Universidad de Castilla-La Mancha

Escuela Superior de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática

Tecnología Específica de

Computación

Aplicación de técnicas de análisis de datos a la fase final de las partidas en el juego Teamfight Tactics

Javier Tomás Fernández Martín

Julio, 2023

Trabajo Fin de Grado

Grado en Ingeniería Informática

Tecnología Específica de Computación

Aplicación de técnicas de análisis de datos a la fase final de las partidas en el juego Teamfight Tactics

Autor: Javier Tomás Fernández Martín

Tutores: José Antonio Gámez Martín y

Juan Carlos Alfaro Jiménez

Julio, 2023

Dedicado a mi familia y a todas
aquellas personas que me
apoyan a diario

Declaración de Autoría

Yo, Javier Tomás Fernández Martín con DNI 49801020V, declaro que soy el único autor del trabajo fin de grado titulado “Aplicación de técnicas de análisis de datos a la fase final de las partidas en el juego Teamfight Tactics” y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a 9 de julio de 2023

Fdo: Javier Tomás Fernández Martín

Resumen

Hoy día con el constante crecimiento en la industria del videojuego y el campo competitivo de la misma, los jugadores cada vez buscan más aplicaciones externas que los ayuden a aprender, mejorar, tomar decisiones más acertadas, obteniendo así mejores resultados y disfrutando más del juego. Como consecuencia, las empresas cada vez más buscan, dentro de lo posible, cómo implementar dichas aplicaciones externas para que sea más cómodo para el jugador y tenga una mejor experiencia.

Este trabajo abordará un problema del videojuego Teamfight Tactics. Este videojuego es un "autochess", que como su traducción al español indica, se asemeja a un ajedrez automático, el cual lo constituyen 8 jugadores que enfrentan sus tableros cada ronda y pierden vida si sus fichas son derrotadas. El objetivo del juego es evitar ser de los 4 primeros eliminados, buscando ser el último superviviente de la partida. Para ello es necesario invertir los recursos disponibles en mejorar el tablero.

El objetivo principal de este trabajo es crear un programa capaz de, dado dos agrupaciones de fichas con sus respectivas variables, determinar cual saldrá victoriosa en el último enfrentamiento. Para realizar esta predicción se recurrirá a la analítica de datos, analizando partidas de la base de datos de la empresa Riot Games, las cuales se obtendrán a través de una API que proporciona dicha empresa.

Hay que remarcar que los datos disponibles corresponden al último enfrentamiento de la partida y no a las fases intermedias de la misma, y por eso este trabajo está limitado a dicho enfrentamiento únicamente.

Agradecimientos

En primer lugar, agradecer a mis tutores José Antonio Gámez Martín y Juan Carlos Alfaro Jiménez, los cuales me han guiado a lo largo del trabajo y ayudado con todo lo que he necesitado.

Agradecer también mi familia y todas las personas que me apoyan tanto con lo que respecta a la carrera como en otros aspectos de la vida.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Motivación

Hoy día la industria de los videojuegos está creciendo cada vez más, como se ve en la **Figura 1-1**, ya que desde 2017 a 2021 han crecido un 15% los ingresos que generan mundialmente, y se espera que sigan creciendo a un ritmo similar. Esto hace que los videojuegos lleguen a una gran variedad de usuarios, y que cada vez más las empresas intenten facilitar la entrada al mundo del videojuego, haciendo interfaces intuitivas, ayudas al jugador novato, etc.

Riot Games es una de las compañías más grandes en la industria del videojuego, y últimamente, aprovechando la repercusión de su juego más popular, League of Legends, está expandiéndose en otros tipos de juegos y de áreas. Como se puede observar en la **Figura 1-2**, en los 3 últimos años ha tenido de media un mínimo 140 millones de jugadores en línea al mes, con más de 10 millones conectados a la vez en su punto álgido.

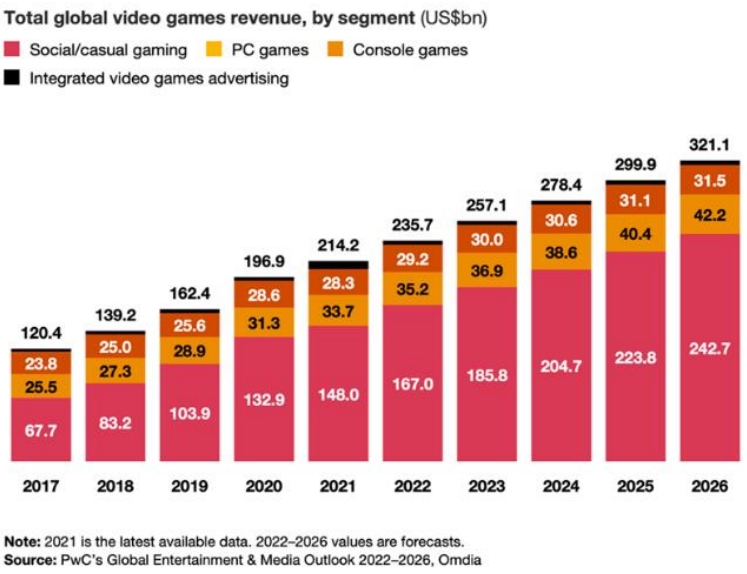

Figura 1-1 Ingresos globales totales de los videojuegos [1]

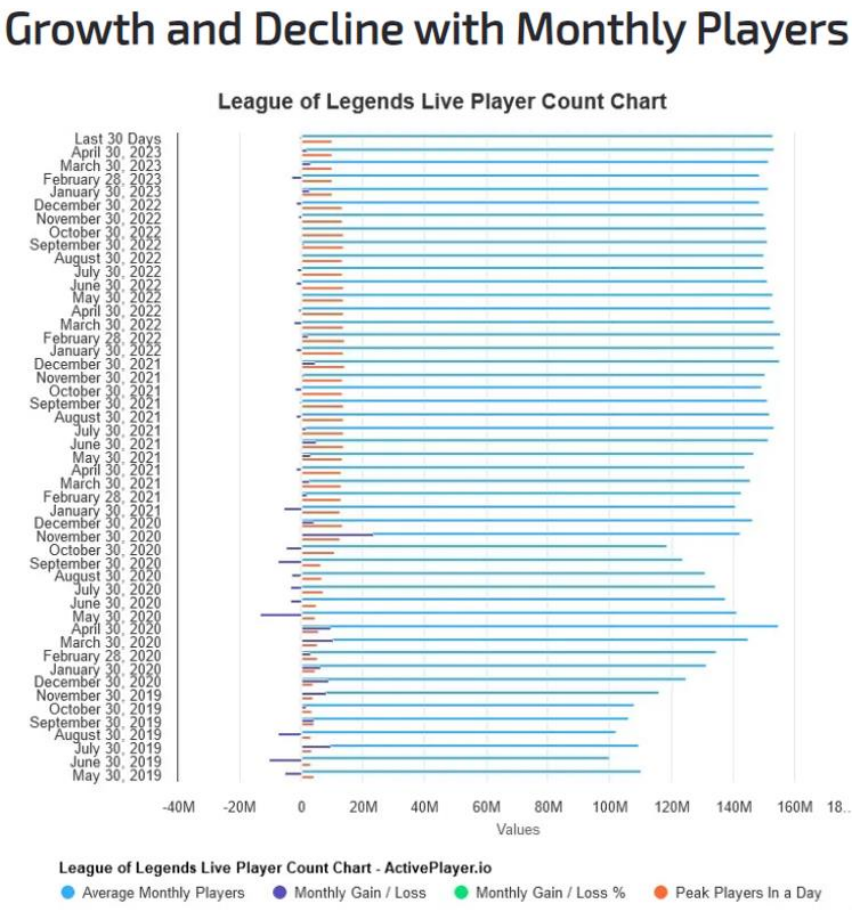

Figura 1-2 Jugadores de League of Legends activos los últimos cuatro años [2]

Uno de estos campos en los que se está expandiendo la compañía Riot Games es un juego de AutoChess llamado "Teamfight Tactics" (desde ahora TFT), el cual salió en 2019 con una enorme cantidad de jugadores incluso nada más ser publicado, llegando a "33 millones de jugadores cada mes" [3], al nivel de modalidades de juego de League of Legends muy populares y esperadas, como se ve en la Figura 1-3. Bien es cierto que este número de jugadores ha bajado, pero aun así se ha mantenido bastante alto. En 2021 tenía "10 millones de jugadores cada día" [4], según los desarrolladores del juego.

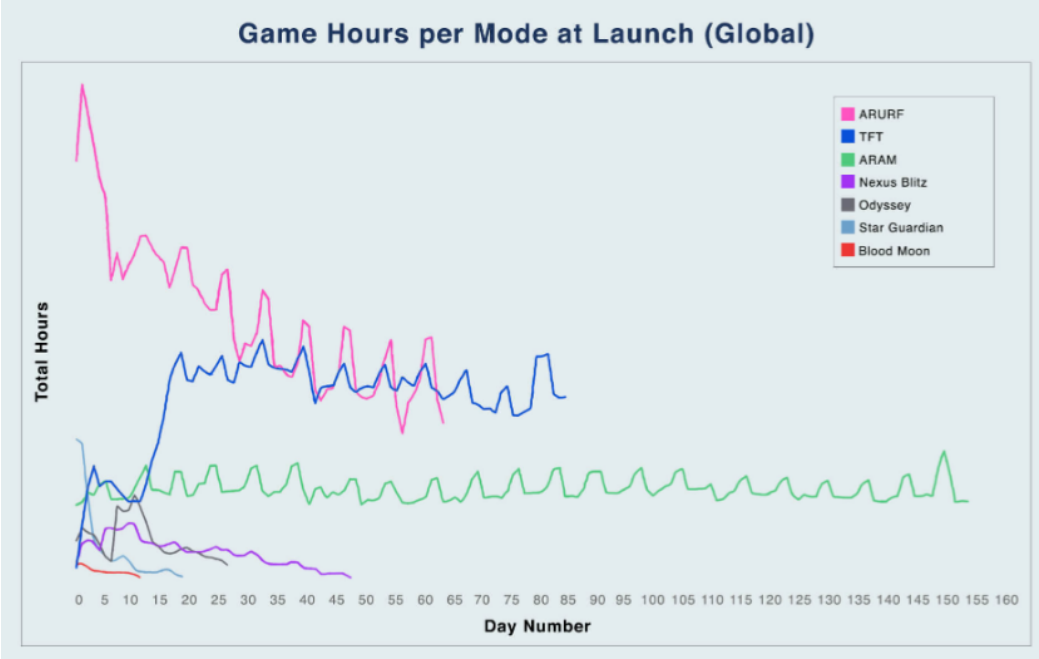

Figura 1-3 Total de número de horas jugadas después de la fecha de lanzamiento[3]

El número de jugadores bajó ya que el TFT es un juego relativamente complicado, y es por eso por lo que cada vez más, la compañía intenta facilitar la entrada al juego a nuevos jugadores, cambiando y añadiendo consejos en la interfaz del propio juego, e incluso recomendando y apoyándose en páginas externas las cuales ayudan el juego a crecer. A pesar de esto, se espera que el juego siga creciendo, ya que la inversión en él va a crecer, según filtraciones de los jugadores que tienen contacto con los desarrolladores [5].

1.2 Objetivos

Como hemos podido observar, las bases del juego son algo complicadas, y muchas veces los jugadores no saben cuándo tienen una composición de fichas más fuerte que el adversario, o creen que tienen una mejor composición y no saben por qué siguen perdiendo, cual ha podido ser la diferencia, etc.

El objetivo de este proyecto es *conseguir un modelo que sea capaz de determinar con un índice de confianza que jugador se alzará con la victoria, dado un escenario en el cual solo quedan dos jugadores enfrentándose entre sí, ya que el resto de jugadores ya han sido derrotados, y es la última ronda del juego*. Planteamos únicamente esta situación debido a cómo funciona la base de datos disponible, que solo muestra el tablero completo en la ronda antes de que el jugador fuese derrotado o se acabase la partida. Para conseguir esto, se tendrá que recopilar una gran cantidad de partidas de los mejores jugadores de Europa, ya que esos jugadores son los más capaces de llegar a mesas óptimas y de donde más información se puede extraer.

El modelo deberá ser capaz de determinar que jugador será el que se alzará con la victoria dado dos tableros con sus respectivas fichas, objetos, aumentos, etc. y con la información usada en la toma de decisiones se podrá obtener tanto un análisis del estado del juego actual como determinar ciertos detalles diferenciales escondidos que algunos jugadores no han encontrado todavía.

1.3 Estructura del proyecto

Aquí se definirá como se ha estructurado la memoria.

- Capítulo 1: Introducción. Se introduce el problema que tenemos que resolver, así como los distintos motivos como para solucionarlo y una pequeña

introducción al contexto en el cual se desarrollará el proyecto. Por último, se marcan objetivos.

- Capítulo 2 : Estado del arte. Se analizarán distintas herramientas que existen actualmente con las cuales se pueden obtener datos similares al resultado deseado del proyecto.
- Capítulo 3: Metodología y Desarrollo. Dedicado a exponer las herramientas utilizadas, obtención de la base de datos, y la metodología seguida.
- Capítulo 4: Experimentos y Resultados. En este capítulo se desarrollarán los diferentes experimentos y se mostrarán sus resultados, explicándolos y analizándolos individualmente.
- Capítulo 5: Conclusiones y trabajo futuro. Por último, se expondrán las conclusiones del proyecto, analizando como se podría mejorar e implementar en el juego los modelos creados usando herramientas ya existentes.

1.4 Competencias

Las diferentes competencias que se trabajarán en este proyecto respectivas a la intensificación de Tecnologías de la Computación durante la realización de este proyecto son:

- Capacidad para tener un conocimiento profundo de los principios fundamentales y modelos de la computación y saberlos aplicar para interpretar, seleccionar, valorar, modelar, y crear nuevos conceptos, teorías, usos y desarrollos tecnológicos relacionados con la informática.
- Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.

- Capacidad para adquirir, obtener, formalizar y representar el conocimiento humano en una forma computable para la resolución de problemas mediante un sistema informático en cualquier ámbito de aplicación, particularmente los relacionados con aspectos de computación, percepción y actuación en ambientes o entornos inteligentes.
- Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.

Capítulo 2

Estado del Arte

2.1 Introducción

En esta sección se desarrollará cómo funciona el TFT para tener un conocimiento general que servirá para entender los datos analizados posteriormente. A continuación, se verán algunos ejemplos de herramientas externas de ayuda al jugador que existen hoy día, y se concluirá el capítulo con una introducción a los algoritmos de aprendizaje automático que se han utilizado en este proyecto.

2.2 Introducción a Teamfight Tactics

Teamfight Tactics es un juego de estrategia en línea que combina elementos de ajedrez y juegos de cartas coleccionables. En este juego, cada jugador tiene su propio tablero, con diferentes fichas, denominadas “campeones”. Las partidas son de un total de 8 jugadores, y cada partida consta de varias rondas, en las que los jugadores enfrentan sus tableros entre sí, perdiendo vida el jugador que haya sido derrotado. Una vez dos jugadores enfrentan sus tableros, la pelea ocurre de manera automática; las fichas se

moverán y atacarán solas. Un jugador es derrotado en una ronda cuando todas las fichas de su tablero se quedan sin vida.

El objetivo del juego es ser el último jugador en pie, aunque se considera victoria todo lo que sea quedar de los 4 últimos vivos, y derrota quedarse sin vida de los 4 primeros (top 4 y bottom 4).

El TFT se actualiza cada cierto tiempo, y cambian mecánicas, fichas, objetos, etc. categorizando cada temporada en "sets". En este trabajo nos centraremos en el set 8, concretamente en el 8.5.

Conceptos básicos de Teamfight Tactics

A continuación, se explicarán los elementos básicos que componen una partida de TFT:

- Tablero: El tablero se divide en dos mitades. La mitad más cercana al jugador es el lugar donde colocará los campeones que tenga, y en la otra mitad aparecerán los campeones del jugador contrario cuando se enfrenten. Las casillas donde el jugador coloca a los campeones son hexágonos, y cada mitad del tablero está compuesta por 4 filas y 7 columnas (**Figura 2-1**).

Figura 2-1 Imagen del tablero de TFT con un campeón seleccionado [6]

- Campeones: Los campeones (**Figura 2-2**) o fichas son las piezas que el jugador puede obtener a lo largo de la partida, posicionándolas en el tablero y mejorándolas.

Figura 2-2 Foto de dos campeones

- Banquillo: Cuando un jugador obtiene campeones, estos automáticamente son colocados en el banquillo. El jugador tiene 9 huecos en el banquillo para acumular y guardar campeones (**Figura 2-3**).

Figura 2-3 Foto del banquillo con varios campeones

- Objetos: Los objetos son mejoras que se pueden efectuar sobre los campeones. Para formar un objeto se deben combinar dos componentes de objetos, obteniendo así uno completo el cual tiene mayor impacto sobre la partida. Cada campeón tiene un máximo de 3 objetos (**Figura 2-4**) que puede llevar equipados. Una vez equipado un objeto en un campeón, ese objeto se queda vinculado al campeón hasta que es vendido, que entonces se recupera el objeto y se puede equipar en otros campeones si se desea.

Figura 2-4 Foto de los objetos equipados en un campeón

- Oro y economía: El jugador obtiene oro después de cada ronda de forma pasiva, aumentando la cantidad de oro que obtiene con diferentes estrategias, como tener una racha de rondas en las que ha salido victorioso, o incluso una racha de derrotas, tener una cantidad determinada de oro con el cual puedes hacer interés, etc. (**Figura 2-5**)

Figura 2-5 Foto de la interfaz donde se muestra el oro y las rachas

- Nivel: El jugador obtiene experiencia después de cada ronda de forma pasiva, con la cuál poco a poco va subiendo de nivel (**Figura 2-6**). El nivel representa el número de fichas máxima que puede tener en el tablero dicho jugador. También puede gastar oro para comprar experiencia.

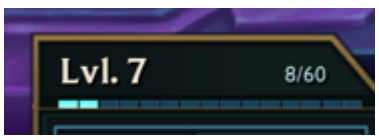

Figura 2-6 Foto de la interfaz donde se muestra el nivel del jugador

- Tienda: Cada ronda se le ofrecen al jugador 5 campeones (**Figura 2-7**), los cuáles puede comprar utilizando oro, y vender recuperando total o parcialmente ese

oro. El precio de los campeones depende de la rareza de estos. El jugador puede actualizar la tienda tantas veces como quiera (o pueda) empleando el oro que tiene.

Figura 2-7 Foto de la tienda con diferentes campeones

Mecánicas de Teamfight Tactics

Una vez vistos los conceptos básicos del juego, se puede explicar mejor cómo funcionan las mecánicas básicas del mismo:

- Mejorar campeones: Los campeones aparecidos en tienda son copias de un mismo campeón. Un jugador puede mejorar su copia de campeón juntando tres del mismo, subiéndolo a "dos estrellas" o a "plata" (**Figura 2-8**). Si el jugador obtiene tres copias de dos estrellas, esa ficha se mejora a "tres estrellas" u "oro". Cuanto más mejorada está una ficha, mejores estadísticas tiene (daño, vida, etc).

Figura 2-8 Foto de un campeón a dos estrellas y a una estrella

- La “pool”: Todas las fichas mostradas a todos los jugadores en todas las tiendas de una partida de TFT son comunes y limitadas. Es decir, es como sacar 5 canicas aleatorias de una bolsa común, siendo la bolsa la pool, y las canicas campeones. Esto tambien quiere decir que hay una cantidad limitada de campeones, y de número de copias de este.

- Economía: Un jugador obtiene 5 unidades de oro de forma pasiva, el cuál aumenta con el interés y las rachas. El interés funciona de la siguiente manera: El jugador obtiene 1 de oro extra si al terminar la ronda tiene mínimo 10 unidades de oro, 2 si tiene 20, 3 si tiene 30, etc. hasta un máximo de 5 de interés, con 50 unidades de oro. La racha, ya sea de victorias o de derrotas, otorga al jugador 1 unidad de oro extra si tiene una racha de 2-3, 2 si tiene racha de 4, o 3 si tiene racha de 5 o más. Además, el jugador obtiene 1 unidad de oro al terminar la ronda si ha salido victorioso de la batalla.

- Aumentos: 3 veces durante la partida, el jugador tendrá que elegir 1 de 3 aumentos que se le ofrecen. Estos aumentos son mejoras o cambios en el estilo de juego, aumentando el poder de determinados campeones, dando más vida al propio jugador, oro, cambios en el interés, etc. Los aumentos salen también de una “pool” común, pero no hay un número de copias de cada aumento, es decir, a todo el mundo le pueden salir los mismos aumentos.

- Rondas PvE: Cada ciertas rondas, habrá una en la cual el jugador se enfrentará a distintos enemigos (bots), que no suelen ser difíciles de derrotar, los cuales otorgarán al jugador oro, campeones, componentes u objetos cuando sean derrotados.

- Rondas PvP: Cada ronda el jugador se enfrentará su tablero contra otro jugador, y el que sea derrotado, perderá vida.

- Carrusel: En esta ronda todos los jugadores aparecen alrededor de un carrusel de campeones los cuales tienen un componente u objeto equipado. Los jugadores salen de dos en dos por turnos y escogen un campeón del carrusel, siendo los dos con menos vida los primeros en escoger, y los dos con más vida los últimos. Esta ronda ocurre solamente una vez por fase (**Figura 2-9**).

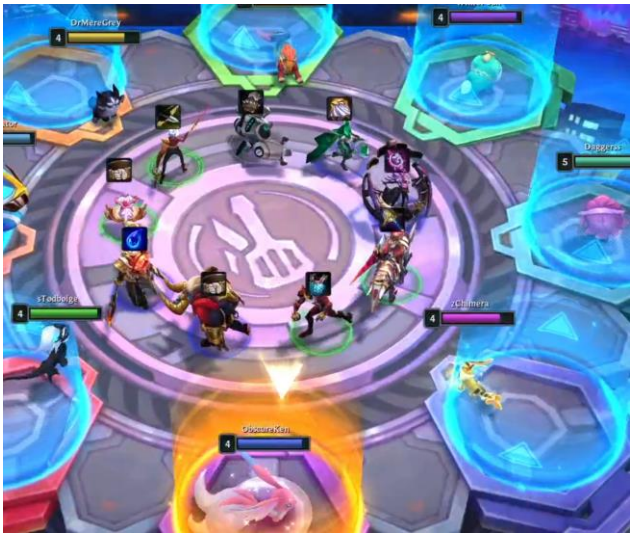

Figura 2-9 Ejemplo de un carrusel

Figura 2-10 Foto resumen de un tablero con anotaciones

Desarrollo normal de una partida de Teamfight Tactics

Antes de nada, hay que definir las diferentes secciones de una partida de TFT. Una partida de TFT se divide en diferentes fases, empezando en la fase 1 con un carrusel inicial, y 3 rondas PvE, teniendo un total de 4 rondas. Luego, cada fase se compone de 3 rondas PvP, 1 carrusel, 2 rondas PvP más, y por último una ronda PvE que supone el final de la fase. Hay tantas fases como sean necesarias para que solo un jugador quede en pie.

Una vez explicado cuál es el funcionamiento normal de una partida, se puede entender mejor como están divididas las rondas en sí, excepto en los carruseles, que ya se han explicado:

- Planificación: El jugador tiene un breve periodo de tiempo (30 segundos normalmente) para ejecutar distintas acciones antes de la fase de pelea. En esta fase el jugador puede comprar fichas, mejorarlas, meterlas en el tablero, moverlas por el tablero, crear objetos, equipar a los a campeones, etc.

- Pelea: En esta fase los campeones que tienen en el tablero los jugadores se enfrentan entre sí en el caso de ser una ronda PvP, si es PvE se enfrentan a los monstruos de la fase, atacando de manera automática. El jugador puede seguir invirtiendo el oro, pero no puede hacer modificaciones en el tablero. Es decir, el jugador no puede mover, añadir o quitar fichas del tablero (aunque sí que puede equiparles objetos), y si consigue 3 copias de un campeón, pero una de ellas está dentro del tablero, el campeón no se mejorará hasta la siguiente ronda.

Funcionamiento del ranking de Teamfight Tactics

Cuando los jugadores juegan partidas competitivas, ganan o pierden puntos de liga (LPS) basado en su resultado. Con estos puntos de liga se puede subir y bajar de división y de rango. Cada 100 LPS, se sube al siguiente rango y se reinician los puntos a 0, y se baja de división si se pierden partidas cuando el jugador tiene 0 LPS.

Cuando se alcanza el máximo rango, se sube a la siguiente división. Cada división tiene 4 rangos, siendo el 4 el más bajo, y el 1 el más alto. Las divisiones ordenadas de menor, (la más novata), a mayor, (la más experta), son: Hierro, Bronce, Plata, Oro, Platino y Diamante (**Figura 2-11**).

Una vez se supera el rango de Diamante I, se asciende a una división especial llamada "Master", la cual agrupa a todos los jugadores capaces de superar todas las divisiones mencionadas anteriormente. A partir de ahí, la agrupación de jugadores es diferente, ya que no se puede subir de rango, pero sí de división y se pueden ganar LPS infinitos sin que el sistema los reinicie a 0 por subir de división.

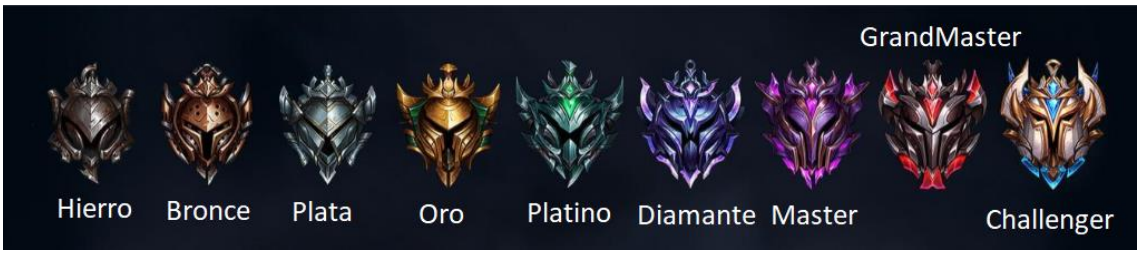

Figura 2-11 Distribución de rangos en Teamfight Tactics [7]

Para subir a la siguiente división, el jugador debe estar en el top 1000 de jugadores de su región, ascendiendo así a GrandMaster. Para ascender a la última división, Challenger, el jugador debe estar en el top 200 de su región.

Como se puede ver en la **Figura 2-12**, en las últimas divisiones hay muy pocos jugadores, estando solamente los más experimentados en esas divisiones. Aún así,

como no se puede bajar de división (de oro a plata por ejemplo), se observa un pico en cada división que es la última de su categoría, como Master, Diamante IV, Platino IV, etc.

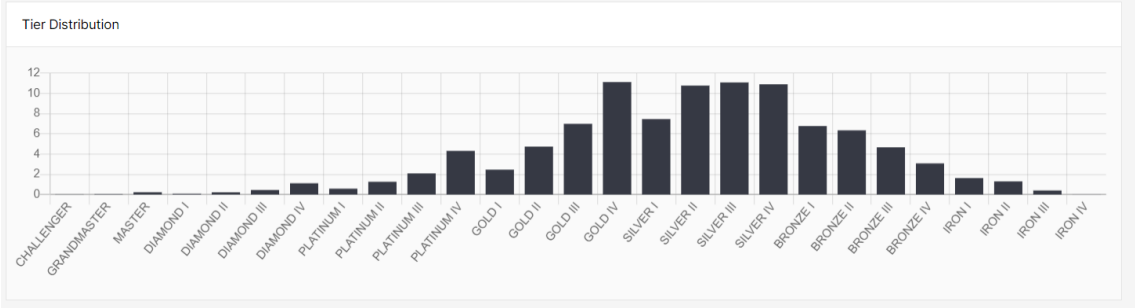

Figura 2-12 Distribución de los jugadores en el ranking de Teamfight Tactics [8]

2.3 Herramientas externas de ayuda al jugador.

Como ya se ha comentado, hay varias herramientas externas al propio juego que tratan de ayudar al jugador lo máximo posible, exponiéndole que conjunto de campeones y aumentos son los óptimos en la mayoría de las situaciones. A continuación, se explicarán un poco las dos más importantes y que más proyección de mejora tienen.

2.3.1 Tactics.tools

Esta página web [9] todavía no tiene aplicación integrada, pero a pesar de ello, es de las herramientas más usadas por los jugadores con más experiencia, debido a la gran variedad de datos que proporciona, y su capacidad para filtrarlos al gusto. Esta página está dividida en varias secciones, pero solo vamos a mencionar las más importantes y el uso que los jugadores le dan:

- Composiciones: En esta sección (**Figura 2-13**), se muestran las composiciones (agrupaciones de campeones) que mayor porcentaje de top 4 tienen, mostrando también el "play rate", que es la cantidad de jugadores que juegan esa composición de media por partida, y el "win rate", que es el porcentaje de

jugadores que ganan una partida con esa composición. Esta es la base de los jugadores novatos, ya que su objetivo de partida normalmente es llegar a la composición que les muestra la página, la cual también muestra los objetos que debe tener cada campeón, y los aumentos más útiles. Para los jugadores avanzados también es muy útil, ya que algunos aumentos u objetos pueden ser mediocres en términos generales, pero que para una composición específica sean mejores que la mayoría, debido a alguna interacción entre las fichas y sus habilidades u otros motivos. Esto último se puede observar si se analiza con cautela cada composición individualmente con diferentes criterios.

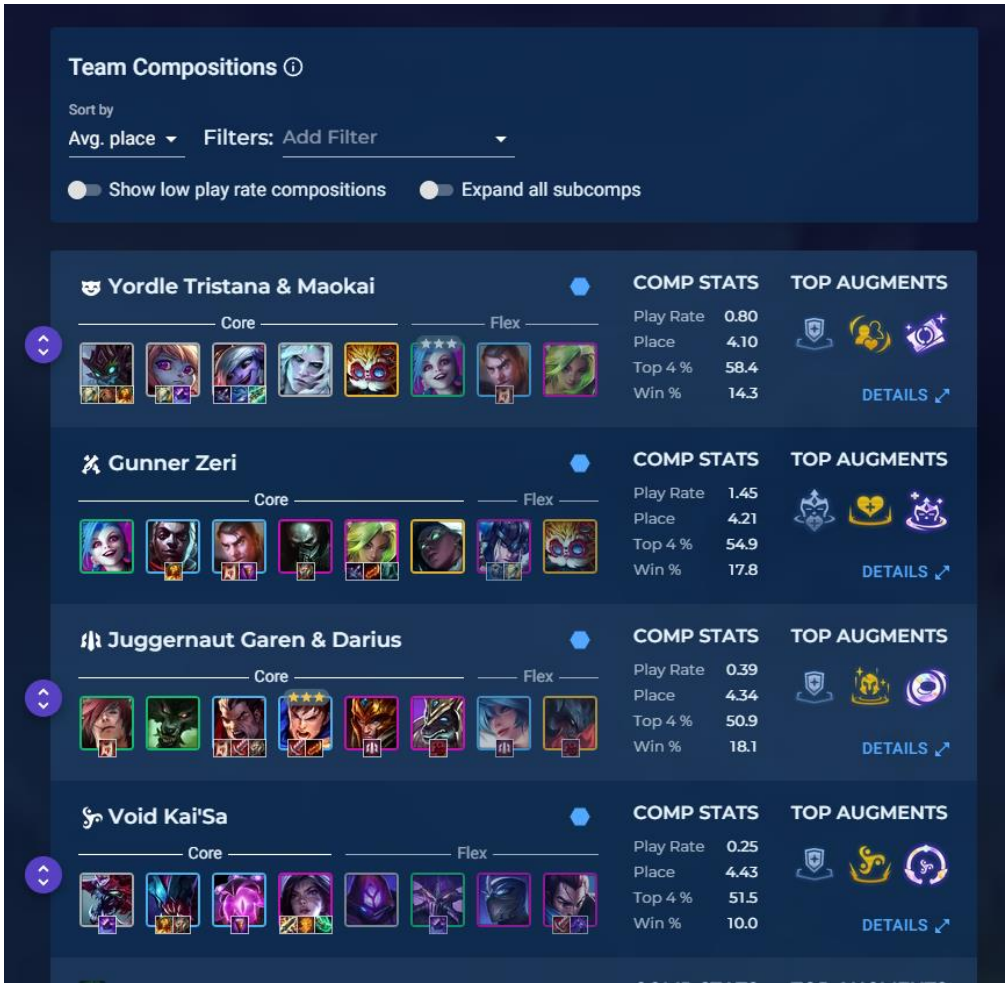

Figura 2-13 Foto de la página web en la sección de composiciones

- Estadísticas: Esta sección, a su vez, se subdivide en objetos, unidades, aumentos, etc. Como funcionan de manera similar y tampoco se va a tratar como tema en profundidad, se tratará únicamente la página de los aumentos (**Figura 2-14**), ya que es la más popular.

  En esta sección, se muestran los distintos aumentos del juego, pudiéndolos filtrar por rareza del aumento, el número de partidas que ha sido seleccionado, el momento de la partida que ha sido seleccionado, etc. Esta sección es de las más importante de toda la página para los jugadores más experimentados, ya que también se pueden mostrar por porcentaje de jugadores que han hecho de top 4 cuando han escogido el aumento. Otra forma de representar lo mismo es el apartado de "posición", el cual muestra la posición media obtenida por todos los jugadores que han elegido ese aumento. Esto ayuda a la toma de decisiones de la partida, ya que si un aumento con 10k de partidas tiene una posición media de 5.6, indica que el aumento tiene un rendimiento por debajo de la media, mientras que si con el mismo número de partidas tiene un 3.8 es de los aumentos más fuertes del juego, y puede dar una ventaja notoria al jugador que lo elija. Un aumento está balanceado cuando tiene una posición media de 4.2 aproximadamente, siendo cualquier valor menor que 4.1 un aumento desbalanceado que le da mucha ventaja al jugador, y cuando es mayor que 4.5, pone al jugador en desventaja ya que es un aumento más débil que la media.

  Esto mismo ocurre para objetos y campeones, aunque estos tienen distintos tipos de filtrado, y se pueden obtener conclusiones distintas.

Augment	Games	Place	Top 4	Win	At 2-1	At 3-2	At 4-2	Top Users
Built Different III	10k	3.81	63.2%	17.9%	3.79	3.87	—	
Stable Evolution	6.3k	3.87	62.4%	17.0%	3.70	3.87	3.99	
Dueling Gunners	18k	3.87	61.6%	20.5%	4.10	3.87	3.82	
March of Progress	12k	3.89	60.4%	21.9%	3.89	—	—	
Morning Light	7.4k	3.92	60.8%	18.7%	4.04	3.69	3.98	
Noxus Crown	4.0k	3.92	60.6%	15.4%	3.80	3.88	4.24	
Unstable Yordle D...	2.6k	3.95	60.1%	21.9%	3.89	4.80	—	
Long Distance Pal...	37k	3.96	60.3%	17.4%	3.95	3.99	3.96	
Mana Burn	8.1k	4.03	59.2%	16.2%	—	—	4.03	
Gifts from the Fallen	51k	4.04	58.3%	17.5%	4.14	4.04	4.02	
Void Crown	5.4k	4.05	59.1%	13.5%	4.14	3.82	4.23	
The Golden Egg	6.8k	4.11	52.8%	22.7%	—	—	4.11	
Unified Resistance I	35k	4.11	57.1%	16.2%	4.27	4.06	4.14	
Tons of Stats!	43k	4.12	57.0%	15.4%	4.07	4.07	4.15	
Overcharged Mana...	9.1k	4.13	56.4%	16.0%	4.06	4.05	4.20	
Final Ascension	2.8k	4.14	57.2%	15.3%	—	—	4.14	

Figura 2-14 Foto de la sección de aumentos ordenada por posición media

2.3.2 MetaTFT

Esta herramienta [10] tiene tanto página web como una aplicación que se puede usar dentro del juego. La página web tiene unas herramientas similares a las mencionadas en Tactics.tools [9], pero con menos capacidad de filtrado en aspectos más avanzados. Lo que nos interesa en esta página, es que guarda una captura de pantalla de todas las rondas, analizando la vida, el oro, y el tablero que tiene el jugador en ese momento, para que se pueda revisar la partida paso por paso al final.

También ofrece una interfaz que reconoce los aumentos que se le ofrecen al jugador y muestra dentro del juego la posición media y el "win rate" sin necesidad de tener la página abierta. Esto es bastante interesante ya que el que sea capaz de mostrar los datos del aumento que aparece en pantalla, quiere decir que hay un modelo entrenado capaz de reconocer y diferenciar los diferentes aumentos a tiempo real. Con esto

implementado ya, también abre puertas a distintas maneras de mostrar al jugador que objetos hacer, que campeones comprar, o como posicionar tu composición.

Todas estas capturas de pantalla a las distintas fases del juego, hace que esta aplicación sea capaz de obtener datos en los puntos intermedios de la partida, teniendo así una gran cantidad de datos y pudiendo optimizar las composiciones iniciales (**Figura 2-15**) que luego llevan a la composición final. De hecho, una herramienta bastante curiosa es que cuando el jugador se enfrenta a otro jugador aparece un porcentaje de probabilidad de victoria. Esa herramienta ahora mismo no funciona con certeza, probablemente debido a que no se tiene en cuenta el posicionamiento de los campeones el cual suele ser decisivo, o quizás porque los datos de las partidas recogidas son de los jugadores con menor nivel, lo cual hace que cuando una pelea no es "evidente", falle normalmente en la predicción. Esto es porque las composiciones más novatas no aportan información relevante, o porque los datos de esas composiciones novatas no se tratan bien.

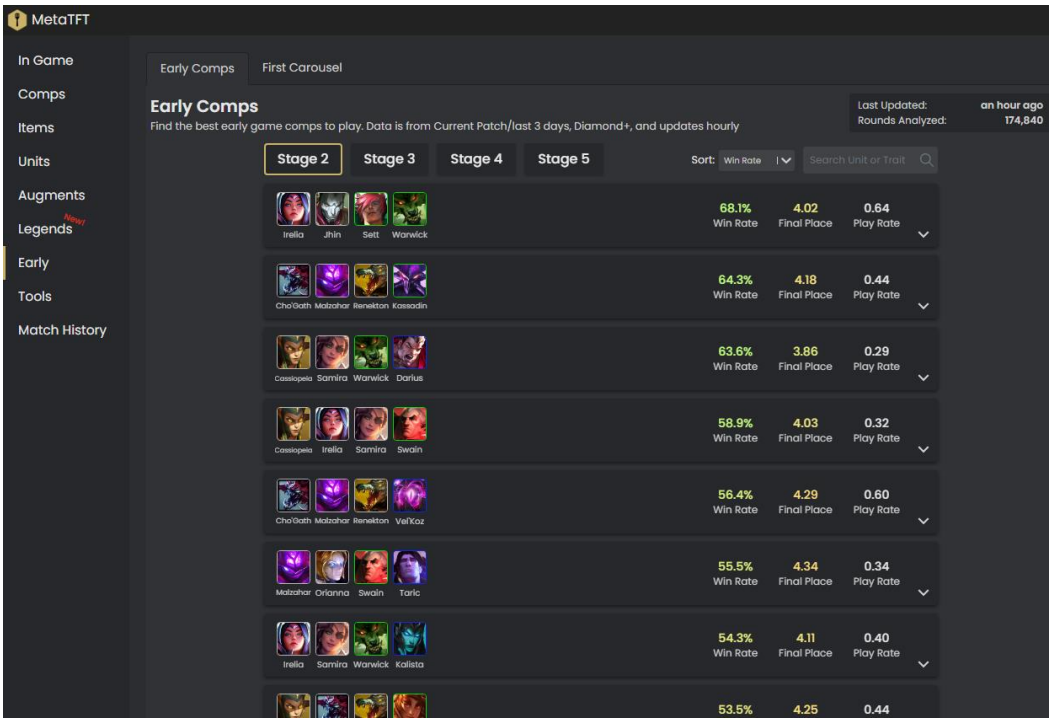

Figura 2-15 Foto del apartado de composiciones iniciales de MetaTFT

2.4 Algoritmos de aprendizaje

Estos algoritmos son un conjunto de instrucciones las cuales permiten a un sistema aprender, mejorar y optimizar su comportamiento ante una tarea específica a partir de unos datos dados. A partir de ellos, se obtienen modelos, los cuales son capaces de hacer predicciones o tomar decisiones. En este trabajo se han usado varios tipos algoritmos de aprendizaje, tanto supervisados como no supervisados, y en esta sección se explicarán los fundamentos de cada uno de ellos.

2.4.1 K-NN

Este algoritmo (K-Nearest Neighbors) es un algoritmo de aprendizaje supervisado, el cual se basa en la proximidad de las instancias del conjunto de entrenamiento. Este algoritmo no se tiene que entrenar, ya que cuando tiene que analizar un nuevo caso calcula la distancia de todas las instancias del conjunto hasta la nueva.

K representa el número de instancias (vecinos) que influyen en el proceso de etiquetado de la que se quiere predecir. Este número es muy importante, ya que como se puede observar en la **Figura 2-16**, si K es 2 o 3, la nueva instancia (el circulo verde) se clasificaría como un triángulo rojo, mientras que si K es 5 se clasificaría como un cuadrado azul.

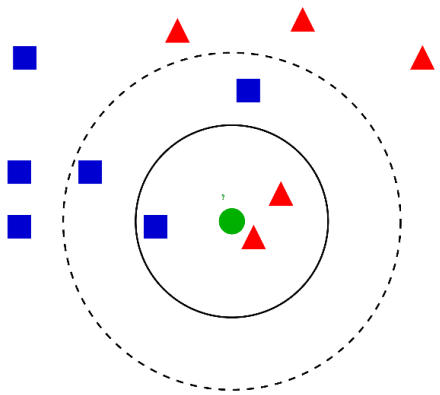

Figura 2-16 Ejemplo de un problema de clasificación con K-NN [11]

2.4.2 Árboles de decisión

Un árbol de decisión representa un grafo dirigido en el cual cada nodo corresponde a una variable de entrada, cada rama de un nodo representa uno de los posibles valores que puede tomar la variable, y cada hoja tiene un valor de la variable clase, como se ve en el ejemplo de la **Figura 2-17**. Para ofrecer un resultado dado una instancia de entrada, un árbol de decisión recorre desde la raíz hasta la hoja correspondiente, eligiendo en todo momento la rama que satisface la condición del atributo requerido en la instancia. La clase elegida es la clase de la hoja en la cual el árbol ha terminado el recorrido.

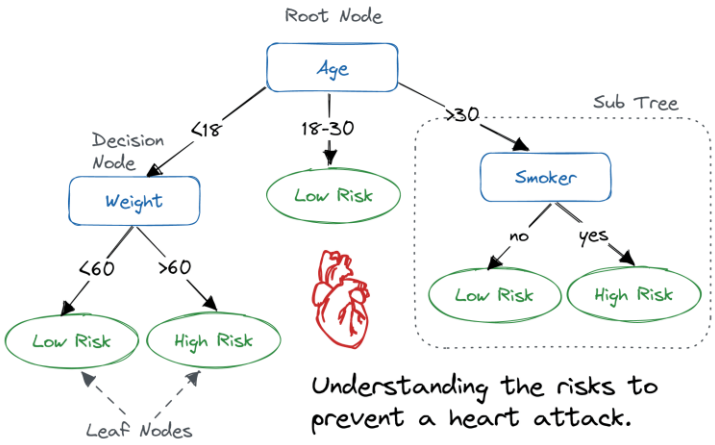

Figura 2-17 Ejemplo de un árbol de decisión [12]

En el proceso de creación de un árbol de decisión a partir de un conjunto de datos, se siguen los siguientes pasos:

- Ramificación: Cada vez que se deba ramificar, se calcula la ganancia de información para cada atributo candidato a ser el criterio de la ramificación. La ganancia de información se calcula comparando la entropía o el índice de Gini (entre otros criterios) de la variable clase, antes y después de la ramificación. Se selecciona el atributo que maximice esta ganancia, reduciendo así la

incertidumbre sobre la distribución de la variable clase en los nodos creados. Este paso se repite recursivamente para cada nodo hijo generado hasta que el árbol ha llegado a la profundidad máxima requerida o no puede ramificar más.

- Poda: Una vez creado el árbol, se comprueba que nodos son necesarios podar, en caso de que haya alguno. Esto es así para evitar el sobreajuste o quitar nodos que no aportan información al modelo, eliminándolos o fusionándolos con otros. Para ello, se recorre el árbol, evaluando la relevancia y generalización de cada nodo. Una técnica común en cuanto a la poda es calcular el error de los nodos hijos de una ramificación y el error del nodo padre, y si el error total disminuye o no supera un error máximo establecido con anterioridad, se efectúa la poda y se fusionan los nodos hijos.

2.4.3 Boosting

Boosting se basa en utilizar varios modelos simples a los cuales se les asocian unos pesos para obtener una predicción a través de una votación ponderada, obteniendo así un modelo más complejo. Estos modelos simples se aprenden de forma secuencial, modificando los pesos de los registros en función del error en su clasificación cometida por el modelo anterior. De esta forma, se presta más atención a los registros mal clasificados con anterioridad, ajustando cada vez más el modelo final. Hay varias técnicas de boosting las cuales tienen sus diferencias entre sí, aunque la idea principal es compartida por todos.

2.4.3.1 Random forest

En este tipo de modelo está compuesto por un conjunto de árboles de decisión. Cada árbol ha sido entrenado con un subconjunto de datos diferente y con unas características aleatorias. Para obtener los subconjuntos de entrenamiento, se utiliza el muestreo con reemplazo, y el árbol se entrena seleccionando unas características (atributos) de manera aleatoria, teniendo así una gran diversidad en los árboles. Para

obtener una predicción, se hace uso de alguna técnica de selección, como el voto por la mayoría, para filtrar la predicción de todos los árboles y devolver una única respuesta.

2.4.3.2 Adaptive Boosting

Este tipo de modelo se enfoca en ajustar los pesos de tal manera que se centren en los ejemplos clasificados incorrectamente, siendo capaz así de enfocarse en los casos más difíciles (**Figura 2-18**). Esto tiene implícitamente asociado un problema con el sobreajuste, ya que se debe tener mucho cuidado para evitar que el ruido de la base de datos se convierta en ejemplos con un peso relevante.

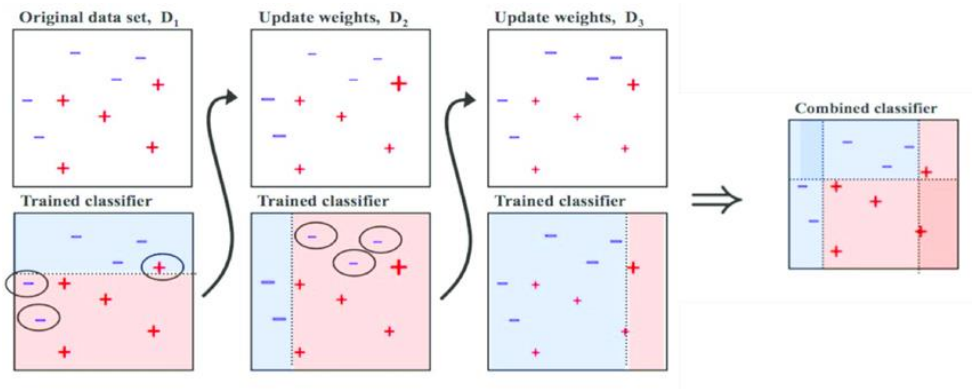

Figura 2-18 Ejemplo del funcionamiento de AdaBoost [13]

2.4.3.3 Gradient Boosting

Esta técnica empieza con un único modelo inicial simple, el cual se utiliza para aprender y ajustar nuevos modelos, los cuales se unen al inicial, hasta obtener varios modelos simples que combinados son capaces de hacer una buena predicción.

Para empezar, se realiza una predicción inicial que se toma como punto de partida y va mejorando con cada iteración. Se calcula el error residual, que representa la diferencia entre los valores reales y las predicciones realizadas por los modelos anteriores, los cuales son árboles de regresión, ya que su variable objetivo son los residuos del error

cometido por los modelos previos- A partir de este error residual, se aprende un nuevo modelo de regresión que tiene en cuenta dicho error y se vuelve a efectuar una predicción, repitiendo este proceso las veces que sea necesario.

Por último, el modelo final obtiene una predicción haciendo una votación ponderada con todos los modelos aprendidos durante el entrenamiento.

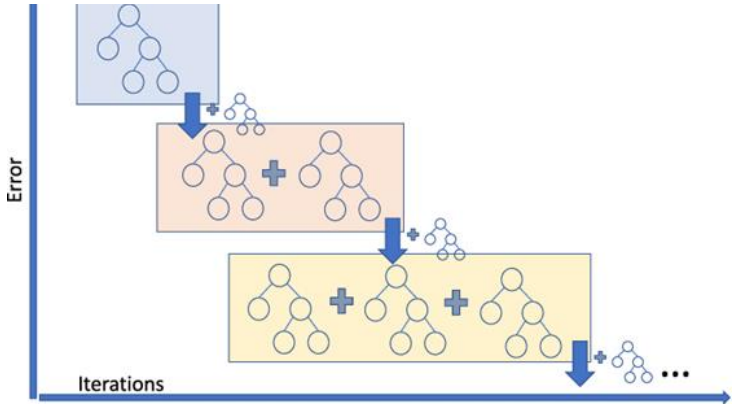

Figura 2-19 Ejemplo del funcionamiento de Gradient Boosting [14]

2.4.3.4 Histogram Gradient Boosting

Es una variable del algoritmo de Gradient Boosting que utiliza histogramas para acelerar el proceso de entrenamiento. En este modelo se discretizan las características numéricas en intervalos, reduciendo así la cantidad de valores únicos usados para determinar el umbral en cada partición del árbol y facilitando los cálculos. En lugar de calcular los gradientes directamente, se utiliza una aproximación basada en histogramas, lo cual requiere menos tiempo de computación.

2.4.4 Perceptrón

Esta técnica de aprendizaje supervisado se usa para la clasificación binaria y es uno de los primeros modelos de redes neuronales (**Figura 2-20**).

Cuando este algoritmo se entrena, se inicializan primero los pesos y el umbral (bias), y luego se le proporciona de entrada una instancia del conjunto de entrenamiento. Cada atributo de la instancia se multiplica por su respectivo peso y se suma junto con el umbral, siendo esto lo que recibe la función de activación que determina la salida del perceptrón. Si la salida es errónea, se ajustan los pesos, teniendo en cuenta ciertos parámetros como, por ejemplo, la tasa de aprendizaje. Se repite esto para todas las instancias del conjunto hasta que el perceptrón no cambie los pesos o se desee parar (para evitar el sobreajuste a la base de datos).

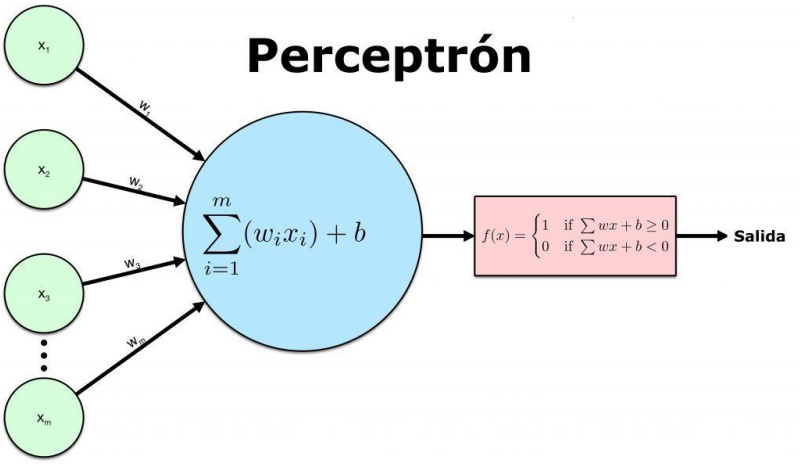

Figura 2-20 Ejemplo del funcionamiento de un Perceptrón [15]

2.4.5 Redes Neuronales (Perceptrón Multicapa)

Hay varios tipos de redes neuronales, siendo el más popular la del perceptrón multicapa. Este tipo de red neuronal está formada por una gran cantidad de perceptrones, los cuales representan a una neurona, inspirado en el funcionamiento del cerebro humano. Este algoritmo está compuesto por distintas capas, las cuales están contienen perceptrones, teniendo así un sistema como el que se muestra en la **Figura 2-21**.

Como se puede observar, cada capa oculta recibe como entrada la salida de otra capa, mientras que la capa de entrada tiene como entrada las características del problema. Esto quiere decir que cada perceptrón de una capa oculta recibe una entrada de cada perceptrón y manda como salida su resultado a cada perceptrón de la siguiente capa, así hasta la capa final.

El entrenamiento de este algoritmo es parecido al anterior, pero en este caso se debe retropropagar los errores hacia atrás, ya que el error real sólo se conoce en la capa de salida. De esta manera, el error influye en todas las capas de la red, haciendo que el modelo sea cada vez más eficiente.

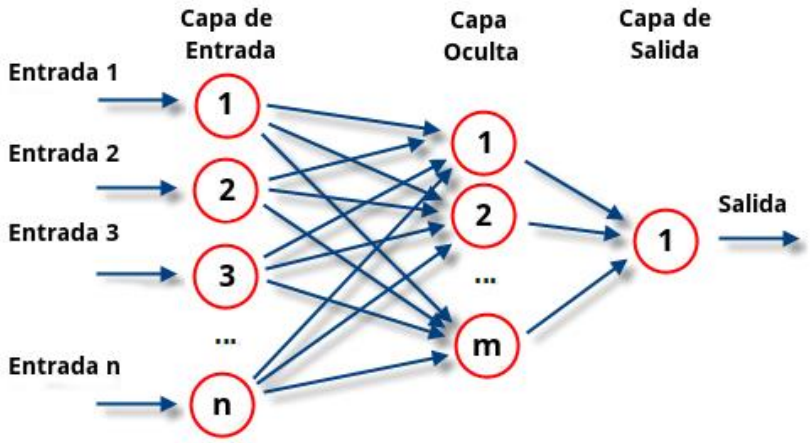

Figura 2-21 Ejemplo de una Red Neuronal Perceptrón Multicapa [16]

2.4.6 Reglas de asociación

Esta técnica de aprendizaje automático no supervisado consiste en encontrar un conjunto de reglas de asociación que tengan un soporte y confianza mínimo. Para ello, se calcula la frecuencia de cada elemento individual (por ejemplo, cada producto vendido en una tienda), y entonces se generan conjuntos que cumplan la frecuencia de compra (soporte) mínima requerida. De esta manera, se va iterando, creando grupos de elementos frecuentes cada vez más grandes, como se muestra en la **Figura 2-22**, los

cuales son filtrados cada iteración en base a ese soporte mínimo. Esto se conoce como el algoritmo APRIORI.

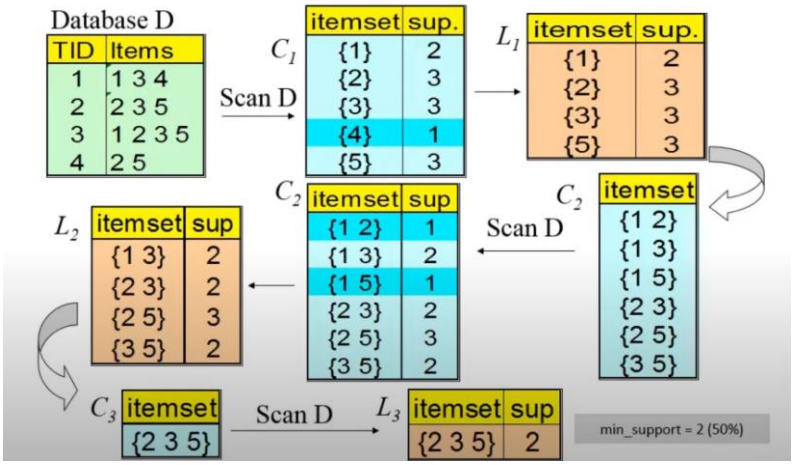

Figura 2-22 Ejemplo del algoritmo APRIORI [17]

2.5 Conclusiones

El TFT es un juego complicado, que tiene bastantes factores a tener en cuenta, pero a pesar de esto hay muchas ayudas tanto externas como con interfaces implementadas dentro del propio juego, para mejorar la experiencia del jugador novato, y que ayuda al jugador más experimentado a efectuar decisiones más acertadas. A pesar de esto, queda todavía mucho por mejorar, como se puede observar con MetaTFT [10] que cada vez más, va mejorando su base de datos, pudiendo hacer predicciones más acertadas y más útiles.

Todo esto es sin tener en cuenta el posicionamiento, que es algo que en niveles más avanzados puede tener un impacto diferencial en la partida, y también despreciando el factor aleatorio de las batallas, ya que hay algunas cosas que se deciden en tiempo de ejecución de manera aleatoria, como si un campeón está a una distancia igual de dos campeones enemigos, hacia cuál se mueve para atacarle.

Un posible futuro es que estas herramientas también ayuden al jugador con el posicionamiento, indicando los hexágonos en los cuales colocar sus campeones de manera que tengan un impacto más relevante en la batalla. Esto de todas maneras, no perjudicaría a los jugadores más experimentados, ya que se efectúan una cantidad considerables de cambios en el tablero en el último momento para coger desprevenido al rival, pero con esta herramienta se podría analizar y optimizar el posicionamiento tanto en tiempo real, como después de la partida, cuando se analice.

Como hemos visto, hay varias técnicas de aprendizaje automático que podrían usarse para realizar algunas predicciones útiles en el juego, lo único que haría falta es comprobar que técnica sería la más eficiente, y estudiar la manera de tratar, mostrar, y analizar los datos.

Capítulo 3

Metodología y Desarrollo

3.1 Introducción

En este capítulo se comentarán las herramientas utilizadas a lo largo del trabajo, así como lo que respecta a la obtención de los datos.

3.2 Herramientas utilizadas

Las herramientas que han sido utilizadas en este trabajo son las siguientes que se explican a continuación.

3.2.1 Python 3

El lenguaje de programación que ha sido utilizado en este proyecto es Python. Este lenguaje de programación se ha convertido en uno de los más populares en el campo del aprendizaje automático [18], ya que tiene una gran variedad de bibliotecas y frameworks específicos para el procesamiento de datos y la construcción de distintos tipos de modelos.

Python también es muy usado por los programadores más novatos por su legibilidad del código, ya que se acerca mucho al lenguaje natural, y permite comprender el código de una manera más sencilla que otros lenguajes de programación más complejos.

3.2.2 Scikit-learn

En lo que a librerías respecta, scikit-learn (desde ahora sklearn) ha sido usada en este proyecto para la creación del mayor tipo de modelos mediante técnicas de aprendizaje automático. Esto es así ya que sklearn proporciona una gran variedad de algoritmos y herramientas para el procesamiento de datos, selección de características, validación de modelos, etc.

Esta librería es de las más populares en el campo del aprendizaje automático debido a su facilidad de uso y su amplia gama de clases y funciones que hacen el desarrollo de modelos complejos mucho más sencillo y rápido.

3.2.3 Librerías adicionales

A pesar de haber utilizado sklearn para la gran mayoría de modelos, se han usado otras librerías para complementar y poder abarcar una mayor variedad en los modelos creados.

La librería principal para la lectura y manipulación de los datos ha sido Pandas. Haciendo uso de los DataFrames que esta librería proporciona, se han podido manejar correctamente los datos, y debido a la integración que tiene con otras librerías, como Sklearn y NumPy, también ha facilitado la creación de modelos y el manejo de los datos.

Para los modelos relacionados con las redes neuronales, se ha usado la librería TensorFlow, la cual es de las más poderosas para el aprendizaje automático en Python.

Esta librería proporciona abstracciones de alto nivel, al igual que Keras, lo que permite al programador enfocarse al diseño e implementación del modelo, sin tener que pensar en los detalles de bajo nivel. Además, tiene una gran variedad de capas y módulos que facilitan la construcción de distintos tipos de redes neuronales.

Para los modelos de reglas de asociación, se utiliza la librería Apyori, la cual proporciona una solución eficiente al problema de tratar los valores nulos en las listas de items.

Por último, también se han usado otras librerías en el proceso de obtención de los datos, como JSON, CSV y Request, las cuales se han utilizado para poder acceder a la base de datos de la compañía, filtrar los ficheros recibidos, y meterlos en un csv que posteriormente se trataría con Pandas.

3.3 Obtención de los datos

Para la obtención de los datos, se ha tenido que seguir los siguientes pasos.

3.3.1 Obtención de la API

Para obtener la API de Riot Games para poder tener acceso a la base de datos de la compañía, hay que entrar en su página de desarrolladores [19] y registrarse. Una vez hecho esto, en el perfil se puede generar una "API Key", la cual se debe usar en las "peticiones" para poder acceder a los datos.

Como se puede observar en la **Figura 3-1**, esta clave tiene ciertas limitaciones:

- Solo puede hacer 100 peticiones cada 2 minutos, con un máximo de 20 peticiones cada segundo
- La clave expira cada día, es decir, no puedes usar la misma clave durante varios días. Una vez expira la clave, se puede generar otra nueva sin ningún problema.

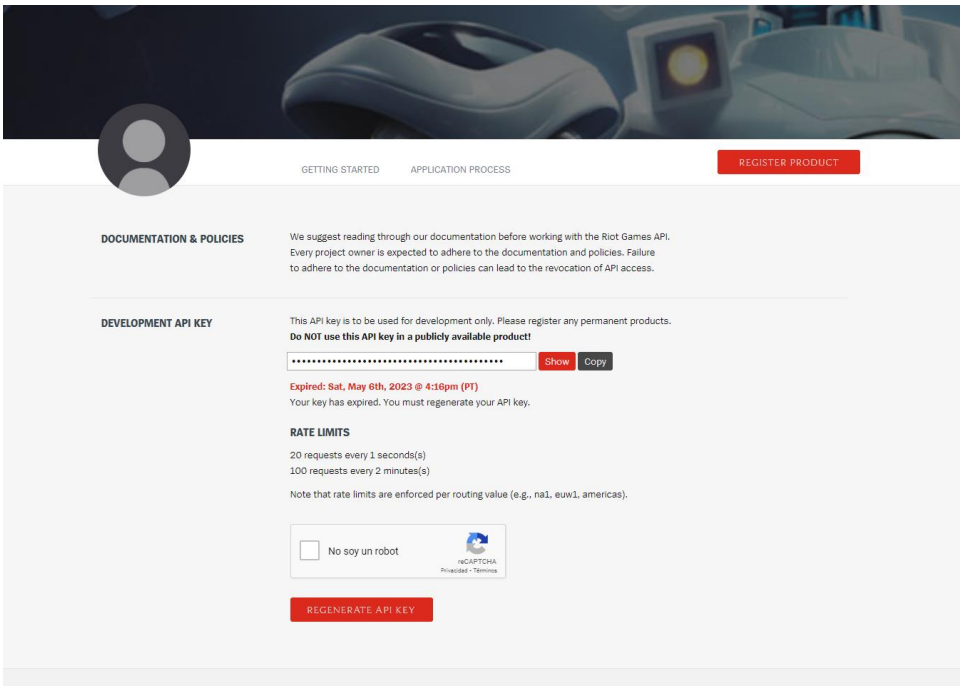

Figura 3-1 Imagen de la sección donde se consigue la clave

3.3.2 Obtención de una partida

Como lo que se quiere obtener son las partidas de los mejores jugadores del juego, se usará el ranking interno de clasificación que tiene el juego. Para ello, se usará la petición que devuelve a los mejores 200 jugadores del ranking del juego. Una vez obtenida una lista como la que aparece en **Figura 3-2**, se guarda el "summonerName". El "summonerName" es el nombre que tiene el jugador en su cuenta de Riot Games.

Figura 3-2 Resultado de la petición de los jugadores con “tier = Challenger”

Con el nombre de invocador, se puede obtener el “puuid”, que representa el identificador del jugador en la base de datos (**Figura 3-3**).

Figura 3-3 Resultado de la petición de información sobre el jugador “HAVALI”

Una vez obtenido todo esto, se pueden buscar las partidas del jugador según los parámetros que aparecen en la **Figura 3-4**.

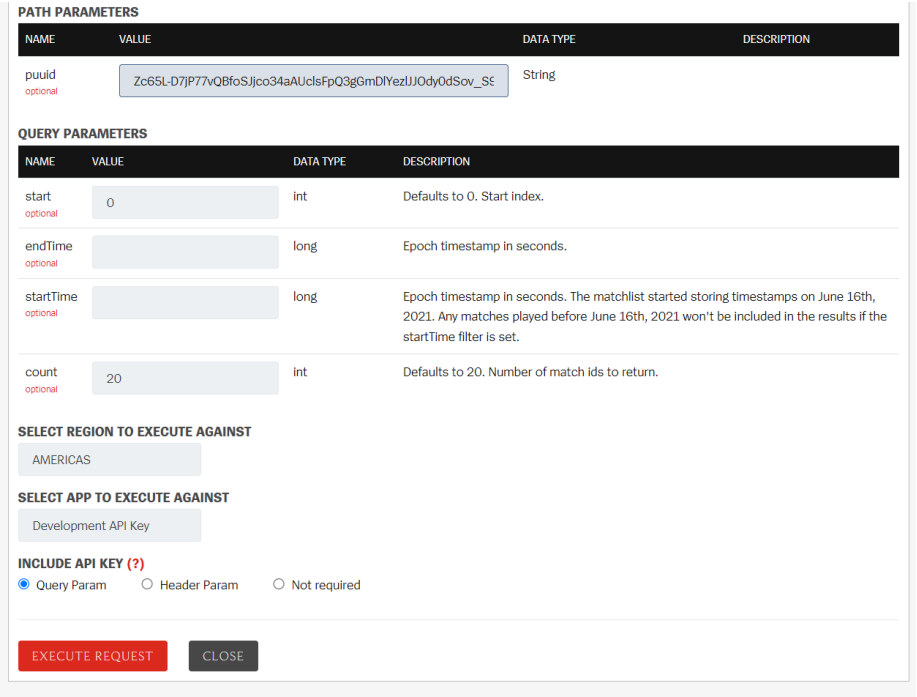

Figura 3-4 Parámetros de la obtención de partidas de un jugador.

Y así obtenemos una lista de las partidas del dicho jugador, como se muestra en la **Figura 3-5**.

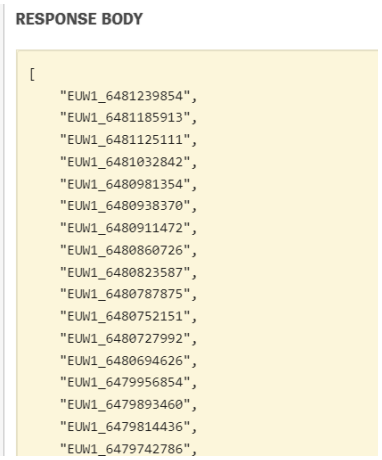

Figura 3-5 Resultado de la petición de la obtención del ID de las partidas de un jugador

Con el ID de la partida, ya se puede acceder a ella para obtener toda la información que se almacena en la base de datos, como la lista de participantes, cierta información de la versión del juego en ese momento, etc. Ver **Figura 3-6**.

RESPONSE BODY


```
{
    "metadata": {
        "data_version": "5",
        "match_id": "EUW1_6481239854",
        "participants": [
            "PPLaLam-g-6ABnsdwZkCHAcrG99mw24P7W-ocSehpfNZUy0uPiDxe5XSrVNoTSqActw19d6VUwF4uw",
            "xsmdmmVx2Si59QUKmYWOHRKKK2_Fj3uYWuHYs1EfAavsP3ZVu8o6fndsAsv5P1B9l76L_Q3HpQCK7Q",
            "cD_v0pAM5ePiPoJ8-1rI-xhOeEs2MmX1AcD4GWZIh3nMi9AaHRWWyhS5eTzCS-jNNsct0d8sOZ_3Fg",
            "Zc65L-D7jP77vQBfoSJjco34aAUclsFpQ3gGmDlYezlJJOdy0dSov_S9Z_2_NWWQ1COxTe_n6HhvqQ",
            "El7FQ-whv0mlaT6_TfLRPYC7SZWhSKgNlicCCgKislTfYFs3wjWFvpxR95z5YhN2YO172BDKJXEDhw",
            "3NShIGuHqVVkNFu05lOa4atCgcKeg-ewoGlqbVoO5EK6vlmMVS9WTRRXhiTfVXgFcpPXJ1Nz3BcqGQ",
            "XNTxcNVkss-s9LCMPDtrxyR1VqrEl3H6JBhmLJMmhNjzHBO_p3NOFKVuHUrlbXkGgb6a3HZjO2F92g",
            "JZBomd7VErOYZWzaBE_g7187SPu9Fes_oWeG4aBdEer_sYXF1V_QDAY8i8lDm3tpSEYr9EGa8-H1XA"
        ]
    },
    "info": {
        "game_datetime": 1688401037026,
        "game_length": 2154.048583984375,
        "game_version": "Version 13.13.518.0539 (Jun 28 2023/03:11:33) [PUBLIC] ",
        "participants": [
```


Figura 3-6 Resultado de la petición para obtener información sobre una partida

3.3.3 Automatización

Una vez explicado cómo se puede obtener la información de una partida de un jugador, solo falta automatizarlo para obtener varias partidas de varios jugadores. Para ello, se hará uso de un programa que se ha desarrollado en Python.

Primero de todo, se ha creado una función la cual se encarga de controlar la cantidad de peticiones por minuto, ya que si se excede el límite expuesto anteriormente, el mensaje que se recibe es de error, y puede dar problemas a la hora de automatizarlo. Es por eso, que lo que hace la función es contar hasta el límite establecido, y entonces esperar 2 minutos, para seguir las normas del uso de la API. Esta función tiene que ser llamada cada vez que se hace una petición al sistema.

Una vez establecido esto, se declaran algunas variables necesarias, como la región en la cual se obtendrán los datos (en este caso Europa), la clave API, y el url que corresponda, el cual está formado por una mezcla de las variables establecidas y una url base que proporciona Riot Games.

El programa en cuestión lo que hace es recorrer y almacenar el nombre de los mejores jugadores del juego, haciendo uso del ranking global del juego. Para el experimento se han cogido los 200 mejores jugadores, que tienen el rango de Challenger, los 800 siguientes, que tienen el rango de GrandMaster, y de Master se han filtrado a solamente los jugadores que tienen una puntuación superior a la mitad de los puntos máximos de la liga. Para ello, se han recorrido todos los jugadores de Master mirando sus puntos, obteniendo un valor mínimo y uno máximo, dividiéndolo entre dos, y almacenando el nombre de los que están en la mitad superior únicamente. Esto se ha hecho así debido a que hay una gran cantidad de jugadores en la mitad inferior de Master debido a que el sistema no deja bajar de división, como se ha comentado en la Introducción a Teamfight Tactics.

Una vez obtenida una lista de nombres, se debe obtener la lista de “puuid” asociada, y con esto, acceder a las partidas de los jugadores, en este caso, las últimas 200 partidas, guardándolas en una lista.

Debido a que se ha accedido a la lista de partidas de un jugador A y de un jugador B, las partidas en las que se hayan enfrentado el jugador A y el B, estarán dos veces, así que hay que filtrar las partidas recogidas y quedarse solamente con el identificador de la partida una vez.

En este punto, el programa guarda en un fichero .txt la lista de las partidas obtenidas, para que en caso de que ocurriese algún fallo, se pudiera continuar desde ahí, o por si hay que seccionar el almacenamiento de datos debido a que el número de partidas sea

demasiado grande como para acceder a todas en un solo día, antes de que caduque la clave y haya que parar el programa.

En este trabajo, en vez de seccionar en varios fragmentos las partidas e irlas recopilando a lo largo de los días, se ha optado por otra solución: se va a obtener únicamente un número de partidas específico que se puedan recopilar en un solo día. Este número de partidas es variable y será representado con **x.** Una posible solución sería simplemente obtener las **x** primeras partidas de la lista, pero esto tiene dos problemas: solo se tendría en cuenta a los **x**/200 jugadores primeros de la lista aproximadamente, y además se obtendrían partidas demasiado espaciadas en el tiempo entre sí, haciendo que las partidas con la misma versión del juego disminuyan.

Como solución, se puede iterar en la lista de partidas obtenidas según la formula:
“partida = lista_games[x + S*R]”.
Donde:

- x = número de la iteración actual. Es un puntero que ayuda al desplazamiento dentro de cada sección. Va de 0 hasta "seccióones"-1.
- S = constante "sección", la cual desplaza a las distintas zonas de la lista.
- R = Va de 0 a "ratio_games_jugador"-1, ayuda a desplazarse de una sección a otra multiplicándolo por S
- ratio_games_jugador: Variable que almacena cuantas partidas se obtienen de cada jugador. En un principio es 200, como se ha mencionado anteriormente, pero al quitar las partidas repetidas, se obtiene un nuevo ratio con una sencilla regla de tres en este caso.

De esta manera, teóricamente se obtendría una partida de cada jugador, sin que estén muy esparcidas en el tiempo, ya que se obtendría la primera del primer jugador, del segundo, etc. Hasta el último, y luego la segunda de cada uno y así.

Por último, para obtener la información de las partidas, se establece el número de partidas que se quiere obtener, y se itera de manera automática obteniendo una lista con la información de cada partida accedida. Esta lista se guarda en un archivo .json como punto de guardado.

3.3.4 Almacenamiento de datos

Una vez obtenido el archivo con la información de las partidas, hay que filtrar que variables son las relevantes para el problema que nos incumbe, y hay que plantear como almacenar dicha información de manera que pueda ser tratada de forma eficiente posteriormente sin perder información.

Selección de variables

No toda la información que se proporciona cuando se accede a una partida es relevante para este caso, así que hay que filtrar las variables que se van a almacenar. En este trabajo solamente se usarán siguientes variables:

- Versión del juego: Para poder filtrar y tener todas las partidas de la misma versión del juego.
- Posición del jugador: Para poder filtrar solamente a los jugadores que han quedado vivos uno contra el otro.
- Campeones en mesa e información sobre su nivel y sus objetos.
- Aumentos.

El resto de variables, como el oro que tenía el jugador en el momento que se terminó la partida, las sinergias de las fichas activadas, etc. No se tienen en cuenta, ya que lo relevante es lo que el jugador tenía dentro de la mesa en el momento de la última pelea.

Creación del CSV

Con esto en cuenta, se crea un csv con la siguiente estructura:

- La etiqueta PX indica el jugador al que se refiere la información recogida.
- Para el tier de las fichas, se sigue la nomenclatura: PX_TFT8_NombreFicha_Tier, indicando 0 si no existía esa ficha en el tablero, 1, 2 o 3, si estaba a 1 estrella, 2 estrellas o 3.
- Para los objetos de las fichas, se sigue la nomenclatura PX_TFDT8_NombreFicha_ObjX, siendo X igual a 1, 2 o 3, dependiendo del hueco de objeto que ocupe. En esta variable se guarda el nombre del objeto que tiene equipado el campeón.
- Los aumentos se guardarán en la variable PX_AugmentY, siendo Y igual a 1, 2, o 3, dependiendo de si es el primer, segundo, o tercer aumento de la partida.
- P1_Win indica con un 1 si ha ganado el jugador 1, si no, es un 0.

Para ambos jugadores se guarda toda esta información, teniendo como resultado un csv en el cual cada instancia representa una partida, con los valores nulos a 0, excepto por el P1_Win. Se llamará valores nulos a todos los valores que sean 0 y representen que un campeón no está en el tablero, o que un campeón no tiene objetos equipados.

3.3.5 Limpieza de datos

Dado que ya ha comprobado que no haya valores nulos cuando se han guardado las partidas, tenemos con certeza una base de datos completa y un csv con todos los registros completos.

3.4 Metodología de desarrollo : SCRUM

3.4.1 Introducción

Para la realización de este proyecto se ha elegido la metodología de ágil SCRUM. Esta metodología de trabajo permite dividir los bloques de trabajos en tareas más pequeñas, obteniendo una mejor organización del proyecto.

Scrum es una metodología de desarrollo de software que se centra en la flexibilidad, colaboración y entrega continua de productos. En esta metodología, se divide el trabajo en iteraciones cortas y fijas de tiempo, llamadas “sprints”, con una duración media de tres semanas, en las cuales se realizan las tareas planificadas para esa iteración. Al final del sprint, se entrega el producto resultado de la iteración.

3.4.2 Roles

Hay tres roles principales en un equipo Scrum:

- Propietario del producto: Este rol representa a los interesados en el producto final. Se encarga de financiar el proyecto, recuperar la inversión, lanzarlo, etc.

- Scrum Manager: Es el responsable de hacer que se cumpla el proceso Scrum. Se encarga de la formación y entrenamiento del proceso e incorporar la metodología Scrum a la empresa.

- Equipo de desarrollo: Se encargan de transformar la pila de trabajos del sprint en un incremento de la funcionalidad del software. Este equipo debe ser auto-gestionado, auto-organizado, y multi-funcional.

3.4.3 Aplicación a este proyecto

En este proyecto se ha optado por la metodología ágil Scrum debido al funcionamiento basado en sprints y reuniones. Se han identificado las tareas básicas necesarias para el correcto desarrollo del trabajo, dividiéndolas en tareas elementales y comprobando el avance del proyecto con reuniones al final de cada sprint.

3.4.3.1 Sprint 1: Obtención de la base de datos

El primer objetivo establecido en este trabajo fue la obtención de la base de datos. Esto se subdividió en las tareas especificadas en la sección de 3.3. Una vez obtenida la base de datos, se organizó una reunión para programar el siguiente sprint.

3.4.3.2 Sprint 2: Selección de variables y creación del primer modelo

El segundo sprint se basó en determinar las variables relevantes para el proyecto y la manera de organizar los datos para representarlos de la manera más óptima posible, sin perder información y evitando llenar el modelo de ruido y datos innecesarios.

Una vez razonado y escogida la manera en la cual poder pasarle los datos a un modelo, se construye un modelo básico de aprendizaje automático, en este caso, un árbol de decisión básico para la predicción de cuál de los dos jugadores ganará la partida introducida como instancia a clasificar, y se comprueba que todo está correctamente.

3.4.3.3 Sprint 3: Creación de modelos básicos

Una vez establecido como se van a representar los datos, se procede a la creación de los distintos tipos de modelos establecidos, para comparar los modelos entre sí y ahondar en los modelos que sean más efectivos al problema planteado.

3.4.3.4 Sprint 4: Obtención del modelo final

Una vez comparado todos los modelos obtenidos usando los algoritmos de clasificación descritos en el capítulo 2.4, se plantearon otro tipo de modelos más específicos para el problema que se está tratando, siendo este el de reglas de asociación. Concretamente la librería Apyori, la cual trata los valores que representan que una ficha no está presente en el tablero de tal manera que no generan ruido en el modelo, como ocurría con los otros tipos de algoritmos. Una vez obtenido y comprobado qué modelo es más eficiente, se efectúa un análisis completo y se finaliza el proyecto.

3.4.3.5 Sprint 5: Escritura de la memoria

Por último, después de la fase de experimentación, se procedió a registrar los resultados, dejando plasmado el trabajo realizado y la progresión del proyecto a lo largo del tiempo.

Capítulo 4

Experimentos y Resultados

4.1 Introducción

En este capítulo se hablará sobre los resultados obtenidos, la comparación entre modelos, y que parámetros se han ajustado según los criterios pertinentes.

Para la creación de todos los modelos, se ha separado la base de datos en un conjunto de entrenamiento y un conjunto de test, dividiéndolo cada uno en dos, los datos y la variable objetivo, en este caso P1_Win.

Todos los casos que tienen un factor aleatorio, como la división de la base de datos en entrenamiento y test, han sido controladas estableciendo una semilla para poder asegurar la reproducibilidad del proyecto.

4.2 K-NN

Empezando por las técnicas de aprendizaje supervisado, se comienza con la técnica de K-NN. Para entrenar este modelo con sklearn, se ha creado un “pipeline” básico para

poder preprocesar los datos. Para el preprocesamiento se ha utilizado un MinMaxScaler para las variables numéricas, y un OneHotEncoder para las categóricas.

Con la técnica de MinMaxScaler se obtiene una base de datos con valores numéricos uniformes, en este caso en concreto, entre 0 y 1. Lo que hace es establecer el valor mínimo (Min) como 0, y el valor máximo (Max) como 1, y el resto de valores intermedios se les asigna el valor que les corresponde en la nueva escala.

Por otro lado, el OneHotEncoder se encarga de transformar las variables categóricas en variables numéricas binarias, para que puedan ser tratadas por el árbol de decisión. Para ello, crea una matriz con la longitud igual al número de etiquetas únicas de la base de datos, y asigna un 1 solamente a una posición, representando de esta manera un valor categórico, como se puede ver en la **Figura 4-1**.

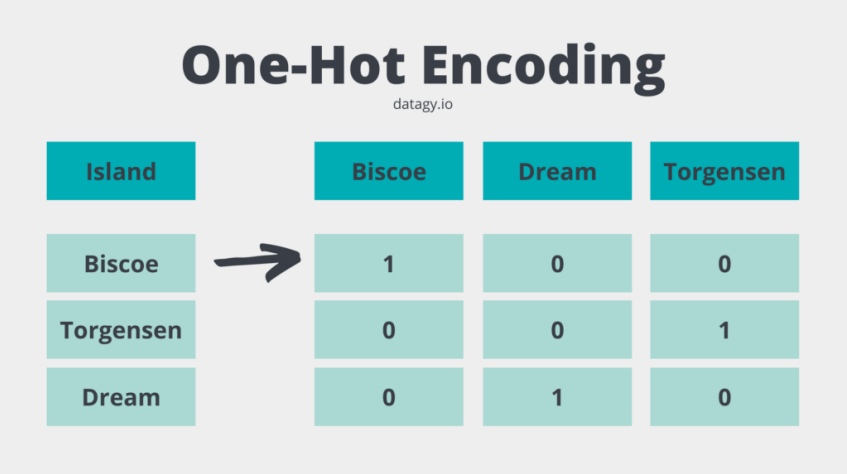

Figura 4-1 Ejemplo de OneHotEncoder [20]

Una vez que se ha creado el pipeline que ayudará a preprocesar los datos para los modelos, se establece el número de vecinos más cercanos a 5, obteniendo una puntuación sobre el conjunto de datos de entrenamiento de *0.7282* y sobre el de test de *0.5976.* Esta puntuación representa el porcentaje de acierto de este sobre el conjunto correspondiente.

Como el resultado puede depender mucho del número de vecinos seleccionado, es interesante optimizar los parámetros de este tipo de modelos. Aunque con el resultado obtenido no se espera una mejora abismal, ni incrementar el test score más del 0.8, se debe considerar hacerlo igualmente ya que es un factor muy determinante. Aprovechando que se tiene que optimizar ese parámetro, se optimizará también la técnica usada para asignar los pesos a los vecinos. Para automatizar la creación de todos los posibles modelos combinando los parámetros establecidos, se ha utilizado una función que realiza una búsqueda exhaustiva de los parámetros para crear un estimador utilizando validación cruzada, y muestra los resultados en una tabla. Se ha utilizado una validación cruzada estratificada de 5x10.

Los resultados de la **Tabla 1** indican, como evidentemente se había predicho, que los resultados de este tipo de modelo no mejoran excesivamente. De hecho, se puede observar que la máxima puntuación obtenida sobre el conjunto de test es de un modelo que sobreajusta en el de entrenamiento, aunque el tercer modelo mostrado no sobreajusta pero tiene una puntuación sobre el conjunto de test ligeramente peor.

	K-Vecinos	Pesos	Media train score	Media test score
1	10	Distancia	1	0.5842
2	8	Distancia	1	0.5797
3	7	Uniforme	0.7	0.5795
4	9	Distancia	1	0.5793
5	7	Distancia	1	0.5792
6	12	Distancia	1	0.5782
7	11	Distancia	1	0.5777
8	9	Uniforme	0.68	0.5762

Tabla 1 Primeros resultados de la optimización de K-NN

4.3 Árbol de decisión

Para el árbol de decisión, también es necesario preprocesar los datos utilizando un pipeline. El resultado obtenido es una puntuación de *1* sobre el conjunto de entrenamiento, indicando un gran sobreajuste, y una puntuación de *0.5592* sobre el conjunto de test. La matriz de confusión, tal y como se explica en la **Figura 4-2**, representa los valores que ha acertado el modelo (True Negatives y True Positives), y los que ha fallado (False Negatives y False Positives), y la generada por este modelo es la que se muestra en la **Tabla 2**.

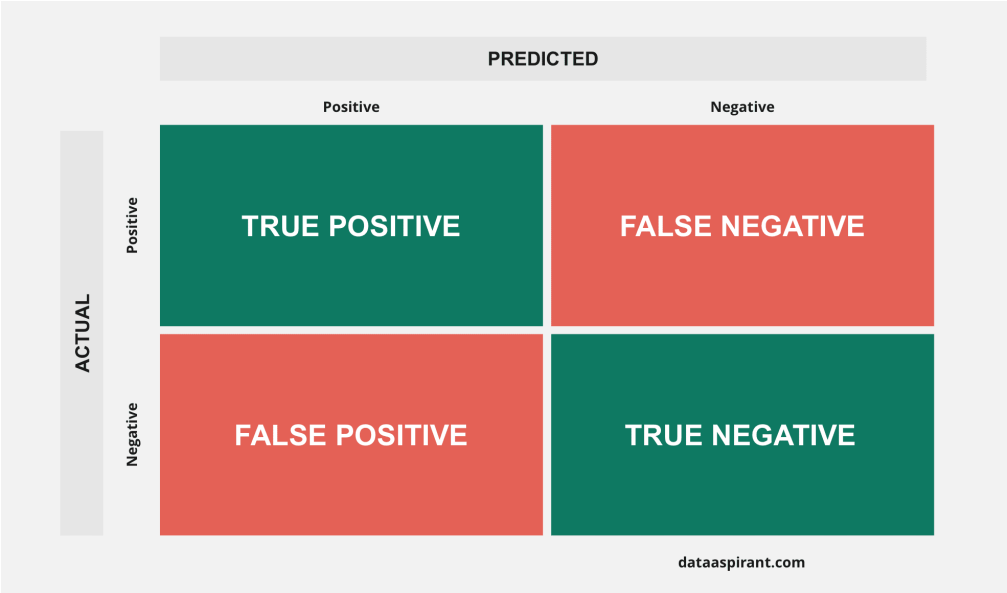

Figura 4-2 Explicación de una matriz de confusión [21]

	Positivos predichos	**Falsos predichos**
Positivos reales	254	173
Negativos reales	197	229

Tabla 2 Matriz de confusión del árbol de decisión

Como se ha obtenido una puntuación de casi 0.56, una posible opción de mejorar el modelo es optimizando los parámetros del árbol. Para esto, se probará con dos criterios para la creación del árbol, Gini y la entropía, así como varios valores para determinar la profundidad máxima del árbol y la relación complejidad-ajuste, es decir, el equilibrio entre la precisión del modelo y su complejidad. Para estos modelos se ha utilizado una técnica de validación cruzada estratificada de 1 x 10.

Para automatizar la creación de todos los posibles modelos combinando los parámetros establecidos, se ha utilizado una función que realiza una búsqueda exhaustiva de los parámetros para crear un estimador utilizando validación cruzada, y muestra los resultados en una tabla. Los resultados obtenidos son los que aparecen en la **Tabla 3**, y como se puede observar en la celda de "Media test score", la mayor puntuación obtenida es 0.57, así que se puede determinar que el problema de la puntuación no es por los parámetros, si no por el cómo se le presentan los datos al árbol o, simplemente, no es el modelo apropiado para el tipo de problema que se está tratando. Además, también se puede observar que si no se le define una profundidad al árbol, este tiende a sobreajustar, como ha pasado en el modelo creado anteriormente y en el cuarto modelo mostrado en la tabla.

	Complejidad-ajuste	**Criterio**	**Profundidad**	**Media train score**	**Media test score**
1	0.0	Gini	4	0.5919	0.5713
2	0.0	Entropía	4	0.5888	0.5674
3	0.0	Entropía	6	0.6141	0.5672
4	0.0	Entropía	No definida	1	0.5642
5	0.0	Gini	6	0.6297	0.5621

Tabla 3 Primeros resultados de la optimización del árbol de decisión

4.4 Random Forest

Una vez descartado el árbol de decisión, se optó por probar con modelos multiclasificadores (ensembles), comenzando con uno de los más básicos, Random Forest. Con este tipo de modelo también es necesario utilizar el pipeline para poder entrenarlo correctamente. Con los resultados obtenidos, se puede determinar que este modelo sobreajusta totalmente, teniendo una puntuación de *1* sobre el conjunto de entrenamiento, y *0.63* sobre el conjunto de test. Como este modelo es el más básico de los ensembles que se van a trabajar, no merece la pena optimizar los parámetros para evitar el sobreajuste, y es mejor enfocarse en modelos más elaborados para obtener una puntuación mejor. Aun así, es un buen punto de partida.

4.5 Cat Boost

Después de probar con el árbol de decisión básico y Random Forest, se eligió el modelo de Cat Boost. Este modelo es un tipo de Boosting el cual es capaz de tratar las variables categóricas sin hacerles ningún tipo de modificación, ya que, con el OneHotEncoder, las dimensiones de la base de datos crecían excesivamente, pudiendo suponer un problema.

Se creó un modelo básico de Cat Boost, sin pipeline, y estableciendo los parámetros básicos, como por ejemplo la tasa de aprendizaje, y se obtuvo una puntuación de *1* sobre el conjunto de entrenamiento, y una de *0.64* sobre el conjunto de test. Esta puntuación es mayor que la obtenida con el árbol de decisión y muy similar al modelo de Random Forest. Esto puede ser por el tipo de algoritmo (Boosting), aunque en este caso específico, puede influir el haber eliminado la transformación de las variables categóricas y poder tratarlas sin hacerles ningún cambio.

En este caso, no se ha especificado un número determinado de árboles que se aprenden durante el entrenamiento, así que el algoritmo usa el número

predeterminado de 1000 árboles. Como se ha obtenido un modelo que sobreajusta, se procede a crear un modelo con menos árboles para evitar el sobreajuste. El resultado con 150 árboles es de una puntuación de *0.89* sobre el conjunto de entrenamiento y una de *0.65* sobre el conjunto de test. Aunque el modelo no sobreajuste aprendiendo menos árboles, la puntuación sigue siendo similar.

4.6 Adaptive Boosting

Como cabe la posibilidad que los modelos de Boosting sean más eficientes ante este problema que el árbol de decisión, se comprueba si es cierto esa hipótesis creando más modelos de tipo Boosting, empezando por el Adaptive Boosting (AdaBoost).

Este tipo de modelo sí que necesita una codificación distinta para las variables categóricas, así que, para poder tratarlo bien, necesita hacer uso del pipeline. Una vez entrenado el modelo, se evalúa obteniendo una puntuación de *0.68* sobre el conjunto de entrenamiento y una de *0.66* sobre el conjunto de test. La puntuación obtenida con este modelo es bastante similar a la que se obtuvo con el CatBoost, pero en este modelo no existe el problema de sobreajuste. Con esto en cuenta, se puede determinar con más certeza que el método Boosting obtiene mejores resultados que el árbol de decisión, aunque para terminar de probar la hipótesis, se deben realizar algunas pruebas más.

Un posible escenario es que los parámetros utilizados en el entrenamiento del modelo no sean óptimos y se haya obtenido una puntuación inferior a la que se debería. Para poder comprobar completamente si ese es el caso, se optimizan los parámetros creando distintos tipos de modelos y comparando las puntuaciones obtenidas entre sí, igual que se hizo con el árbol de decisión. En este caso, los parámetros optimizados son el número de árboles que aprenderá el modelo, la tasa de aprendizaje, el criterio

utilizado para la creación del árbol, y su profundidad máxima. Se usará también una validación cruzada estratificada de 1 x 10.

Como se puede observar en la **Tabla 4**, la puntuación sobre el conjunto de test casi no mejora con respecto a la obtenida anteriormente, independientemente de los parámetros modificados. Con esto se puede concluir que este tipo de modelos no pueden obtener una puntuación mejor.

	Criterio	**Profundidad**	**Tasa de aprendizaje**	**N.º de árboles**	**Media train score**	**Media test score**
1	Gini	1	1	100	0.6978	0.6543
2	Gini	1	0.95	100	0.6970	0.6530
3	Entropía	1	1	100	0.6973	0.6529
4	Entropía	1	0.95	100	0.6971	0.6523
5	Gini	1	0.95	50	0.6789	0.6417
6	Gini	1	1	50	0.6779	0.6405

Tabla 4 Primeros resultado de la optimización del AdaBoost

4.7 Gradient Boosting

El siguiente modelo creado fue Gradient Boosting, haciendo uso también del pipeline para tratar las variables categóricas. Se optó por este modelo debido a su similitud con el AdaBoost, para poder reafirmar los resultados obtenidos con él, y comprobar cuál de los dos funciona mejor ante el problema planteado.

Con este modelo se obtiene una puntuación de *0.74* sobre el conjunto de entrenamiento y *0.65* sobre el conjunto de test. Como la puntuación obtenida ante el

conjunto de test es muy similar a la del modelo anterior, en este caso no es necesario la optimización de los parámetros puesto que se pueden esperar unos resultados parecidos.

4.8 Histogram Gradient Boosting

Por último, en cuanto a lo que respecta a las técnicas de Boosting, se prueba el Histogram Gradient Boosting, debido a como trata internamente los datos, creando histogramas y haciendo otro tipo de operaciones. Estos cambios en cómo se tratan los datos no deberían suponer un cambio en los resultados, pero sí una optimización con respecto al tiempo de CPU.

Para generar el modelo, es necesario hacer uso del pipeline creado para codificar las variables categóricas de manera que el modelo las pueda manejar. Una vez entrenado el modelo, la puntuación sobre el conjunto de test es de *0.63* y sobre el conjunto de entrenamiento de *0.*89. Como se esperaba, no varía casi el resultado con respecto a los otros modelos de Boosting, aunque el tiempo de ejecución ha sido más breve.

Además, este modelo tiene un ligero sobreajuste, ya que tiene una puntuación muy elevada sobre el conjunto de entrenamiento, la cual no se ve reflejada en la puntuación del conjunto de test.

4.9 Perceptrón

Para empezar en el campo de las redes neuronales, es interesante empezar primero con un modelo de regresión logística llamado Perceptrón, para comparar el cambio ante una red neuronal completa. Con este modelo también es necesario utilizar el pipeline ya que no es capaz de manejar variables categóricas. Una vez entrenado se obtiene un resultado obtenido bastante inferior a los modelos anteriormente creados,

con una puntuación de *0.61* sobre el conjunto de entrenamiento y de *0.58* sobre el de test, así que se descarta el Perceptrón como modelo solución.

4.10 Red Neuronal

Como último modelo de aprendizaje supervisado, se comprueba el resultado de utilizar redes neuronales ante el problema planteado. Como a una red neuronal no se le puede pasar un pipeline de la manera que se estaba haciendo con el resto de modelos, y no puede trabajar tampoco con las variables categóricas para la implementación de sklearn, es necesario hacer el preprocesamiento del pipeline sobre la base de datos. Además, antes de empezar con el entrenamiento es importante crear un "callback", en donde se guardará la mejor versión del modelo obtenida durante el entrenamiento.

Para todos los modelos creados se ha usado el método de activación "relu" para las capas densas intermedias, y "sigmoid" para la capa de salida. Además, se ha "Adam" como algoritmo para ajustar pesos y sesgos, la función de perdida de "entropía cruzada binaria", y "accuracy" como métrica

El primer modelo creado para la red neuronal consta de una capa densa con 32 neuronas, una de dropout con un valor de 0.15 para evitar el sobreajuste, y la capa densa de salida con 1 neurona. Este modelo se entrena 20 épocas, ya que como se puede observar en la **Tabla 5**, por muchas épocas que haya y por mucho que aumente el "Accuracy" (puntuación sobre el conjunto de test), no mejora el "Validation accuracy" (puntuación sobre el conjunto de validación).

Para comprobar si la red neuronal puede mejorar, se crea otro modelo algo más complejo con más capas y más neuronas, teniendo cuidado con no sobreajustar el modelo.

Época	Accuracy	Validation accuracy
1	0.5199	0.5545
2	0.5982	0.5780
3	0.6194	0.5967
4	0.6413	0.6155
…	…	…
19	0.7287	0.6354
20	0.7340	0.6366

Tabla 5 Resultado del primer modelo de red neuronal

En el segundo modelo, se prueba una red con más capas, con capas densas con 32, 64, 128 y 64 neuronas respectivamente, terminando con la capa de salida con 1 neurona. Para evitar el sobreajuste, se ha intercalado una capa de dropout después de cada capa.

Época	Accuracy	Validation accuracy
1	0.5109	0.5240
2	0.5669	0.5838
3	0.6067	0.5932
4	0.6355	0.6120
…	…	…
19	0.7856	0.6295
20	0.8000	0.6284

Tabla 6 Resultado del segundo modelo de red neuronal

Como se puede observar, aunque se obtenga una mayor puntuación sobre el conjunto de entrenamiento, no hay casi diferencia en la puntuación sobre el conjunto de validación, obteniendo un valor incluso un poco menor. De hecho, si se evalúan los modelos guardados en los callbacks, el primer modelo obtiene una puntuación mejor sobre el conjunto de test, de *0.66*, mientras que el segundo modelo obtiene una puntuación de *0.63*. Con estos dos modelos viendo que no se obtiene mejoras sustanciales, se da por terminado el estudio de las redes neuronales en este proyecto.

4.11 Finalización del sprint

Se da por terminado los modelos de aprendizaje supervisado, finalizando el sprint. En este momento se comparan los resultados obtenidos hasta ahora y se plantea como afrontar el siguiente sprint.

Como se puede observar en la **Tabla 7**, la red neuronal básica es el modelo que mejor resultado ha dado, seguida de todos los modelos de Boosting. Esto nos plantea que quizás probando distintos tipos de redes neuronales, modificando los parámetros obtenidos se pueden obtener mejores modelos de redes neuronales.

De todas formas, viendo el resultado del resto de modelos, se plantea el motivo de haber obtenido una puntuación tan baja. Al final, se concluye que puede ser debido a la gran cantidad de valores nulos que tiene la base de datos, ya que en un tablero de TFT final suele haber 8 o 9 fichas, pero cada instancia tiene el valor de los campeones que no están en el tablero con un valor 0, teniendo como resultado que cada instancia tiene más ruido que información útil, propagando este ruido a los modelos y siendo imposible entrenar un modelo eficiente.

Modelo	Mejor Puntuación
Redes Neuronales	0.6624
AdaBoost	0.6543
CatBoost	0.6495
Gradient Boosting	0.6471
Random Forest	0.6307
Histogram Gradient Boosting	0.6284
K-NN	0.5967
Perceptrón	0.5850
Árbol de decisión	0.5713

Tabla 7 Comparación de los resultados obtenidos

Como se puede comprobar en la **Tabla 8**, en las características "Px_TFT8_Personaje_Tier", no se obtiene cuantos valores son 0 debido a que se están filtrando por "0", es decir, por un string que sea 0, no por un valor numérico. A pesar de esto, se puede observar por las características que hacen referencia a los objetos que casi todos los valores de la gran mayoría de los personajes son 0, ya que la base de datos tiene 4263 instancias, y, por ejemplo, el campeón Ashe tiene el objeto 3 vacío en 4256 instancias, es decir, en el 99,8% de los casos.

Con estas primeras conclusiones, se decide explorar las reglas de asociación, sobre todo la librería Apyori, la cual esos valores a 0 de cada instancia se ignoran, usando únicamente los valores válidos para la creación de reglas de asociación, planteando así el siguiente sprint.

Característica	N.º de valores “0”
P1_TFT8_Ashe_Tier	0
P1_TFT8_Ashe_Obj1	4219
P1_TFT8_Ashe_ Obj2	4248
P1_TFT8_Ashe_ Obj3	4256
P1_TFT8_Blitzcrank_Tier	0
P1_TFT8_Blitzcrank_ Obj1	4105
P1_TFT8_Blitzcrank_ Obj2	4166
P1_TFT8_Blitzcrank_ Obj3	4200
…	…
P2_Augment3	3
P1_Win	0

Tabla 8 Cantidad de valores nulos en la base de datos

4.12 Reglas de asociación

Para poder entrenar un modelo usando reglas de asociación, lo primero es tratar los datos para poder entrenarlo. Para ello, hay que pasar de la base de datos actual a una lista de transacciones, quitando así todos los valores nulos existentes. También, la variable objetivo pasa de ser valores 0 o 1, a ser un item de la transacción el cual es “P1_Win” o “P1_Lose”. Se obtiene una lista de transacciones como la que aparece en la **Tabla 9**, y como se puede observar, se ha guardado el tablero del jugador 1 completo sin valores nulos, incluyendo el resto de los valores que son útiles en el problema que se está manejando.

Transacción	Valores
1	['P1_TFT8_Poppy_Tier 2', 'P1_TFT8_Annie_Tier 2', 'P1_TFT8_Annie obj1 TFT_Item_WarmogsArmor', 'P1_TFT8_Annie obj2 TFT8_Item_TitansResolve_GenAE', 'P1_TFT8_Annie obj3 TFT8_Item_HandOfJustice_GenAE', 'P1_TFT8_GnarBig_Tier 3', 'P1_TFT8_GnarBig obj1 TFT_Item_TitansResolve', 'P1_TFT8_GnarBig obj2 TFT_Item_Bloodthirster', 'P1_TFT8_GnarBig obj3 TFT_Item_LastWhisper', 'P1_TFT8_Morgana_Tier 2', ... 'P2Augment1 TFT8_Augment_GenAEEmblem', 'P2Augment2 TFT8_5_Augment_GnarSupport', 'P2Augment3 TFT6_Augment_CelestialBlessing2', 'P1_Lose']

Tabla 9 Ejemplo de transacción

Con la lista de transacciones ya se puede entrenar el modelo. Cabe destacar, que debido a la gran cantidad de reglas que se genera, el tiempo de entrenamiento de este modelo puede ser desorbitado, siendo con creces de los que más ha tardado en entrenar. Además, para obtener reglas relevantes, se indica por parámetro el índice de confianza mínimo, para obtener solamente las que tengan un índice de confianza mayor que 0.7. Una vez entrenado, se obtienen reglas de asociación como las que se muestran en la **Tabla 10**. Como se puede observar, hay reglas de asociación que no aportan la información que se busca, es decir, que no dicen que jugador va a ganar. Para solucionar esto y obtener solamente las reglas importantes, se filtran para obtener únicamente las que tengan “P1_Win” o “P1_Lose”.

Regla	Antecedente	Consecuente	Soporte	Confianza
1	P1_Augment1 TFT8_Agument_AnnieSupport	P1_TFT8_Annie_Tier 2	0.0114	0.8298
2	P1_Augment1 TFT8_Agument_EzrealSupport	P1_TFT8_Ezreal_Tier 2	0.0117	0.8000
3	P1_Augment1 TFT8_Agument_GenAEEmblem	P1_TFT8_Annie_Tier 2	0.0173	0.8082
4	P1_Augment1 TFT8_Agument_GenAEEmblem	P1_TFT8_Zoe_Tier 2	0.0164	0.7671

Tabla 10 Ejemplos de reglas de asociación

El resultado del filtrado es similar al que aparece en la **Tabla 11**, y como se puede observar se han obtenido varias reglas de asociación con más de un item en el antecedente.

Regla	**Antecedente**	**Consecuente**	**Soporte**	**Confianza**
1	P1_TFT8_Alistar_Tier 2, P2_TFT8_Aphelios obj1 TFT_Item_LastWhisper	P1_Lose, P2_TFT8_Aphelios_ Tier2	0.0103	0.7292
2	P1_Augment1 TFT8_Agument_EzrealSupport	P1_TFT8_Samira_ Tier2, P1_Win	0.0173	0.7024
3	P1_Augment1 TFT8_Agument_GenAEEmblem	P1_TFT8_Samira_ Tier2, P1_Win	0.0111	0.7037
4	P1_Augment1 TFT8_Agument_GenAEEmblem	P1_TFT8_Samira_ Tier2, P1_Win	0.0132	0.7377

Tabla 11 Ejemplo de las reglas de asociación filtradas

Para comprobar el accuracy de este tipo de modelos, se ha creado una función que, transacción por transacción, comprueba cuantas reglas se disparan con dicha transacción, y anota lo que estas reglas predicen. Una vez recorridas todas las reglas, efectúa una predicción en base al voto por la mayoría, y anota si ha acertado con la predicción o no, guardando así el número de aciertos y el número total de predicciones efectuadas. Por último, se dividen estos dos números al finalizar todas las transacciones de entrada, obteniendo así un porcentaje de acierto. Utilizando esta función se obtiene una puntuación sobre el conjunto de entrenamiento de *0.65* y sobre el conjunto de test de *0.59*.

Para comprobar que la puntuación obtenida es fiable, se volvió a entrenar un modelo para un momento distinto del estado del juego. Para esto, había que volver a generar la base de datos especificando la versión del juego que se quería obtener. Una vez hecho esto, entrenado el modelo de reglas de asociación, y filtrado las reglas, se obtiene una lista de reglas finales mucho menor que en el modelo anterior. Se eligió precisamente esa versión del juego porque en ese momento predominaba una composición la cuál una vez conseguida, ganaba gran cantidad de partidas, y eso se traduce en una cantidad menor de reglas. En este caso, la puntuación obtenida sobre el conjunto de entrenamiento es de *0.51*, siendo de los peores resultados que se han obtenido en el proyecto.

Con esto se puede intuir que a pesar de que el estado del juego sea más estable, si el modelo genera un mayor número de reglas de asociación, será más fiable.

4.13 Conclusiones

Tras haber probado distintos tipos de modelos, se puede concluir que las redes neuronales son las que mejor tratan este tipo de problema. Es posible que algún modelo de red neuronal sea muy efectivo para el proyecto actual, y cabe la posibilidad

que, debido a que no hay manera de determinar qué configuración de red neuronal es la óptima, no se haya podido encontrar en las pruebas efectuadas la red neuronal adecuada.

Puede que exista una manera más eficiente de representar los datos, de forma que no haya tanto ruido en el conjunto de entrenamiento y el resto de modelos clásicos probados en este experimento funcionen mejor, pero actualmente con los datos representados de la forma en la que se ha establecido, el modelo que mejor resultados ha dado ha sido el modelo de redes neuronales.

Debido al alto tiempo de ejecución de las reglas de asociación, no se han podido efectuar una gran cantidad de experimentos sobre estos modelos, a pesar de que es el tipo de algoritmo que más sentido tiene que funcione, debido a como se tratan los valores nulos y evita el ruido.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1 Conclusiones

El objetivo principal de este trabajo era crear un programa capaz de, dado dos tableros de dos jugadores, determinar que jugador saldrá victorioso en el último enfrentamiento. Para realizar esta predicción se ha recurrido a la analítica de datos, analizando partidas de la base de datos de la empresa Riot Games, las cuales han sido obtenidas a través de una API que proporciona dicha empresa.

Durante el proyecto, se han utilizado distintas técnicas de aprendizaje automático, razonando por qué se ha usado cada una y qué modelo podría mejorar los resultados obtenidos.

También se ha aplicado una metodología de desarrollo ágil, la cual ha facilitado la organización del proyecto, dividiéndolo en sprints y valorando al final de cada sprint los resultados y el trabajo futuro.

El lenguaje programación utilizado durante el desarrollo del proyecto ha sido Python, lo cual ha permitido ahondar en el lenguaje de programación, descubriendo librerías que no había utilizado todavía y familiarizándome más con las que si conocía.

El mayor obstáculo con el cual se ha tenido que lidiar ha sido la representación de los datos, ya que la solución planteada en este trabajo hace que se genere una base de datos con un gran número de características, teniendo muchos valores nulos, y siendo casi imposible para los modelos obtener un buen resultado. Como solución se ha planteado las regalas de asociación, las cuales han conseguido lidiar con esos valores nulos y modificar la forma en la cual se muestra la información en la base de datos.

Otro inconveniente a la hora del desarrollo del proyecto ha sido el tiempo de ejecución de algunas partes del proyecto, ya que para la obtención de la base de datos el programa podía demorarse casi 24 horas debido a la gran cantidad de tiempo que hay que esperar entre peticiones al servidor. Esto también ha influido en el tamaño de la base de datos, puesto que a las 24 horas caducaba la API, y solo se ha obtenido lo que se ha podido recopilar durante ese periodo de tiempo. Además, los modelos de reglas de asociación han tenido un tiempo de entrenamiento bastante alto, no inferior a 5 horas, debido a la gran cantidad de reglas que generan, aunque una gran parte de ellas no sean utilizadas.

Hay que remarcar que los datos disponibles corresponden al último enfrentamiento de la partida y no a las fases intermedias de la misma, y por eso este trabajo está limitado a dicho enfrentamiento únicamente.

5.2 Competencias trabajadas

Tal y como se especificó en apartado de Competencias, se muestra en esta sección el trabajo realizado con respecto a cada competencia:

- Se ha valorado, interpretado, seleccionado y modelado distintos tipos de técnicas de aprendizaje automático, ampliando el conocimiento de los principios fundamentales y modelos de la computación.
- Se han desarrollado aplicaciones informáticas que aplican los fundamentos, paradigmas, y técnicas propias de los sistemas inteligentes.
- Se ha obtenido, formalizado y representado el conocimiento humano en una forma computable para la resolución de problemas mediante un sistema informático relacionado con aspectos de computación, percepción y actuación en ambientes o entornos inteligentes.
- Se han desarrollado técnicas de aprendizaje computacional y se han diseñado e implementado un sistema que la utiliza, extrayendo automáticamente información y conocimiento a partir de grandes volúmenes de datos.

5.3 Trabajo futuro

Como se ha ido valorando a lo largo del proyecto, hay varias secciones en las cuales se pueden seguir trabajando e intentando mejorar el proyecto:

- Integración del modelo creado a una interfaz in-game que muestre al usuario el porcentaje de victoria que tiene en la mesa final, dando una aplicación al modelo creado.
- Plantear nuevas maneras de representar los datos de manera que no haya un exceso de valores nulos para que la gran mayoría de modelos planteados en este proyecto puedan extraer información más útil del conjunto de entrenamiento, mejorando su resultado.
- Ampliación de la base de datos con la interfaz in-game creada, la cual podría captar los datos intermedios de la partida y así poder entrenar modelos que no

solamente ayuden a la batalla final, si no a todas las batallas que ocurren a lo largo de la partida.

- Continuar haciendo experimentos optimizando los parámetros en las reglas de asociación.

Bibliografía

[1] «Los videojuegos están en auge y se espera que la industria siga creciendo.», *Foro Económico Mundial*, 9 de septiembre de 2022. https://es.weforum.org/agenda/2022/09/el-juego-esta-en-auge-y-se-espera-que-siga-creciendo-este-grafico-le-dice-todo-lo-que-necesita-saber/

[2] «League of Legends Live Player Count and Statistics», 11 de julio de 2022. https://activeplayer.io/league-of-legends/

[3] «TFT Update: Numbers & What's Next», 25 de septiembre de 2019. https://nexus.leagueoflegends.com/en-us/2019/09/tft-update-numbers-whats-next/

[4] T. W. published, «Teamfight Tactics hit 10 million daily players at a recent peak», *PC Gamer*, 17 de marzo de 2021. [En línea]. Disponible en: https://www.pcgamer.com/teamfight-tactics-draws-10-million-daily-players-at-its-peaks/

[5] *EL SET 10 DE TFT SERÁ UN ANTES Y DESPUÉS*, (31 de mayo de 2023). [En línea Video]. Disponible en: https://www.youtube.com/watch?v=K1Tc72x-y5g

[6] M. eSports, «Riot transforma el tablero de TFT», *Movistar eSports*, 18 de octubre de 2019. https://esports.as.com/league-of-legends/tacticas-maestras--tft/tablero-TFT-hara-grande_0_1292270766.html

[7] «/dev: Ranked Teamfight Tactics», 5 de julio de 2019. https://nexus.leagueoflegends.com/en-us/2019/07/dev-ranked-teamfight-tactics/

[8] «TFT All Tier Leaderboards EUW »:, *TFT Stats, Leaderboards, League of Legends Teamfight Tactics - LoLCHESS.GG*. https://lolchess.gg/leaderboards?mode=ranked&region=euw

[9] «TFT Meta, Stats, Comps, Match History and more Teamfight Tactics tools», *tactics.tools*. https://tactics.tools/

[10] «MetaTFT - Discover the TFT Meta & Stats for Set 9». https://www.metatft.com

[11] T. Srivastava, «A Complete Guide to K-Nearest Neighbors (Updated 2023)», *Analytics Vidhya*, 25 de marzo de 2018. https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

[12] «Python Decision Tree Classification Tutorial: Scikit-Learn DecisionTreeClassifier». https://www.datacamp.com/tutorial/decision-tree-classification-python

[13] V. Alto, «Understanding AdaBoost for Decision Tree», *Medium*, 31 de enero de 2020. https://towardsdatascience.com/understanding-adaboost-for-decision-tree-ff8f07d2851

[14] A. Choudhury, «What is Gradient Boosting? How is it different from Ada Boost?», *Analytics Vidhya*, 24 de diciembre de 2020. https://medium.com/analytics-vidhya/what-is-gradient-boosting-how-is-it-different-from-ada-boost-2d5ff5767cb2

[15] J. M. Alvarez, «El perceptrón como neurona artificial», *Blog de Jose Mariano Alvarez*, 11 de junio de 2018. https://blog.josemarianoalvarez.com/2018/06/10/el-perceptron-como-neurona-artificial/

[16] L. Salcedo, «Introducción a las Redes Neuronales- Parte #1 - Tipos de Redes Neuronales», *Mi Diario Python*, 31 de julio de 2018. https://pythondiario.com/2018/07/introduccion-las-redes-neuronales-parte.html

[17] *Reglas de asociación*, (18 de febrero de 2019). [En línea Video]. Disponible en: https://www.youtube.com/watch?v=XvaAJhLlNyg

[18] «¿Qué es Python y a qué se debe su popularidad?», *Computer Hoy*, 26 de marzo de 2022. https://computerhoy.com/reportajes/tecnologia/python-debe-popularidad-1022077

[19] «Riot Developer Portal». https://developer.riotgames.com/apis

[20] Nik, «One-Hot Encoding in Scikit-Learn with OneHotEncoder • datagy», *datagy*, 23 de febrero de 2022. https://datagy.io/sklearn-one-hot-encode/

[21] «3_confusion_matrix.png (1968×1160)». https://dataaspirant.com/wp-content/uploads/2020/08/3_confusion_matrix.png

Anexo I. Estructura del código

I.1 Código utilizado durante el proyecto y organización

El repositorio con el código fuente del proyecto y los resultados obtenidos del estudio es accesible a través de esta dirección en la plataforma GitHub:

https://github.com/VibingFrog/CodigoTFG_JavierTomasFernandezMartin

Este repositorio contiene un archivo llamado “Leeme.txt” el cuál explica la organización del proyecto y que contiene cada archivo.