo 1.

## 7.2. Juego Contextual

Estos agentes no actúan en la red de forma aleatoria. En cambio, se encuentran en constante interacción en un escenario que puede ser modelado como un juego repetido, una idea central en la teoría de juegos. Esta rama de las matemáticas se ocupa de los conflictos y la cooperación entre agentes inteligentes racionales. Dichos agentes, en su afán por maximizar su recompensas deben seleccionar un camino teniendo en cuenta las decisiones de los demás.

El hecho de que la elección de un agente afecta a todos los demás hace que este juego sea especialmente interesante. Si muchos jugadores utilizan las mismas carreteras, se produce congestión, tardarán más tiempo y sus recompensas se verán disminuidas. Por ello, los agentes siguen una estrategia determinada por un algoritmo que pretende reducir sus tiempos de viaje eligiendo el mejor camino teniendo en cuenta las decisiones de los demás agentes. Esto es debido a que las elecciones del resto de agentes afectarán a la congestión de la red.

En este juego de enrutamiento, los agentes están sujetos a muchos factores que cambian con el tiempo, como las capacidades de la red. Estos factores cambiantes determinan las reglas del juego en cada momento, y solo los agentes cuya estrategia esté determinada por un algoritmo que contempla el contexto podrán observar dichas circunstancias y adaptarse a ellas. Como resultado, pueden tomar decisiones mejor informadas. Por lo tanto, el sistema de enrutamiento es un ejemplo de un tipo de juegos que denominado *juegos contextuales*. Estos juegos contextuales permiten a los jugadores observar la información contextual, lo que les puede hacer lograr un mejor rendimiento.

En una simulación se selecciona uno de los posibles algoritmos: *algoritmo Hedge*, *algoritmo GPMW* o *algoritmo cGPMW*. El algoritmo seleccionado controla a un número arbitrario de agentes y actúan conforme a su lógica. Normalmente hay 20 agentes manipulados por el algoritmo. El resto de los 528 agentes que no están controlados tienen una dinámica muy diferente. Mientras que los agentes controlados tienen 5 posibles caminos para llegar al destino, que son los más cortos, los agentes que no están controlados tienen un único camino. El resto de agentes no necesitan tomar ninguna decisión porque siempre optarán por el mismo camino.

El número de agentes controlados es uno de los parámetros que se establece al principio de la simulación. Sin embargo, los agentes controlados no son los mismos de una simulación a otra. En cada una, se obtienen índices de los agentes controlados aleatoriamente dentro del número de agentes totales,  $N$ .

## 7.3. La Red

Tras haber descrito el funcionamiento de este sistema, este apartado se centrará en cómo es la red en la que operan los agentes, o jugadores de la partida. Como ya se ha mencionado, esta red tiene 24 nodos, que guardan además información de su posición en el eje de abscisas y en el eje de ordenadas. En cuanto a las carreteras, en esta red hay 76 carreteras y cada una de ellas, entre un nodo origen y destino, tiene una capacidad, una longitud y un tiempo de viaje. Todo esto afecta directamente a los agentes al enviar paquetes y es fundamental señalar que la capacidad de una carretera viene dada por el contexto  $z_t$ . A partir de ahora, será lo mismo referirse a la capacidad que al contexto.

El escenario en el que se encuentran los agentes no es estático, sino que está constantemente cambiando. De hecho, existe un conjunto predefinido de contextos diferentes, y en cada ronda se escoge uno de estos contextos de manera aleatoria. Este cambio en el contexto puede verse como una perturbación en las capacidades de las

carreteras, donde cada carretera ve su capacidad variar en cierto grado en función del contexto.

En este sistema, la red realiza dos funciones clave: el cálculo de los vectores de estrategia (los caminos de cada agente) y el cálculo del tiempo de viaje. Obtener los vectores de estrategia es un punto fundamental que se hace antes de iniciar la simulación. El cálculo de estos vectores se realiza sobre un grafo donde se calculan los caminos más cortos para las demandas esperadas. Las demandas esperadas en la red representan los paquetes que deben enviar todos los agentes, y para poder cumplir con el envío de dichos paquetes, se calculan los 5 caminos más cortos desde el nodo origen hasta el nodo destino. Ha sido usado el algoritmo Yen para cumplir esta función. Dado un grafo donde hay un nodo origen y un nodo destino, consiguiendo el camino más corto. En esta función de cálculo de los vectores de estrategia, se ha obtenido los  $k$ -caminos más cortos,  $k$  es 5, para los 528 agentes que reflejan las demandas esperadas en la red.

En este caso, el algoritmo Yen ha sido útil para encontrar una solución adecuada para encontrar los  $k$ -caminos más cortos. La forma en la que este algoritmo actúa es la siguiente: encuentra el camino más corto entre un nodo origen y un nodo destino y lo almacena como el primer camino posible. Después encuentra el segundo camino más corto entre el origen y destino. Posteriormente el tercero más corto... En definitiva, este algoritmo almacenará los caminos por orden de distancia entre un origen y un destino.

Por otra parte, la red también se encarga del cálculo del tiempo de viaje. Esto se realiza para un agente en concreto o para todos los agentes cuando es requerido al finalizar una ronda  $t$ . Cuando se calcula el tiempo de viaje para todos los jugadores se usa esta función gracias al vector de estrategias, que permite conocer el camino empleado por todos los jugadores, y usando el vector de estrategias se puede obtener el tiempo que tarda cada agente de acuerdo a dos aspectos: el camino que utiliza y las unidades de tiempo requeridas. Las unidades de tiempo que hacen falta dependen básicamente del uso de una fórmula matemática que toma el tiempo de viaje de las carreteras, las capacidades de las carreteras y las ocupaciones en esa ronda  $t$ .

El tiempo de un viaje representa la recompensa obtenida por un agente. En cada ronda el agente calculará el tiempo de su viaje asociado al brazo elegido y recibirá su recompensa. En este caso, la función de recompensas es antagonista indiferente. Esto es debido a que es dinámica y cambia a lo largo de las rondas, pero no variará según las acciones tomadas por el algoritmo. Las recompensas se modifican con la ocupación de las carreteras y las capacidades (contexto). Los contextos son obtenidos aleatoriamente de una distribución con todos los contextos posibles y en cada ronda se escoge uno. Dicho contexto no será seleccionado para perjudicar el algoritmo, pero sí evolucionará durante el tiempo por lo que las recompensas son indiferentes. Esto se detalla a fondo en el apartado 5.

Para finalizar, tras profundizar en esta sección es posible hacerse una idea de la importancia de la red en este sistema porque es la estructura base del juego donde se desarrollan los bandidos multi-brazo y cumple funciones imprescindibles que son vitales para las simulaciones. Toda esta información es calculada para el uso en la inicialización del juego o durante la propia simulación.

## 7.4. Algoritmos

El principal objetivo de este apartado es indagar en cada uno de los algoritmos desarrollados para entender su funcionamiento y poder ilustrar conceptos teóricos explicados en este trabajo con una implementación práctica detallada. Los algoritmos a tratar son los siguientes: el *algoritmo Hedge*, el algoritmo GPMW y el algoritmo cGPMW.

### 7.4.1. Algoritmo Hedge

En primer lugar, se comenzará con el algoritmo Hedge porque es el más sencillo de entender a pesar de su complejidad y sirve como base para explicar los otros. No obstante, este algoritmo es muy diferente a los otros dos porque tiene retroalimentación total. Antes de definir el algoritmo y detallarlo, deben ser mencionadas sus principales características.

Es un algoritmo que posee retroalimentación total, es decir, conoce las recompensas de cada uno de sus  $K = 5$  brazos en una ronda dada. Esta retroalimentación puede conseguirse con la función de cálculo de tiempos de viaje de la red, que le permite obtener el tiempo de viaje para cada uno de sus brazos. Para ello, tiene en cuenta las capacidades de las carreteras, es decir, el contexto. Su principal objetivo, al igual que el del resto de

algoritmos, es minimizar el tiempo de viaje en cada ronda mediante la mejora de toma de decisiones. Esto significa que tratará de elegir el camino que reduzca el tiempo de viaje y, consecuentemente, mejore las recompensas.

Debido a esto, al existir retroalimentación total, es un algoritmo que no se puede comparar directamente con los otros dos. Ni GPMW, ni cGPMW tienen retroalimentación total. El algoritmo Hedge conoce toda la información posible. Tener retroalimentación total no es comparable con conocer el contexto. Mientras que saber el contexto es una información que podría ser considerada útil, la retroalimentación total es equivalente a conocer toda la información de la red. A pesar de que el Hedge juega un único brazo en cada ronda, es como si jugará todos. Por esto, dicho algoritmo servirá de introducción y ejemplo pero a la hora de comparar algoritmos, el estudio se centrará en los dos siguientes.

Antes de abordar este algoritmo en profundidad, hay que señalar una premisa fundamental de los bandidos contextuales que sí cumple el cGPMW pero no el algoritmo actual. En un bandido contextual, el algoritmo observa el contexto  $x_t$  de la ronda antes de seleccionar un brazo. Esta es la clave que debe ser entendida y asimilada. Si el algoritmo tiene en cuenta el contexto observado antes de elegir una acción será un bandido contextual. Debido a este motivo, el algoritmo Hedge, no es un bandido contextual, ya que no usa el contexto (capacidades) para seleccionar un camino. En cambio, el algoritmo Hedge elige en función del tiempo, es decir, toma el camino que menos tiempo tarde porque le reportará mejores recompensas. Por otro lado, aunque Hedge no tenga en cuenta el contexto para decidir un camino, sí lo utiliza (además de las ocupaciones de las carreteras y el vector de estrategias) para calcular el tiempo de viaje para cada brazo, y obtener la retroalimentación total.

Una vez que han sido mencionados los aspectos más relevantes, se expone el algoritmo para despues ahondar en sus características exhaustivamente.

---

#### **Algoritmo 7.12 Algoritmo Hedge en Juego Contextual.**

---

- 1: Al inicializar: Se establece una tasa de aprendizaje  $\gamma$ , se inicializan los *pesos* a 1, se calculan la recompensa máxima y la recompensa mínima. Se obtien los  $K$  caminos que puede tomar.
  - 2: **for** cada ronda  $t = 1, 2, \dots$  **do**
  - 3:   Se juega un brazo  $a$  elegido aleatoriamente en función del vector *pesos*.
  - 4:   Se actualiza el vector *pesos*:  $\forall a \in A : pesos_a \leftarrow pesos_a \cdot e^{\gamma \cdot (-l_a)}$ .
  - 5: **end for**
- 

Como se puede observar, antes de la simulación el algoritmo Hedge se encarga de inicializar los parámetros necesarios. Primero se inicializa a uno el valor de los pesos de todos los brazos. La variable *pesos* representa la importancia de cada brazo. La probabilidad de dicho brazo se calcula dividiendo su peso entre la suma de todos los pesos. Un peso mayor se traducirá en más posibilidades de que ese brazo sea seleccionado. Como todos parten con un peso de 1, todos tienen a priori las mismas probabilidades de ser elegidos. Una probabilidad alta de un brazo A supone que se elegirá seguramente antes que un brazo B. Estas probabilidades cambian a lo largo de la simulación por medio de la función actualizar.

Después, establece un valor a  $\gamma$ , que representa la tasa de aprendizaje y es arbitraria. Su oficio es actualizar en menor o mayor cantidad los pesos tras haber obtenido una recompensa en la actualización. Como se encuentra en una exponencial en negativo, se puede razonar que una  $\gamma$  alto influirá menos en el cambio de los pesos mientras que una  $\gamma$  de un valor menor influirá más en los pesos. Por defecto, toma el siguiente valor para que haya un equilibrio en esta variable entre el número de brazos y el número de rondas:

$$\gamma = \sqrt{\frac{8 \log(K)}{T}}$$

Por último, se obtienen los valores de las recompensas máximas y mínimas que se calculan antes de la simulación, estos valores permitirán escalar las recompensas en la actualización y representan el valor máximo y el mínimo que puede conseguir un agente con cualquier contexto y cualquier brazo. Se calculan haciendo múltiples simulaciones de aproximación con todos los contextos y brazos para obtener el máximo y el mínimo. Dichas aproximaciones se realizan antes de la simulación real, y ayudan a conseguir estos valores. También se definen los brazos o caminos que puede tomar este agente, y en cada ronda elegirá entre uno de estos brazos.

Tras desarrollar esto, de este algoritmo, aparte de la inicialización, deben diferenciarse dos funciones claves que realiza: la elección de un brazo y la actualización. La selección del brazo es un proceso muy sencillo y lógico

de acuerdo a la variable *pesos* y su probabilidad, siendo:

$$\text{probabilidad}_a = \frac{\text{peso}_a}{\sum_{\forall a} \text{peso}_a}$$

Se elige un brazo aleatoriamente en función de los pesos de cada uno de los brazos. Esto se realiza al principio de la ronda y, tras jugar un brazo, se obtiene una recompensa. A raíz de esta recompensa, tiene lugar el segundo punto vital que es la actualización.

La actualización es el eje de este algoritmo ya que se encarga de actualizar el vector *pesos* y afecta directamente en la selección de brazos. Básicamente lo que hace la actualización es obtener una recompensa para cada brazo y recalcular el vector *pesos*. Esto se hará de la siguiente forma: A través de la función de la red se calcula el tiempo esperado para cada uno de los brazos. Hay que recordar que para calcular el tiempo esperado de cada brazo se usan las capacidades, las ocupaciones totales y el vector de estrategias. Posteriormente, se usan las recompensas mínimas y máximas para que la recompensa calculada sea menor que la recompensa máxima y mayor que la recompensa mínima, luego se normaliza para que la recompensa esté entre 0 y 1, y después se calcula las pérdidas, para cada brazo  $l_a$ , que son igual a uno menos la recompensa, es decir, las pérdidas son lo que se deja de ganar.

Para actualizar los pesos con las pérdidas de cada brazo se aplicará la siguiente fórmula usando  $\gamma$  como se ha mencionado, y también las pérdidas conseguidas, se calculan los pesos para cada brazo:

$$\forall a \in A : \text{pesos}_a \leftarrow \text{pesos}_a \cdot e^{\gamma \cdot (-l_a)}.$$

La variable  $l_a$  refleja las pérdidas del brazo  $a$  en esa ronda  $t$ . Mediante esta fórmula es posible observar cómo el peso de los brazos que obtengan peores recompensas disminuirá, mientras que aquellos brazos que obtengan buenas recompensas mantendrán valores altos y, por consiguiente, serán más veces elegidos en rondas futuras.

El funcionamiento de los pesos se puede explicar con un ejemplo. Hay que imaginar que existen 5 brazos que representan las mejores rutas o caminos para un agente. Pasada una ronda  $t$ , el agente jugará el brazo uno, pero como hay retroalimentación total, sabrá cuánto ha tardado usando su camino y cuánto habría tardado usando los otros cuatro caminos. Si se supone que el camino que ha elegido ha sido el que más tiempo tarda, se actualizará el vector *pesos* de acuerdo a esta lógica. El brazo uno, que es el elegido, y el que ofrece peor recompensa ya que tarda más tiempo, disminuirá su peso. Esto provocará que el brazo uno tendrá un peso menor en la siguiente ronda  $t + 1$ , y habrá menos probabilidades de elegir este brazo. Por otra parte, el resto de brazos aumentarán su peso respecto al brazo uno.

Esta estrategia permite al algoritmo penalizar los caminos que sean lentos y tengan malas recompensas y premiar los caminos rápidos que las tengan buenas. De este modo, a lo largo de las rondas, tras repetir esta actualización de los pesos se acabarán teniendo pesos altos en los caminos que sean más rápidos mientras que los caminos más lentos tendrán pesos menores. Así, el algoritmo tenderá a elegir caminos con pesos altos, que le retornarán mejores recompensas a la larga. Y usando esta estrategia en el largo plazo el algoritmo mejorará sus recompensas.

No obstante, es imprescindible considerar que dicho algoritmo Hedge no considera el contexto, porque trata de elegir los mejores caminos pero no discierne si en una ronda hay mucha o poca congestión en un camino. Simplemente el algoritmo recibe, con retroalimentación total, el tiempo que tarda en usar cada camino y según los tiempos de viaje, que varían cada ronda según la ocupación (contexto), se actualizará. Es decir, este algoritmo realizará actualizaciones a su vector *pesos* en función del tiempo de viaje de cada camino, y la recompensa asociada a este tiempo de viaje, y sin diferenciar entre contextos distintos cada ronda, el algoritmo mantendrá con altos pesos los caminos más rápido a lo largo de las rondas. Es decir, los caminos que mejores recompensas medias den serán más elegidos.

A pesar de que este algoritmo no considera el contexto, es el algoritmo que mejores rendimientos ofrece ya que tiene retroalimentación total. Al contrario que los otros algoritmos, este sabe en realidad cuáles son las pérdidas y recompensas para todos los brazos como si los hubiese elegido. Esta es una diferenciación muy importante ya que aunque no tiene en cuenta el contexto, se podría decir que este algoritmo juega con otras reglas y alcanza menores remordimientos que el resto.

Debido a esto, el análisis se centra en la comparativa entre el GPMW y el cGPMW para ver de qué manera puede influir la información contextual. Por otro lado, el algoritmo Hedge es muy importante ya que además de enriquecer el trabajo y se muestra un bandido multi-brazo con retroalimentación total en un caso real, también

se explica como punto de partida para familiarizarnos con el funcionamiento de estos algoritmos en la red. También hay que señalar que es el algoritmo usado por defecto por los jugadores que no son controlados por el algoritmo elegido (GPMW o cGPMW) en la simulación, por lo que es imprescindible para poder desarrollar adecuadamente este programa.

#### 7.4.2. Algoritmo GPMW

La investigación ahora se centra en el algoritmo GPMW. A pesar de no ser un algoritmo estrictamente contextual, sí que considera las capacidades totales de las carreteras. Por esto, se puede afirmar que estudiar dicho algoritmo va a aportar mucho valor a este trabajo y va a enriquecer los conocimientos obtenidos acerca del desarrollo de estos bandidos. El algoritmo GPMW va a ser comparado con el cGPMW que sí es estrictamente contextual.

Una vez que ha sido aclarado este aspecto, se puede desarrollar exhaustivamente el funcionamiento de este bandido. A diferencia del algoritmo Hedge, este algoritmo no tiene retroalimentación total, solo conoce el coste de su brazo elegido. Sin embargo, el algoritmo GPMW simula retroalimentación total con costes falsos. En el primer algoritmo, Hedge, se calculaba con la red la recompensa de cada brazo dado el contexto de ese momento. En el algoritmo de este apartado la retroalimentación total es simulada con predicciones, no con resultados reales como lo hacía el algoritmo Hedge. El aspecto de las predicciones es vital para entender este algoritmo.

Este algoritmo realiza predicciones para simular las recompensas que esperaría obtener para todos los brazos en la ronda en la que se encuentre. Gracias a esta técnica, el algoritmo actualizará los pesos que otorga a cada brazo según lo buenas que sean sus predicciones. Hay un peso por cada brazo y los brazos con mejores rendimientos, o predicciones ahora, tendrán pesos mayores por lo que tenderán a ser elegidos. El funcionamiento de los pesos es similar al apartado anterior excepto por una cuestión.

Una diferencia a tener en cuenta respecto al algoritmo Hedge es la forma de calcular los pesos. En cada ronda el algoritmo actualiza los pesos de acuerdo a la tasa de aprendizaje y lo hará con las pérdidas acumuladas.

$$\forall a \in A : pesos_a \leftarrow pesos_a \cdot e^{\gamma \cdot (-L_a)}.$$

En este caso, está utilizando las pérdidas acumuladas en el exponente que multiplica los pesos. Dichas pérdidas acumuladas las representa como  $L_a$ . Anteriormente, usaba las pérdidas de la ronda, pero este algoritmo tiene en cuenta las pérdidas de todas las rondas anteriores.

Esto se debe a que en Hedge los pesos se actualizan en función de la pérdida de la ronda actual porque se supone que la distribución de probabilidad de las acciones óptimas es relativamente estable de una ronda a otra. Por eso, con las pérdidas de la ronda actual se considera que es suficiente. Por otra parte, el algoritmo GPMW considera la pérdida acumulada teniendo en cuenta las rondas anteriores porque su estrategia no depende solo de las observaciones de pérdidas de la ronda actual, sino también las observaciones previas, ya que el GPMW parte de la premisa de que la distribución de probabilidad de las acciones óptimas puede cambiar a lo largo del tiempo.

Una vez que se ha mencionado estos aspectos se mostrará el esquema que sigue este algoritmo, y después hay una explicación detallada de cada paso:

---

#### **Algoritmo 7.13 Algoritmo GPMW en Juego Contextual.**

---

- 1: Al inicializar: Se establece una tasa de aprendizaje  $\gamma$ , se inicializan los *pesos* a 1, se calculan la recompensa máxima y la recompensa mínima. Se obtienen los  $K$  caminos que puede tomar. Se inicializan el historial (ocupaciones totales y unidades enviadas) y el historial de recompensas.
  - 2: **for** cada ronda  $t = 1, 2, \dots$  **do**
  - 3:   Se juega un brazo  $a$  elegido aleatoriamente en función del vector *pesos*.
  - 4:   Se actualizan el historial y el historial de recompensas con el brazo  $a$  y su recompensa  $r$ .
  - 5:   Se simula la retroalimentación total:  $\forall a \in A : r_{\text{simulada}}(a) = f(\text{historiales}, \text{ocupaciones}_t)$
  - 6:   Se actualiza el vector *pesos*:  $\forall a \in A : pesos_a \leftarrow pesos_a \cdot e^{\gamma \cdot (-L_a)}$
  - 7: **end for**
- 

Lo primero que realiza este algoritmo es una inicialización muy similar a la del algoritmo anterior pero añadiendo nuevos atributos porque la metodología utilizada es más compleja. Se obtienen la tasa de aprendiza-

je,  $\gamma$ , las recompensas máximas y mínimas y los  $K$  caminos al igual que antes. También el algoritmo tendrá dos piezas cruciales: el historial y el historial de recompensas de rondas pasadas. Es decir, en una ronda  $t$ , tendrá los historiales hasta esa ronda de todas las anteriores. Ahora se verá qué valor tienen y para qué sirven estos atributos.

Por un lado, el historial de recompensas refleja la recompensa obtenida en cada una de las rondas anteriores. Por otro lado, el historial almacena dos vectores en cada ronda. Primero almacena la estrategia jugada por el brazo elegido  $a$  y los carreteras que usa ese brazo. También guarda las ocupaciones totales de las carreteras que necesita este camino, brazo, elegido. En definitiva, el historial contendrá la información de las carreteras jugadas por el jugador en una ronda, tanto la ocupación total de esa carretera, como las paquetes que el brazo envía por esas carreteras, de acuerdo a su camino escogido. Usando estos historiales, y las ocupaciones dada la ronda en la que se encuentre, podrá calcular las recompensas simuladas de la siguiente manera, tal y como se detalla en [12]:

- En primer lugar, el algoritmo actualiza los historiales: el historial de recompensas con la nueva recompensa de la ronda  $t$  y el historial que refleja las ocupaciones totales para las carreteras del camino  $a$  elegido y los paquetes enviados del camino  $a$  elegido.
- Una vez hecho esto, usa un modelo de *regresión gaussiana* y lo entrena para las rondas anteriores, siendo la variable  $X$  el historial y la variable independiente el historial de recompensas.
- Posteriormente calcula las otras ocupaciones, que son las ocupaciones totales menos las que requiere el jugador, para saber cuáles son los paquetes enviados por todos los demás jugadores, a excepción de el propio agente.
- Despues, este algoritmo obtiene para cada brazo los paquetes que tiene que enviar para cada carretera, y junto con otras ocupaciones, lo utiliza como  $X$  para predecir con la regresión gaussiana, que ya ja sido entrenada, y obtiene una predicción de la recompensa para cada brazo y la varianza de esta predicción.
- Con un valor de beta,  $\beta$ , de 0.5, calcula el límite superior de confianza con la predicción obtenida para cada brazo:  $UCB_a = \hat{\mu}_a + \beta_t \hat{\sigma}_a$ . La varianza la proporciona el modelo de regresión gaussiana.
- El algoritmo GPMW actualiza el vector *pesos* usando el vector *UCB* al igual que para el algoritmo Hedge. Las recompensas serán el vector *UCB* para todos los brazos, comprueba que están dentro de la recompensa máxima y recompensa mínima para después escalarlas. Calcula las pérdidas para cada brazo  $l_a$ , haciendo la diferencia entre uno y las recompensas escaladas. Actualiza con las pérdidas acumuladas que tenía ya de otras rondas:  $L_a = L_a + l_a$  para cada brazo  $a$ . Usa  $L_a$  y modifica el vector *pesos*:  $\forall a \in A : pesos_a \leftarrow pesos_a \cdot e^{\gamma \cdot (-L_a)}$ .

A continuación, se detallará qué es la regresión gaussiana y qué utilidad tiene en esta implementación. La regresión gaussiana es un método de aprendizaje automático que proporciona una forma de inferir una función desconocida a partir de datos de entrenamiento ruidosos. Es un modelo de regresión no paramétrico, lo que significa que puede adaptarse a datos de cualquier forma sin tener una forma predefinida. Por otro lado, en el contexto de la regresión gaussiana, un kernel (o función de covarianza) es una función que mide cuánto se parecen dos puntos. En otras palabras, define el grado de correlación o similitud entre dos puntos en el espacio de entrada. En la implementación original, usar el modelo de regresión gaussiana (con el kernel) permite explotar la similitud entre dos contextos ya que se mide la similitud entre los puntos. De este modo, un bandido que considere el contexto como el siguiente algoritmo, cGPMW, consigue alcanzar muy buenas recompensas al sacar partido de la regresión gaussiana para mismos contextos y captura muchas formas diferentes de relaciones entre datos de entrada y salida.

Respecto a dicho modelo de regresión gaussiana, cabe señalar que en Python se usa la librería GPy mientras que en la adaptación que ha sido implementada en C++, se usa “rvm regression trainer” de la biblioteca dlib. Esto es un modelo de regresión de vectores de soporte (RVM) que permite entrenar dicho modelo para que pueda realizar predicciones para obtener las recompensas simuladas. Para el modelo RVM no ha sido necesario utilizar el kernel, la función de covarianza porque el modelo de regresión de vectores utilizado no permite pasarle por parámetro un kernel.

En cualquier caso, este algoritmo GPMW, usa un modelo de aprendizaje automático, un modelo de regresión, para predecir las recompensas de todos los brazos y simular retroalimentación total. Por ello, la regresión juega un rol clave para simular las predicciones de cada brazo, ya que en función de estas predicciones se actualizan los pesos de cada brazo y, en consecuencia, se juegan más aquellos brazos que tengan mejores predicciones.

Esto se puede entender con un ejemplo. Si hay una situación en la que dada una ronda  $t$  se elige un brazo  $a$ , el algoritmo actualizaría los historiales para entrenar el modelo con toda la información hasta  $t$ , que se incluye también. El algoritmo haría predicciones para los 5 brazos que tiene, y si el modelo de regresión predice una muy buena recompensa para el brazo 2 frente al resto de brazos, el peso del brazo 2 aumentará frente a los demás brazos, y será más probable elegir el brazo 2 a partir de la siguiente ronda. Las predicciones definen cuáles serán los brazos más utilizados ya que modificarán los pesos de los brazos en cada ronda. Si las predicciones afectan mucho o poco para cambiar el peso de un brazo dependerá del  $\gamma$ , o tasa de aprendizaje, (al igual que el Hedge).

#### 7.4.3. Algoritmo cGPMW

Por último, se explica el algoritmo más complicado pero también más eficaz. Este es el único algoritmo contextual de los tres. Dicho algoritmo es una extensión o mejora del algoritmo GPMW, y hay que tener en cuenta que se encuentran muchas similitudes entre estos dos algoritmos. Además, el peso de los brazos se actualizará usando el vector *pesos* al igual que lo hacen los otros dos algoritmos. Por ello, este algoritmo trabajara con nomenclatura similar así como con conceptos desarrollados en las subsecciones anteriores del algoritmo Hedge y el algoritmo GPMW.

Para empezar con el algoritmo cGPMW, se comienza explicando porqué es el único bandido contextual. A diferencia de los otros dos, este algoritmo observa el contexto antes de tomar una decisión respecto a qué acción quiere jugar. Por ello, tiene una nueva estructura, observa el contexto y calcula una estrategia antes de seleccionar la acción. Esta función de calculo de estrategia le permite considerar el contexto para elegir el mejor brazo de acuerdo al contexto vigente. El cálculo de estrategia es algo similar a lo que hace el algoritmo GPMW cuando simula la retroalimentación total. No obstante, el algoritmo cGPMW computa su estrategia teniendo en cuenta el contexto y realizando las operaciones de manera diferente.

La principal idea de este algoritmo contextual es utilizar el contexto para optimizar las recompensas. Su fundamento se basa en que un agente podrá mejorar las acciones jugadas para un mismo contexto si se repite durante el tiempo de manera que aprenderá cómo elegir adecuadamente cada contexto. Esto es lo que se conoce en la teoría de juegos contextuales como que en un juego contextual dónde contextos y acciones similares probablemente produzcan recompensas parecidas. El agente tratará de explotar estas situaciones similares con el objetivo de maximizar sus recompensas mediante el aprendizaje de que brazo es adecuado en cada situación o contexto.

Esto se entiende mejor con un ejemplo. Hay que recordar que los contextos representan las capacidades de las carreteras. Se supone un caso muy sencillo, donde hay dos contextos: un contexto si hace un día con un tiempo soleado y otro si nieva. Si nieva las carreteras reducen su capacidad. La primera vez que nieva el agente sabrá que las capacidades de la carretera son más pequeñas y puede mandar menos paquetes, sin embargo, como no ha experimentado este contexto no sabrá qué acción es mejor. Si se suceden las rondas y el agente pasa por varios días con nieve, comenzará a aprender que acciones son las mejores para los días con nieve. Por esto, puede que aunque para los días soleados (sin nieve) el mejor camino es el uno, mientras para los días con nieve el mejor camino sea el cuatro. El agente podría acabar dándose cuenta de esto tras varios días con nieve donde ha probado todos los caminos. Esta es la base de la explotación de los bandidos contextuales en los juegos, el agente se aprovecha de repetidas situaciones de un contexto para aprender cuál es la mejor estrategia, o brazo, que puede seguir.

Respecto al funcionamiento del algoritmo, la mejor forma de verlo es mostrar el esquema y luego explicarlo, el cGPMW opera de la siguiente manera que se presenta en el algoritmo 14. Cabe señalar que sigue un esquema similar al del algoritmo GPMW pero tiene sus propias diferenciaciones. Primero se inicializa, después observa el contexto y posteriormente realiza las operaciones necesarias para mejorar su toma de decisiones.

---

**Algoritmo 7.14 Algoritmo cGPMW en Juego Contextual.**


---

- 1: Al inicializar: Se establece una tasa de aprendizaje  $\gamma$ , se inicializan los *pesos* a 1, se calculan la recompensa máxima y la recompensa mínima. Se obtienen los  $K$  caminos que puede tomar. Se inicializan el historial y el historial de recompensas.
  - 2: **for** cada ronda  $t = 1, 2, \dots$  **do**
  - 3:   Se observa el contexto  $x_t \in X$ .
  - 4:   Se calcula la estrategia  $y$  y se simula la retroalimentación total para dicho contexto  $x_t$ :  $\forall a \in A : r_{\text{acumulada}}(a|x_t) = f(\text{historiales}, \text{ocupaciones}, \text{capacidades}_t)$
  - 5:   Se actualiza el vector *pesos*:  $\forall a \in A : \text{pesos}_a \leftarrow \text{pesos}_a \cdot e^{\gamma \cdot (-L_a)}$
  - 6:   Se juega un brazo  $a$  elegido aleatoriamente en función del vector *pesos*.
  - 7:   Se actualizan el historial y el historial de recompensas con el brazo  $a$  y su recompensa  $r$ .
  - 8: **end for**
- 

Tras haber mostrado el esquema, es apreciable el principal rasgo de este algoritmo. Primero observa el contexto,  $x_t$ , y después calcula la estrategia a seguir antes de elegir un brazo. Por lo tanto, ya ha desarrollado una estrategia al haber observado un contexto específico antes de elegir un brazo. A priori, podría pensarse que es una ventaja porque aprovechará ese contexto en específico al crear una política acorde a las capacidades de la red antes de tomar una decisión, antes de elegir un brazo. Este es un factor diferencial en contraste con el resto de algoritmos como el algoritmo GPMW que, primero, no observa el contexto, y segundom actualiza sus pesos y realizan predicciones al final de la ronda, no antes de elegir la acción.

Ahora se profundiza en los pasos clave de esta función y se detendrá en desarrollar estas fases para facilitar la comprensión y poder abordar el algoritmo con todo detalle:

- En primer lugar, cGPMW Observa el contexto,  $x_t$ , al comienzo de la ronda. Este contexto refleja las capacidades de las carreteras. El contexto de cada ronda  $t$  se obtiene aleatoriamente entre los  $x \in X$  contextos posibles.
- Después el algoritmo la estrategia dado el contexto. Para calcular la estrategia, al igual que con el algoritmo anterior aprovecha la simulación de retroalimentación total. Sin embargo, el algoritmo cGPMW tiene varias diferencias apreciables respecto a su antecesor.
  - El modelo es entrenado con regresión gausiana con los siguientes datos históricos: el historial de recompensas y el historial que se compone por tres factores: las ocupaciones totales, los paquetes enviados por el agente según el camino elegido y las capacidades. Al contrario que los algoritmos GPMW, se almacenan las capacidades de cada ronda, es decir, los contextos.
  - El algoritmo cGPMW itera desde las rondas anteriores hasta la ronda  $t - 1$ .
  - En cada ronda, realiza una predicción para cada brazo con el modelo de regresión gaussiana, dadas las capacidades actuales, y calcula la recompensa para cada brazo con la siguiente fórmula:  $UCB_a = \hat{\mu}_a + \beta_t \hat{\sigma}_a$ . Beta es una variable que se sigue usando para reducir a la mitad la varianza. La varianza es obtenida por el modelo de regresión gaussiana.
  - Una vez ha calculado la recompensa para esa ronda, escalará dicha recompensa (para cada brazo), de acuerdo a las mismas reglas que necesitaba el algoritmo GPMW.
  - Escala las recompensas sobre 0 y 1 usando la recompensa máxima y la recompensa mínima que obtenida en la incialización.
  - Cuando se obtienen las recompensas escaladas, el algoritmo calcula la recompensa acumulada escalada hasta la ronda  $t - 1$ . De este modo, se tienen en cuenta la predicción de las recompensas de todas las rondas anteriores dado un contexto concreto para cada uno de los brazos.
  - Tras haber calculado todas las recompensas acumuladas para cada brazo haciendo predicciones en las rondas anteriores con las capacidades actuales, utilizará las pérdidas acumuladas, y simplemente restará a  $t$  rondas anteriores las recompensas acumuladas:  $L = T - \text{recompensasacumuladas}_a$  para cada brazo  $a$ .  $L_a$  representaría las pérdidas acumuladas para las rondas anteriores con un contexto particular para un brazo. Se hace de esta forma ya que las recompensas acumuladas se han ido sumando durante  $T$  rondas y estarán entre un valor de 0 y  $T$  para cada brazo.
  - Actualiza los pesos el vector *pesos*:  $\forall a \in A : \text{pesos}_a \leftarrow \text{pesos}_a \cdot e^{\gamma \cdot (-L_a)}$ .

- El siguiente paso a seguir por este algoritmo es elegir un brazo  $a$  aleatoriamente dado el vector  $pesos$  que ha sido modificado anteriormente con el cálculo de la estrategia.
- Actualizará los historiales para el brazo  $a$  y la recompensa obtenida durante esa ronda,  $r_t$ .

Cabe señalar, que el algoritmo cGPMW realiza predicciones en todas las rondas anteriores a la ronda en la que se encuentre y, como resultado, necesita mucho tiempo para ejecutarse. En comparación con el algoritmo GPMW tarda bastante más porque aunque en las primeras rondas el algoritmo cGPMW no requiere mucho tiempo, cuando ya hay un número considerable de rondas, como 40, tendrá que hacer predicciones con la regresión para las 39 rondas anteriores por lo que el tiempo de ejecución aumenta exageradamente a partir de un número alto de rondas.

Como se puede observar, al igual que el algoritmo GPMW los pesos se actualizan en función de las pérdidas acumuladas y no las pérdidas de la ronda. Esto se hace porque se considera que la distribución de las acciones óptimas cambia durante la partida. Se usa este planteamiento ya que se puede pensar que las capacidades del oponente pueden variar a lo largo del tiempo y, por lo tanto, también lo harán las recompensas esperadas.

Entender el funcionamiento de este algoritmo hace comprensible que tener en cuenta el contexto puede ser muy beneficioso. En este caso, se predicen como actuarán todos los brazos dadas distintas ocupaciones para este contexto. Esta información es de gran utilidad ya que el algoritmo conocerá cómo de bueno sería cada camino con las capacidades que se dan en la ronda.

Por este motivo, se puede afirmar que frente al algoritmo GPMW, el algoritmo cGPMW aprovecha el contexto para realizar predicciones más precisas y, en consecuencia, toma mejores decisiones. Por ejemplo, si se predice cuál va a ser el rendimiento de enviar paquetes a través de un camino y no es conocida su capacidad, no se sabrá si puede haber congestión que retrase el tiempo de envío y haga que el camino sea una mala opción. Por el contrario, si antes de mandar los paquetes es sabido si el camino tiene mucha capacidad, se podrá intuir si es una buena alternativa para tardar poco en enviar los paquetes. También sería lo mismo si se observa que el camino tiene poca capacidad ya que habrá mucha congestión debido a la limitada capacidad del mismo, y si esta información del camino es conocida, esté no será considerado como un buen brazo. Cuando se hace referencia a la capacidad de un camino, se está hablando de las capacidades de las carreteras que componen dicho camino.

En segundo lugar, el algoritmo cGPMW también podrá aprender más rápido ya que identificará mejor los patrones entre brazos y recompensas al conocer la información contextual. Esto sucede porque el algoritmo puede ajustar su política de selección de brazos para tomar decisiones más informadas en función del contexto actual. Se parte de la base de que las recompensas están relacionadas con los contextos y los brazos. Al considerar todas las variables involucradas, rendirá mejor. Se podría decir que el algoritmo comprenderá patrones mejor ya que es el único algoritmo que apreciará la toma de decisiones de manera global, considerando la red en su totalidad y abarcando más información que la que conocen otros algoritmos. Esto mejorará sus decisiones y al tener más información para decidir aprenderá más rápido que otros algoritmos.

Otro factor importante es que se podrá adaptar mejor a cada situación. Como ya se ha comentado, hay ciertas limitaciones para algoritmos que no tienen en cuenta el contexto porque toman la decisión en base al mejor brazo en general sin considerar situaciones específicas. Pasa lo contrario con este algoritmo, que sabrá perfectamente que aunque un brazo pueda ser una alternativa mediocre, para un contexto determinado puede ser la mejor vía. Para ilustrar esto, en una red donde un agente tiene cinco distintos caminos y se supone que el camino tres es una opción que ofrece rendimientos regulares, que no es muy rápido ni muy lento, este brazo tres podría considerarse como un brazo mediocre. Sin embargo, podría darse una situación que es muy puntual y ocurre esporádicamente pero provoca que el resto de caminos tengan poca capacidad, como que se aumenten las restricciones en las carreteras por la contaminación excepto en las carreteras del camino tres. Dicho camino a pesar de no ser especialmente bueno, sería tremadamente útil ante esta coyuntura ya que el resto de caminos se verían gravemente perjudicados. Mientras que el algoritmo GPMW no apreciaría esto, el algoritmo cGPMW sacaría partido de esta situación y por eso es destacable su habilidad para adaptarse a diversas situaciones.

Por último, el algoritmo cGPMW puede desarrollar una buena estrategia de cara a la exploración-explotación. Al ser conocedor del contexto puede centrar más su exploración en ciertos contextos donde hay mucha incertidumbre mientras que puede mantener una constante explotación en contextos seguros. Esto puede comprenderse claramente con el mapa de carreteras. Si se supone que una de las variables del contexto es el calendario, dada una época del año festiva, se puede pensar que en estos días las carreteras tendrán mucha ocupación y ante dicha situación habrá mucha incertidumbre por saber qué carreteras serán más concurridas que otras. El algoritmo puede pensar que al aumentar el riesgo le conviene explorar más en un sábado de una semana festiva como

Semana Santa que ante un sábado normal donde las carreteras tendrán ocupaciones similares. De esta manera podría centrarse en desarrollar una estrategia adecuada en estos contextos específicos donde la incertidumbre aumenta considerablemente.

## 7.5. Conclusiones

Esta aplicación práctica se inició con dos objetivos. El primero era realizar una traducción de la implementación original del juego en Python [18] a C++, un lenguaje de más bajo nivel que permitiría escudriñar los detalles para entender el funcionamiento del juego. El segundo objetivo era observar las diferencias entre los algoritmos que no tenían en cuenta el contexto: Hedge y GPMW, y los que sí: cGPMW.

En cuanto al primer objetivo, se ha conseguido implementar correctamente en C++ los algoritmos estudiados, tal y como queda reflejado en los repositorios [26] y [27]. Gracias a esto, se ha obtenido un conocimiento muy avanzado acerca de las partes o procesos que conforman los algoritmos usados en los problemas de bandidos multi-brazo. Por ejemplo, se ha aprendido que la forma de implementar un algoritmo es mediante una clase que tenga como atributos imprescindibles el número de brazos, la distribución actualizada de las probabilidades de elección de cada brazo y demás parámetros necesarios, como la tasa de exploración, que ajustan el funcionamiento del algoritmo. Además, la clase del algoritmo debe implementar dos métodos que son ejecutados en cada ronda: el primero ejecuta la fase de elección de brazo y, tras recibir la recompensa correspondiente, el segundo actualiza la distribución de probabilidades de los brazos en base a dicha recompensa. Por otra parte, no solo se han desarrollado correctamente todos los algoritmos, sino que también se ha creado una red y una simulación en C++ que imita la original en Python.

Respecto al segundo objetivo, este capítulo se ha adentrado en las características de los algoritmos contextuales y los no contextuales para poder explicarlos de manera clara y precisa. Asimismo, se ha conseguido entender y explicar las diferencias entre estos algoritmos. Por tanto, se puede considerar que este objetivo ha sido cumplido con creces.

Pero no todo fue sencillo, durante el proceso de adaptación se encontraron ciertas limitaciones de C++ con respecto a Python que impedían obtener los mismos resultados que en los análisis realizados por Giuseppe Sessa y sus colaboradores reflejados en la figura 12, extraída de [12], y en la figura 13, extraída de [14].

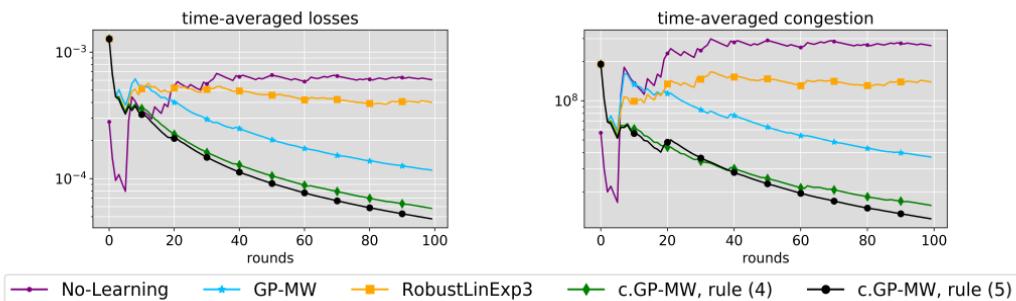


Figura 12: Coste medio y congestión media.

Para saber los brazos o caminos que pueden ser escogidos por los agentes era necesario conocer previamente los  $k$  caminos más cortos (5 en este caso). Con este propósito, en el código de Python se emplea una librería conocida como Networkx que simplifica el proceso. Sin embargo, en C++ no existe dicha biblioteca, lo que supuso la primera dificultad para la traducción a dicho lenguaje. Finalmente, tras la realización de muchas pruebas fallidas, se alcanzó el éxito al adecuar el código publicado en [19] al desarrollo previo, consiguiendo desarrollar la misma función que la librería Networkx en Python.

El aprovechamiento de este código permitió implementar el algoritmo Yen K Caminos Más Cortos, que a su vez hace uso del algoritmo Dijkstra Camino Más Corto. Previamente, fue necesaria la familiarización con el tipo de grafos con los que trabajan estos algoritmos. Además, hubo que conectar el algoritmo Yen con la red de nodos y carreteras, almacenada en la estructura de datos “SiouxNetwork”, y con el vector de estrategias que se explica a continuación. Por ello, para adaptar adecuadamente este algoritmo hubo que modificar gran parte del código ya desarrollado y del propio Yen K Caminos Más Cortos. En conclusión, esta tarea se llevó a cabo

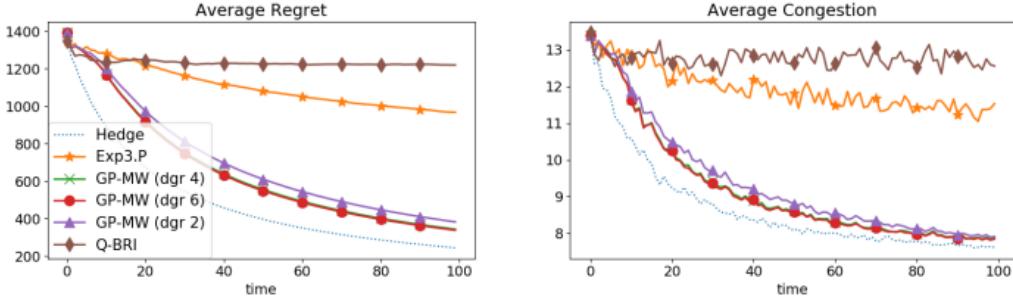


Figura 13: Remordimiento medio y congestión media.

correctamente y se pudo solventar la dificultad.

En el código desarrollado estos caminos se guardan en un vector de estrategias de tres dimensiones, concretamente [528][5][76], compuesto por un primer vector de 528 jugadores o posiciones, un segundo vector para cada jugador de 5 caminos o posiciones y un tercer vector para cada camino. Cada uno de los caminos tiene un tamaño de 76 carreteras de la red, y en cada carretera o posición del vector del camino aparecen las demandas de ese jugador en dicho camino. Se recuerda que cada uno de los 5 caminos refleja uno de los 5 brazos del agente. Por ejemplo, en la posición 20 del vector de estrategias hay cinco brazos asociados a ese jugador (el de la posición 20). El brazo 1 se define por un vector de 76 posiciones y cada una de estas 76 posiciones representa una carretera. Por lo tanto, si la posición 2 tiene un valor de 10 ( $[20][1][2] = 10$ ) y la posición 14 tiene un valor de 5 ( $[20][1][14] = 5$ ), significará que el brazo o camino 1 del jugador 20 utilizará las carreteras 2 y 14, con una demanda total de 15.

Por otro parte, el problema más importante que surgió se debe, de nuevo, a factores técnicos propios de C++, que carece de determinadas librerías que sí están disponibles en Python. Concretamente, hubo problemas al intentar adaptar una librería llamada GPy de Python a C++. Esta librería se encarga de realizar la regresión gaussiana usada por los algoritmos GPMW y cGPMW. En su defecto, en C++, se ha trabajado con la función de regresión “rvm regression trainer” de la biblioteca dlib, también conocida como RVM (Relevance Vector Machine), que consiste en un algoritmo de aprendizaje automático. El único hiperparámetro que requiere RVM es la tasa de aprendizaje, cuyo valor se modificó con el objetivo de no obtener sobreajuste en las predicciones realizadas durante las pruebas. En definitiva, a pesar de que con esta alternativa sí fue posible realizar la regresión correctamente, los resultados conseguidos no fueron iguales a los de Python.

Cabe señalar que inicialmente se probó con el regresor “krr trainer”, también implementado en la biblioteca dlib, pero los resultados fueron sustancialmente peores que los obtenidos con el regresor RVM. Además, aunque se consiguió elaborar en una función que calcula y optimiza los Kernels en C++ utilizando la biblioteca Eigen, esto no fue de utilidad ya que no era compatible con la función de regresión RVM de la biblioteca dlib. Por otra parte, como la biblioteca dlib, a diferencia de GPy, no proporciona la varianza de las predicciones, se decidió optar por calcular la varianza de los datos de entrenamiento y usarla como varianza para las predicciones de la adaptación.

Realmente, todas las decisiones han sido tomadas con el fin de mejorar las predicciones realizadas por la adaptación en C++ y desarrollar buenos algoritmos. De todas formas, se ha llegado a la conclusión de que la traducción de este juego contextual y sus algoritmos en Python a C++ ha requerido de excesivo tiempo y esfuerzo debido a las limitaciones técnicas de C++ en comparación con las ventajas y capacidades del lenguaje Python.

Con todo lo anterior, se considera que la implementación desarrollada sirve como base sólida para entender los bandidos contextuales y sus algoritmos, y enriquece las aportaciones de este apartado contribuyendo al avance del conocimiento en este campo.

Una vez expuestas las características del juego en C++, se procede al análisis del mismo. Para ello, se comparan los resultados del algoritmo GPMW y el algoritmo cGPMW para un agente aleatorio en una simulación concreta. Es decir, en esta simulación, se observa el comportamiento de un único agente para los dos algoritmos. De esta manera, ambos algoritmos disponen de los mismos brazos o caminos para elegir y sus rendimientos pueden ser contrastados bajo el mismo escenario. No se considera el algoritmo Hedge porque tiene retroalimentación total. El interés de este análisis radica en observar las diferencias entre un algoritmo que tiene en cuenta

el contexto, cGPMW, y otro que no observa dicho contexto, GPMW.

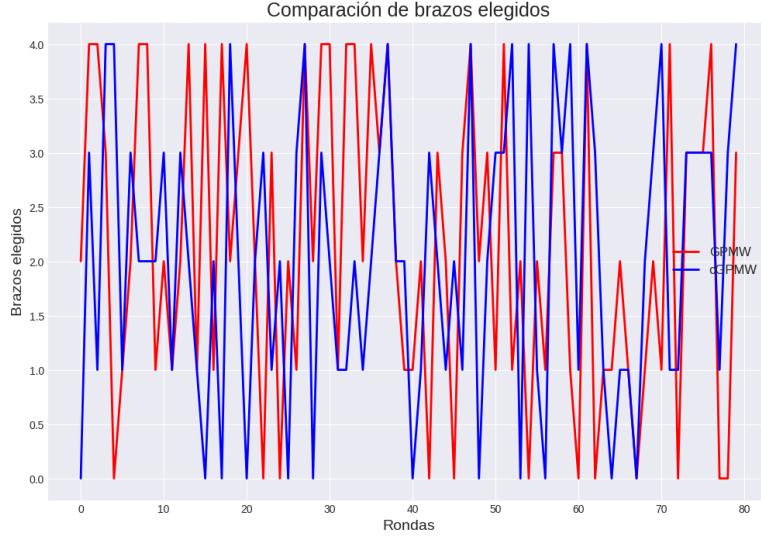


Figura 14: Brazos elegidos.

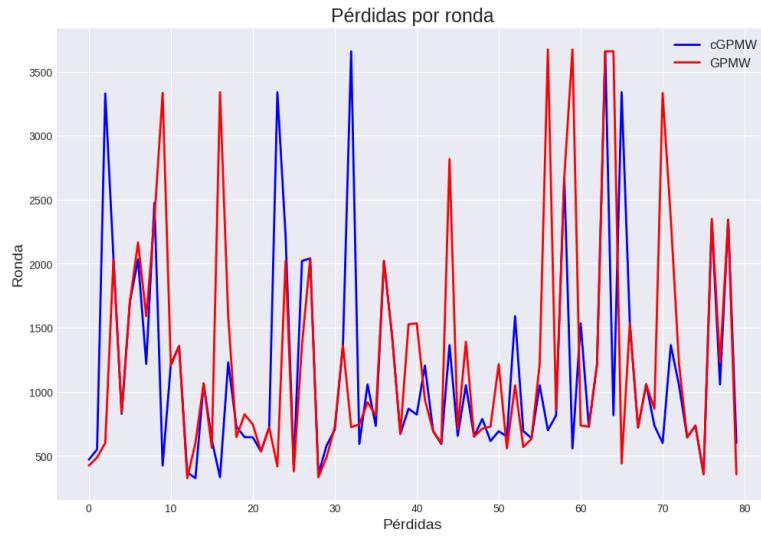


Figura 15: Pérdidas por ronda.

Para obtener las figuras 14 y 15, primero se han guardado los resultados de la simulación de C++ en un fichero. Después, con Python se ha analizado dicho fichero y se han creado estas gráficas que estudian el comportamiento y rendimiento de los algoritmos. Junto a la figura 16, estas reflejan, en cada ronda, los brazos elegidos por cada algoritmo y su efecto inmediato en las pérdidas. Cabe mencionar que en la primera figura los 5 brazos se representan con los valores enteros del intervalo  $[0, 4]$ .

Al analizar estas gráficas, se observa que el algoritmo cGPMW no siempre elige mejor que GPMW, hay ocasiones en las que este segundo elige un camino que implica menos pérdidas. De todas formas, por regla general el algoritmo GPMW es el que obtiene mayores pérdidas, o peores recompensas, a lo largo del horizonte temporal.

Por último, la figura 17 muestra verdaderamente qué algoritmo consigue minimizar las pérdidas medias a lo largo de la simulación. En este caso, se puede apreciar que el algoritmo cGPMW es el que consigue menores pérdidas medias al final de la simulación. En definitiva, se puede concluir que el algoritmo cGPMW, que considera el contexto, alcanza un mayor rendimiento que el algoritmo GPMW en escenarios contextuales.

Debido a la pérdida de eficacia por el proceso de traducción del lenguaje que ha sido reflejada en esta sección y a que solo se ha podido comparar el rendimiento de dos algoritmos entre sí, no ha sido posible obtener

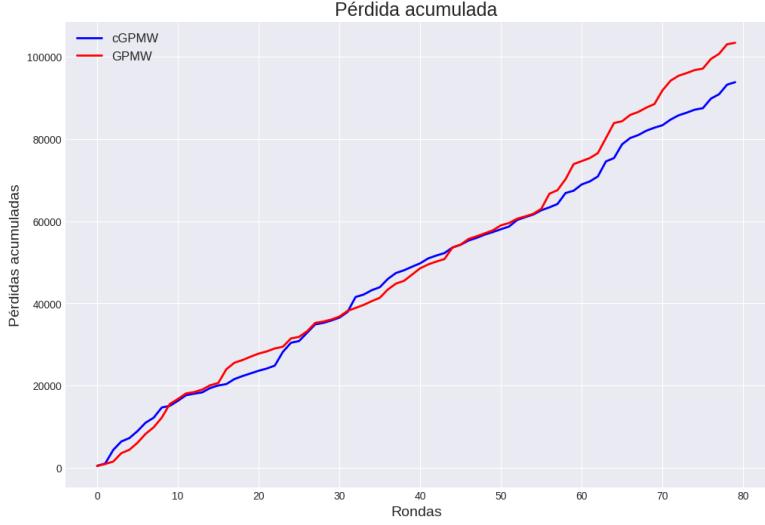


Figura 16: Pérdidas acumuladas.

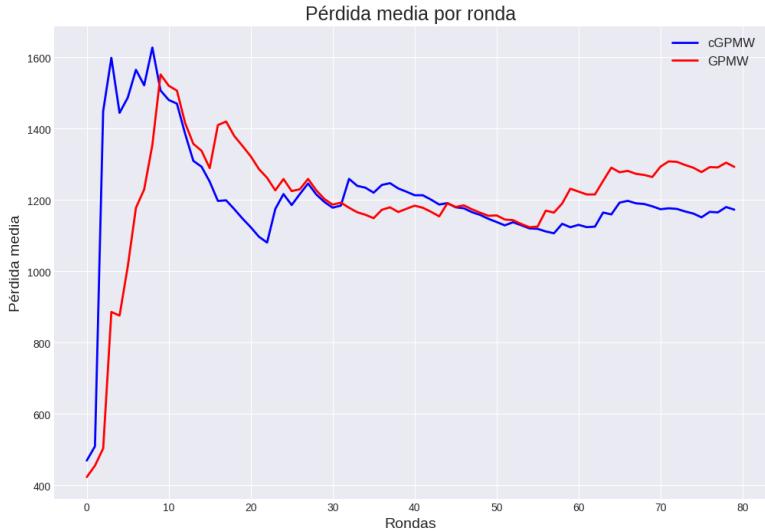


Figura 17: Pérdidas medias.

resultados tan significativos y variados como los que ofrecen los autores de [12] en sus gráficas. Sin embargo, como se puede comprobar en la siguientes explicaciones, la implementación desarrollada en C++ permite llegar a las mismas conclusiones que en [12] respecto a los algoritmos que compara (cGPMW y GPMW), y por tanto se reafirma su valor aportado al trabajo.

Como ya se ha mencionado, el resultado obtenido en [12] y en [14] está reflejado en las figuras 12 y 13, respectivamente. Estas gráficas representan los rendimientos de diferentes algoritmos ante un mismo problema y permiten comprobar qué algoritmos son más eficaces en un escenario contextual. Si se atiende a la figura 12, se comprueba como los algoritmos Exp3 y GPMW son los que peor se comportan ya que no tienen en cuenta el contexto. Se debe tener en cuenta que Exp3 es un algoritmo diseñado para problemas de bandidos antagonistas. En cambio, el algoritmo cGPMW que si tiene en cuenta el contexto parece ser el que menor coste medio consigue, en ambas versiones. La diferencia entre ambas versiones es muy ligera y se basa en el cálculo de la distribución de probabilidad de elección de cada brazo. En base a esta primera gráfica se puede concluir que los algoritmos que sí tienen en cuenta el contexto para tomar su decisión son los que mejor se desenvuelven en un problema de bandidos contextuales.

Si se observa la figura 13 se contemplan las mismas conclusiones que en la figura 12, pero destaca la presencia del algoritmo Hedge en la comparación. Con su rendimiento se sitúa a la cabeza del resto de algoritmos, siendo el algoritmo que mejor rendimiento medio obtiene. Esto se debe a que Hedge recibe retroalimentación total de las recompensas obtenidas por cada uno de los brazos y esta información adicional le permite trabajar más

eficazmente que el resto. Una vez dicho esto, se postula que se han analizado correctamente los tres algoritmos del capítulo que aparecen en las subsecciones 7.4.1, 7.4.2 y 7.4.3. Se aprecian las diferencias entre el algoritmo Hedge, que tiene retroalimentación total y los otros dos modelos. Por ello, se ha comparado el rendimiento entre los algoritmos GPMW y cGPMW de acuerdo al análisis de los autores y a las gráficas desarrolladas en este trabajo. Por tanto, se puede afirmar que se ha cumplido el segundo objetivo con éxito ya que se ha comprendido qué algoritmos se comportan mejor frente a un problema de bandidos contextuales.

En definitiva, el valor de este capítulo para la comprensión de los bandidos contextuales es indudable. Se han logrado estudiar, entender, implementar y explicar algoritmos complejos y técnicas avanzadas. Además, ha permitido asimilar que los algoritmos contextuales son unos algoritmos realmente interesantes que ofrecen rendimientos sobresalientes y permiten trabajar en un marco perfecto para desarrollar otras aplicaciones, como las expuestas en las siguientes secciones.

## 8. Bots de Comercio

### 8.1. Planteamiento

La Inteligencia Artificial (IA) está transformando el ámbito financiero, particularmente a través de las soluciones de *trading* automatizadas conocidas como bots de comercio. Los bots de comercio son aplicaciones que ejecutan órdenes de compra y venta de activos financieros de manera automatizada, basándose en algoritmos y procesos estandarizados. El principio que rige su comportamiento es el mismo que el de cualquier inversor humano: comprar bajo y vender alto. Estos algoritmos son capaces de superar factores sentimentales, como el entusiasmo en un mercado alcista, vendiendo rápidamente activos cuando detectan un cambio de tendencia en vez de permanecer por “el miedo a perderse algo mejor”. De esta forma, minimizan los riesgos y maximizan las ganancias.

La arquitectura y codificación de estos bots son factores determinantes para su éxito, ya que deben garantizar un equilibrio entre riesgo y retorno. Aunque son herramientas automatizadas, requieren la supervisión del operador para monitorizar las condiciones del mercado y ajustar las estrategias si fuese necesario. Entre sus ventajas, destaca la eliminación de errores humanos, ya que el algoritmo ejecutará exactamente lo que esté programado para hacer. La calidad de los resultados obtenidos por un bot dependerá de la idoneidad y efectividad de su programación para adaptarse al mercado.

En el contexto de la economía digital y la transformación de las finanzas, los bots de comercio son cada vez más relevantes. Son especialmente útiles en el comercio de criptomonedas, que opera 24/7, ya que pueden seguir las fluctuaciones del mercado durante todo el día, siempre minimizando el riesgo gracias a que están programados para tener en cuenta muchos factores a la vez. Además, pueden realizar operaciones a una velocidad que supera a cualquier humano. A pesar de la necesidad de conocimientos y experiencia para el *trading*, muchas personas optan por utilizar bots para evitar el estrés y las exigencias de tiempo que supone la observación constante del mercado.

En un mundo donde la automatización es cada vez más cotidiana y ante la necesidad de generar ingresos en una economía digital en constante evolución, los bots de comercio han venido para quedarse. Debido a su eficiencia y comodidad, se prevé que su uso y popularidad continúen creciendo en los próximos años.

Sin embargo, no todo es perfecto y los bots de comercio presentan varios inconvenientes a la hora de ponerlos en práctica. El primero es el más claro y es saber cuál es el mejor bot de comercio para operar en un determinado mercado. Cada bot sigue su propia estrategia, la cual a priori se desconoce si es suficientemente buena. Por otro lado, aunque pueda ser una ventaja que los bots estén operativos de manera constante, esto también conlleva que su manera de actuar permanece inalterable. De esta forma, si el comportamiento del mercado cambia y la estrategia del bot no se ajusta adecuadamente a estas nuevas variaciones, el inversor podría incurrir en significativas pérdidas.

Para hacer frente a estos inconvenientes, se ha desarrollado una solución con dos objetivos principales. En primer lugar, explorar cuál es el mejor bot de comercio dentro de un conjunto de bots. En segundo lugar, reajustar las prioridades o pesos de los bots según las recompensas obtenidas, permitiendo así adaptarse a las posibles variaciones en el comportamiento del mercado. Gracias al desempeño conjunto de los algoritmos Exp4 y Hedge se pueden conseguir ambos objetivos. Exp4 garantiza una exploración constante de todos los bots para poder actualizar sus pesos en caso de que las condiciones del mercado sean mejor aprovechadas por bots que antes no obtenían las mejores recompensas y, por tanto, tenían pesos bajos y no eran explotados con frecuencia. Por consiguiente, con este modelo de solución se consigue una exploración y explotación constante del mejor bot en cada episodio de la vida del activo en el mercado.

La estructura del problema que se plantea es la siguiente:

- Dentro del horizonte temporal solo se maneja un título del mercado. Además, dicho título es considerado como una entidad singular, lo que significa que no se puede ajustar la cantidad adquirida en función de si las expectativas de incremento de su precio son muy altas o altas, respectivamente. Solo se permite la compra o la venta del título. Esta simplificación del problema facilita la obtención de conclusiones más claras.
- Cada política dentro del conjunto entregado al algoritmo representa un bot de comercio. En cada ronda, el bot recibe un contexto específico proporcionado por el algoritmo y, basándose en este, realiza la recomendación de un brazo. Cada bot construye su propia estrategia utilizando un subconjunto de las

variables totales del contexto, diferente al subconjunto observado por el resto de los bots. Incluso si más de un bot basa su estrategia en el mismo subconjunto de variables del contexto, la manera en la que cada bot interpreta estas variables, es decir, su estrategia, siempre será diferente de la de los demás bots.

Es importante recordar que el algoritmo Exp4 es el encargado de elegir los brazos después de recibir las recomendaciones de los bots. Para ello, en función de la tasa de exploración, sigue la recomendación de algún bot teniendo en cuenta sus pesos o elige un brazo de manera aleatoria.

- Los brazos representan todas las acciones posibles: comprar el título, vender el título o mantener la posición. Para entender la casuística del problema, es crucial definir el concepto de la posición actual del algoritmo respecto al título. En términos financieros, una posición larga se refiere a la compra de un activo anticipando un aumento en su precio en el futuro. Por otro lado, una posición corta se adopta cuando un inversor vende un activo anticipando una disminución en su precio.

Esta posición actual introduce una restricción global que obliga a diferenciar entre los brazos disponibles y los brazos posibles. Por ejemplo, si en la ronda anterior el algoritmo Exp4 optó por comprar, se entiende que adoptó una posición larga con respecto al título. En este caso, y dado que el título se considera único para simplificar el problema, las únicas acciones disponibles para la siguiente ronda son vender el título o mantener la posición larga. En contraste, si el algoritmo eligió vender o mantener una posición corta en una ronda determinada, las acciones disponibles para la siguiente ronda serán comprar o mantener la posición corta. En resumen, si se ha comprado o vendido en una ronda previa, no será posible realizar la misma acción en la ronda siguiente. Por ende, a pesar de que el problema contempla tres posibles acciones o brazos, debido a la restricción global comentada, el número de brazos disponibles en cada ronda se reduce a dos.

- El contexto que recibe el bot es la situación diaria en el mercado del título, es decir se cuenta con un contexto diario en cada ronda. La información que aporta cada contexto diario es la posición actual del algoritmo respecto al título y el conjunto de indicadores que se suelen usar para operar en el mercado de valores: precio de la acción o divisa, bandas de Bollinger, canal de Donchian, estocástico, Ichimoku, MACD, momentum, RCI, RSI, volumen, media móvil, retrocesos de Fibonacci, etc.
- La recompensa obtenida por el algoritmo se basa en su elección de brazo y el precio futuro del título:
  - Si decide comprar y el precio sube la recompensa es positiva.
  - Si decide comprar y el precio baja la recompensa es nula. De nuevo, con el propósito de simplificar el problema y facilitar la obtención de conclusiones más claras, se ha decidido no incluir recompensas negativas, a pesar de que estas podrían presentarse en situaciones reales.
  - Si decide vender y el precio sube la recompensa es nula.
  - Si decide vender y el precio baja la recompensa es positiva.
  - Si decide mantener la posición larga y el precio sube la recompensa es positiva. Esto implica que en las rondas previas el algoritmo ha comprado un activo, adoptando una posición larga, con la expectativa de que su valor aumentará con el tiempo. Al decidir mantener esta posición (es decir, no vender el activo), y al verificarce un incremento en el valor del activo, se genera una ganancia o recompensa positiva. Esto se debe a que el valor del activo que posee el algoritmo ha aumentado, y si decidiera venderlo en ese momento, obtendría un beneficio.
  - Si decide mantener la posición larga y el precio baja la recompensa es nula.
  - Si decide mantener la posición corta y el precio sube la recompensa es nula.
  - Si decide mantener la posición corta y el precio baja la recompensa es positiva.

A pesar de que el algoritmo Exp4 trabaja con los costes estimados de cada experto, durante esta sección se hablará en términos de recompensas para facilitar la comprensión de los diferentes análisis realizados.

- El horizonte temporal es un tema complejo que requiere de un análisis separado ya que es vital seleccionar un intervalo de tiempo adecuado para poder explicar correctamente el comportamiento de la solución planteada.

Cada ronda representa uno de los contextos diarios proporcionados por Yahoo Finanzas, la fuente de datos para este problema. Esta fuente suministra información relativa al precio del activo deseado para casi cada día del año, excepto en algunos días para los cuales no dispone de datos. No obstante, el factor diferencial

que modifica por completo los resultados obtenidos es la separación entre los contextos diarios que recibe el algoritmo. Por ejemplo, un horizonte temporal de  $T = 1000$  rondas tendría implicaciones diferentes si la separación entre rondas es diaria en comparación a si es semanal. En el primer caso, se analizaría un periodo de poco más de 1000 días, debido a los defectos de información de la fuente de los datos. Si la separación entre rondas es semanal, se analizaría un periodo de más de 1000 semanas o 7000 días. Dado la influencia sustancial que esto tiene sobre los resultados, el tratamiento de esta disyuntiva será postergado hasta el comienzo del análisis de la solución.

Se ha decidido plantear los contextos con datos históricos en vez de con datos en tiempo real con el objetivo de simplificar el proceso de obtención de datos. Aunque las conclusiones obtenidas reflejen la forma en la que se comporta la solución en el espacio temporal al que pertenecen los datos del contexto, estas se pueden extrapolar para intuir su rendimiento en escenarios futuros con características similares a las de dicho intervalo de tiempo.

Dado el contexto en cada ronda, la función de recompensas otorga recompensas en base al precio del activo en dicho contexto y al precio del activo en el siguiente contexto o ronda. Este siguiente contexto diario mencionado sigue siendo un dato histórico, así se evita que la función de recompensas tenga que sincronizarse con los mercados para esperar el precio del día siguiente.

Con el objetivo de analizar cómo se comporta la solución planteada en distintos escenarios, es recomendable variar tanto el activo observado como el punto de inicio de descarga de los datos históricos. Por ejemplo, estas variaciones permiten analizar el desempeño de la solución tanto en mercados alcistas como bajistas.

Por otra parte, para analizar correctamente la solución planteada, es fundamental la elección del conjunto de políticas dado al algoritmo. Esta elección debería facilitar al algoritmo Exp4 el aprendizaje para discernir entre políticas de mayor y menor rendimiento. En consecuencia, aunque es beneficioso desarrollar bots sofisticados que puedan alcanzar recompensas altas, también es de gran importancia incluir bots de “bajo rendimiento” para comprobar cómo el algoritmo reduce su peso a lo largo de las rondas en comparación con los bots de mejor rendimiento. A continuación, se describen los bots que conforman el conjunto de bots que se proporciona al algoritmo.

## 8.2. Expertos

### 8.2.1. Experto en Posiciones Cortas

El primer bot que se incluye en el conjunto de bots presentado al algoritmo es un bot experto en, como su propio nombre indica, vender. Es un bot pesimista al que no le gusta el riesgo y no contempla otra alternativa que no sea vender o mantener la posición de venta. Es decir, no hay condiciones bajo las cuales este experto decida comprar.

Entonces si, por hacer caso a otro bot en la ronda previa, el algoritmo incurre en una posición larga poseyendo el título, este bot recomendará la venta del título. En cambio, si el algoritmo presenta una posición corta con respecto al título, el bot experto en posiciones cortas sugerirá la elección del brazo correspondiente a mantener dicha posición.

Las conclusiones que se pueden sacar de este experto son limitadas, ya que su comportamiento es completamente independiente de las condiciones del mercado. Sin embargo, este experto será útil para comparar su rendimiento con los ofrecidos por los bots de comercio que sí tienen en cuenta factores económicos y/o financieros y que se presentan más adelante. Sin embargo, es importante tener en cuenta que esta estrategia no es realista en la mayoría de las situaciones del mundo real, ya que no permite aprovechar las oportunidades de compra cuando las condiciones del mercado son favorables.

### 8.2.2. Experto en Posiciones Largas

Se continua incluyendo el bot con el comportamiento contrario al del bot anterior. En este caso, se trata del bot experto en posiciones largas. Al igual que su bot “hermano”, este es un bot simple y directo: solo escogerá brazos que supongan comprar o mantener la posición de compra. Es decir, no se comporta de forma adversa al riesgo y está asumiendo, de forma optimista, que los precios del activo aumentarán a largo plazo.

Por ejemplo, si el algoritmo presenta una posición corta este bot tomará la decisión de comprar el título. En cambio, si el algoritmo se encuentra en una posición larga respecto al título, el bot especializado en posiciones

largas recomendará conservar dicha posición.

En términos de conclusiones, este bot no tiene en cuenta ninguna información sobre el precio del activo o cualquier otro indicador económico y/o financiero. Por lo tanto, es probable que esta estrategia no funcione bien en un mercado volátil donde los precios cambian rápidamente. Sin embargo, en un mercado que está constantemente al alza, esta estrategia podría ser efectiva.

### **8.2.3. Experto en Posiciones Cambiantes**

El bot experto en posiciones cambiantes, al contrario que los expertos en posiciones cortas y largas, no se mantiene en una sola posición constante, sino que alterna entre posiciones largas y cortas. Es decir, si la posición del algoritmo es corta el bot recomienda comprar, y si la posición es larga sugiere vender. Por lo tanto, si el algoritmo solo hiciese caso a este bot, después de comprar un activo, lo venderá en la siguiente oportunidad, y viceversa.

De forma análoga a los dos algoritmos anteriores, este algoritmo sigue una estrategia simplificada y no tiene en cuenta ninguna información de carácter económico o financiero. A pesar de que esta estrategia no parece tener mucho sentido, puede ser interesante para mercados financieros volátiles ya que los precios de los activos pueden fluctuar considerablemente en cortos períodos de tiempo. Concretamente, puede resultar efectiva si la compra del título coincide cuando su precio está bajo y la venta cuando el precio está alto. De esta forma, el experto podría, en teoría, aprovechar estas fluctuaciones de precios para generar ganancias.

No obstante, este enfoque tiene sus riesgos. Aparte de que su simpleza puede implicar que sus decisiones no respondan correctamente a la evolución del mercado, las fluctuaciones de los precios no siempre son predecibles y pueden no seguir el patrón esperado, lo que podría resultar en pérdidas. Adicionalmente, este enfoque podría incurrir en costos de transacción significativos debido al alto volumen de operaciones, lo cual podría erosionar cualquier ganancia obtenida.

### **8.2.4. Experto en Cruce de Medias Móviles**

A continuación, se presenta el primer bot que contempla argumentos financieros para tomar sus decisiones. Concretamente, este bot interpreta la media móvil simple o SMA (Simple Moving Average) para operar en el mercado. Este indicador representa el promedio de los precios de cierre del título en los últimos  $n$  períodos.

Las SMA se utilizan a menudo para determinar la dirección de la tendencia. Si la SMA se mueve al alza, la tendencia es alcista. Si la SMA se mueve a la baja, la tendencia es bajista. El cruce de los precios por encima de la SMA suele utilizarse para activar señales de compra, mientras que cuando cruzan por debajo de la SMA, es posible que se desee vender [23].

Este bot no monitoriza la variable SMA sino que controla de forma conjunta dos variables derivadas de la SMA. Para analizar el corto plazo, lo hace a través de la variable SMA\_corta que calcula el promedio del precio para los últimos  $n = 5$  días. En cambio, para evaluar el largo plazo amplia el rango del cálculo a  $n = 20$  días para obtener la variable SMA\_larga.

Según la teoría, cuando la SMA\_corta cruza por encima de la SMA\_larga es posible que el comportamiento futuro del mercado sea alcista, y la mejor acción sea comprar. En cambio, si la SMA\_corta cruza por debajo de la SMA\_larga, es preferible adoptar una posición en corto. La figura 18 muestra un ejemplo que ilustra esta explicación.



Figura 18: Cruce de SMA de periodo largo, medio y corto [24].

Las gráficas de color rojo, verde y naranja de la figura representan las SMA de largo, medio y corto plazo, respectivamente. Se puede comprobar que cuando las medias móviles de medio y corto plazo cortan con la de largo plazo se produce un cambio en la tendencia del precio del título. La estrategia de cruce de medias móviles es una técnica de análisis técnico popular que se basa en la idea de que las tendencias a corto plazo pueden predecir las tendencias a largo plazo. Concretamente, este bot resuelve los posibles contextos como sigue:

- Si está en posesión del título: Mantiene su posición larga a menos que la media móvil a corto plazo (SMA\_corta) sea menor que la media móvil a largo plazo (SMA\_larga). En ese caso vendería el título porque representaría una señal de que el precio va a descender próximamente.
- Si no está en posesión del título: Mantiene su posición corta a menos que salte la señal de compra cuando la media móvil a corto plazo sea mayor que la media móvil a largo plazo.

Sin embargo, esta estrategia puede no funcionar bien en mercados laterales o con baja volatilidad, ya que las señales de compra y venta pueden ser menos precisas en esas condiciones.

#### 8.2.5. Experto en RSI

El bot experto en el oscilador RSI, al igual que el bot anterior, solo trabaja monitorizando una fuente de datos. El índice de fuerza relativa o RSI (Relative Strength Index) es un oscilador de impulso utilizado en el análisis técnico. El RSI mide la velocidad y la magnitud de las variaciones recientes del precio de un valor para evaluar las condiciones de sobrevaloración o infravaloración del precio de ese valor. Fue desarrollado por J. Welles Wilder Jr. e introducido en su libro [15].

Cabe destacar que el RSI puede hacer algo más que señalar valores sobrecomprados y sobrevendidos. También puede indicar valores que pueden estar preparados para un cambio de tendencia o un retroceso correctivo en el precio. Por ejemplo, si el precio de un activo está alcanzando nuevos máximos o mínimos, pero el RSI no está registrando nuevos máximos o mínimos respectivamente, podría estar indicando una desaceleración en el impulso y una posible inversión de la tendencia. [25]

Para calcular el valor del RSI, primero se calculan las ganancias medias y las pérdidas medias para un periodo anterior de  $n = 14$  días, por regla general. Una vez obtenido el valor de dichas variables, se calcula el valor del indicador como  $RSI = 100 - (100 / (1 + (Ganancias\ Medias / Pérdidas\ Medias)))$ .

La estrategia de este bot sigue las señales tradicionalmente usadas para comprar y vender del RSI. Es decir, una lectura del RSI igual o superior a 70 indica que el precio está superando el umbral de sobrecompra y, por tanto, el bot recomienda vender, o mantenerse en caso de que la posición del algoritmo ya fuese corta. Por el otro lado, una lectura de 30 o inferior indica una situación de sobreventa y el bot sugiere comprar, o mantenerse en caso de que el algoritmo ya presente una posición larga.

Normalmente, este oscilador se suele monitorizar conjuntamente con otros indicadores para mejorar la calidad del análisis. Esto se debe en parte a que, como todos los índices, puede producir señales falsas. Por este motivo, si el bot de comercio o cualquier agente no tiene otras referencias seguirá las señales falsas sin detectarlas, incurriendo en pérdidas. Por otro lado, como este oscilador no presenta información de la dirección de la

tendencia, el bot podría no recibir señales de sobrecompra o sobreventa por parte del RSI mientras que el activo puede estar en una tendencia alcista o bajista fuerte. Además, el RSI tampoco proporciona niveles de soporte y de resistencia que podrían ser útiles para establecer objetivos de precio para vender cuando esté alto o paradas de las pérdidas para vender cuando el precio disminuya hasta un determinado nivel. Por estos motivos, se suele recomendar el análisis técnico combinando los datos de varios índices al mismo tiempo.

### 8.2.6. Experto en MACD y Bandas de Bollinger

Se introduce, por primera vez, un bot que hace uso de más de un indicador al mismo tiempo. Este bot se rige a partir de dos indicadores, el MACD y las bandas de Bollinger. El indicador MACD (Moving Average Convergence Divergence) es una herramienta muy común dirigida al análisis de mercado. Este indicador define la diferencia entre una media móvil exponencial (EMA) del corto plazo y otra del largo plazo. Lo normal es que se consideren los 12 y 26 días anteriores como el corto y largo plazo, respectivamente. Matemáticamente, la línea MACD se calcula tal que  $\text{MACD} = \text{ema}(12, \text{Precio}) - \text{ema}(26, \text{Precio})$ . Además, a partir de la línea MACD calculada anteriormente, se genera una línea de señal conocida como EMA en base a, normalmente, los 9 días anteriores de tal forma que  $\text{Señal} = \text{ema}(9, \text{MACD})$ . La diferencia entre la línea MACD y la línea de señal se denomina Histograma MACD, por tanto,  $\text{Histograma} = \text{MACD} - \text{Señal}$ . En [16] se puede encontrar la interpretación de este indicador.

Por otra parte, las bandas de Bollinger son otro instrumento de análisis técnico que se utiliza para medir la volatilidad del precio. La volatilidad de un título en un determinado intervalo de tiempo es la variabilidad de su rentabilidad en relación a su rentabilidad media en dicho periodo. A partir de la media móvil simple (SMA) del precio de los 20 días anteriores y su desviación típica, se puede establecer una banda superior e inferior tal que  $\text{Banda Superior} = \text{sma}(20, \text{Precio}) + 2 * \text{std\_dev}(20, \text{Precio})$  y  $\text{Banda Inferior} = \text{sma}(20, \text{Precio}) - 2 * \text{std\_dev}(20, \text{Precio})$ . Según el tamaño del ancho de banda, que como señala John Bollinger en [21] representa cómo de separadas están las bandas externas con respecto a la línea central, se puede cuantificar la volatilidad del título.

El bot experto en MACD y las bandas de Bollinger que se plantea trabaja de la siguiente manera:

- Si está en posesión del título: Mantiene su posición larga a menos que se de alguna de las siguientes 2 condiciones para vender:
  - La señal del MACD es mayor que el valor del MACD. Esta condición indica una posible tendencia bajista.
  - El precio de cierre es mayor que la banda superior de Bollinger. Podría indicar que el precio está sobrevalorado y podría corregirse a la baja.
- Si no está en posesión del título: Mantiene su posición corta a menos que se de alguna de las siguientes 2 condiciones para comprar:
  - La señal del MACD es menor que el valor del MACD. Esta condición indica una posible tendencia alcista.
  - El precio de cierre es menor que la banda inferior de Bollinger. Podría indicar que el precio está infravalorado y podría corregirse a la alza.

### 8.2.7. Experto en Todo

Por último, se ha querido incluir en el conjunto de bots de comercio proporcionado al algoritmo un bot que se comporte de acuerdo al movimiento de muchos de los indicadores habituales en el mercado actual. En cada ronda, este bot observa un conjunto de indicadores financieros y la posición actual del algoritmo: corta o larga. A continuación, evalúa una serie de condiciones basadas en estos indicadores. Por ejemplo, comprueba si el precio de cierre es superior a la banda superior de Bollinger, si el MACD es superior a su línea de señal, si el RSI es superior a 50, etc. Suma el número de condicionantes de su estrategia que son verdaderos y luego realiza una recomendación basada en si este número es mayor o igual a 6.

Si en una determinada ronda el algoritmo está en posesión del título y el número de condiciones verdaderas es mayor o igual a 6, el experto decide mantener la posición larga. De forma inversa, si el número de condiciones verdaderas es menor que 6, decide vender el título. La lógica es similar si el algoritmo tuviese una posición corta. Es decir, si no posee el título en una ronda en la que la cantidad de condiciones verdaderas supera o iguala el valor 6 el bot experto en todo recomendará al algoritmo comprar y en caso contrario mantener su posición.

Entre todos los bots que serán evaluados y explotados por el algoritmo Exp4, este es el bot cuya estrategia presenta la mayor complejidad de construcción debido a la variedad de indicadores que rigen su comportamiento. Una de las objeciones que se podrían plantear a esta estrategia es que considera la importancia de todos los indicadores por igual. Esta objeción se fundamenta en el hecho de que dependiendo de las condiciones del mercado unos indicadores pueden ser más relevantes que otros.

En cambio, la diversidad de señales que presenta permite al bot incrementar su flexibilidad de decisión con respecto al resto de bots ya que adquiere una visión más completa del estado del mercado. Es decir, su comportamiento no siempre queda definido por los mismos indicadores. Con todo esto, el bot experto en todo es una gran incorporación al conjunto de bots.

Se podría haber seguido incluyendo más estilos de bots de comercio en el conjunto de bots, pero la parte relevante de esta sección, en relación al trabajo de fin de grado, se centra en el análisis del comportamiento de la solución planteada para resolver el problema que se viene explicando. De todas formas, el código desarrollado incluye la implementación de otros bots que pueden ser incluidos, si se desea, en el conjunto de bots contemplado por el algoritmo Exp4, como por ejemplo el experto en Ichimoku, el experto en Fibonacci, etc.

### 8.3. Análisis de la Solución Planteada

Antes de comenzar, en la figura 19 se presenta la leyenda para identificar a cada bot del conjunto de bots entregado al algoritmo Exp4.

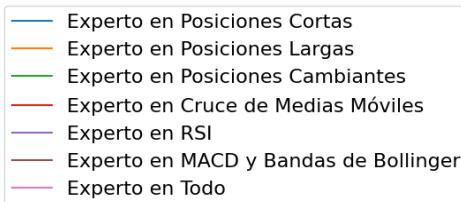


Figura 19: Leyenda de los bots de comercio.

De esta forma es posible identificar con el color azul al bot experto en posiciones cortas, con el naranja al bot experto en posiciones largas, con el verde al bot experto en posiciones cambiantes, etc. La finalidad con la que se ha separado la leyenda de la presentación de las gráficas mostradas a continuación es incrementar la visibilidad de los detalles, permitiendo mejorar la comprensión de las situaciones reflejadas.

#### 8.3.1. Consideraciones previas

Previo al análisis de la solución planteada, en este apartado se busca justificar dos decisiones tomadas con respecto a la implementación de la solución para superar ciertas limitaciones que a priori presentan los algoritmos Exp4 y Hedge de forma conjunta. Concretamente, ha sido necesario modificar la función de recompensas y el cálculo de las probabilidades que presentan los expertos de elegir cada brazo.

#### Modificación en la Función de Recompensas

En un principio, se quiso establecer 1 como valor para la recompensa positiva y 0 para la recompensa nula. Pero debido a la forma ( $\frac{c_t(a_t)}{\Pr[a_t=a_{t,\pi} | \bar{p}_t]}$ ) que el algoritmo Exp4 tiene de establecer los costes estimados para aquellos bots que eligen, en una determinada ronda, el mismo brazo que el seleccionado finalmente por el algoritmo en dicha ronda, se tuvo que realizar una modificación. Esto se debe a que, si el algoritmo acertaba con su

decisión de brazo, aquellos bots cuya recomendación coincidía obtenían una recompensa de 1, que equivale a un coste de 0. Sustituyendo dicho coste en la fórmula mencionada anteriormente, dichos bots obtenían un coste estimado de 0. Por defecto, Exp4 por defecto otorga un coste estimado de 0 a los bots que no recomiendan el mismo brazo que el elegido por el algoritmo. Por ende, en este escenario en el que el algoritmo acierta con su elección, no habría diferencia entre los costes estimados de aquellos bots que acertases con su elección y los que no. Como resultado, tal y como explica la fórmula ( $w_{t+1}(a) = w_t(a) \cdot (1 - \epsilon)^{\hat{c}_t(a)}$ ) que el algoritmo Hedge aplica para actualizar los pesos de los bots, ninguno de los pesos se modificaría. Esto implicaría que tomar buenas decisiones no proporcionaría ninguna “ventaja” a los bots.

Por tanto, con la intención de solucionar este *impasse* se ha tomado la decisión de considerar una recompensa positiva de 0,9 que evita el problema anterior y, para mantener un equilibrio, una recompensa “casi nula” de 0,1.

### Modificación de las Probabilidades Individuales de Elegir un Brazo

Según el libro de Slivkins [2], que constituye el pilar fundamental del desarrollo de este trabajo, la definición teórica del algoritmo Exp4 hace referencia a que, en cada ronda, cada experto debe devolver la probabilidad con la que ha recomendado su brazo con la finalidad de que Exp4 pueda calcular la probabilidad conjunta, entre todos los expertos, de escoger cada uno de los brazos posibles. Esto se debe a que Exp4 necesita tener calculada la variable  $\Pr[a_t = a_{t,\pi} | \vec{p}_t]$  para usarla como denominador en su paso 6. Su aplicación en el denominador lleva implícita la condición de que su valor no puede ser 0 o cercano a 0, ya que en ambos casos el resultado de la división sería nulo.

Para ello, como ya indica Slivkins en una de sus anotaciones del capítulo 6, la solución consiste en asegurar que las probabilidades individuales de cada experto de elegir o recomendar cada uno de los brazos sean mayor que 0. Esto supone ir en contra de la lógica de los conceptos aprendidos hasta el momento, dado que un bot de comercio es un experto que asocia contextos a brazos siguiendo una estrategia predefinida e invariante. Es decir, dado un determinado contexto, el bot siempre decide recomendar el mismo brazo de acuerdo a su estrategia, lo que implica una probabilidad de elección del 100 % sobre dicho brazo, quedando reducidas a 0 las probabilidades de ese bot de elegir el resto de brazos disponibles.

Para comprobar que la solución mencionada por Slivkins es necesaria, se ha probado a programar las probabilidades individuales de elección de brazo de los bots tal y como se ha mencionado anteriormente, siguiendo estrictamente la lógica y no los comentarios de Slivkins. La figura 20 ilustra el problema que sucede con los pesos de los bots si las probabilidades individuales de recomendación de cada bot son 100 % para el brazo indicado, bajo cada contexto, por su estrategia y 0% para el resto.

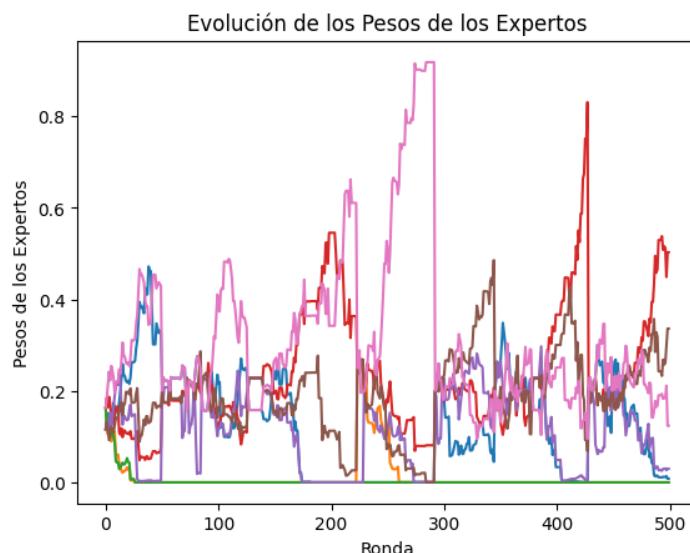


Figura 20: Evolución de los Pesos de los Expertos con  $\gamma = 0,2$ .

Antes de adentrarnos en el análisis, es esencial recordar que la probabilidad conjunta entre todos los bots

de recomendar el brazo  $a_t$ , finalmente elegido por el algoritmo, es usada como denominador en el cálculo de los costes estimados, con los cuales Hedge actualiza los pesos de los bots. Además, es necesario recordar que el cálculo de esta probabilidad tiene en cuenta las probabilidades individuales de recomendación de cada bot y los pesos de estos bots.

Cuanto mayor sea el valor de  $\gamma$ , más probabilidades hay de que el algoritmo elija aleatoriamente un brazo, sin tener en cuenta los pesos de los bots. Por tanto, a pesar de que algunos bots ya hayan sufrido una caída de sus pesos y el algoritmo, difícilmente, vaya a seguir sus recomendaciones, con  $\gamma > 0$  Exp4 podría llegar a elegir el mismo brazo que los bots “infravalorados”.

En el caso de que, en una determinada ronda, el algoritmo elija un brazo que solo ha sido recomendado por uno o pocos bots cuyos pesos son muy bajos, se estaría incurriendo en una situación en la que la probabilidad conjunta definida anteriormente reciba un valor de 0 o cercano a 0.

Esta situación sucede varias veces durante el transcurso de la ejecución reflejada en la figura 20. Cada vez que se da esta situación, como por ejemplo en la ronda 292, supone un antes y un después en la evolución de los pesos de los bots. Se deja en el anexo 13 una depuración “casera” obtenida durante la ejecución mencionada en caso de que se quiera apreciar con mayor detalle lo acontecido en la ronda 292.

Como se ha querido demostrar, es necesario que las probabilidades individuales de elección de cada bot sean mayores que 0 para todos los brazos. Si se interpreta textualmente la definición de una política (asociación de contextos a brazos con probabilidad del 100 %), los resultados que se obtienen carecen de valor porque en el cálculo de los costes estimados se divide el coste real entre una probabilidad de 0 o cercana a 0. Por tanto, para el resto de la sección se adopta una interpretación menos realista pero que evita el problema y permite realizar un análisis riguroso. Esta variante consiste en programar que el bot informe al Exp4 de que su probabilidad de recomendar el brazo que indica su estrategia según el contexto es del 90 %, mientras que la del otro brazo no escogido es del 10 %. Este simple y pequeño reparto de la probabilidad total es suficiente para paliar los problemas generados por la interpretación literal de la teoría del problema.

### 8.3.2. Análisis del Horizonte Temporal

En la introducción del problema de los bots de comercio se mencionaba que el horizonte temporal convendría ser definido tras un análisis previo debido a la complejidad que presenta. Este es uno de los aspectos más relevantes del planteamiento del problema porque los costes estimados en los que incurre cada experto en cada una de las rondas dependen en gran medida de la configuración del horizonte temporal.

Por ejemplo, para analizar 6 años se puede considerar una ronda como cada uno de sus días, meses, trimestres, años, etc. Lo que varía es la separación temporal entre cada dato del precio del activo. Por tanto, se puede analizar el mismo intervalo de tiempo combinando distintas cantidades de separación temporal con distinto número de rondas u horizonte temporal.

Debe recordarse que el bot recibe una recompensa u otra dependiendo del brazo que ha recomendado y del siguiente precio proporcionado por la fuente de los datos. En este sentido, no es lo mismo que el siguiente precio sea el del día siguiente que el de una semana después o un mes después. En definitiva, las recompensas de los bots de comercio quedan al amparo de la decisión que se tome en este aspecto.

Para un activo (Apple Inc.) y un intervalo de tiempo (2015-2021) determinados al azar, se han analizado las recompensa medias obtenidas por el algoritmo para diferentes días de separación entre cada dato, y los resultados quedan reflejados en la figura 21.

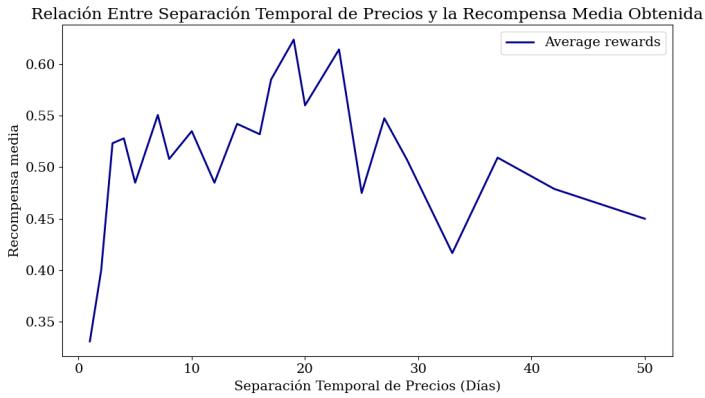


Figura 21: Relación entre la Separación Temporal de Contextos (en días) y la Recompensa Media Obtenida

Las recompensas medias obtenidas por el algoritmo dependen de cómo de bien se comportan los bots en el escenario establecido. Por tanto, a partir de la figura anterior, se puede deducir que los bots de comercio rinden mejor con una separación temporal de 15 y 25 días entre los contextos diarios que recibe el algoritmo Exp4. La explicación detrás de esta conclusión se encuentra en la forma en la que son diseñados los indicadores. En la solución planteada, estos se forman analizando los  $n = 9, 14, 12, 20, 26$  días anteriores. Entonces, es lógico que la amplitud del intervalo de los datos históricos, que los bots tienen en cuenta para escoger brazos, influya en cuál es el mejor escenario para los bots. Por ejemplo, un bot que recomienda en base a los datos relativos al último año de la vida del activo tiene información acerca de la evolución del mercado a largo plazo. Este bot nunca será capaz de igualar el rendimiento a corto plazo de un bot que ha sido informado únicamente de los precios de la semana anterior, ya que este último tiene una fiel imagen del comportamiento del activo a corto plazo.

Cabe destacar que la existencia de esta disyuntiva es independiente del valor de los hiperparámetros  $\gamma$  y  $\varepsilon$ , asociados a la tasa de exploración y aprendizaje, respectivamente. Por lo tanto, teniendo en cuenta los valores de  $n$  comentados, para el análisis final de los resultados se ha considerado una separación temporal de 18 días entre los precios consecutivos considerados por la función de recompensas.

Aparte de aprovechar al máximo el potencial de los bots planteados, también se consigue disminuir la correlación entre los resultados obtenidos por los bots. Hasta el momento no se había mencionado esta circunstancia, pero es importante tener en cuenta que, aparte de que el número de brazos disponibles en cada ronda es muy reducido (2), los bots interpretan el mercado y recomiendan brazos de forma muy similar entre sí. En consecuencia, sus recompensas esperadas tienen comportamientos parecidos. Por tanto, aminorando la correlación entre los rendimientos de los bots se ayuda a que el algoritmo Exp4 aprenda, de manera óptima, cuáles son los mejores bots. En las figuras 22 y 23 se pueden observar las diferencias entre las correlaciones en función de la separación temporal entre las rondas.

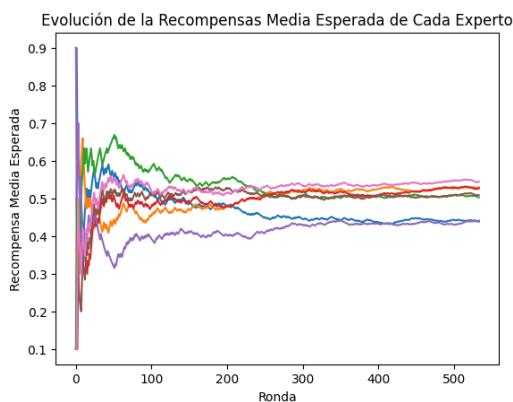


Figura 22: Correlación entre los rendimientos esperados de los bots con 3 días de separación entre rondas.

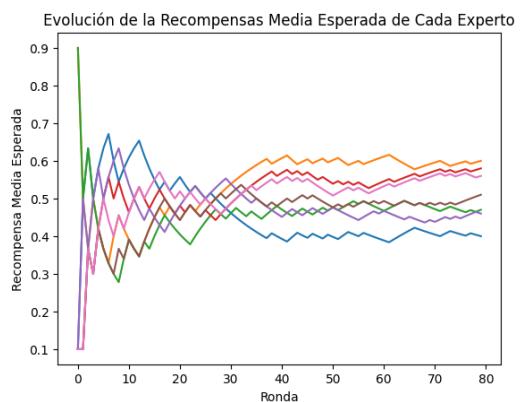


Figura 23: Correlación entre los rendimientos esperados de los bots con 20 días de separación entre rondas.

### 8.3.3. Análisis de la Tasa de Aprendizaje

En esta sección se estudia qué tasa de aprendizaje es óptima para el problema problema planteado. Se define también como  $\varepsilon$ . Para ello, se realiza un análisis con el objetivo de comprender el comportamiento del Exp4 bajo diferentes tasas de aprendizaje.

La tasa de aprendizaje es un hiperparámetro que tiene mucha importancia en el modelo Exp4. Define la rapidez con la que el algoritmo compone una distribución de pesos sobre los expertos que ya presenta claras diferencias entre las prioridades relativas otorgadas a cada experto, es decir la rapidez con la que aprende. Una tasa alta castiga mucho a los bots que toman decisiones erróneas al principio de la ejecución, reduciendo demasiado su peso y dejando de tener sus recomendaciones en consideración. Esto puede ser perjudicial para el rendimiento final obtenido por el algoritmo si los bots menospreciados son a la larga los mejores. Por el contrario, con una tasa de aprendizaje muy baja, el algoritmo tardaría demasiado en aprender y seguiría muchas veces las sugerencias de bots que no son buenos, reduciendo la recompensa media final.

Por tanto, se definen dos ideas que se tratarán de demostrar seguidamente:

- La primera es que una tasa de aprendizaje baja implica que Exp4 tarda mucho en aprender pero terminará aprendiendo correctamente cuáles son los bots buenos. Por tanto, aunque termine aprendiendo bien, su recompensa media final no será muy elevada debido al lastre que supone probar demasiadas veces todos los bots, incluyendo los malos, al principio.
- La segunda idea es que una tasa de aprendizaje alta no garantiza que Exp4 aprenda cuáles son los mejores bots, porque puede castigarles mucho al principio. En consecuencia, con la tasa alta, la recompensa media será muy elevada si Exp4 aprende adecuadamente desde el principio cuáles son los bots que mejor rinden.

Como ya se ha mencionado, este aprendizaje rápido no siempre es efectivo, ya que depende de la aleatoriedad. Habrá veces que el algoritmo no aprenda idóneamente y explote en mayor medida los peores bots durante el resto del horizonte temporal, perjudicando enormemente la recompensa media final.

En la figura 24, se lleva a cabo una prueba del algoritmo con diferentes tasas de aprendizaje. Como resultado, se observan variaciones en la recompensa media. Además, se constata que el bot más frecuentemente seleccionado y el bot con mayor peso en la última ronda no siempre son los mismos. También, permite apreciar la segunda idea comentada anteriormente: con tasas de aprendizaje altas muy similares (0.4 y 0.5) Exp4 obtiene resultados completamente distintos. Con una  $\varepsilon = 0,4$  alcanza una de las peores recompensas medias, mientras que con una tasa  $\varepsilon = 0,5$  la recompensa media es mayor. Esto refleja la aleatoriedad en las recompensas a consecuencia de una tasa de aprendizaje alta, tal como se quería demostrar.

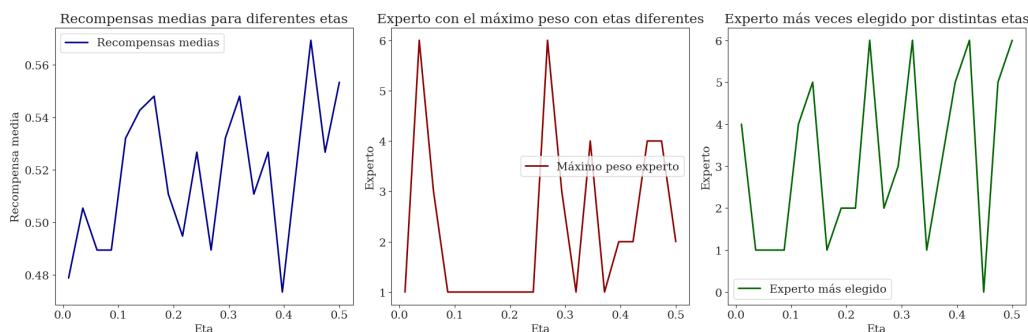


Figura 24: Resultados de Ejecutar la Solución con de Distintas Tasas de Aprendizaje  $\varepsilon$ .

Paralelamente y de forma adicional a la ejecución del Exp4, se han guardado en variables auxiliares las recompensas esperadas asociadas a las recomendaciones de los bots, simulando que los bots trabajan individualmente el problema, aunque realmente Exp4 decide el transcurso de la solución teniendo en cuenta todos los bots. Es decir, a pesar de que el algoritmo sigue la recomendación de un solo bot y estima las recompensas para el resto de bots, las variables auxiliares guardan las recompensas que todos los bots obtendrían por su elección. Esta simulación permite el análisis a posteriori de qué bots obtienen las mejores recompensas independientemente de las decisiones tomadas por algoritmo Exp4. Por ende, posibilita comprobar si la solución planteada a partir del algoritmo Exp4 consigue aprender a explotar los bots más eficientes.

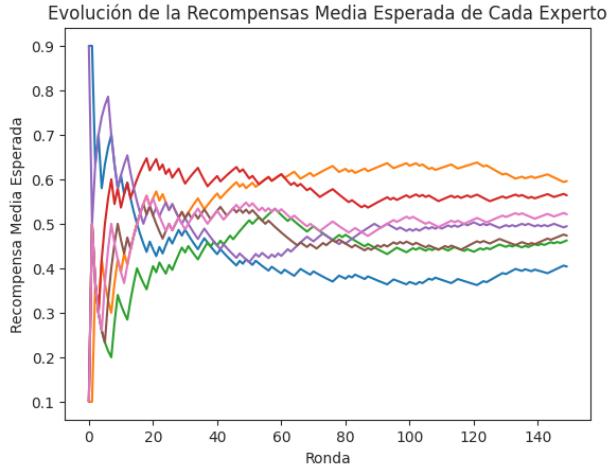


Figura 25: Recompensa Media Esperada de Cada Bot para  $\varepsilon = 0,39$ .

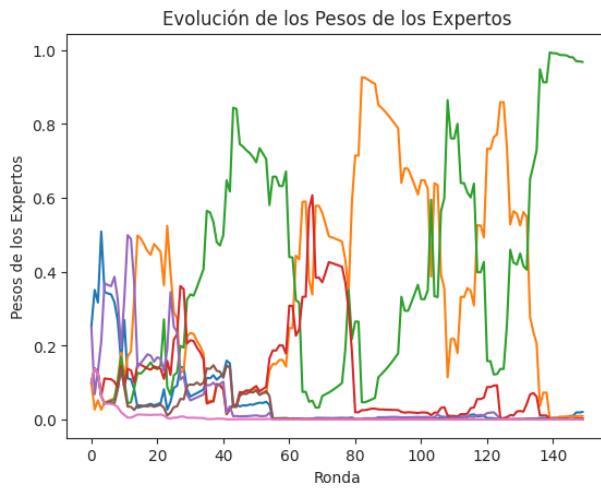


Figura 26: Pesos de los Bots para  $\varepsilon = 0,39$ .

Con una tasa de aprendizaje  $\varepsilon = 0,39$ , la figura 25 expone la simulación argumentada anteriormente, y la figura 26 ilustra la evolución de los pesos de los bots desde el punto de vista del algoritmo Exp4, sin tener en cuenta la simulación. Esta segunda gráfica permite observar el proceso de aprendizaje del algoritmo ya que refleja cómo varía su consideración de todos los bots. La combinación de ambas gráficas da lugar a la manifestación de la existencia o no de un aprendizaje adecuado por parte de Exp4. Por ejemplo, si el bot experto en RSI es el que mejor recompensas medias ofrece a lo largo de la simulación y Exp4 nunca le otorga un peso relativo alto, esto significa que el algoritmo no está aprendiendo a explotar el mejor bot.

Los resultados, en este escenario con una tasa de aprendizaje alta, son evidentes. El segundo mejor bot según la simulación, el bot experto en cruce de medias móviles (rojo), al principio llega a obtener pesos de más de 40 %, pero tras alguna mala recomendación, su peso disminuye hasta casi 0 dificultando su posterior reconsideración. Por tanto, queda claro que Exp4 no está aprendiendo bien porque penaliza demasiado al bot rojo al principio, a pesar de que este sea en realidad uno de los bots con mayores recompensas medias esperadas. De esta forma la segunda idea ha quedado probada.

Por otro lado, se analiza el comportamiento del algoritmo con una tasa demasiado baja, concretamente  $\varepsilon = 0,01$ , mediante las figuras 27 y 28.

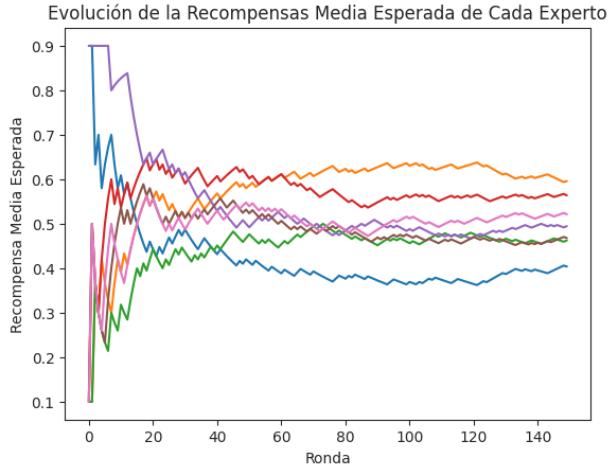


Figura 27: Recompensa Media Esperada de Cada Bot para  $\varepsilon = 0,01$ .

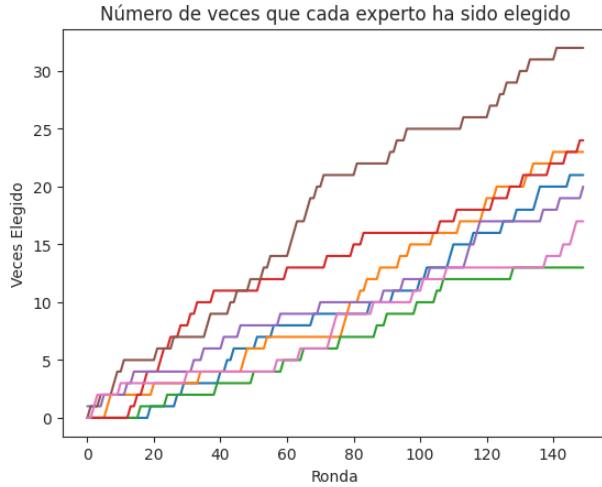


Figura 28: Elección de Bots para  $\varepsilon = 0,01$ .

La figura 27 ilustra las recompensas esperadas obtenidas con la simulación. Cabe destacar que en este caso es ligeramente diferente a la simulación representada anteriormente. Esto es porque las recomendaciones de los bots dependen de la posición larga o corta que presente el algoritmo. El desarrollo de las posiciones del algoritmo dependerá de cómo el algoritmo decida en cada ejecución. Como sus decisiones no son deterministas, las posiciones presentadas a los bots varían entre ejecuciones y, en consecuencia, sus recompensas esperadas también. Por este motivo, las gráficas se parecen pero no son idénticas. Estas diferencias se dan por el simple hecho de ser distintas ejecuciones, independientemente de si se varía la tasa de aprendizaje o la de exploración. De todas formas, la variación de las recompensas esperadas es tan pequeña que se suele mantener el orden de los mejores bots entre ejecuciones.

Como se puede observar en la figura 28, todas los bots se eligen casi uniformemente durante las primeras 120 rondas; hay muy poca variación exceptuando el bot marrón experto en los indicadores MACD y bandas de Bollinger. A partir de la ronda 50, el algoritmo elige más veces el bot marrón que los dos mejores bots según 27, el experto en posiciones largas (naranja) y el experto en medias móviles (rojo). Aunque finalmente aumente el peso de estos dos mejores bots, ha elegido demasiadas veces a los peores bots, y esto provoca que la recompensa media final del algoritmo sea muy baja. Además, como la tasa de aprendizaje es tan pequeña, a pesar de que pasen las rondas, el algoritmo sigue manteniendo al bot marrón con mucho peso, aún siendo uno de los peores bots, como evidencia la figura 27.

Por lo tanto, ha quedado demostrado que elegir una tasa de aprendizaje baja no es beneficioso para el algoritmo porque la recompensa obtenida se reduce. Tampoco se debe optar un valor alto ya que hay mucho riesgo de que no aprenda las mejores políticas. Por ello, se ha detectado una buena zona de trabajo, para obtener mejores recompensas, cuando las tasas de aprendizaje se encuentran en el intervalo de 0,15 y 0,25 aproximadamente.

En el análisis final de los resultados se ejecutará la solución con un valor  $\varepsilon = 0,2$ .

#### 8.3.4. Análisis de la Tasa de Exploración

La tasa de exploración  $\gamma$  influye en la cantidad de veces que el algoritmo explora un brazo aleatoriamente sin seguir las recomendaciones de los bots. La relación entre el valor de esta tasa y la recompensa media obtenida se puede analizar a través de la figura 29.

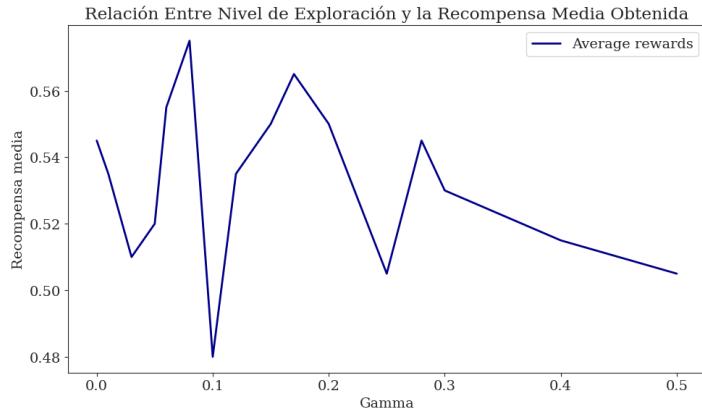


Figura 29: Relación entre el Nivel de Exploración y la Recompensa Media.

Como ya se comentó en la introducción, el número de brazos disponibles para cada ronda es  $K = 2$ . Este número de brazos es una cantidad bastante limitada y no existe la necesidad de realizar una gran exploración con el objetivo de comprobar el rendimiento medio de todos los brazos. En muy poco tiempo, por el simple transcurso de las rondas, el algoritmo habrá explorado sobradamente cada uno de los dos brazos. De todas formas, no está de más otorgar un cierto valor a este hiperparámetro para prevenir situaciones en la que los bots se estanquen con una misma recomendación y no consigan ver el potencial de otro brazo dado un contexto determinado. Por ello, para el análisis final de los resultados se usará una  $\gamma = 0,05$ .

#### 8.3.5. Puesta en Práctica de la Solución

Con el objetivo de solucionar el problema de los bots de comercio de manera óptima, se ha considerado una separación entre rondas de 18 días, una tasa de aprendizaje  $\varepsilon = 0,2$  y una tasa de exploración  $\gamma = 0,05$ . Se ilustra de nuevo la leyenda de los bots en la figura 30.

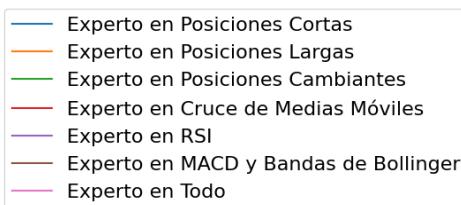


Figura 30: Leyenda.

A continuación se plantean un escenario para comprobar cómo se comporta la solución. En este caso, el algoritmo se sitúa entre mediados del año 2015 y finales del 2019 para operar en el mercado sobre las acciones de la empresa NVIDIA Corporation (NVDA) dedicada al desarrollo de software y hardware. Para ser capaces de interpretar el comportamiento del algoritmo es necesario ser conscientes de la evolución del precio del título durante el periodo mencionado, reflejada en la figura 31

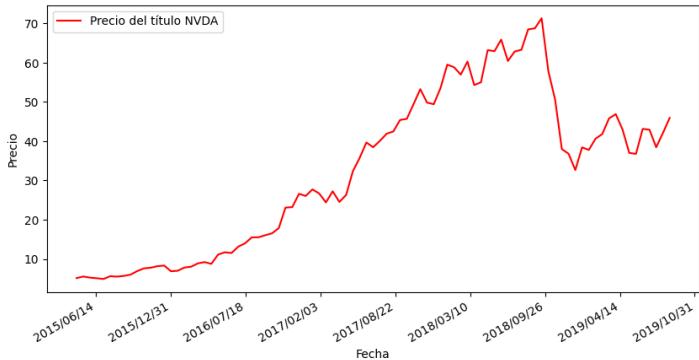


Figura 31: Precio NVDA.

Para evaluar lo sucedido durante la ejecución de la solución, se tiene en cuenta la variación de los pesos de cada bot y la simulación de las recompensas esperadas si todos los bots recibiesen la recompensa real correspondiente en vez de la recompensa estimada. Dichos datos están representados en las figuras 32 y 33, respectivamente.



Figura 32: Evolución de los Pesos de los Bots.

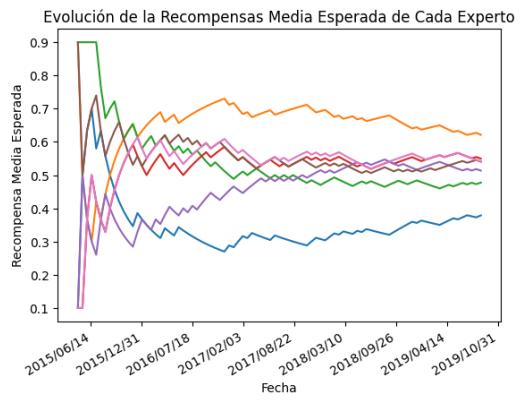


Figura 33: Simulación Evolución Recompensas Medias por Experto.

En este caso, la simulación indica que el bot que mejor se comporta es el bot experto en posiciones largas. Esto es coherente, debido a que el precio ha seguido una tendencia alcista durante los primeros tres años favoreciendo así las compras de dicho título. Otros expertos que también se comportan bien son el experto en todo y el experto en cruce de medias móviles. Gracias a otra gráfica, se comprueba cómo el algoritmo ha ido variando los pesos de los expertos para finalmente otorgar mayor preferencia a aquellos bots que, como se han identificado en la simulación, mejor se comportan. Por tanto, se llega a la conclusión de que el algoritmo planteado está aprendiendo con el paso del tiempo cuáles son los bots óptimos que debe explotar en este escenario. Es decir, la solución planteada ha logrado su objetivo y es todo un éxito.

Gracias a este éxito, el algoritmo ha conseguido aumentar progresivamente su recompensa media. Debe ser recordado que los rendimientos de los bots están bastante correlacionados y esta tarea puede resultar bastante difícil cuando se llega al límite de excelencia de los bots incluidos en el conjunto dado al algoritmo, que se ubica entre 0.5 y 0.6 de recompensa media. En este caso, la ejecución empezó mal pero con el paso del tiempo consiguió adaptarse, actualizando los pesos de los bots, y remontar la situación hasta alcanzar el nivel comentado, tal y como revela la figura 34.

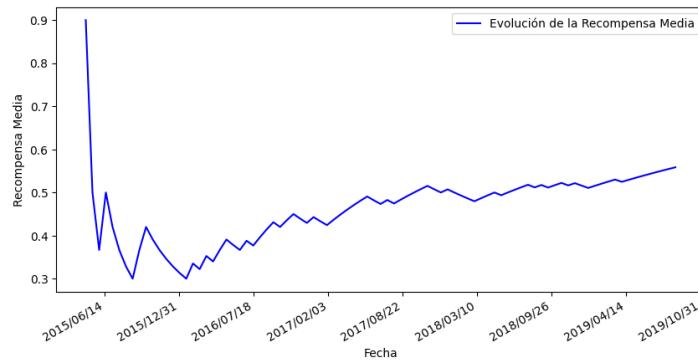


Figura 34: Recompensa Media Obtenida por la Solución para NVDA.

Cabe destacar que la solución implementada para esta sección, reflejada en los repositorios [26, 27] de los autores, permite cambiar el activo y el periodo analizado. De esta forma, variando los parámetros de entrada, se puede profundizar más en el análisis de qué bots de comercio se comportan mejor ante las múltiples situaciones que se pueden presentar en los mercados financieros.

## 9. Sistema Recomendador de Películas

En esta sección se desarrollará un sistema recomendador de películas utilizando un *bandido contextual con clase política*, descrito en la sección 6.3. Este sistema recomendador es un tipo de motor de sugerencias que utiliza técnicas avanzadas de aprendizaje automático para proporcionar recomendaciones personalizadas de películas a los usuarios. Las recomendaciones se harán con el algoritmo Exp4 explicado anteriormente. Se crearán varias políticas con el objetivo de probar la eficiencia de dicho algoritmo. El propósito del sistema recomendador es sugerir películas que valoren positivamente los usuarios.

Cabe señalar que toda la implementación ha sido realizada en Python. Este lenguaje de programación permite aprovechar sus múltiples librerías para trabajar con conceptos bastante interesantes. Gracias a Python se han aplicado políticas complejas y se han obtenido resultados reveladores.

Para explicar esta implementación se profundizará en los puntos claves en diferentes subsecciones. Por ello, primero se detallará la base de datos utilizada, después el sistema de recomendación y cómo funcionan las recompensas, luego las políticas y, por último, serán analizados los resultados y se expondrán las conclusiones.

### 9.1. Base de Datos

En primer lugar, hay que poner el foco en la base de datos que ha sido utilizada. Se trata de “MovieLens 1M”. Es una base de datos que se ajusta perfectamente a las necesidades de este sistema recomendador. Cuenta con un millón de valoraciones de películas y hay más de 4000 películas valoradas.

La base de datos es propiedad de GroupLens que se refiere al grupo académico de la Universidad de Minnesota. Dicho grupo ha desarrollado múltiples algoritmos en el área de los sistemas recomendadores, convirtiéndose así en un pionero en el campo.

Por otro lado, lo que hace de esta base de datos idónea para el trabajo es la información contextual asociada las valoraciones. Es decir, cada valoración de una película está relacionada con un usuario del que se dispone de sus datos personales. De dicho usuario se conoce su género, edad y trabajo. Debido a esto, la base de datos representa una gran oportunidad ya que permite analizar la información contextual de los usuarios y esto permitirá crear un muy buen sistema recomendador. Hay 6000 usuarios almacenados en este banco de datos, y se garantiza que cada usuario ha realizado al menos 20 votaciones.

De las características del usuario, el género se refleja con *F* de mujer (female en inglés) y *M* de hombre (male en inglés). La edad se muestra con números que indican lo siguiente: 1 significa menores de 18 años, 18 son personas entre 18 y 24 años, 25 es de 25 a 34, 35 es de 35 a 44, 45 son los que tienen desde 45 a 49, 50 es desde 50 a 55 y por último, 56 es para personas mayores de 56 años. La ocupación del usuario se denota con un número del 0 al 20. Cada número indica un trabajo o una rama, por ejemplo el 7 simboliza ejecutivo y el 1 académico o educador.

Por otra parte, de cada película se conoce su título y su género. El género de una película también se denomina categoría. Concretamente, una película puede pertenecer a los siguientes 18 géneros (están todos en inglés): *action, adventure, animation, children's, comedy, crime, documentary, drama, fantasy, film-noir, horror, musical, mystery, romance, sci-fi, thriller, war y western*. Además, hay que mencionar que una película puede formar parte de uno, dos, tres o cuatro géneros.

Es una base de datos perfecta porque aporta el contexto del usuario. Gracias a MovieLens 1M se pueden hacer recomendaciones a los usuarios conociendo sus valoraciones y su información contextual. Esto aporta un gran valor a esta implementación ya que se contará con información de miles de personas distintas que será usada para crear un sistema de recomendación altamente sofisticado. Se espera que con este volumen de información el modelo sea capaz de discernir patrones sutiles y tendencias en las preferencias de los usuarios.

Esta base de datos puede ser encontrada en [17] donde también se puede acceder a más detalles de la información recopilada en esta sección.

## 9.2. Iniciación

Tras conocer cómo es la base de datos que será utilizada, se explicará el sistema. A continuación, se enuncian algunos de los aspectos fundamentales del modelo propuesto: los brazos, el contexto y la función de recompensas,

- El contexto se determina a partir de la información contextual del usuario: género, edad y ocupación.
- Los brazos son las categorías de las películas. Hay 18 brazos que han sido mencionados en el apartado anterior.
- La recompensa tiene un valor en el intervalo [1, 5] y representa la valoración del usuario. Será normalizada en el siguiente rango [0, 1].
- La distribución de recompensas se considera que es IID, definida en apartados anteriores. Se trabaja con una distribución de recompensas de esta índole porque se parte de la premisa de que las preferencias de los usuarios no cambiarán. Los gustos de los usuarios serán constantes conforme pasa el tiempo. Es decir, habrá las mismas probabilidades de que a un usuario U le guste una película A al principio de la simulación que en la última ronda. Esto es conveniente para abordar esta implementación, y además se puede decir que es realista. Normalmente los gustos cinematográficos de un usuario, especialmente en relación con los géneros, tienden a ser bastante estables y no suelen experimentar cambios drásticos a lo largo del tiempo.

Hay  $T$  rondas, y en cada ronda se recomienda una película a un usuario. En cada ronda  $t$  se utiliza  $x_t$  para hacer referencia al contexto de un usuario. Sin embargo, el enfoque de cada ronda puede ser desde una perspectiva distinta, hay rondas de entrenamiento y rondas de evaluación, esto será explicado en el próximo apartado.

La función de recompensas es del siguiente modo. Dada una ronda, el sistema recomendador selecciona un brazo, género, para un usuario. Con dicho género de película el modelo recomienda una película al usuario. El procedimiento a seguir es el siguiente: se accede a la base de datos (MovieLens 1M) y se coge una lista de películas de esa categoría que haya votado el usuario. Estas películas están en la base de datos pero aún no se las ha sugerido el sistema recomendador. Para las películas de esta lista se conoce la valoración que ha dado el usuario. Se escoge la primera película de la lista obtenida de ese género para el usuario. Si el sistema recomendador estuviera haciendo una segunda recomendación de esta categoría a este usuario, el modelo selecciona la segunda película de la lista, si hiciera una tercera recomendación, la tercera película sería escogida... Tras elegir una película, el peso del brazo es actualizado según la recompensa obtenida. El brazo es el género elegido y se actualiza con la valoración del usuario a esa película en la base de datos real. Esta evaluación corresponde con la nota que ha puesto el usuario a esta película, y así lo recoge la base de datos de MovieLens.

Al recomendarle películas de esta forma, se obtiene la valoración real que ha dado ese usuario a una película en particular. Gracias a este mecanismo, las recompensas no son simuladas sino que representan realmente las preferencias de los usuarios. Por otra parte, la metodología llevada a cabo también se conoce como *acciones estructuradas* porque no solo se está eligiendo la categoría, que es el brazo, sino que también el sistema debe seleccionar una película. Consecuentemente, este sistema recomendador toma varias decisiones.

Las recompensas se caracterizan porque hay *retroalimentación parcial*. Como las películas tienen distintos géneros, cuando se recomienda una película, no se está recomendando únicamente una categoría, sino que se recomiendan varias. Entonces, a pesar de que se elige solo un brazo, el algoritmo juega varios brazos, que son los géneros asociados a la categoría. Por ello, hay que actualizar la recompensa para los brazos jugados, o también llamados brazos activados. Por lo tanto, se obtiene retroalimentación parcial porque son conocidas las recompensas de más brazos del que es elegido.

Por ilustración, considérese que en una ronda determinada se requiere efectuar una recomendación de película a un usuario con un contexto específico. Se imagina que el algoritmo en primera instancia elige el brazo *children's*. El sistema buscaría una película que haya votado este usuario en la base de datos MovieLens que sea del género *children's*. Cabe mencionar que al referirse al sistema recomendador se puede llamar como algoritmo, modelo o sistema. Una vez el modelo ha obtenido la película (suponiendo que el usuario tiene películas de ese género en la base de datos), comprueba si la película tiene otros géneros. Por ejemplo, si el modelo ha elegido *E.T.*, verá que también pertenece a las categorías de *drama*, *fantasy* y *sci-fi* además de *children's*. Si la recompensa es de 4, o 0.8 sobre 1, el sistema se actualizará para las cuatro categorías, los cuatro brazos, con la recompensa obtenida (0.8).

Si, por el contrario, no hay ninguna película de esta categoría seleccionada para ese usuario en MovieLens, el sistema pasa de ronda. No obstante, esto es poco probable ya que cada usuario ha votado más de 20 películas y cada una tiene varias categorías. Igualmente, se realizan las suficientes rondas como para obtener resultados representativos y concluyentes. Por esto, no es un problema que para ciertos usuarios no se pueda recomendarles una película en una ronda concreta. Estos son casos aislados y el algoritmo ya tiene bastante información para trabajar.

### 9.3. Funcionamiento

Para llevar a cabo esta implementación, se ha adoptado un enfoque fundamentado en el aprendizaje automático tradicional. Al haberse utilizado el algoritmo Exp4 para discernir entre políticas eficientes e ineficientes, es necesario asegurarse de que dichas políticas, o expertos, hayan pasado un entrenamiento previo para que ofrezcan recomendaciones de calidad. Se conocen como recomendaciones de calidad aquellas sugerencias que obtengan buenas valoraciones. De otra forma, las políticas no realizarían buenas recomendaciones si no son entrenadas. Por lo tanto, para garantizar un buen rendimiento del algoritmo, deben entrenarse las políticas al principio.

Concretamente, de las cuatro políticas que han sido implementadas, deben entrenarse las dos más complejas: el algoritmo de filtro colaborativo basado en vecindad y la red neuronal. Sin dicha fase de entrenamiento estas dichas políticas no serán capaces de realizar recomendaciones de calidad. Las otras dos políticas, que son las simples, actualizarán las recompensas medias de cada brazo durante esta etapa.

Debido a esto, hay dos etapas en la simulación: una primera fase de entrenamiento y una segunda fase de evaluación. Por ello, se sigue una perspectiva en el desarrollo similar a la de los algoritmos de aprendizaje automático. Hay que recordar que la base de datos contiene 6000 usuarios, y se ha optado por una forma tradicional según el aprendizaje automático de dividirlos en un 80% – 20%, 80% de los usuarios para entrenamiento y 20% para evaluación. Además, para poner el prueba al algoritmo con más ejemplos, se recomendará varias veces al 20% de usuarios, concretamente 4 veces a cada usuario. De este modo, será posible evaluar el algoritmo con más datos y realizará más recomendaciones. Serán unos mil usuarios aproximadamente y se les recomendará de manera circular, primero al usuario uno, después al dos... así hasta el último y, tras este, se repetirá con el usuario uno, luego el dos, y así sucesivamente. Han sido elegido arbitrariamente las 4 vueltas para tener más de 4000 recomendaciones y son consideradas más que suficientes. El único inconveniente es que su tiempo de ejecución es muy alto, tarda aproximadamente dos horas entre las dos fases.

Inicialmente, se trabajaba con una recomendación para cada usuario en la fase de entrenamiento. Sin embargo, como hay suficientes datos en la base de datos y es conveniente aprovechar esta información para entrenar a los expertos, se han recomendado 4 películas a cada usuario de entrenamiento. Así, se obtiene más información de utilidad para las políticas con el objetivo de mejorarlas y hacerlas eficaces.

La ventaja de separar en dos conjuntos de usuarios es que los expertos aprenderán con gran volumen de usuarios, y su contexto, y recomendarán a otros usuarios con una información contextual diferente. Se estudiará si realmente el sistema recomendador sabe sugerir correctamente a partir de la información que han extraído con los usuarios de entrenamiento, y será posible determinar si el algoritmo es capaz de generalizar desde los datos de entrenamiento. Esto significa que el propósito del sistema recomendador es extraer patrones de los usuarios de entrenamiento y establecer relaciones entre el contexto de dichos usuarios y los géneros que puede recomendar. De esta manera, tratará de adaptar la recomendación de una categoría al contexto de un usuario nuevo (género, edad y ocupación) con el objetivo de que el usuario proporcioné una buena valoración a la película. El sistema recomendador seleccionará géneros para nuevos usuarios, del conjunto de evaluación, y así se podrá saber si el modelo es capaz de hacer buenas recomendaciones para usuarios que no conocía, es decir, a los que no les había mostrado películas anteriormente.

Además, como durante la fase de evaluación se realizarán cuatro recomendaciones (cuatro selecciones de brazo y cuatro películas) para cada usuario, el sistema recomendador se asegurará de que no se muestre la misma película a un usuario. El sistema almacenará las películas que le ha recomendado con anterioridad, y así garantizará que le recomiende películas que no le había recomendado previamente. Esto mismo hará en la fase de entrenamiento con las películas recomendadas a este otro grupo de usuarios.

Hay que destacar un aspecto crucial para entender esta implementación. En la fase de entrenamiento aprenden las políticas con los usuarios de entrenamiento, su contexto y recompensa, pero el algoritmo Exp4 no se

utiliza en esta primera simulación. En la segunda fase, de evaluación, las políticas no aprenden porque ya han aprendido en el entrenamiento y serán consideradas expertas, y deberían saber hacer buenas recomendaciones. En esta segunda parte es Exp4 el que aprende a diferenciar cómo de buenas son estas políticas y actualiza sus pesos. Por ello, se estará midiendo no solo la eficiencia de las políticas, sino también la habilidad de este algoritmo para aprender y explotar las mejores políticas.

Han sido utilizadas dos estrategias diferentes en cada parte de la simulación. En la primera etapa, se hace énfasis en explorar muchos brazos, mientras que en la segunda se quiere maximizar la recompensa. La razón por la que el sistema desea explorar en la primera parte es porque quiere que las políticas aprendan del uso de muchos brazos, incluso los malos, para que dichas políticas tengan suficientes relaciones entre brazos, contextos y recompensas. Para realizar esto, el modelo ha aprovechado un algoritmo de los bandidos estocásticos que ha sido desarrollado en el trabajo y que además es la primera de las cuatro políticas, dicho algoritmo es el Épsilon-Avaricioso con un épsilon de 0.95. Debe recordarse que esta variable épsilon definirá la tasa de exploración del bandido, por lo que explorará prácticamente siempre. Esta política permitirá practicar una exploración exhaustiva en la fase de entrenamiento y esto permitirá probar todos los brazos en múltiples ocasiones. Las otras políticas aprenderán de los datos almacenados de cada ronda: la recompensa para los brazos activados y la información contextual asociada a esa recompensa. Por otro lado, en la segunda etapa, se pretende mejorar el rendimiento del algoritmo y de esto se encargará el algoritmo Exp4.

En último lugar, hay que seleccionar los hiperparámetros: la tasa de exploración y la tasa de aprendizaje para Exp4. Se ha tomado una tasa de exploración de 0 ya que se puede considerar que con la exploración en la fase de entrenamiento es suficiente. Los expertos tendrán la capacidad de elegir un brazo teniendo información de todos los brazos disponibles. Respecto a la tasa de aprendizaje, se toma un valor de  $\epsilon = 0,05$  porque el objetivo es que aprenda pero no con demasiada rapidez para que no deseche instantáneamente ninguna política y no castigue en excesiva medida. Por el contrario, el algoritmo Exp4 aprenderá consistentemente y también obtendrá buenas recompensas con la elección de este hiperparámetro. Esto ya fue estudiado en la sección 8. Sin embargo, en este caso al disponer de muchas rondas, una tasa de aprendizaje de este valor es una buena opción.

Cabe señalar que al elegir una tasa de exploración,  $\gamma$ , de 0, se evitarán tener errores con Exp4. Esto es debido a que si la tasa no es 0, habrá ocasiones en las que el Exp4 explorará y no hará caso de la recomendación de un experto. Si la tasa de exploración el algoritmo Exp4 es cero, en todas las rondas se dejará aconsejar por las políticas y nunca explorará por su cuenta. En este caso, siempre se dejará guiar por un experto porque se considera conveniente para obtener la mejor recompensa media. En la fase de entrenamiento ya se ha realizado suficiente exploración por lo que no es necesario tener una tasa de exploración mayor que cero, ya que se conoce información de todos los brazos. Al plantearlo así, se puede establecer que una política retorne una probabilidad de 0 de jugar un brazo. Esto significa que hay políticas que para un contexto dado nunca elegirán ciertos brazos, siempre y cuando se aplica una tasa de exploración de 0. Por el contrario, si se supone que hay una tasa de exploración mayor a 0, y en una ronda el Exp4 decidirá explorar por el valor  $\gamma$ , si en esa ronda se observará que todos los expertos tienen una probabilidad de 0 de elegir ese brazo explorado (elegido aleatoriamente por  $\gamma$ ), el valor  $p_t$ , que es la probabilidad de elegir ese brazo por todos los expertos, sería 0. Esto causaría problemas en los costes falsos porque los costes falsos son igual a la división de los costes, lo que se deja de ganar, entre  $p_t$ , y en este caso se estarían dividiendo entre 0:  $\hat{c}_t(e) = c_t(a_t)/0$ . Debe recordarse la fórmula que puede producir este problema del Exp4:

$$\hat{c}_t(e) = \begin{cases} \frac{c_t(a_t)}{\Pr[a_t=a_{t,\pi} | p_t]} & \text{si } a_t = a_{t,\pi}, \\ 0 & \text{en caso contrario.} \end{cases}$$

Por el contrario, en este modelo nunca va a ser 0 dicha variable  $p_t$  porque si un brazo es elegido significa que se hace caso a los expertos y al menos uno de estos tendrá una probabilidad mayor de 0 de elegir ese brazo:  $\forall t \in T : p_t > 0$ . Por ello, es imprescindible mantener el valor de la tasa de exploración a 0 por el modo en el que han sido realizadas las políticas para no obtener este error. Además, se considera que la elección de 0 de la tasa de exploración es una elección sensata y adecuada que permitirá optimizar las recompensas de este sistema.

## 9.4. Políticas

En esta sección se abordarán conceptos muy interesantes de las políticas con las que se han trabajado. Se procederá a explicar y profundizar en las cuatro políticas que han sido implementadas. Hay dos políticas más sim-

ples, cuya metodología ya se ha estudiado, y se han aplicado otras dos políticas más, que son más complicadas, y aportan mucho valor a este caso. Estas dos últimas políticas aparecen en las siguientes subsecciones 9.4.2 y 9.4.3.

#### 9.4.1. Políticas Simples

A continuación, se indagará las políticas más sencillas, que no por ello son prescindibles. A pesar de su simplicidad, van a aportar ideas muy interesantes y ayudarán a llegar a mejores conclusiones.

Primero, hay que comenzar con la política más simple: elegir el mejor brazo siempre. Se selecciona el mejor brazo de acuerdo a su recompensa media  $\mu(a)$ . Para desarrollar esta estrategia ha sido utilizado el Épsilon-Avaricioso, o Epsilon-Greedy en inglés, con un épsilon igual a 0. Hay que tener presente que el épsilon en esta política representa la tasa con la que el algoritmo explorará entre los brazos. Por lo que si se da un  $\epsilon = 0$ , nunca explorará, y esto asegura que siempre explotará el mejor brazo.

Durante la fase de entrenamiento la recompensa media de cada brazo será almacenada. De este modo, la política sabrá cuál es el brazo que tiene mejores recompensas. En la fase de evaluación, con Exp4, dicha política elegirá siempre el brazo que mejor recompensa haya tenido durante el entrenamiento. Por ejemplo, si el mejor brazo en la primera parte es la categoría *musical*, siempre seleccionará este brazo: *musical*.

Lo interesante de esta estrategia es ver cómo rendirá un algoritmo que solo elija el mejor brazo sin tener en cuenta el contexto. Realmente se está poniendo a prueba si la información contextual es de utilidad, ya que en caso de que esta primera política sea la mejor, se podrá deducir que el contexto no está mejorando el algoritmo. Debido a esto, con recomendar siempre el mejor género de películas ya se conseguirían buenas recompensas sin tener en cuenta la información contextual.

Esta alternativa resulta atractiva puesto que puede ser que el género cinematográfico más popular resulte ser siempre la mejor opción. Aunque un sistema de recomendación puede contar con géneros de nicho como *fantasy* o *sci-fi*, estos pueden tener menos éxitos con una audiencia más general. En lugar de ello, podría ser preferible optar siempre por recomendar el género que consistentemente obtiene mejores calificaciones promedio entre todo el público, en este caso serían los usuarios de entrenamiento. Por ejemplo, un género como el *suspense* podría ser una elección más acertada para una recomendación que busque satisfacer a una amplia gama de espectadores.

En segundo lugar, hay otra política simple: Épsilon-Avaricioso con un épsilon igual a 0.95 ( $\epsilon = 0.95$ ). Esta política realmente se centrará en explorar entre todos los brazos en la inmensa mayoría de las ocasiones. Las ventajas que ofrece implementar esta política son dos:

Por un lado, se saca partido de esta política para la fase de entrenamiento. Tal y como se ha explicado en el apartado anterior, dicha política es usada para explorar en la primera etapa de entrenamiento con el objetivo de que los expertos puedan recabar datos entre los 18 brazos con sus recompensas y contextos. Su propósito es que con la información recabada, las políticas puedan hacer buenas recomendaciones en la segunda parte al Exp4.

Por otro lado, también sirve como política para observar cómo de buena será una estrategia que es prácticamente aleatoria (explora casi siempre). Esto permitirá extraer conclusiones sobre si elegir aleatoriamente puede tener sentido. Se verá si un Épsilon-Avaricioso que explora será premiado por el algoritmo Exp4. Debido a esto, se puede decir que la primera política representaría la explotación y esta segunda la exploración.

Una vez que han sido expuestas las dos primeras políticas, las que se denominarían simples, se desarrollarán las políticas complejas. Hay un apartado dedicado para cada una de estas dos estrategias.

#### 9.4.2. Filtro Colaborativo Basado en Vecindad

Este apartado ahonda en la tercera política: filtro colaborativo basado en vecindad. Para desarrollar esta idea, la información ha sido obtenida de esta fuente [11]. Este algoritmo nace de la necesidad de algoritmos especializados en recomendar como los que se usan en páginas web, en tiendas online como Amazon, películas de Netflix o noticias de Google.

Por ejemplo, se presenta el caso de Netflix. Netflix se fundó como una empresa de alquiler de discos de vídeo digital (DVD) por correo, que luego se expandió a la entrega de contenido en línea, o también llamado streaming. Actualmente, el principal negocio de Netflix es proporcionar en línea películas, series y otros programas de televisión por medio de suscripciones. Netflix permite a los usuarios calificar las películas y programas en una escala de 5 puntos. Almacena las acciones de los usuarios en términos de lo que ven. Estas calificaciones se usan para hacer recomendaciones personalizadas, lo cual mejora la experiencia del usuario y puede ayudar a mejorar la lealtad y retención del cliente.

Para mantenerse como uno de los líderes en el mercado, Netflix ha probado con muchos sistemas de recomendación y ha desarrollado competiciones donde se ponían a prueba distintos sistemas como el que se tratará ahora. También se usaba una metodología similar a la que se desarrolla en este sistema, entrenaba sus algoritmos con unos usuarios de entrenamiento y luego medía su rendimiento con otros usuarios de evaluación, o test.

Además, cabe señalar que esta política se centra en analizar la información contextual del usuario. El objetivo del TFG es desarrollar bandidos contextuales multi-brazo teniendo en cuenta este tipo de datos. A pesar de que se pueden analizar las características de las películas (que son los ítems), este filtro colaborativo tendrá la perspectiva que ha sido comentada: recomendar en base al contexto del usuario. Por otro lado, hay más información de los usuarios en la base de datos que de los ítems para realizar el sistema recomendador, por lo que es conveniente aplicar el primer enfoque.

Para comenzar, se explicará lo que son los algoritmos de filtro colaborativo basados en vecindad, que también son denominados como algoritmos basados en memoria. Estos algoritmos parten de la premisa de que usuarios parecidos tienen un patrón de comportamiento similar. Conductas que son semejantes se pueden observar a la hora de valorar ítems, y por esto, los ítems parecidos obtienen valoraciones similares de usuarios con un cierto grado de semejanza. Se llama ítem a un objeto o artículo que el usuario va a valorar; en Amazon el ítem es un producto, como un libro, mientras que en el recomendador de Google el ítem es una noticia. Hay dos tipos de algoritmos de vecindad:

- Filtro colaborativo basado en usuarios. En este caso, las valoraciones de usuarios similares respecto a un usuario objetivo A son usadas para hacerle recomendaciones a dicho objetivo A. Las valoración que se espera que tenga A de un ítem es la valoración media ponderada del grupo de iguales. El proceso de ponderar la media de cada miembro del grupo de iguales será explicado posteriormente.
- Filtro colaborativo basado en ítems. En esta estrategia, como se pretenden hacer recomendaciones de un ítem objetivo B, deben determinarse un conjunto S de ítems, que son similares al ítem B. Luego, para predecir la valoración de un usuario A del ítem B, se tienen en cuenta las valoraciones de ese usuario A para el ítems pertenecientes al conjunto S. De esta manera, se verá cómo ha evaluado A objetos análogos al que se le quiere recomendar. La media ponderada de dichas evaluaciones es utilizada para predecir cuál va a ser la valoración del usuario A del ítem B.

Una vez que han sido observadas las dos principales clases de este algoritmo, se puede apreciar la principal diferencia. En el primero tipo, se recomienda en base a valoraciones que han dado usuarios similares mientras que en la segunda clase se predice la valoración del ítem de acuerdo a la valoración que ha dado ese mismo usuario a ítems similares.

Lógicamente, en este caso debe aplicarse con un filtro colaborativo basado en usuarios para explotar su información contextual. La idea es trabajar con un grupo de iguales de un usuario objetivo, que es aquel al que se le desea recomendar una película. Por ello, un grupo de iguales es creado y los miembros pertenecen a los usuarios de entrenamiento. Se usarán sus valoraciones para recomendar la mejor categoría posible de película a usuarios objetivos, que son miembros conjunto de usuarios de evaluación.

Además, hay dos formas de enfocar el algoritmo de vecindad:

- La primera forma sería predecir la valoración de la combinación de un usuario y un ítem. Es el método más simple y primitivo de los sistemas recomendadores. En este caso, es necesaria la valoración de un usuario, y se predice un ítem. La predicción de esta calificación se basa generalmente en las valoraciones del usuario de otros ítems y de las evaluaciones de otros usuarios a este ítem.
- La segunda alternativa es determinar los  $k$  ítems más próximos, también llamados  $top - k$  ítems, o a partir de ahora los usuarios más próximos, es decir, el grupo de iguales. Este enfoque opta por identificar los  $k$  usuarios o ítems más relevantes.

La implementación utiliza la segunda opción y gracias al grupo de iguales es posible hacer una recomendación a otro usuario en base a su contexto. El algoritmo elegirá un brazo u otro para un usuario en función del grupo de iguales. Este será el grupo de usuarios que más se puedan asemejar al usuario objetivo. Se toma 5 como el número de usuarios de un grupo de iguales.

El algoritmo desarrollado se basa en [11] y es una adaptación al modelo. Utiliza los usuarios más similares al usuario al que el sistema recomendador pretende recomendar. Para calcular la similitud entre dos usuarios, se usa el coeficiente de la similitud del coseno. Cabe señalar que el grupo de iguales van a ser del entrenamiento, y el usuario al que se le va a recomendar, pertenecerá a la evaluación. Así, se garantiza que en ningún caso el propio usuario pertenezca al grupo de sus usuarios semejantes. Esta estrategia permitirá agrupar a los usuarios que tienen un contexto parecido al usuario objetivo para realizar una recomendación. Por esto, ciertamente se estará evaluando la relación entre el contexto, los brazos y las recompensas. Si esta política es eficaz, se podrá afirmar que recomendar según las preferencias de usuarios con contextos parecidos (edad, trabajo y sexo) es una buena idea.

Tras haber explicado los fundamentos de esta política, se muestra el coeficiente de similitud del coseno y cómo se ha utilizado en el código:

$$\text{Similitud del coseno: } \text{CosSim}(u, v) = \frac{u \cdot v}{\|u\| \cdot \|v\|}$$

Aquí  $u$  y  $v$  son vectores,  $u \cdot v$  es el producto escalar, y  $\|u\|$  y  $\|v\|$  son las normas magnitudes de  $u$  y  $v$ , respectivamente:

$$\|u\| = \sqrt{u_1^2 + u_2^2 + \dots + u_n^2}$$

En este caso, los vectores representarían la información contextual de un usuario: sexo, edad y ocupación. Ha sido utilizada la distancia del coseno, que se define a continuación:

$$\text{Distancia del coseno: } \text{CosDist}(u, v) = 1 - \text{CosSim}(u, v)$$

donde  $\text{CosSim}(u, v)$  es la similitud del coseno entre  $u$  y  $v$ , como ha sido mostrada anteriormente. Los usuarios con menor distancia respecto a el usuario objetivo formarán el grupo de iguales. La explicación se muestra con el siguiente ejemplo. Si el usuario objetivo es un hombre de 45 años, educador, si hay otro usuario, perteneciente al conjunto de usuarios de entrenamiento, que tiene 50 años, es hombre y también educador, la distancia del coseno será muy pequeña entre estos dos usuarios. Por ello, será muy probable que ese usuario pase a formar parte del grupo de iguales usuarios. Con este y cuatro usuarios más, se formaría del grupo de iguales. Con dicho conjunto el algoritmo trabajara para maximizar la recompensa de dicho usuario objetivo. Se hace de la siguiente manera:

En primer lugar, es observado el contexto  $x_t$  de dicho usuario objetivo. Después, son obtenidos los  $k = 5$  usuarios que más se parecen usando la distancia del coseno. Para cada usuario del grupo de iguales se guardan las recompensas de cada brazo en un vector. Se calcula la recompensa media de cada brazo según las recompensas de los 5 usuarios. Y finalmente, es seleccionado el brazo que tenga mejor recompensa media.

El objetivo consiste en mostrar la categoría de película que haya tenido más éxito medio entre usuarios similares. Volviendo al caso anterior, si hubiera un varón de 45 años que es educador. El algoritmo de filtro colaborativo obtiene los 5 usuarios que más se asemejan. Después, calcula la recompensa media para cada brazo según las votaciones de estos 5 usuarios análogos. Para ilustrarlo, se podría suponer que las películas de *action* tienen una media de 0,61, las de *drama* 0,72, las de *film-noir* 0,83.... Tras haber calculado la media de cada categoría, el algoritmo escogerá la categoría que más media, en este caso *film-noir*, y recomendará una película de *film-noir* al usuario objetivo.

Realmente, se parte de la premisa de que buenos brazos en contextos anteriores ofrecerán altos rendimientos en el contexto actual de acuerdo al grupo de iguales. Esta hipótesis es puesta a prueba en el apartado de resultados donde se analizará si ha influido de manera determinante el contexto o simplemente con una política de

elegir el mejor brazo el algoritmo Exp4 obtendría buenas recompensas. Asimismo, podría suceder que el contexto sea un factor esencial para la generación de recomendaciones, pero esta política específica no resulte ser la más adecuada a tener en cuenta. Consecuentemente, recomendar según las preferencias del grupo de iguales puede no ser la estrategia más inteligente para este caso y, por consiguiente, habrá estrategias más acertadas para recomendar considerando el contexto.

Al adaptar esta política al algoritmo Exp4 se ha tenido que realizar una función que calcule las probabilidades de que un brazo sea elegido. Dicho método calcula las medias de los brazos. Si ante un contexto, la media de un brazo  $a_i$  es mayor que la media del resto de brazos devolverá 1 para el brazo  $i$  ya que será elegido con total certeza. Por el contrario, si la media no es la más alta, devolverá 0. Como se puede observar, seleccionará siempre el brazo que más media tenga para el grupo de iguales.

El pseudocódigo del algoritmo sería el siguiente:

---

**Algoritmo 9.15** *Filtrado colaborativo con algoritmo de vecindad.*

---

- 1: Al inicializar: Se almacenan los contextos de los usuarios de entrenamiento, con sus brazos activados y sus recompensas asociadas.
  - 2: **for** cada ronda  $t = 1, 2, \dots$  **do**
  - 3:   Se observa el contexto  $x_t \in X$  de un usuario.
  - 4:   Para todos los contextos de los usuarios de entrenamiento  $\forall x_i \in X_{train}$ , se calcula la distancia del coseno respecto a  $x_t$ :  $CosDist(x_t, x_i) = 1 - CosSim(x_t, x_i)$ .
  - 5:   Se obtienen los usuarios del grupo de iguales, los  $k$  contextos más similares a  $x_t$ , que corresponden a las  $k$  distancias más pequeñas dentro de los usuarios de entrenamiento ordenados por  $CosDist(x_t, x_i)$  de menor a mayor.
  - 6:   Se calcula la recompensa media para todos los brazos  $\forall a \in A$  de las valoraciones del grupo de iguales.
  - 7:   Se selecciona el brazo  $a_t$  con mejor recompensa media.
  - 8: **end for**
- 

Por último, debe mencionarse alguno de los factores por los cuales se ha decidido implementar este experto. Es un algoritmo que permite la personalización. Los sistemas de filtrado colaborativo proporcionan recomendaciones personalizadas basadas en el comportamiento de usuarios similares. A diferencia de otras técnicas de recomendación como el filtrado basado en el contenido, el filtrado colaborativo basado en usuarios no requiere ninguna información específica sobre los ítems, esto hace que encaje perfectamente con el modelo. Su flexibilidad también es un aspecto a señalar ya que por ejemplo, se puede establecer arbitrariamente el número de  $k$  vecinos así como otros ajustes del modelo. También, un punto a favor es que es una política que aporta mucho valor a la investigación. Esta estrategia se está trabajando desde una perspectiva diferente y permite estudiar algoritmos especializados en sistemas de recomendación.

#### 9.4.3. Red Neuronal como Bandido Contextual

Por último, este apartado se adentrará en la política final que es una de las ideas más estimulantes con la que se ha trabajado en esta implementación. Esta política que ha necesitado de muchas pruebas, hace uso de una red neuronal como si fuera un bandido contextual. Para implementarla, se utiliza Tensorflow, y Keras, en Python. Estas herramientas permiten trabajar con una red neuronal de manera flexible y adaptarla como un bandido. Hay que señalar que la idea es propia de los autores del trabajo. Dicha idea ha sido posible materializarla gracias a las fuentes mencionadas y a los conocimientos adquiridos con este trabajo.

La información con la que se ha trabajado para desarrollar la red neuronal y los conceptos teóricos detrás de la explicaciones están en [20]. Las redes neuronales se implementan adecuadamente gracias a dicha fuente. El conocimiento adquirido en este campo se traslada al marco de los bandidos multi-brazo. Estos modelos pueden aportar mucho valor por los siguientes motivos:

- Son algoritmos que ofrecen buenas alternativas porque tienen capacidad de manejar grandes cantidades de datos y características. En este caso, hay muchos usuarios y 1 millón de valoraciones de películas, por lo que se realizan muchas recomendaciones y esto es un punto a favor de las redes neuronales.

- Las redes neuronales tienen la habilidad de establecer relaciones interesantes entre múltiples características. En este sistema de recomendación, se tienen en cuenta el contexto como datos de entrada y las recompensas (valoraciones de los usuarios). Estos algoritmos tienen la capacidad de extraer complejas relaciones entre las variables con las que están trabajando para optimizar los resultados.
- La red neuronal se adapta convenientemente al modelo. Esto es debido a que como es necesario entrenar a los expertos, es la ocasión ideal para crear una red neuronal con los datos de entrenamiento. De este modo, para la fase de evaluación habrá una red totalmente entrenada y será posible evaluar su rendimiento.

De manera resumida, la aplicación de una red neuronal a un problema de bandido multi-brazo se hace para que esta red modele la relación entre el contexto y las recompensas de los brazos. La entrada a la red neuronal es la información contextual y la salida es una distribución de probabilidades sobre los brazos de manera que se elige un brazo en función de esta distribución. Esta política ofrecerá otro enfoque diferente para capturar las relaciones entre el contexto y los brazos. Al contrario que la tercera política, filtro colaborativo usando el algoritmo de vecindad, las redes neuronales calcularán distintas conexiones entre contextos, brazos y recompensas sin utilizar el grupo de iguales. Este contraste entre las dos políticas será de gran utilidad para analizar los resultados finales y entender cómo es más conveniente considerar el contexto en este caso de uso.

La idea ha sido utilizar una red neuronal multiclasa. Dicha red tiene 18 clases o neuronas en su última capa. Es decir, una clase para cada brazo. Para un contexto la red neuronal devuelve las probabilidades de lo bueno que será cada brazo. Esto significa que dado un contexto, la red neuronal ofrecerá probabilidades para cada brazo. La suma de todos los brazos dará 1, y hay 18 probabilidades, una por cada brazo. En las primeras implementaciones la red neuronal elegía aleatoriamente en esta función de probabilidad pero dado que las recompensas son IID (no hay un adversario), se obtendrán recompensas más altas si es seleccionado el brazo con mayor probabilidad. No es imprescindible escoger el brazo de acuerdo a la aleatoriedad porque las recompensas no son adversarias, sino que son IID. Por ello, tras varias pruebas, la red neuronal finalmente escoge siempre el brazo con mayor probabilidad.

Una vez explicada cómo actúa la red neuronal como bandido multi-brazo, se detalla cómo es exactamente esta red y por qué ha sido diseñada de una determinada manera. El trabajo se ha realizado con una red de tres capas densas, que están completamente conectadas. Dichas capas tienen 120, 70 y 18 neuronas respectivamente. Esto es debido a que se ha implementado un modelo complejo, con muchas neuronas. Según la documentación expuesta en [20] se ha optado por esta estrategia con el objetivo de crear un modelo complejo añadiendo regularización. Esto es debido a que al tener un modelo complejo (muchas neuronas), el modelo alcanza mucha varianza porque da demasiado peso a los parámetros del modelo y se produce un sobreajuste respecto a los datos de entrenamiento. Para evitar dicho sobreajuste, se usa la regularización. El propósito fundamental del modelo es realizar recomendaciones de calidad con los usuarios de evaluación, no con los de entrenamiento. El fin es crear un modelo que generalice bien. Por ello, todas las medidas que se han tomado han sido de acuerdo a este objetivo. Por ejemplo, se ha ajustado la tasa de aprendizaje, la regularización, las funciones de activación y la pérdida para garantizar esta generalización. Aunque aquí se comentan los hiperparámetros elegidos, se han hecho pruebas y simulaciones con múltiples opciones. Debido a esto, se mostrarán las elecciones finales que mejor optimizan la red. El proceso de selección de estos parámetros ha sido iterativo y ha requerido de muchas simulaciones para maximizar la recompensa.

La elección de la tasa de aprendizaje ha sido de 0,05. Este es un hiperparámetro crucial en las redes neuronales que determina cómo de rápido se quieren actualizar los pesos de este modelo. Si la tasa de aprendizaje fuera muy grande, el modelo puede converger muy rápidamente. Sin embargo, como es posible entrenar al modelo con muchos datos de entrenamiento, se ha establecido una tasa igual a 0,05 para que converga de manera adecuada y aprenda correctamente.

Para llevar a cabo la regularización, se ha utilizado una regularización L2 que previene el sobreajuste. Esta regularización añade una penalización a los pesos grandes del modelo para que no se ajuste demasiado a los datos de entrenamiento disminuyendo la varianza del modelo y mejorando la generalización. El valor elegido de la regularización,  $\lambda$ , ha sido 0,015 en la capa de entrada y la capa intermedia.

Respecto a las funciones de activación, *relu* ha sido la función de activación elegida para la capa de entrada y la oculta (la del medio). Esta función ayuda a hacer más rápido el proceso de descenso de gradiente, que es la forma en la que se entrena la red. En la capa de salida es utilizada la función de activación *softmax* que se usa a menudo en la clasificación multiclasa, que en este caso se realiza con 18 clases o brazos.

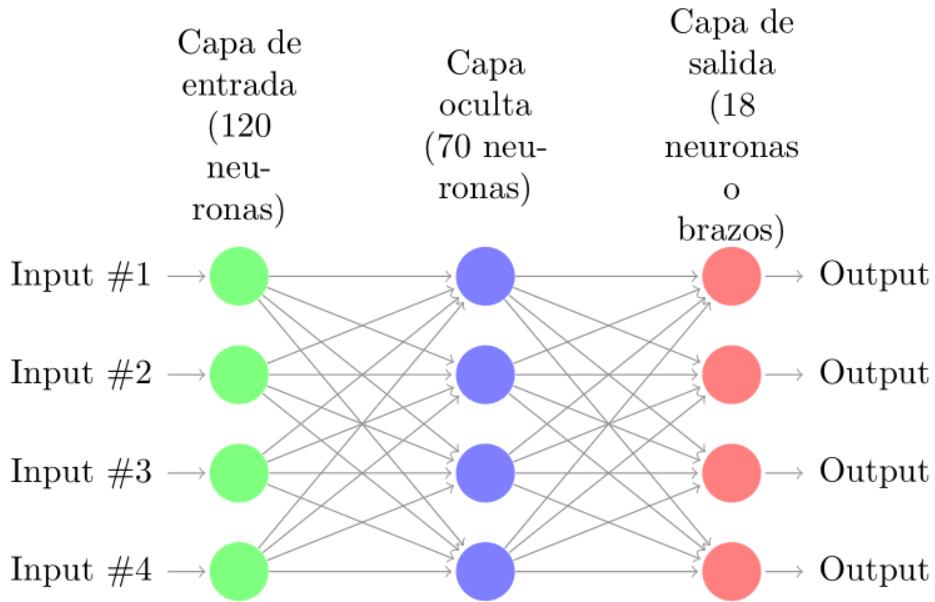


Figura 35: Red neuronal como bandido multi-brazo.

También hay que mencionar que se han usado unas épocas con un valor de 170. El objetivo es que la red neuronal entrene lo suficiente y pueda converger adecuadamente sin sobrepasarse con los datos de entrenamiento. Después de muchas pruebas y depuraciones alrededor de esta cifra se encuentran resultados correctos.

Finalmente, el modelo se compila con la pérdida de “categorical crossentropy” y el optimizador “Adam”. Categorical crossentropy es una función de pérdida comúnmente usada en problemas de clasificación multiclas. Adam es un algoritmo de optimización que se utiliza para actualizar los pesos de la red en función de los datos de entrenamiento. La tasa de aprendizaje se pasa a Adam que permite controlar la velocidad a la que el modelo aprende.

La figura 35 muestra cómo se vería la red neuronal de manera simplificada. Hay tres capas: La primera capa de entrada de 120 neuronas, la capa intermedia (u oculta) 70 neuronas, y la capa de salida de 18 neuronas (una para cada brazo). Como se puede apreciar, las entradas, son el contexto del usuario mientras que cada una de las neuronas de salida representaría un brazo. La figura es de gran ayuda ya que permite visualizar la estructura de la red.

Por ejemplo, durante el entrenamiento, si se tienen el brazo 2 con una recompensa de 0.8 para un contexto  $x_t$ , se entrena la red neuronal con el contexto como entrada. En la salida, al brazo 2 se le asigna una recompensa de 0.8, mientras que al resto de brazos se les da una recompensa de 0. De esta manera solo se actualizarán los brazos activados en cada ronda. Como cada película tiene varias categorías, cuando la red es entrenada con la valoración de un usuario, se hará varias veces. Una vez para cada categoría perteneciente a la película elegida. En cada una de estas veces será usada la misma recompensa para ese brazo y el mismo contexto como entrada. Si en el ejemplo mencionado también se activará el brazo 7, porque la película pertenece a ambos géneros, tanto el brazo 2 como el brazo 7 serían actualizados con una recompensa de 0.8. Igual que antes, el resto de brazos tendrían una recompensa de 0.

A continuación, se muestra un algoritmo que ilustra cuáles son los pasos a seguir de la red neuronal. Hay una exposición de estos dos pseudocódigos, uno para el entrenamiento y otro para la evaluación:

Al ver la estructura del código, los pasos que sigue la red neuronal quedan muy claros. Mediante este entrenamiento consigue establecer relaciones entre contextos y las recompensas asociadas a los brazos activados. Hay que tener en cuenta que una película tiene varias categorías y hay retroalimentación parcial, por lo que obtiene la recompensa para varios brazos, los brazos activados que corresponden a los géneros de la película seleccionada.

Aquellos brazos que tengan buenas recompensas serán elegidos con mayor probabilidad durante la siguiente etapa. Por lo tanto, este entrenamiento es fundamental y en función de cómo se realice la red neuronal ofrecerá unos buenos rendimientos o no.

Hay que mencionar que en el código la red neuronal es entrenada al final del entrenamiento, es decir, no

---

**Algoritmo 9.16** *Red neuronal en el entrenamiento.*

---

- 1: **for** cada ronda  $t = 1, 2, \dots$  **do**
  - 2:   Se observa el contexto  $x_t \in X$  de un usuario.
  - 3:   Se recibe el brazo elegido, que también es brazo activado,  $a \in A$  y los brazos activados  $a_i, a_j \dots \in A$ .
  - 4:   Para cada brazo activado  $a \in A$ , se entrena la red con  $x_t$  (como variable de entrada), y con la recompensa obtenida  $r_t$  como salida de ese brazo. El resto de brazos no activados,  $a \in A$  **and**  $\text{!activado}$ , les asigna una recompensa  $r_t = 0$ .
  - 5: **end for**
- 

entrena ronda a ronda como se hace en un bandido convencional que se actualizan los pesos tras observar una recompensa, sino que entrena tras haber pasado por todos los usuarios de entrenamiento. Esto se hace para ahorrar costes de tiempo y ejecutar el algoritmo más rápidamente. Por lo tanto, entrenará con todos los usuarios de una sola vez.

---

**Algoritmo 9.17** *Red neuronal en la evaluación.*

---

- 1: Al inicializar: Entrena la red neuronal con los usuarios de entrenamiento.
  - 2: **for** cada ronda  $t = 1, 2, \dots$  **do**
  - 3:   Se observa el contexto  $x_t \in X$  de un usuario.
  - 4:   Se calcula una distribución de probabilidades para los  $K$  brazos dado ese contexto  $x_t$ .
  - 5:   Se selecciona el brazo  $a_t$  con mayor probabilidad.
  - 6: **end for**
- 

Después de haber entrenado dicha red neuronal, se puede considerar que ya es una experta y el algoritmo Exp4 la utilizará como una de sus políticas. Como se ha recalcado antes, la red neuronal no entrena durante la evaluación del Exp4 porque ya es una experta y el objetivo es observar cuáles son las mejores y peores políticas. Por ello, en esta fase de evaluación la red no será actualizada.

En dicha etapa de evaluación, la red neuronal calculará una distribución de probabilidades dado un contexto teniendo en cuenta los datos con los que había sido entrenado anteriormente. Aquí entra en juego las relaciones que puede deducir entre los contextos y la recompensas. Las recomendaciones serán de calidad en función de la capacidad de la red neuronal para capturar la relación entre estas variables. Por otro lado, dado un contexto, la red neuronal tendrá la probabilidad de 1 de elegir un brazo si es el brazo que más probabilidades tiene. De este modo, el resto de brazos tienen una probabilidad de 0 de ser elegidos.

Además de estas funciones, la red neuronal también debe ser capaz de dar la probabilidad de elegir un brazo dado un contexto. Esto es necesario porque el algoritmo Exp4 necesita esta probabilidad para ajustar el peso de sus políticas por lo que se ha tenido que implementar este método. La red neuronal devolverá 1 si el brazo tiene la mayor probabilidad de ser el mejor y se elegirá con certeza, mientras que devolverá 0 en caso contrario.

Por último, se ha podido elegir entre dos estrategias para la red neuronal cuando selecciona un brazo a recomendar: elegir el brazo aleatoriamente de acuerdo a la distribución de probabilidades o seleccionar el brazo con mayor probabilidad. Para escoger la mejor opción y optimizar la red neuronal, se ha realizado la simulación de una prueba para el algoritmo Exp4 con ambas estrategias y se seleccionará la que mejor resultados de. La resultante será utilizada como la política de red neuronal y competirá contra la política del mejor brazo, la política Épsilon-Avaricioso que explora y la tercera política del filtro colaborativo basado en algoritmos de vecindad.

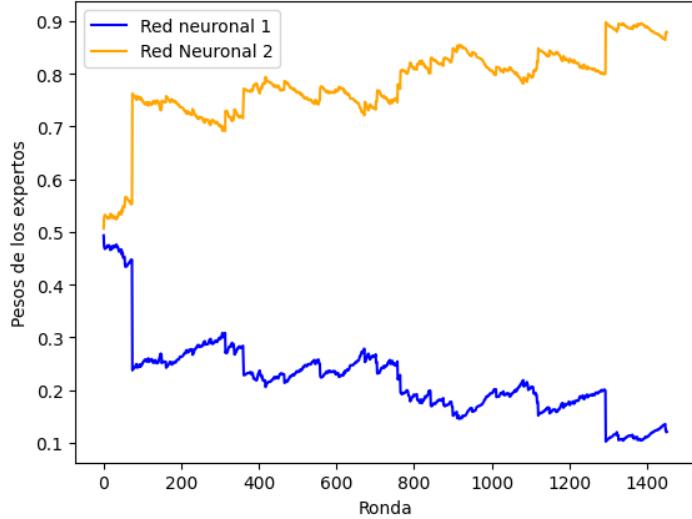


Figura 36: Comparación de redes neuronales.

Esta simulación se ha realizado para solo un 30 % de los usuarios pero los resultados son reveladores. La red neuronal 1 es la que elige el brazo aleatoriamente de acuerdo a una distribución donde cada brazo tiene una probabilidad de ser elegido según lo bueno que sea y la red neuronal 2 elige simplemente el brazo que mayor probabilidad tenga sin añadir aleatoriedad. Se observa en la figura 36 que la red neuronal 1 es premiada por el Exp4 y obtiene mejores pesos ya que sus recompensas son mejores. Al contrario, la red neuronal 2 consigue peores resultados y su peso va disminuyendo en contraste con la otra red neuronal. Debido a este análisis, en la simulación final, se utilizará una red neuronal que elija el brazo con mayor probabilidad sin tener en cuenta la aleatoriedad. Hay que recordar que esto no es un problema porque las recompensas son IID y significa que no hay un adversario. Esto corresponde con la sección 5.

## 9.5. Resultados

En este último apartado se analizan los resultados por medio de gráficas y se ofrecen las conclusiones oportunas respecto a cada figura. Antes de comentar los resultados finales, se expone una simulación inicial que es esclarecedora para desarrollar algunas hipótesis en este sistema recomendador. Los resultados estudiados corresponden a la fase de evaluación del Exp4 en cualquier caso.

### 9.5.1. Resultados Iniciales

Al realizar este sistema de recomendación se ha llegado a muchas conclusiones pero antes, hay unos resultado preliminares que deben ser comentados ya que se llega a una conclusión enriquecedora para el planteamiento en general.

En primer lugar, hay que señalar que las mejores políticas han sido siempre dos: la red neuronal y la política de explotación, de elegir el mejor brazo. La política de exploración (Épsilon Avaricioso con un épsilon de 0.95) no ha obtenido buenos resultados, como se podía esperar no es buena idea elegir casi siempre aleatoriamente una película. Sin embargo, lo que sí es sorprendente es el mal rendimiento de la política de filtro colaborativo. Se podían tener muchas expectativas puestas en este experto y no ha tenido ninguna relevancia en las simulaciones, alcanzando siempre pesos ínfimos y siendo descartada completamente por el Exp4.

Por otro lado, también hay que destacar el buen rendimiento de la política de elegir siempre el mejor brazo. Es algo que llama la atención ya que esta política no ha tenido en cuenta el contexto, sino que simplemente elige la categoría que mejor nota media haya tenido durante la fase de entrenamiento. Este aspecto es revelador ya que ha llegado a superar a una mala red neuronal. A continuación, se muestra el gráfico de una red neuronal, con unos hiperparámetros no apropiados, y consiguientemente, una mala red neuronal frente a la política del mejor brazo. En este caso la política de mejor brazo generalmente es la mejor estrategia para el algoritmo Exp4.

Esto expone unos resultados sorprendentes. Incluso al final de la simulación, la política del mejor brazo acaba ganando la partida completamente a la red neuronal. La gráfica en cuestión es la siguiente:

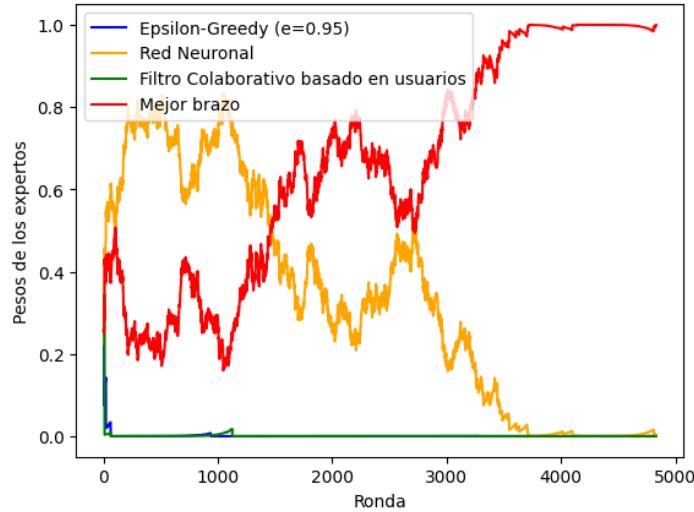


Figura 37: Primera evaluación Exp4.

Tal y como se ha comentado, los resultados no son los esperados: el filtro colaborativo es una total decepción y elegir siempre el mejor brazo parece que puede llegar a ser una alternativa a tener en cuenta ya que acaba siendo el experto favorito por el Exp4. No obstante, tras obtener estos resultados, es deducible que se pueden mejorar los hiperparámetros de la red neuronal y ver si realmente puede superar a todas las políticas con unos parámetros adecuados o por lo menos mantenerse como una buena opción a largo plazo. Cabe señalar que en esta simulación la red neuronal tenía 64 neuronas, 60 neuronas y  $K$  neuronas (18), una tasa de aprendizaje de 0,001 y unos épocas de 10, por lo que se ha intentando mejorar estos aspectos con el objetivo de conseguir la mejor red neuronal posible para ver si es una política viable. También es señalable que solamente la red neuronal es entrenada con una sola ronda de entrenamiento, con una recomendación para cada usuario de entrenamiento, por lo que no contaba con tanta información.

Por lo tanto, se puede concluir en estos primeros resultados que aunque pudiera parecer un buen experto, la política de filtro colaborativo es superada de manera superlativa por la política del mejor brazo, así como también pasa con la política del Épsilon-Avaricioso con un épsilon de 0,95 (una política que explora). Estas dos políticas ya han quedado en evidencia y se ha demostrado por el Exp4 que no son políticas dominantes en este sistema. Además, para una red neuronal que podría denominarse compleja, no se han conseguido buenos resultados frente a la política del mejor brazo, con regularización, muchas neuronas, optimización con Adam y 10 épocas. Es necesario modificar algunos de los aspectos de la red y debe ser entrenada más, con más épocas. Consecuentemente, se ha tratado de mejorar la red neuronal para que obtenga mejores resultados y comprobar si es capaz de rendir mejor y mantenerse como un alternativa viable.

### 9.5.2. Resultados Finales

Tras haber obtenido unos resultados preliminares en el apartado anterior 9.5.1, se procederá a detallar los aspectos tenidos en cuenta para mejorar la red neuronal con el objetivo de ver si una red más optimizada para este caso obtendría mejores resultados. Por otro lado, se parte de la base de que las otras dos políticas van a ser irrelevantes y el análisis debe basarse en la política de elegir siempre el mejor brazo y la red neuronal.

Se prueba con unos hiperparámetros que fueron lo suficientemente buenos para que la red neuronal superase a la política de mejor brazo. Esta red tenía unas 200 épocas, 120 neuronas en la primera capa, 60 en la siguiente, la tasa de aprendizaje ya se mantuvo 0,05. La última capa siempre debe ser de 18 neuronas. Al obtener tan buenos resultados, se aumenta aún más la complejidad de la red, sin embargo se obtuvo sobreajuste porque la red neuronal ya no superaba a la política del mejor brazo. Por lo tanto, ha debido generalizar mal. La conclusión a la que se llega es que se había producido un sobreajuste y había que encontrar un punto medio para que la red neuronal no sobreajustara y se consigan las mejores recompensas.

Los parámetros que finalmente han sido mantenidos de la red neuronal son los siguientes: Se ha utilizado una tasa de aprendizaje de 0,05, una tasa de regularización de 0,015, unos épocas de 170, unas neuronas de 120 en la primera capa, 70 en la segunda y 4 rondas de entrenamiento.

Los resultados obtenidos para Exp4 con esta red neuronal mejorada y el resto de políticas igual que antes son los siguientes:

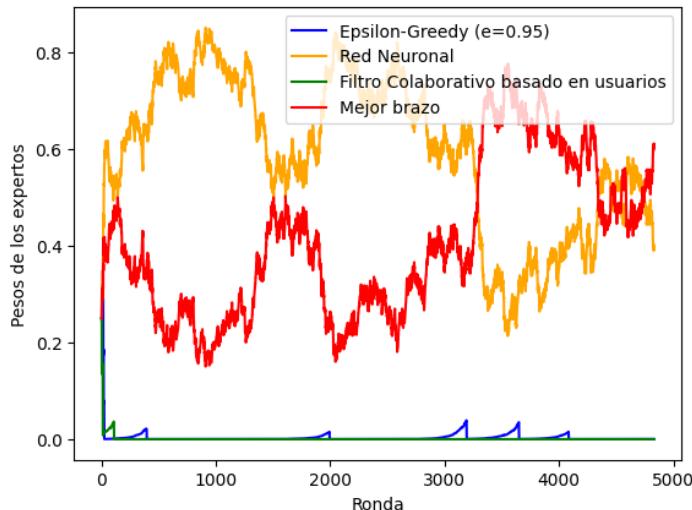


Figura 38: Evaluación Exp4.

Con esta figura 38, se aprecia que al optimizar la red se convierte en una política dominante para el Exp4 y que puede ayudar a mejorar la recompensa media. Por el contrario, se sigue observando que las políticas de un Épsilon-Avaricioso que explora o un filtro colaborativo siguen sin ser buenos expertos. Por otro lado, elegir el mejor brazo sigue siendo una alternativa que puede ser ventajosa para el algoritmo tal y como refleja la figura.

Durante tres cuartas partes de la simulación la red neuronal es la política más acertada. En las rondas finales se produce una alternancia entre la política de elegir el mejor brazo y la red neuronal. Debido a esto, se puede deducir que ambos expertos ofrecen recomendaciones buenas para el algoritmo Exp4 y tiene en cuenta a los dos. Posteriormente se ofrece la recompensa media de la simulación a lo largo de las rondas, pero su valor final es de 0.7536 por lo que se puede estar satisfechos con el desempeño del Exp4.

Respecto a la política de filtro colaborativo que utiliza el algoritmo de vecindad, se puede llegar a la siguiente conclusión: los resultados son claramente negativos, esto indica que dicha política no captura adecuadamente las relaciones entre el contexto y los brazos para maximizar la recompensa. Por ello, se puede deducir que en este caso particular no es buena idea recomendar de acuerdo con las preferencias del grupo de iguales. A pesar de que el grupo de iguales tiene un contexto similar al usuario objetivo, si el sistema recomendador tiene en cuenta las preferencias de dicho conjunto, los resultados son malos. Sin embargo, esto no sugiere que el contexto no aporte información valiosa. Aunque el algoritmo de filtrado colaborativo no haga recomendaciones de calidad, la red neuronal sí que obtiene buenas recompensas y tiene en consideración el contexto. Sin embargo, la política de la red neuronal crea relaciones distintas entre el contexto y los brazos seleccionados. Por lo tanto, con un enfoque diferente como el que utiliza la red neuronal si que se puede aprovechar la información contextual para mejorar las recompensas.

En definitiva, la estrategia óptima en el sistema recomendador es combinar la política de la red neuronal y la del mejor brazo. El Exp4 aprovechará ambas estrategias durante la simulación. Para ello, regulará los pesos de estas dos políticas castigando a la que ofrezca malas recompensas y dando oportunidades a la otra. Hará esto de forma indefinida conforme avanzan las rondas. Intercalar dichos expertos se supone que es la mejor alternativa. Por lo tanto, para algunos usuarios el sistema recomendador sugerirá el mejor brazo, una opción bastante segura, mientras que para otros usuarios recomendará un género según la recomendación de la red neuronal. Esta política tiene en cuenta el contexto y personalizará la recomendación adaptándose a los datos contextuales del usuario para ofrecer una recomendación única. Consecuentemente, el sistema recomendador encuentra un equilibrio entre sugerir siempre el mejor género y hacer recomendaciones personalizadas en base al contexto. Esto consiste en la coordinación del experto del mejor brazo y la red neuronal. La mayoría de veces el sistema recomendar aconsejará el brazo elegido por la red neuronal, que escoge dicho brazo adaptándolo al

contexto, y en otras ocasiones se opta por recomendar el mejor brazo.

Además se ofrece otra simulación, con parámetros de la red neuronal ligeramente distintos a los anteriores para ver las diferencias. Esta segunda simulación sí que tiene una red neuronal correctamente implementada y las recompensas medias son parecidas.

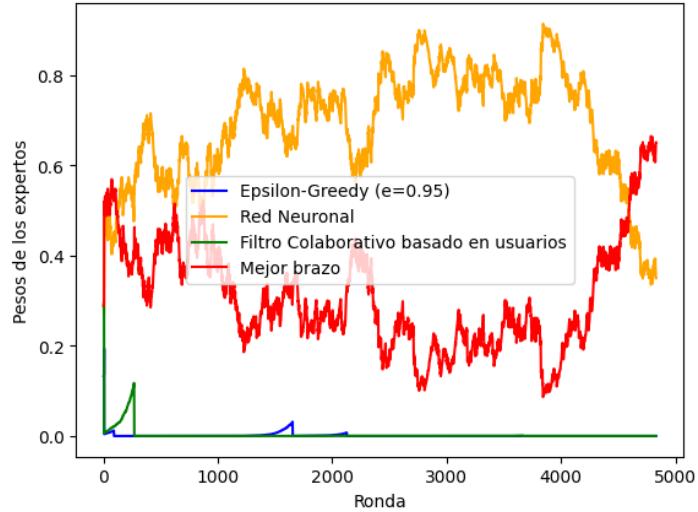


Figura 39: Evaluación auxiliar Exp4.

La figura 39 refuerza la posición que se ha defendido anteriormente. Tanto la política de la red neuronal como la de elegir el mejor brazo son las más útiles y ambas serán relevantes para el Exp4. También se observa el mismo patrón, la red neuronal es el experto dominante durante la mayor parte de las rondas pero la política de elegir el mejor brazo sigue siendo interesante y el Exp4 la tendrá en cuenta. Respecto a las otras dos políticas, los resultados observados son los mismos en todas las simulaciones.

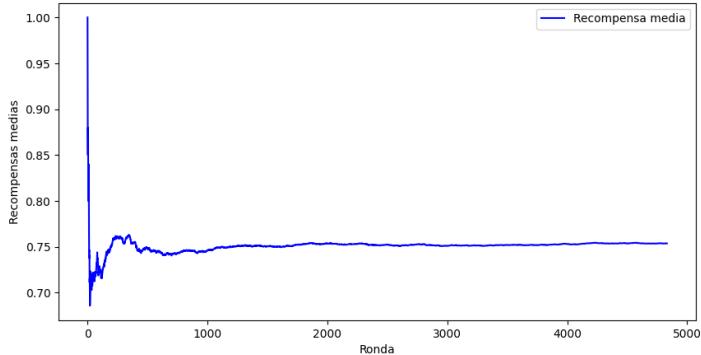


Figura 40: Recompensas medias.

Tras añadir la figura 39, se procede con el análisis de la simulación a analizar, que corresponde con 38. Es imprescindible analizar el gráfico convencional de recompensas medias ilustrado en la figura 40 ya que se aprecia cómo de bueno ha sido el algoritmo en general. En este caso, hay una recompensa media en la última ronda de 0,7536 (ha sido mostrada en pantalla). Esto equivaldría a valoraciones medias más próximas a 4 sobre 5. Puede ser considerado como un buen rendimiento porque siempre son notables las recomendaciones sugeridas por el sistema.

Por otro lado, también es añadido un histograma para ver cómo se ha dado el reparto de recompensas y analizar este punto más a fondo:

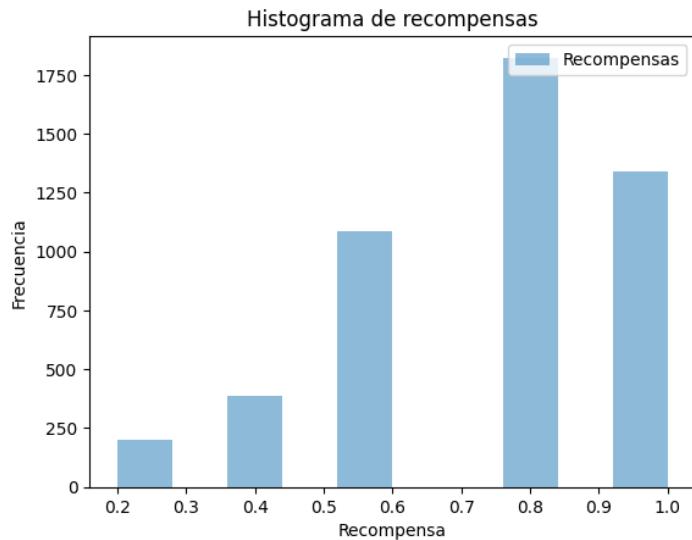


Figura 41: Histograma de recompensas.

Gracias a esta figura se ve que la mayoría de veces el algoritmo obtiene una recompensa de 0.8. También es señalable que la segunda recomendación que más veces hace este algoritmo es de 1, valoraciones de 5 estrellas, y también de vez en cuando recompensas de 0.6. Es difícil ver que el modelo consiga recompensas de 0.4 o 0.2, y son las menos representativas.

Los resultados son significativos. Las recomendaciones que hace este sistema recomendador son sobresalientes y la gran mayoría de veces acierta, o al menos obtiene una valoración suficiente. Esta figura 41 consolida este análisis de que el Exp4 está realizando un buen trabajo.

Se puede concluir esta sección afirmando que se cumplen los objetivos. Además, los resultados adquiridos son correctos no solo por el Exp4 que es capaz de discernir entre expertos buenos y malos, sino que también alcanzaron buenos rendimientos las políticas que han sido implementado. Concretamente, se ha observado la mejoría en la red neuronal tras haber realizado muchas simulaciones y adaptado muchos parámetros. Este proceso ha servido para aprender cómo ajustar esta red neuronal para este modelo. Por lo tanto, el sistema recomendador diseñado en este apartado puede considerarse eficiente.

### 9.5.3. Análisis Detallado

Después de haber expuesto las anteriores conclusiones, se mostrarán algunos gráficos para ilustrar hechos importantes sobre ciertos aspectos específicos. Estas figuras y su análisis siguen siendo parte de la simulación del Exp4 en la fase de evaluación que ya ha sido expuesta por la figura 38. A pesar de adentrarse en la estadística, el análisis presentado en este apartado es fundamental, no solo porque permite interpretar los resultados relevantes del modelo, sino que ofrece información clave que debe ser tomada en cuenta para realizar un sistema recomendador de este calibre. Con esta consideración, se analiza la eficacia de las políticas de acuerdo a un sistema recomendador de películas que pueden ser de gran utilidad para implementaciones venideras en sistemas recomendadores. Así, se adquiere una comprensión más estratégica de cómo desarrollar algoritmos, centrándose la atención en el estudio de las relaciones entre los brazos y las recompensas. El análisis permite identificar las características de los brazos que conducen a recompensas más altas, mejorando así el rendimiento del algoritmo. Por otro lado, se reconoce el papel crucial del contexto en la modelación de estos vínculos. Se comprende que el contexto puede afectar a las recompensas obtenidas. De esta manera, en lugar de considerar las decisiones de manera aislada, se da relevancia a las circunstancias específicas que rodean cada decisión.

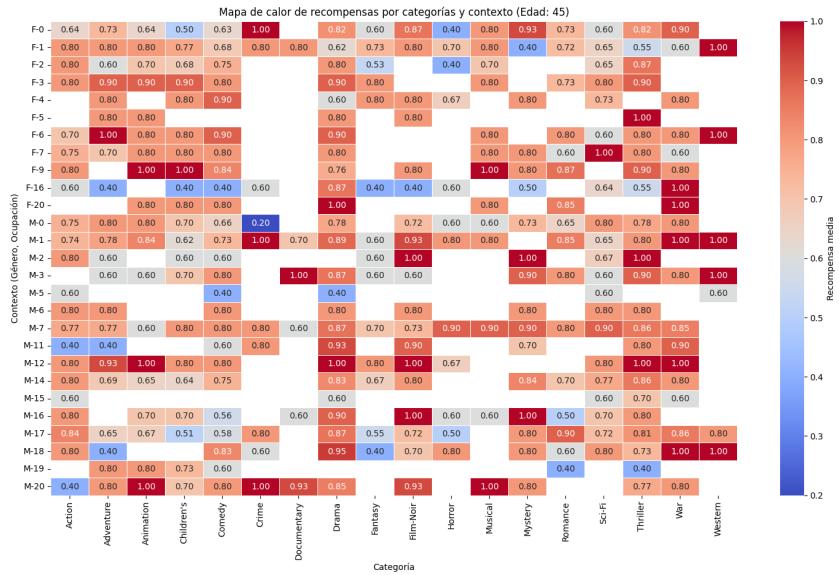


Figura 42: Mapa de calor de recompensas por brazo para personas de entre 45 y 49 años.

El mapa de calor que se está viendo representa las recompensas promedio de grupos de personas en el conjunto de datos. Cada grupo de personas se define por una combinación única de género, edad y ocupación. Un ejemplo serían mujeres de 56 años que sean directivas. Por lo tanto, no son calificaciones individuales, sino que es el promedio de un grupo específico. Este enfoque permite observar patrones y diferencias en las recompensas promedio entre diferentes grupos demográficos. Esto también es aplicable en otras figuras como 54, 55 o 43.

Este mapa sirve como representación de las recompensas medias para un grupo de edad específico. Es muy visual porque muestra las recompensas medias para todos los brazos que se han jugado y a la izquierda aparece el género y la ocupación de las personas. De este modo, se desglosan las recompensas por el contexto específico de cada grupo de edad. También se ha obtenido este mapa para el resto de grupos. Como este es lo suficientemente representativo para mostrar estos resultados, el resto de mapas serán adjuntados en el anexo 13.

Por otro lado, también se puede analizar la información contextual con el siguiente gráfico:

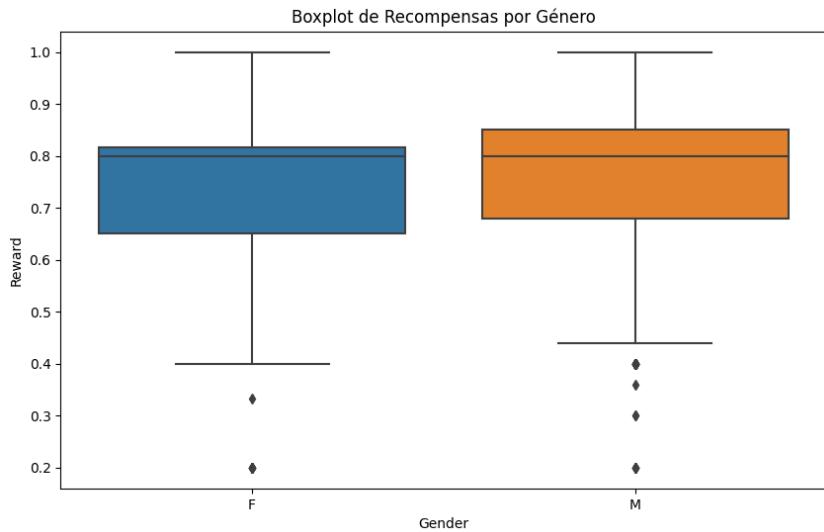


Figura 43: Recompensas por género.

La información de este gráfico puede ser muy útil también porque se aprecian cuáles son las diferencias en las valoraciones de las películas entre hombres y mujeres.

La figura 43 representa un diagrama de caja y bigotes, también llamado bloxplot. Este gráfico cuenta con distintos elementos: la caja muestra el rango intercuartílico (IQR) que es la distancia entre el primer cuartil (Q1) y el tercer cuartil (Q3). Estos son el 25 % y el 75 % de los datos, respectivamente. También se puede ver la

línea en la caja que es la mediana, medida que divide el conjunto en dos partes iguales. Los bigotes representan la dispersión de los datos fuera del rango intercuartílico. Los puntos que se encuentran fuera de los bigotes reflejan los datos atípicos.

Se observa que la caja y los bigotes para los hombres están más elevados en comparación con los de las mujeres. Esto indica que, aunque la recompensa promedio central puede ser similar para ambos géneros (tienen una mediana idéntica), la distribución generada de las recompensas obtenidas por el algoritmo para los hombres tiende a ser más alta que para las mujeres.

También se estudia un gráfico de violín de recompensas por brazo ya que permite entender muy bien cuáles son las recompensas de cada brazo. La figura en cuestión se presenta aquí.

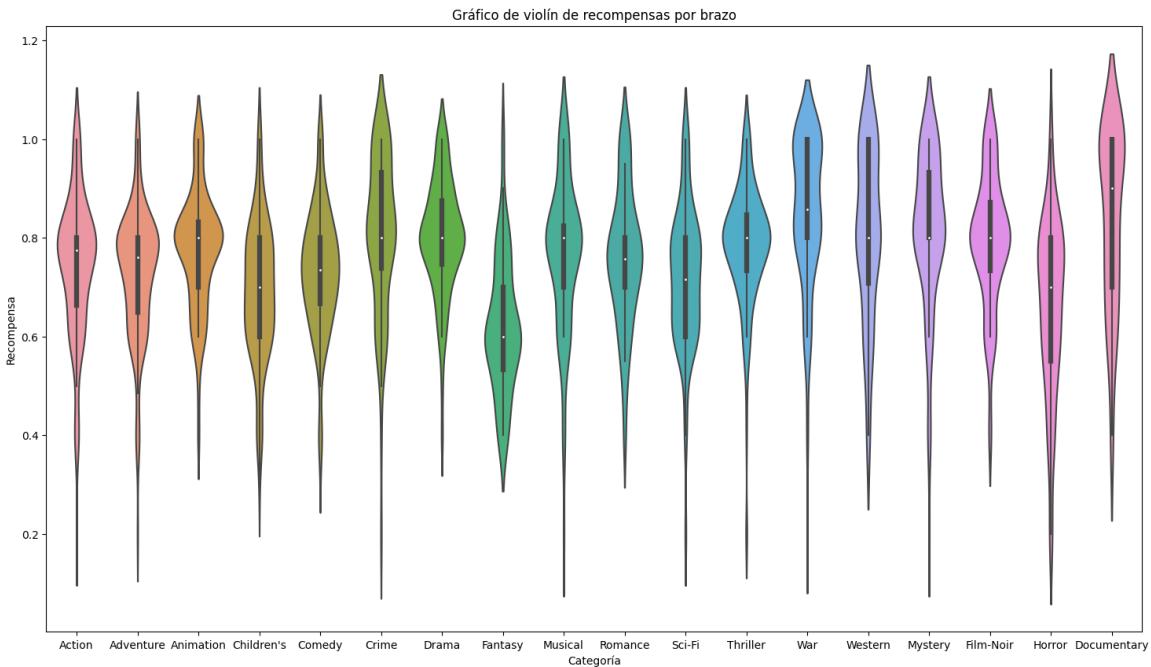


Figura 44: Gráfico violín de los brazos y sus recompensas.

En el gráfico violín se puede observar las recompensa obtenidas por el algoritmo para cada brazo, categoría. Este proporciona mucha información ya que muestra varias medidas estadísticas avanzadas. El círculo blanco corresponde con la mediana, el rectángulo negro es el rango intercuartílico, la diferencia entre el cuartil tres y el cuartil uno. También es posible ver la densidad, que es la anchura de la figura, de manera que la zona donde haya más densidad corresponde con la moda. Las líneas negras o también llamadas bigotes, representan el 95 % de los datos y reflejan los valores típicos. Con esta ilustración se puede observar qué recompensas medias suele tener cada brazo, categoría de película, y ver dónde se concentran los valores con el rango intercuartílico, la densidad y la mediana. Toda esta información se puede encontrar en [22].

Dicha representación es de gran utilidad para interpretar diferencias entre distintos brazos de manera precisa. De esta gráfica se puede extraer que hay géneros como el de *fantasy* que tienen una mediana especialmente baja. Esto denota que la mitad de las recompensas obtenidas para este género son inferiores a la mediana, que es aproximadamente 0,6. Por lo tanto, el sistema recomendador no ha obtenido altas recompensas habitualmente para este género y la mitad de las veces que se sugiera este género se puede esperar una recompensa menor de 0,6.

Por otro lado, también se aprecia que hay varios brazos que tienen una zona muy ancha cerca del 1 como por ejemplo la categoría *documental*. Se puede deducir que los usuarios que valoran el género de documental dan 5 estrellas con más frecuencia que una nota inferior cuando el algoritmo les ha recomendado estas películas. Debido a esto, dicho brazo *documental* es una buena alternativa para usuarios que acostumbran ver este género.

Además, se pueden analizar otros aspectos, como que en el brazo *romance*, su caja es muy pequeña en comparación a las demás, y esto hace ver que el 50 % de sus recompensas se encuentran muy concentradas en torno a 0,76 y 0,82 aproximadamente. Por el contrario, géneros como el de *children's* tienen cajas muy largas por lo que el rango intercuartílico se encuentra más disperso. Para este género su rango estaría entre el 0,6 y 0,8. Se ve una diferencia significativa respecto al de *romance*. Por ello, escoger el brazo *romance* proporciona

ciertas garantías de que su recompensa no variará mucho. Por otro lado, otras categorías como *children's* sí que tienen más volatilidad y por lo general se arriesga más al recomendar una película de este género.

Otro comentario que se puede hacer es que viendo la forma en la que se distribuyen las recompensas de un brazo es posible hacerse una idea de sus valoraciones. En este caso, se observa que el brazo *sci-fi* tiene la misma anchura desde 0,5 hasta 0,8 por lo que será igualmente probable obtener una valoración entre esos valores. Existen posibilidades parecidas de que un usuario de un 3 a dicha película que un 4. Además, como su caja es considerablemente larga, podría afirmarse que el género *sci-fi* no ofrece mucha certeza de que las recompensas sean estables y altas ya que los valores se encuentran muy dispersos. En definitiva, este brazo no es una alternativa muy segura.

Hay películas como las de *horror*, o terror, que tienen una caja larguísima y también unos bigotes de gran longitud. La dispersión de las recompensas para este brazo es muy exagerada y no ha proporcionado nada de seguridad a este sistema recomendador sugerir las películas de horror.

Por último, se pueden extraer patrones. Se aprecia que las tres primeras categorías: *action*, *adventure*, y *animation* tienen una moda cerca del 0,8 según las recomendaciones que ha hecho este sistema. Esto se ve ya que es la zona más ancha del violín es la que esta alrededor de este valor. Por esto, es muy probable que la mayoría de veces que el sistema recomendador elija alguno de estos géneros obtenga una valoración de 4/5.

## 10. Aportaciones individuales

En este apartado se comentan cuáles han sido las aportaciones individuales de cada miembro y cómo han influido en el desarrollo de la totalidad del proyecto.

El trabajado realizado se ha dividido principalmente en dos vertientes: el estudio teórico y las implementaciones prácticas. A pesar de que el desarrollo del código ha requerido de más tiempo y esfuerzo, no por ello el trabajo de investigación de los fundamentos de los bandidos ha sido menos significativo.

Por un lado, respecto al trabajo teórico, este ha sido estudiado, en colaboración, tanto por Iván como por Alejandro. No obstante, ha habido puntos en los que Iván ha sido la cabeza de la investigación teórica. Por ejemplo, el estudio de los bandidos antagonistas surgió de su trabajo, siempre en sinergia y con el valioso apoyo de Alejandro. En este segmento del trabajo fue esencial la comprensión detallada de algoritmos de considerable complejidad, como Exp3, Hedge y Exp4, junto a la incorporación de conceptos sofisticados que desempeñarían un papel crucial en el desarrollo del trabajo. En cuanto a los bandidos estocásticos, la aportación de ambos dos es significativa. Lo mismo sucede con la teoría de los bandidos contextuales: Lipschitz, pocos contextos y clase política, ya que han sido un claro ejemplo de trabajo en equipo. Alejandro ha liderado la implementación de sus respectivas implementaciones, contando en todo momento con el soporte de Iván.

En cuánto a las primeras implementaciones que aparecen en la memoria, para exponer casos de uso relativos a los bandidos de pocos contextos y de tipo Lipschitz, tanto Iván como Alejandro han participado conjuntamente en su desarrollo. Para el desarrollo del algoritmo de pocos contextos hubo que familiarizarse con el algoritmo UCB, que primero se explica dentro del ámbito los bandidos multi-brazo estocásticos, con las distribuciones de probabilidad usadas y con la forma en la que operan los bandidos. En la implementación de los bandidos contextuales lipschitzianos, se ha tenido que aprender una técnica básica de discretización (para aplicarla a los contextos) y también a demostrar la condición de Lipschitz mediante el estudio de propiedades matemáticas. Todo esto ha sido posible después de un estudio analítico de las referencias incluidas en dicho apartado. Estos primeros desarrollos son fruto de la colaboración de los autores de este trabajo, que han trabajado juntos desarrollando estos códigos. Consecuentemente, las explicaciones de este capítulo también corresponden a un trabajo en pareja.

Por otro lado, el desarrollo de los juegos contextuales ha sido una de las mayores dificultades a las que ha habido que hacer frente. Ha requerido un estudio exhaustivo de numerosos artículos, referenciados en el propio apartado. Dichos artículos son de una complejidad muy elevada. La implementación de la red del juego contextual ha requerido de muchísima participación conjunta. Alejandro se ha centrado más en la creación del código de este modelo e Iván en el estudio metódico de los algoritmos y sus requerimientos, aunque ambos han aportado en todo. Para dicha práctica se ha necesitado en varias ocasiones comprender algoritmos avanzados de C++ para adaptarlos al código desarrollado, como por ejemplo el algoritmo Yen que requería modificar la implementación de la red de ciudades y carreteras que había sido elaborada hasta ese momento. También, se han producido múltiples estructuras de datos y estudiado diferentes modelos de aprendizaje automático, posteriormente definidos en la memoria del trabajo, como el de regresión gausiana. Además, para trabajar este apartado, previamente era necesario entender el código en Python para poder trasladarlo a C++, requiriendo de un alto dominio de ambos lenguajes. En esta sección, los algoritmos a los que más tiempo se le ha dedicado son Hedge, GPMW y cGPMW.

Respecto a las implementaciones finales, cabe señalar que Alejandro se ha enfocado en el sistema recomendador de películas del último apartado, mientras que Iván se ha centrado en los bots de comercio. Sin embargo, ambos han estado constantemente conectados con los dos proyectos.

En los bots de comercio ha sido imprescindible una amplia comprensión de los bandidos contextuales con clase política, ya que estos eran la base del aplicativo. Concretamente, ha sido necesario desarrollar al máximo algunos aspectos de los bandidos contextuales y antagonistas, enfocados principalmente en el algoritmo Exp4. Una de las tareas más destacables es el análisis del impacto en los resultados de los hiperparámetros (tasa de aprendizaje y la tasa de exploración) y del horizonte temporal, qué son las variables que definen el comportamiento del algoritmo Exp4. También, se han tenido que estudiar aspectos financieros relevantes. Dichos fundamentos han permitido trasladar conceptos teóricos financieros a un caso de uso real.

Por último, en el desarrollo del sistema recomendador, se han tenido que estudiar múltiples algoritmos y diversos conceptos en una amplia gama de campos. Ha sido necesario aprender a trabajar con el Exp4 así como con sus políticas: redes neuronales, Épsilon-Avaricioso y el algoritmo de filtro colaborativo basado en vecindad.

Para ello, se ha tenido que estudiar, en cierta medida, desde las redes neuronales que son una instancia del aprendizaje automático, hasta algoritmos de recomendación como el filtro colaborativo especializado en sistemas recomendadores. También se ha trabajado con otros bandidos estocásticos como el Epsilon-Avaricioso. Este trabajo ha sido muy estimulante, aunque necesitado de mucho trabajo al haber tratado demasiado contenido. Como resultado, el sistema recomendador creado representa la puesta en práctica de múltiples conocimientos y mucho trabajo en distintos campos, requiriendo el estudio de múltiples algoritmos. Un punto que ha presentado mucha dificultad ha sido modelar una red neuronal como bandido multi-brazo contextual. Para esto, se ha tenido que estudiar a fondo el problema y adaptar esta solución, que ha resultado en algo muy interesante. Cabe destacar que la red neuronal ha requerido de un gran esfuerzo para optimizar sus parámetros. Por último, también ha sido necesario aprender a adaptar la información de una base de datos para que sea tratable por el modelo. Los resultados han sido analizados por medio de la estadística. Estudiar los resultados estadísticamente ha sido un aspecto crucial, no solo para entender un sistema recomendador, sino también para que los resultados obtenidos en este sistema puedan ser considerados por implementaciones venideras.

En definitiva, ambos autores han realizado un esfuerzo conjunto durante el desarrollo de todo el trabajo. Los dos han tocado todos los puntos involucrados en este proyecto. Se ha conseguido sacar partido del buen equipo formado, generando así una gran oportunidad para profundizar en este complejo aspecto del aprendizaje automático. Fruto de esta conexión ha surgido un equipo dinámico en el que los roles han ido variando con el tiempo gracias a la versatilidad de sus integrantes. Además, sendos autores han trabajado en todas las fases de desarrollo de las aplicaciones prácticas. Esto ha permitido exprimir al máximo sus virtudes y aprovechar la sinergia creada para construir un proyecto muy sólido, que aporta mucho valor debido a su complejidad y que conlleva mucho esfuerzo por detrás. Debido a esto, se puede afirmar que ha salido a relucir un gran trabajo de manera exitosa, ofreciendo muchas implementaciones y estudios de suma relevancia en el campo de los bandidos multi-brazo y, consecuentemente, aportando conocimientos al aprendizaje por refuerzo.

## 11. Conclusiones

Este trabajo cumple con creces las expectativas iniciales y los objetivos planteados. Por un lado, se trabajan adecuadamente los conceptos teóricos fundamentales de los bandidos y sus algoritmos, y se hace énfasis en los estocásticos y antagonistas para abarcar correctamente el marco de los bandidos contextuales. Por otro lado, los bandidos contextuales han sido estudiados en profundidad incluyendo el extenso y exhaustivo desarrollo de múltiples aplicaciones prácticas. Esta puesta en práctica de los bandidos contextuales cubre diferentes ámbitos y puede ser un punto de partida para el despliegue de aplicaciones reales en la sociedad actual. Además de haber desarrollado algoritmos en distintos campos, también se expone su metodología al detalle para hacerlos comprensibles.

Con esto en consideración, los resultados obtenidos son más que satisfactorios. Han salido a relucir las ventajas de los bandidos multi-brazo contextuales ya que son ciertamente útiles en diversas aplicaciones prácticas. En este trabajo hay diferentes implementaciones que cubren el ámbito financiero y de las inversiones y también aspectos fundamentales del mundo digital como los sistemas recomendadores. Estos temas tienen un claro impacto en la sociedad moderna. Además, se ha estudiado una variante en la que los algoritmos trabajan en un juego contextual dando lugar a otro posible uso de estos bandidos. Debido a su versatilidad en una amplia gama de campos y su escalabilidad para desarrollar construcciones de software, que no son triviales y permiten explorar problemas muy complejos, los bandidos contextuales son una rama tremadamente significativa del aprendizaje por refuerzo. Permiten obtener soluciones pragmáticas a problemas del mundo real encontrando un correcto equilibrio entre la exploración y explotación para obtener buenos rendimientos que reflejan la eficacia de estos algoritmos. Esto ha quedado demostrado de manera notoria a través de las implementaciones ofrecidas en este trabajo.

## 12. Conclusions

This work more than fulfills the initial expectations and the stated objectives. On the one hand, the fundamental theoretical concepts of bandits and their algorithms are adequately worked out, and emphasis is placed on stochastics and adversarial bandits to properly cover the framework of contextual bandits. On the other hand, contextual bandits have been studied in depth including the extensive and exhaustive development of multiple practical applications. These implementations of contextual bandits cover different domains and can be a starting point for the deployment of real applications in today's society. In addition to having developed algorithms in different fields, their methodology is also exposed in detail to make these algorithms understandable.

With this in consideration, the results obtained are more than satisfactory. The advantages of contextual multi-armed bandits have come to light as they are certainly useful in various practical applications. In this work, different implementations are covering the financial and investment domain and also fundamental aspects of the digital world such as recommender systems. These topics have a clear impact on modern society. In addition, a variant has been studied in which the algorithms work in a contextual game giving rise to another possible use of these bandits. Because of their versatility in a wide range of domains and their scalability for developing software constructs, which are non-trivial and allow the exploration of very complex problems, contextual bandits are a tremendously significant branch of reinforcement learning. They enable the discovery of pragmatic solutions to real-world problems by finding the right balance between exploration and exploitation to obtain good performances that reflect the effectiveness of these algorithms. This has been notoriously demonstrated through the implementations offered in this work.

## 13. Anexo

### 13.1. Depuración de la Evolución de Pesos de los Bots con Probabilidades Individuales de Elección de Brazo de 100 % y 0 %

La depuración “casera” comentada en la sección 8.3.1 se presenta a continuación. Además, la figura 45 vuelve a exponer la evolución de los expertos, también reflejada en dicha sección.

**Ronda: 292**

#### Cálculo de variables

Brazo elegido por el algoritmo: P.LARGA

Recompensa obtenida por el algoritmo = 0.1

Coste para los bots que han recomendado el brazo P.LARGA =  $1 - 0.1 = 0.9$

Probabilidad Individual de cada bot de elegir el brazo que elija = 100 %

Probabilidad Conjunta entre los bots de elegir P.LARGA =

$$= 100 \% * 1.42\text{e-}008 + 100 \% * 4.86\text{e-}259 = 1.421292709769473\text{e-}08$$

#### Obtención de costes estimados

Prob[Elegir Bot PCortas] = 2.76e-252 | Elige brazo: MANTENER | Coste estimado: 0.0

Prob[Elegir Bot PLargas] = 1.42129271e-008 | Elige brazo: P.LARGA | Coste estimado: 63322635.36

Prob[Elegir Bot PCambiantes] = 4.86e-259 | Elige brazo: P.LARGA | Coste estimado: 63322635.36

Prob[Elegir Bot Cruce de MM] = 8.10e-002 | Elige brazo: MANTENER | Coste estimado: 0.0

Prob[Elegir Bot RSI] = 9.96e-004 | Elige brazo: MANTENER | Coste estimado: 0.0

Prob[Elegir Bot MACD y BB] = 2.71e-005 | Elige brazo: MANTENER | Coste estimado: 0.0

Prob[Elegir Bot Todo] = 9.18e-001 | Elige brazo: MANTENER | Coste estimado: 0.0

#### Actualización de pesos y de la distribución de probabilidad

Nuevos Pesos de los bots: [1.e+250, 1.e-001, 1.e-001, 1.e+250, 1.e+250, 1.e+250, 1.e+250]

Nuevas Probabilidades, respectivamente: [2.e-001 2.e-252 2.e-252 2.e-001 2.e-001 2.e-001 2.e-001]

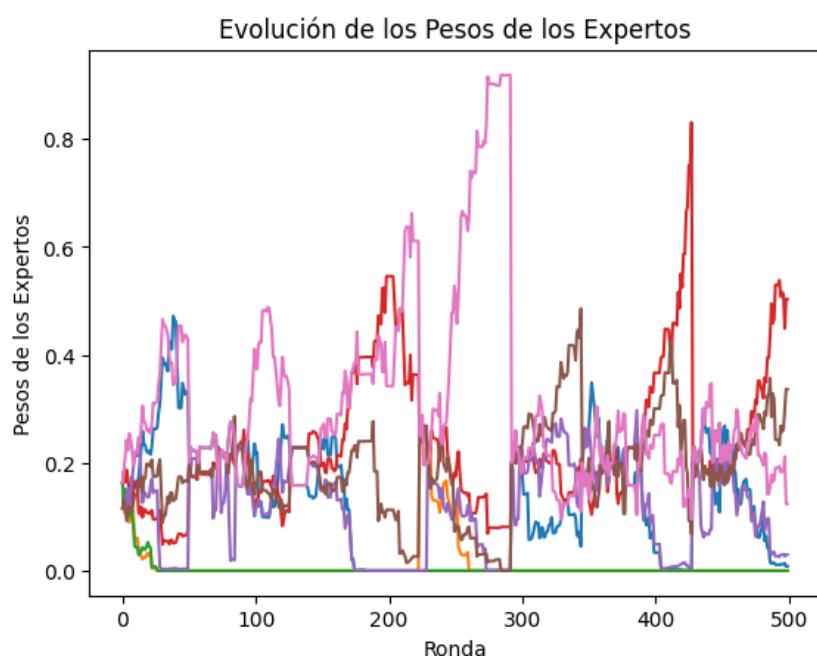


Figura 45: Evolución de los Pesos de los Expertos con  $\gamma = 0.2$ .

Como se puede observar en la depuración, en la ronda 292, el algoritmo ha fallado al elegir el mismo brazo

que han recomendado dos de los tres bots con menores probabilidades de ser escogidos. Esto ha provocado que la probabilidad conjunta de elegir dicho brazo sea bajísima. Por consiguiente, el coste estimado para los bots que han sugerido la opción de comprar ha resultado ser elevadísimo. Lo normal es obtener costes estimados entre los valores 0 y 5, y en este caso el valor es 63322635.36.

La obtención de tan elevados costes estimados por parte de los bots de comercio expertos en posiciones largas y posiciones cambiantes ha provocado que su probabilidad de ser elegidos en la siguiente ronda sea reducida a prácticamente 0. También, ha conllevado la distorsión de las probabilidades de elegir el resto de bots, repartiendo entre estos la probabilidad total restante de forma igualitaria. Concretamente, como son siete bots en total, y dos de ellos han recibido una probabilidad de casi 0, a los cinco restantes se les ha otorgado una probabilidad cercana al 20 %. En la figura 45, se puede confirmar esta situación al contemplar que los pesos de estos cinco bots, tras la ronda 292, parten desde el mismo punto (0.2 en el eje de las ordenadas). En cambio, los pesos de los bots perjudicados adquieren el valor más bajo, sin poder remontar la situación en lo que resta de rondas.

### 13.2. Resultados Sistema Recomendador Películas

Primero se ofrecen otras dos simulaciones del Exp4 con las mejoras hechas en la red neuronal para que se puedan observar otras ejecuciones como referencia. La red neuronal y elegir el mejor brazo son siempre las mejores políticas y al mostrar otras pruebas se confirma esta teoría.

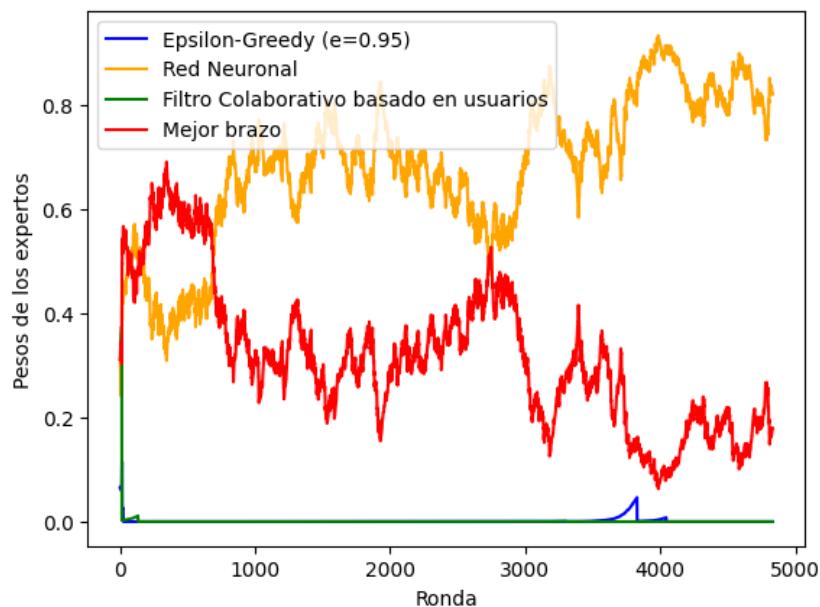


Figura 46: Simulación auxiliar 1.

En esta figura 46 se había intentando usar más neuronas: 120 en la primera capa (igual que antes) y 73 en la segunda. Hay una tasa de regularización de 0.015 en ambas capas y son aumentadas las épocas hasta 200.

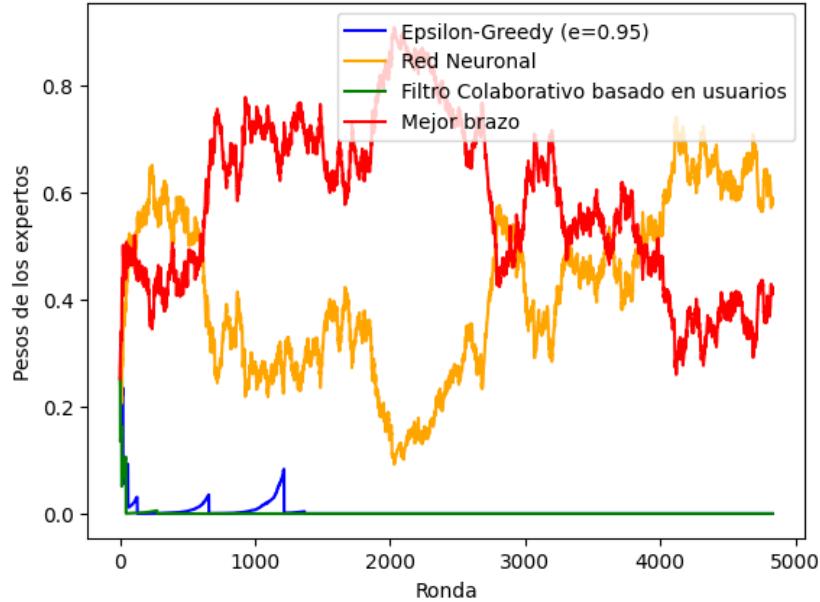


Figura 47: Simulación auxiliar 2.

Esta segunda simulación auxiliar se realizó con los parámetros finales de la red neuronal, y puede servir para ver cómo difieren los resultados según lo que se equivoquen los expertos: la red neuronal y la política del mejor brazo. En todas las simulaciones se puede apreciar que la red neuronal tiene un buen comportamiento y es una política a tener en cuenta siempre.

Además, tras haber ofrecido otras simulaciones, se muestran algunos de los resultados obtenidos en el sistema recomendador de películas que hace uso de un bandido contextual de clase política. Los mapas de calor para los grupos de edad faltantes son presentados a continuación:

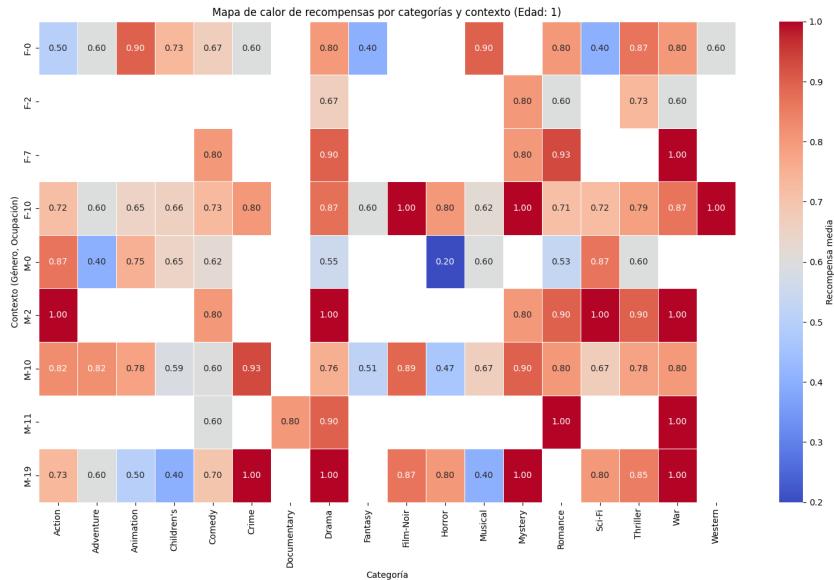


Figura 48: Mapa calor de recompensas por brazo para personas de menos de 18 años.

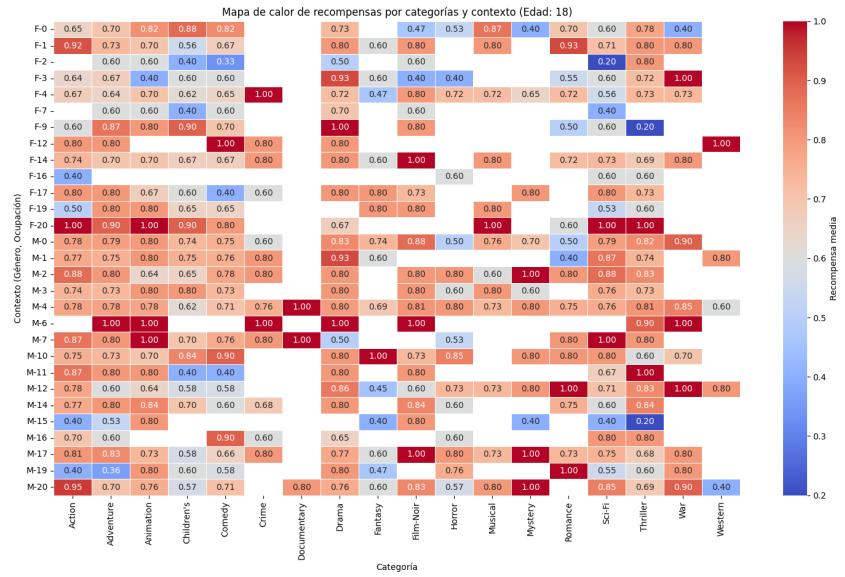


Figura 49: Mapa calor de recompensas por brazo para personas de entre 18 y 24 años.

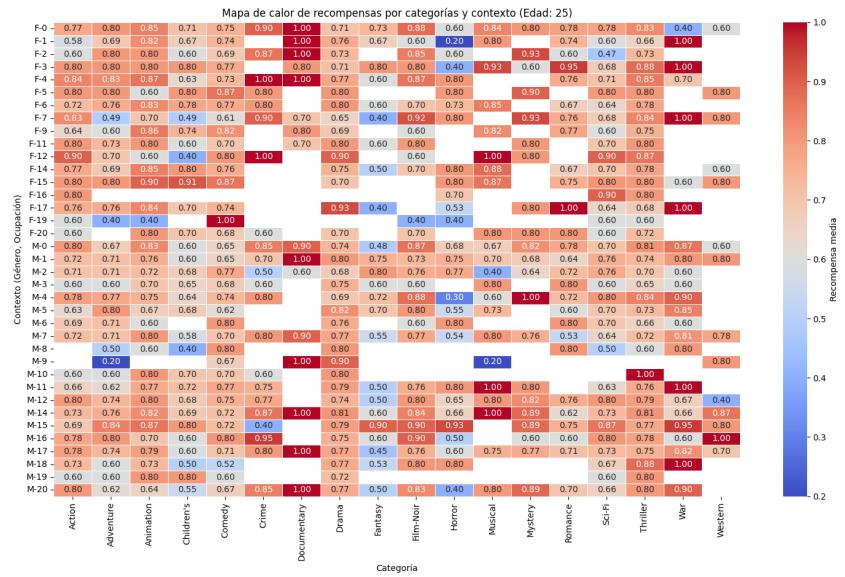


Figura 50: Mapa calor de recompensas por brazo para personas de entre 25 y 34 años.

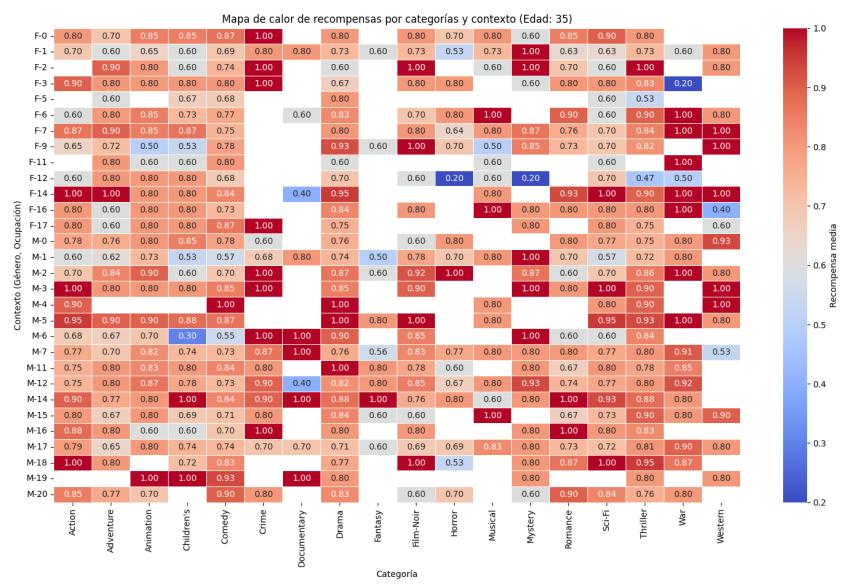


Figura 51: Mapa calor de recompensas por brazo para personas de entre 35 y 44 años.

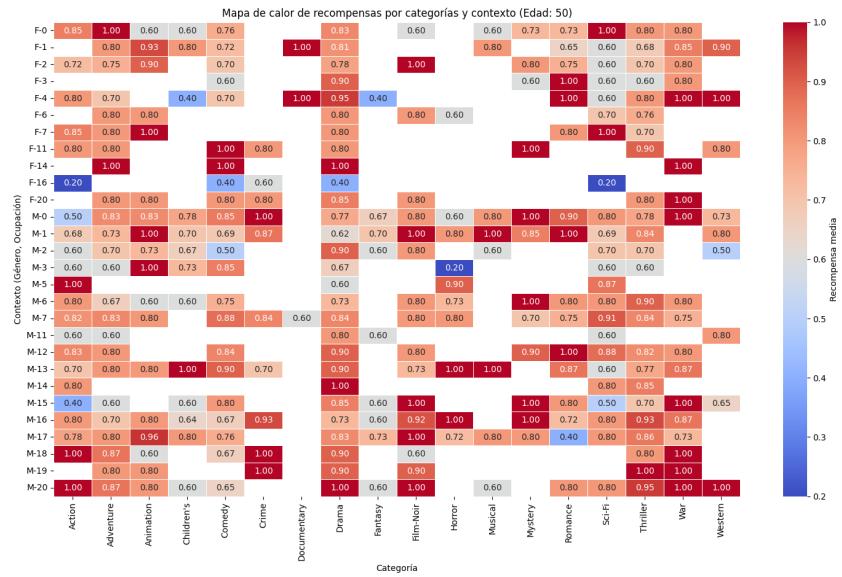


Figura 52: Mapa calor de recompensas por brazo para personas de entre 50 y 55 años.

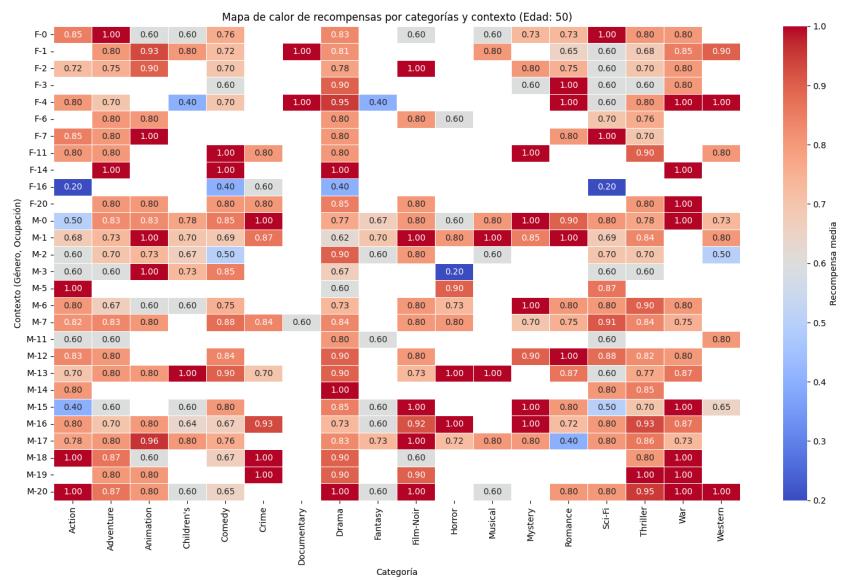


Figura 53: Mapa calor de recompensas por brazo para personas de más de 56 años.

También se adjuntan otras figuras auxiliares 54 y 55 que no se añadieron en la sección 9:

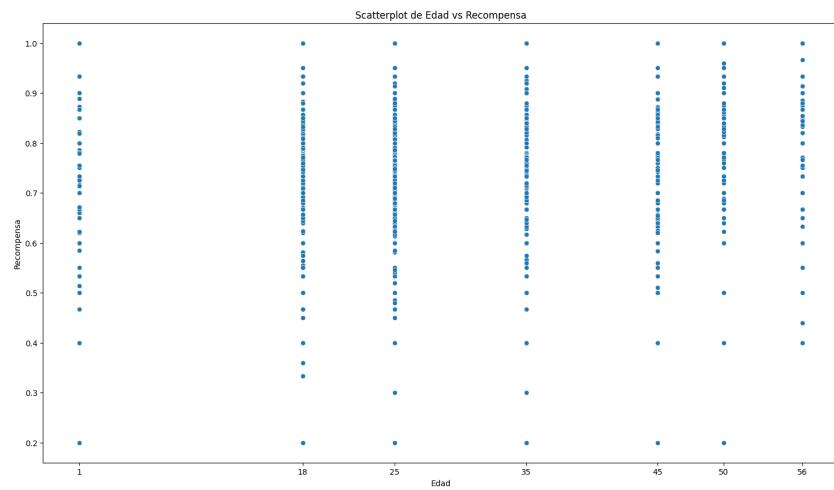


Figura 54: Recompensas para la cada grupo de edad.

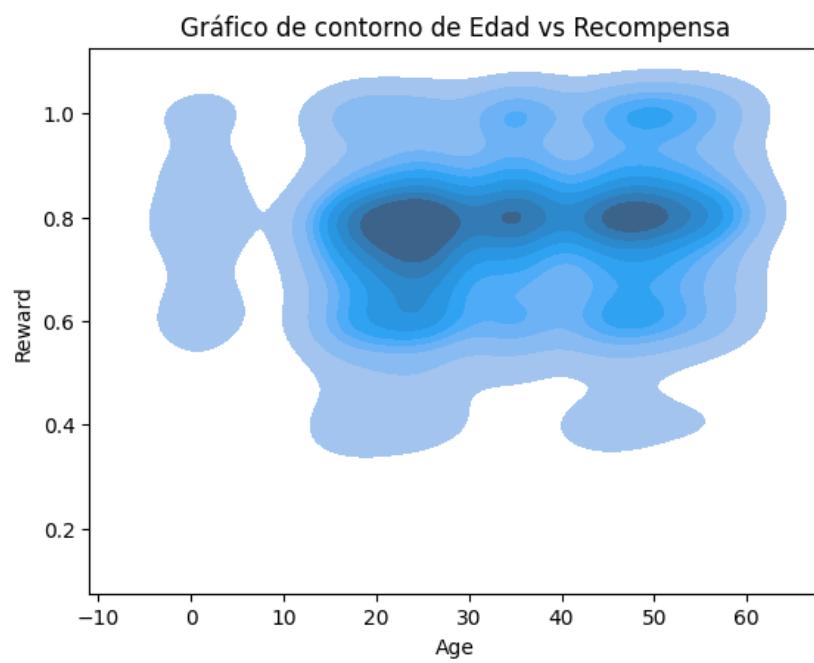


Figura 55: Gráfico de contorno de la edad y la recompensa.

## Referencias

- [1] W. Thompson. Sobre la probabilidad de que una probabilidad desconocida supere a otra en vista de la evidencia de dos muestras. *Biometrika* 1933.
- [2] Aleksandrs Slivkins. *Introduction to Multi-Armed Bandits*, 2019.
- [3] Tor Lattimore and Csaba Szepesvári. *Bandits Algorithms*. Cambridge University Press, 2020.
- [4] Sébastien Bubeck and Nicolò Cesa-Bianchi. *Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems*, 2012.
- [5] Djallel Bouneffouf, Irina Rishz. *A Survey on Practical Applications of Multi-Armed and Contextual Bandits*, 2019.
- [6] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The non-stochastic multi-armed bandit problem, 2012.
- [7] Reinforcement Learning by Richard S. Sutton and Andrew G. Barto. *The MIT Press*. Cambridge, 2018.
- [8] Yevgeny Seldin, Csaba Szepesvári, Peter Auer, Yasin Abbasi-Yadkori. Evaluation and Analysis of the Performance of the EXP3 Algorithm in Stochastic Environments, 2012.
- [9] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.
- [10] Walter Rudin. *Real and complex analysis*, 1987.
- [11] Charu C. Aggarwal, *Recommender systems*. Springer, 2016.
- [12] Pier Guisseppe Sessa, Ilija Bogunovic, Maryam Kamgarpour, Andreas Krause. *Contextual Games: Multi-Agent Learning with Side Information*, 2020.
- [13] Larry J. LeBlanc, Edward K. Morlok, William P. Pierskalla. An efficient approach to solving the road network equilibrium traffic assignment problem, 1975.
- [14] Pier Guissepe Sessa, Ilija Bogunovic, Maryam Kamgarpour, Andreas Krause. No-regret Learning in Unknown Games with Correlated Payoffs, 2019.
- [15] J. Welles Wilder Jr. *New Concepts in Technical Trading Systems*, 1978.
- [16] Raul Canessa C. (<https://www.tecnicasdetrading.com/2010/06/macd-moving-average-convergence-divergence.html>), Indicador MACD – Uso e interpretación del MACD. URL:<https://www.tecnicasdetrading.com/2010/06/macd-moving-average-convergence-divergence.html> (fecha de acceso: 2023-04).
- [17] MovieLens (<https://grouplens.org/datasets/movielens/1m/>), Base de datos de películas MovieLens 1M. URL:<https://grouplens.org/datasets/movielens/1m/> (fecha de acceso: 2023-03)
- [18] Pier Guissepe Sessa (<https://github.com/sessap/contextualgames>), Contextual Games: Multi-Agent Learning with Side Information. URL:<https://github.com/sessap/contextualgames> (fecha de acceso: 2023-02).
- [19] Yan Qui (<https://github.com/yan-qii/k-shortest-paths-cpp-version>), K-Shortest Path Algorithm. URL:<https://github.com/yan-qii/k-shortest-paths-cpp-version> (fecha de acceso: 2023-03).
- [20] Andrew Ng (<https://www.coursera.org/specializations/machine-learning-introduction>), Machine Learning program by DeepLearning.AI and Stanford University. URL:<https://www.coursera.org/specializations/machine-learning-introduction>.
- [21] John Bollinger (<https://editorial.blob.core.windows.net/miscelaneous-input/43nU6TUt1Bpht3fs4zs8DU7xSqDBbdNJ99KrXi1a/John%20Bollinger-637370664925974496.pdf>), Así uso hoy en día en el mercado mis herramientas. URL:<https://editorial.blob.core.windows.net/miscelaneous-input/43nU6TUt1Bpht3fs4zs8DU7xSqDBbdNJ99KrXi1a/John%20Bollinger-637370664925974496.pdf> (fecha de acceso: 2023-04).
- [22] Seaborn. (<https://seaborn.pydata.org/generated/seaborn.violinplot.html>). *seaborn.violinplot*. URL: <https://seaborn.pydata.org/generated/seaborn.violinplot.html> (fecha de acceso: 2023-03).

- [23] Fidelity (<https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/sma>), Simple Moving Average (SMA). URL:<https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/sma> (fecha de acceso: 2023-04).
- [24] Eva Blanco Garzón (<https://admiralmarkets.com/es/education/articles/forex-indicators/media-movil-simple>), Qué son las Medias Móviles y cómo utilizarlas en 3 estrategias de trading. URL:<https://admiralmarkets.com/es/education/articles/forex-indicators/media-movil-simple> (fecha de acceso: 2023-04).
- [25] Jason Fernando (<https://www.investopedia.com/terms/r/rsi.asp>), Relative Strength Index (RSI) Indicator Explained With Formula. URL:<https://www.investopedia.com/terms/r/rsi.asp> (fecha de acceso: 2023-04).
- [26] Iván Hernández Roldán (<https://github.com/ivanhernandezroldan/computer-science-final-degree-project-2023.git>), computer-science-final-degree-project-2023. URL:<https://github.com/ivanhernandezroldan/computer-science-final-degree-project-2023.git>.
- [27] Alejandro Magarzo Gonzalo (<https://github.com/amagarzogonzalo/computer-science-final-degree-project-2023.git>), computer-science-final-degree-project-2023. URL:<https://github.com/amagarzogonzalo/computer-science-final-degree-project-2023.git>.
- [28] Durand, Audrey and Achilleos, Charis and Iacovides, Demetris and Strati, Katerina and Mitsis, Georgios D. and Pineau, Joelle (<http://proceedings.mlr.press/v85/durand18a/durand18a.pdf>), Contextual Bandits for Adapting Treatment in a Mouse Model of de Novo Carcinogenesis. URL:<http://proceedings.mlr.press/v85/durand18a/durand18a.pdf> (fecha de acceso: 2023-04).