



Universidades de Burgos, León y Valladolid

Máster universitario

Inteligencia de Negocio y Big Data en Entornos Seguros



**TFM del Máster Inteligencia de Negocio y Big
Data en Entornos Seguros**

**Uso de técnicas de aprendizaje no
supervisado para la ayuda en
videojuegos tipo MOBA**

Presentado por Iván Iglesias Cuesta
en la Universidad de Burgos — 3 de septiembre de 2021

Tutores: Dr. José Francisco Díez Pastor y
Dr. César Ignacio García Osorio

Universidades de Burgos, León y Valladolid



Máster universitario en Inteligencia de Negocio y Big Data en Entornos Seguros

D. José Francisco Díez Pastor, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

D. César Ignacio García Osorio, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Exponen:

Que el alumno D. Iván Iglesias Cuesta, con DNI 45573756S, ha realizado el Trabajo final de Máster en Inteligencia de Negocio y Big Data en Entornos Seguros titulado «Uso de técnicas de aprendizaje no supervisado para la ayuda en videojuegos tipo MOBA».

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 3 de septiembre de 2021

Vº. Bº. del Tutor:

Vº. Bº. del Tutor:

D. José Francisco Díez Pastor

D. César Ignacio García Osorio

Resumen

Desde hace unos años, el mundo de los deportes electrónicos, o *eSports*, no para de crecer. Con la gran cantidad de jugadores dedicados a estos títulos, como entretenimiento y como carrera profesional, se generan constantemente datos que pueden explotarse para ayudar a nuevos jugadores a iniciarse, y analizar las propias tendencias internas del juego. Generalmente la barrera de entrada en estos juegos es elevada, al ser necesario tener conocimientos de sus mecánicas, para poder desenvolverse en partida y para disfrutar de la experiencia, por lo que los primeros pasos suelen ser frustrantes. Por ello, en este trabajo, usando de ejemplo *League of Legends* (uno de los *eSports* más populares), se plantea un proceso de recopilación, tratamiento y obtención de conocimiento, a partir de las partidas jugadas. Adicionalmente se ha creado una aplicación web para mostrar los resultados obtenidos del análisis de partidas.

Descriptores

MOBA, League of Legends, Riot Games, aprendizaje no supervisado, conjuntos frecuentes, ETL, Django, MongoDB.

Abstract

For a number of years now, the world of eSports has been growing steadily. With the large number of players dedicated to these titles, both as entertainment and as a career, data is constantly being generated that can be exploited to help new players get started, and to analyse the game's own internal trends. Generally, the entry barrier in these games is high, as it is necessary to have knowledge of the game mechanics to be able to play the game and enjoy the experience, so the first steps are often frustrating. For this reason, in this project, using the example of League of Legends (one of the most popular eSports title), a pipeline for collecting, processing and obtaining knowledge from past games is proposed. Additionally, a web application is created to display the results obtained from the games analysis.

Keywords

MOBA, League of Legends, Riot Games, unsupervised learning, frequent itemsets, ETL, Django, MongoDB.

Índice general

Índice general	iii
Índice de figuras	vi
Índice de tablas	vii
Memoria	1
1. Introducción	3
2. Objetivos del proyecto	5
2.1. Objetivos principales	5
2.2. Objetivos personales	5
3. Conceptos teóricos	7
3.1. ETL	7
3.2. Aprendizaje no supervisado	7
3.3. Conceptos sobre <i>League of Legends</i>	8
4. Técnicas y herramientas	15
4.1. Scrum	15
4.2. Git	15
4.3. GitHub	16
4.4. Jupyter	16
4.5. Django	16
4.6. MLXtend	16
4.7. Cassiopeia	16

4.8. MongoDB	17
5. Aspectos relevantes del desarrollo del proyecto	19
5.1. Problema encontrado y propuesta de solución	19
5.2. Desarrollo de <i>notebooks</i> y web	20
5.3. Control de velocidad de peticiones por ratios de la API	20
5.4. Tolerancia y recuperación de fallos	21
5.5. Proceso de recopilación y entrenamiento	22
5.6. Elección del algoritmo	24
6. Trabajos relacionados	27
6.1. LoL Data Solution [12]	27
7. Conclusiones y Líneas de trabajo futuras	29
7.1. Conclusiones	29
7.2. Líneas de trabajo futuras	29
Apéndices	31
Apéndice A Plan de Proyecto Software	33
A.1. Introducción	33
A.2. Planificación temporal	33
A.3. Estudio de viabilidad	38
Apéndice B Especificación de Requisitos	41
B.1. Introducción	41
B.2. Objetivos generales	41
B.3. Catálogo de requisitos	42
Apéndice C Especificación de diseño	43
C.1. Introducción	43
C.2. Diccionario de datos	43
C.3. Diseño procedimental	45
Apéndice D Documentación técnica de programación	47
D.1. Introducción	47
D.2. Estructura de directorios	47
D.3. Manual del programador	49
D.4. Instalación y ejecución del proyecto	50
Apéndice E Documentación de usuario	53

Índice general

v

E.1. Introducción	53
E.2. Requisitos de usuarios	53
E.3. Instalación	53
E.4. Manual del usuario	53
Bibliografía	61

Índice de figuras

3.1. Proceso de selección de campeón	9
3.2. Mapa de <i>League of Legends</i>	10
C.1. Extracción de jugadores	45
C.2. Extracción de partidas	46
C.3. Generación de transacciones y conjuntos frecuentes	46
E.1. Pantalla de inicio	54
E.2. Listado de campeones	55
E.3. Buscador	55
E.4. Ficha del campeón	56
E.5. Ficha de otro campeón	57
E.6. Búsqueda por jugador	58
E.7. El jugador no está en partida	58
E.8. No existe el jugador con el nombre introducido	59

Índice de tablas

3.1. Sistema de ligas	14
5.2. Tiempo de ejecución de los algoritmos	25
A.1. Costes de personal	38
A.2. Tabla de dependencias	38
C.1. Entidad del jugador	43
C.2. Entidad de la partida	44
C.3. Transacciones	45
C.4. Conjuntos frecuentes de objetos	45

Memoria

Introducción

Las competiciones de deportes electrónicos, o *eSports*, al igual que los deportes tradicionales, mueven grandes cantidades de dinero a la vez que atraen a un número muy elevado de espectadores a sus retransmisiones.

Por lo general los juegos de los que se realizan competiciones son gratuitos, por lo que cualquier persona puede adentrarse en ese mundillo para pasar un rato entretenido, o ponerse la meta de llegar a ser profesional.

Sin embargo, llegar a ser profesional es un objetivo complicado por la gran cantidad de horas necesarias para conseguir las capacidades necesarias para llegar a esos niveles de competencia. Además, el rango de edad en el que es más necesario dedicar más horas coincide con etapas de escolarización todavía obligatorias, pudiendo crear conflicto de intereses.

En ambos ámbitos, profesional y casual, se genera constantemente una gran cantidad de datos, tomando la forma de un registro de partidas jugadas. Para este trabajo me he propuesto analizar ese histórico de partidas en uno de los *eSports* más predominantes del momento, *League of Legends*, un videojuego dentro del tipo MOBA (*Multiplayer Online Battle Arena*).

Por dar un poco de contexto a la importancia de este título en el mundo actual, en el año 2020 el juego obtuvo un beneficio de 1.750 millones de dólares [10]. En comparación con otros mercados más tradicionales, en el mismo año, Inditex obtuvo 1.100 millones de euros de beneficios [8].

Además de tener números impresionantes en cuanto a beneficios, se estima que, en lo que llevamos de 2021, cada mes juegan de media, hasta 115 millones de personas [13]. Este elevado número de jugadores genera una gran cantidad de datos, en forma de partidas jugadas, que se pueden usar para analizar el juego y extraer conocimiento sobre cómo se juega.

Mediante el uso de aprendizaje no supervisado, se puede extraer conocimiento del juego y ponerlo a disposición de las personas que empiezan a jugar y hacer más fácil esta entrada. Además de servir de ayuda en el ámbito profesional, para facilitar la preparación de un equipo ante una partida de competición.

Objetivos del proyecto

2.1. Objetivos principales

- Desarrollar un proceso ETL que sea capaz de recopilar los datos necesarios usando la API oficial de Riot Games¹.
- Aplicar técnicas de aprendizaje no supervisado sobre los datos recopilados, en este caso algoritmos para la obtención de conjuntos frecuentes de objetos.
- Ser capaz de extraer conocimiento útil a partir de los datos obtenidos.
- Desarrollar una aplicación en la que se pueda consultar el conocimiento extraído.
- Que el producto final sea capaz de ayudar a los nuevos jugadores.

2.2. Objetivos personales

- Aplicar lo aprendido durante el máster en un campo novedoso.
- Dar a conocer el mundo de los deportes electrónicos en un ambiente donde sean menos conocidos.
- Desarrollar un proyecto de ideación propia.

¹Riot Games es la desarrolladora de League of Legends

Conceptos teóricos

3.1. ETL

El término ETL (*Extract, Transform, Load*) se refiere al proceso de obtención de datos desde una o varias fuentes, su transformación y carga final en un lugar centralizado para su posterior uso.

3.2. Aprendizaje no supervisado

Dentro de la disciplina de aprendizaje automático, los algoritmos de aprendizaje no supervisado usan datos que no están clasificados o etiquetados previamente, con el objetivo de obtener las relaciones que existan entre los mismos. El tipo de algoritmos más conocidos de este grupo son los de agrupamiento o *clustering*, los cuáles se encargan de encontrar y agrupar las instancias de los datos de entrada en sus grupos correspondientes, atendiendo a criterios como la similitud o las distancias que las separan.

En este trabajo el tipo de algoritmos usados son los de conjuntos de elementos frecuentes. Estos son capaces de encontrar elementos que aparecen de forma conjunta frecuentemente en un gran listado de transacciones. Por ejemplo, estos algoritmos son muy usados para obtener que artículos se compran juntos para hacer sugerencias de compra en tiendas en línea.

El algoritmo Apriori es capaz de solucionar este problema. Su funcionamiento se basa en recorrer el listado de transacciones y contando las veces que aparece un elemento en el listado, si es mayor que un valor elegido previamente se considera frecuente, si no, se considera infrecuente. A continuación se hace la contabilidad de los pares de elementos, para identificar también los pares frecuentes, con la particularidad de que se eliminan los

conjuntos que contengan un elemento infrecuente. Se calculan los conjuntos frecuentes de tamaño 2 y se generan los conjuntos de tamaño 3, eliminando los que contenga conjuntos infrecuentes de tamaño 2. Este proceso iterativo se repite hasta que no se obtienen más conjuntos frecuentes.

3.3. Conceptos sobre *League of Legends*

3.3.1. El juego

League of Legends es un videojuego de estrategia multijugador del tipo *MOBA* (*Multiplayer Online Battle Arena*), desarrollado por Riot Games y lanzado en 2011, en el que dos equipos de cinco jugadores se enfrentan para destruir la base del equipo enemigo [9]. Cada jugador controla dentro del juego a un personaje llamado **campeón**, que pueden seleccionar antes de empezar a jugar, y es único entre los diez jugadores. Al igual que el campeón, los jugadores pueden seleccionar una posición antes de entrar a partida.

Para asegurar la unicidad se lleva a cabo un proceso de **selección de campeones** (figura 3.1) entre los diez jugadores, ya divididos en dos equipos. Se empieza con una fase de prohibición, en la que cada jugador bloquea un campeón para evitar que se pueda jugar en la partida actual. A continuación se lleva a cabo la fase de selección, en la cual los jugadores van seleccionando su campeón en orden y alternando el equipo. Empieza el primer jugador del primer equipo, seguido van el primer y segundo jugador del segundo equipo, luego segundo y tercero del primer equipo, tercero y cuarto del segundo equipo, cuarto y quinto del primer equipo, y por último, quinto del segundo equipo. La ordenación de los jugadores es aleatoria.

3.3.2. Mapa

Los jugadores se enfrentan en un mapa en forma de cuadrado (figura 3.2), con las bases de cada equipo localizadas en zonas opuestas del mapa, una en la esquina inferior izquierda, y la otra en la superior derecha. Conectando cada base se encuentran tres **líneas o calles**, **superior o top**, **central o mid** e **inferior o bot**. El espacio entre las calles se denomina **jungla**. Conectando las dos esquinas, que no pertenecen a las bases, se encuentra el **río** que se encarga de separar el terreno del mapa dominado por cada equipo. De forma general los jugadores se reparten de la siguiente manera, uno en **top**, uno en **mid**, dos en **bot** y el restante en la **jungla**.

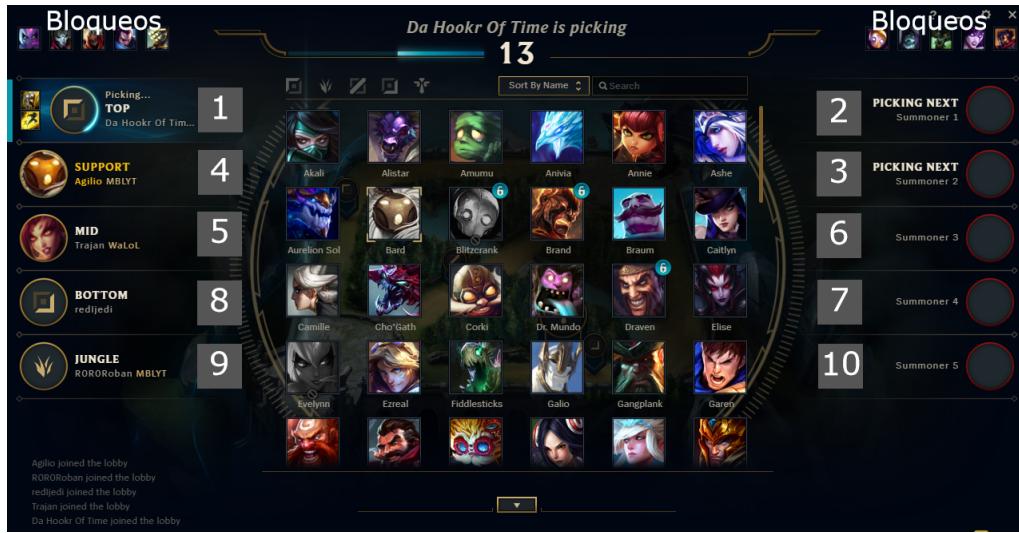


Figura 3.1: Proceso de selección de campeón

3.3.3. Conseguir la victoria

Para ganar la partida hay que destruir el **nexo** del equipo enemigo, es la **estructura** que está más alejada de la base propia. Las **torres** son otro tipo de estructuras situadas por el mapa, que disparan a los campeones del equipo contrario. Existen tres torres por cada línea y equipo, además de dos adicionales que protegen cada nexo. Las torres se tienen que derribar en el orden que se van encontrando en cada línea, y para destruir el nexo, como mínimo, hay que haber derribado todas las torres de una calle. Aunque un jugador podría moverse por la jungla para llegar a cualquier torre, estas no reciben daño si la anterior sigue en pie.

La forma en la que un equipo consigue ventaja sobre el rival es mediante el **oro**, este se consigue de varias formas. La principal forma es asesinando a los **campeones** enemigos. Otras formas de conseguir oro es destruyendo las **estructuras** enemigas. La última forma de conseguir oro es matando **monstruos y súbditos**, los primeros se encuentran en la jungla, los segundos recorren las calles en **oleadas**. Tanto los campeones como los monstruos y súbditos vuelven a aparecer pasado un tiempo concreto, las estructuras, una vez destruidas, no pueden recuperarse.

El oro permite comprar **objetos**, que mejoran las **habilidades** del campeón, haciendo que sea más sencillo derrotar a los campeones enemigos, lo que proporciona más oro para más objetos, causando un efecto bola de

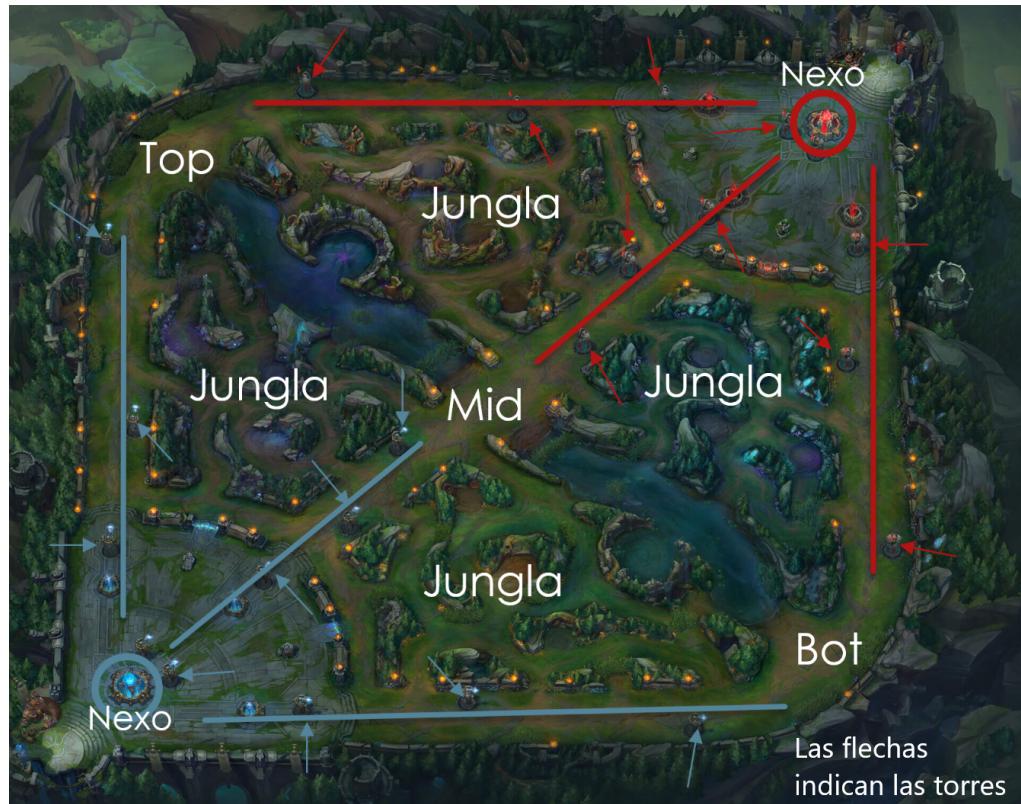


Figura 3.2: Mapa de *League of Legends*.

nieve y la eventual victoria del equipo. Al principio de cada partida, cada jugador empieza con 500 unidades de oro.

Para ver estos conceptos de una forma más visual, Riot Games preparó un vídeo informativo de cara a los mundiales de 2020 para que la gente que no tuviera mucho conocimiento del juego y su funcionamiento pudiera ver y disfrutar la competición. El vídeo se titula [¿Así que queréis ver el Mundial? | Mundial 2020 - League of Legends²](#) y se encuentra disponible en YouTube.

3.3.4. Campeones

Los campeones son los diferentes personajes disponibles que un jugador puede seleccionar antes de la partida. Actualmente hay 156 disponibles, añadiéndose a la lista uno nuevo en franjas de tiempo que van desde uno a seis meses. Cada uno tiene un rol asignado que determina su forma de

²https://youtu.be/ERkt_1TY1kU

jugar, su función e incluso posición dentro del mapa. Los distintos roles son **Tirador, Apoyo, Asesino, Luchador, Tanque y Mago**.

Para que un campeón acabe en una categoría u otra hay que prestar atención a varios factores, entre los que se encuentran su **tipo de ataque** básico, el efecto de sus habilidades y la forma en la que sus estadísticas modifican las **habilidades**. En las secciones **3.3.5** y **3.3.6** se explican en detalles estos conceptos.

3.3.5. Habilidades

Cada campeón tiene un conjunto único de habilidades, una **pasiva** y cuatro **activas**, se diferencian en que la pasiva está siempre causando un efecto y las activas causan su efecto cuando decide el jugador. Se pueden definir como las operaciones que un jugador puede realizar para interactuar con el mapa, con otros jugadores, monstruos y súbditos.

Cada habilidad puede tener un efecto o combinar varios, los cuales se aplican sobre uno mismo, un **aliado** o **enemigo**. Los efectos más comunes son las **curaciones**, realizar **daños** o control de adversario, donde se engloban **ralentizaciones** o **inmovilizaciones**. Estos efectos tienen unos valores numéricos que constan de dos partes, un valor base y uno variable que depende de las estadísticas del campeón.

Explicado con un ejemplo, una **habilidad** de un campeón hace daño a un enemigo con un valor base de 150 puntos de vida y un valor variable que corresponde al 30 % del daño de ataque del campeón. Si en un momento determinado el campeón tiene 300 de daño de ataque, el daño total de la habilidad se calcula como $150 + (0,3 \times 300) = 240$.

En el caso de los daños al rival, existen tres formas en las que se pueden realizar: **daño físico, mágico y verdadero**. Esto está determinado por la habilidad y rol del campeón.

3.3.6. Estadísticas

Las estadísticas son diferentes valores numéricos que determinan las capacidades de cada campeón en un área en concreto del juego. **Estos valores se ven modificados por la compra de objetos** (ver sección **3.3.7**). A continuación se describen las estadísticas y que representan.

Daño de ataque: Daño realizado con ataques básicos.

Probabilidad de crítico: Probabilidad de que un ataque básico haga el doble de daño.

Velocidad de ataque: Cantidad de ataques básicos que se pueden realizar por segundo.

Poder de habilidad: Modifica el daño que realizan las habilidades.

Velocidad de movimiento: Velocidad a la que un campeón se desplaza por el mapa.

Armadura: Cantidad en la que se ve reducido el daño físico que se recibe.

Resistencia mágica: Cantidad en la que se ve reducido el daño mágico que se recibe.

Penetración de armadura: Cantidad de la armadura del rival ignorada a la hora de realizar daño físico.

Penetración mágica: Cantidad de la resistencia mágica del rival ignorada a la hora de realizar daño mágico.

Vida: Daño que tiene que recibir un personaje para morir.

Maná: Coste de usar habilidades.

Robo de vida: Porcentaje de vida recuperado al dañar a un rival.

3.3.7. Objetos

Dentro de la base de cada equipo en el mapa está localizada la **tienda**. Aquí los jugadores pueden comprar objetos con el oro que han ido ganando con el progreso de la partida. Su función es modificar las estadísticas del campeón, para que las habilidades del mismo sean más eficaces contra los rivales. Los objetos que se compran se quedan guardados en el **inventario** del campeón, el cual está limitado a seis objetos.

En el juego actual existen 222 objetos disponibles, clasificados en cinco categorías:

Iniciales: Objetos más relevantes al inicio de la partida, generalmente mejoran una estadística pero no pueden combinarse para formar un objeto de categoría superior.

Básicos: Objetos que mejoran una estadística.

Épicos: Objetos formados por la combinación de varios objetos básicos que mejoran varias estadísticas.

Legendarios: Objetos formados por la combinación de objetos épicos y/o básicos que mejoran varias estadísticas y proporcionan algún efecto adicional.

Míticos: Igual que los anteriores, pero limitado a uno en el inventario.

3.3.8. Ligas

Al igual que otros deportes, *League of Legends* posee un sistema de ligas dentro de las cuales los jugadores que lo deseen pueden clasificarse, en base a la habilidad que demuestren en sus partidas.

En la tabla 3.1 se muestran las diferentes ligas disponibles ordenadas de menor a mayor competencia. También se incluye el porcentaje de jugadores localizados en cada liga [11] y otros atributos explicados a continuación. El porcentaje no es fijo, los mostrados se refieren al estado de las ligas en agosto de 2021. Esto se debe al movimiento de jugadores entre ligas en base a las partidas que juegan y a los nuevos jugadores que empiezan a jugar.

Desde **Hierro** a **Diamante**, cada una cuenta con cuatro divisiones (subcategorías dentro de cada liga), que van desde IV a I. Las tres restantes tienen una única división. Además, tanto **Gran Maestro** como **Aspirante** tienen plazas limitadas, 700 y 300 respectivamente.

La localización de cada jugador se basa en un sistema de puntos, ganándolos al salir victorioso y perdiéndolos al ser derrotado, en un rango aproximado de 10-25 puntos por partida. Estos se otorgan en base a una medida numérica oculta a los jugadores llamada *Match Making Rating* (MMR) [3].

En las ligas con divisiones, cuando el jugador alcanza 100 puntos, pasa a la siguiente división, o en el caso de estar en la división superior, subiría de liga. Por encima de Diamante no hay límite de puntos, y estando los jugadores ordenados por estos, la clasificación se realiza según el límite de plazas mencionado anteriormente.

Nombre	Porcentaje	Divisiones	Límite usado	Cantidad del límite
Hierro	1,9 %	IV, III, II, I	Puntos	100 por división
Bronce	19 %	IV, III, II, I	Puntos	100 por división
Plata	37 %	IV, III, II, I	Puntos	100 por división
Oro	28 %	IV, III, II, I	Puntos	100 por división
Platino	11 %	IV, III, II, I	Puntos	100 por división
Diamante	1,4 %	IV, III, II, I	Puntos	100 por división
Maestro	0,11 %	I	Plazas	Sin límite
Gran Maestro	0,027 %	I	Plazas	700 plazas
Aspirante	0,011 %	I	Plazas	300 plazas

Tabla 3.1: Sistema de ligas

Técnicas y herramientas

En esta sección se van a explicar, por un lado, las técnicas que se han seguido durante el desarrollo del proyecto y por otro, las herramientas utilizadas para llevarlo a cabo.

4.1. Scrum

Scrum es una metodología de desarrollo ágil basada en los principios del manifiesto ágil³. Esta metodología está pensada para equipos por lo que ha sido adaptada para el desarrollo de este proyecto.

Scrum consiste en ciclos de trabajo iterativos denominados *sprints*, con duración de una semana generalmente, en los que al terminar se entrega un producto funcional. Al final de cada *sprint* se lleva a cabo una reunión con los tutores para comentar el desarrollo del *sprint*, enseñar los avances y planear el siguiente ciclo de desarrollo.

4.2. Git

Git es un sistema distribuido de control de versiones. No se han considerado otras alternativas al ser un sistema ya conocido. Actualmente puede ser considerado el sistema con más usuarios y de más aceptación.

³<http://agilemanifesto.org/>

4.3. GitHub

GitHub es un servicio de alojamiento de repositorios de código basado en *git*.

Esta plataforma se ha utilizado para alojar el proyecto, gestionar las tareas mediante *issues* y planificar *sprints* mediante *milestones*. El repositorio del proyecto se encuentra en <https://github.com/IvanBeke/TFM>.

4.4. Jupyter

[Jupyter](#)⁴ es una plataforma, en formato de aplicación web, que permite desarrollar cuadernos que combinan texto y código. Desde ellos se puede ejecutar el código agrupado en celdas y ver la salida al mismo tiempo. Muy útil para realizar pruebas y mostrar resultados en tareas de análisis de datos.

4.5. Django

[Django](#)⁵ es un *framework* de Python para crear aplicaciones web enfocado al rápido desarrollo y a facilitar gran parte del trabajo de los pasos necesarios para obtener un producto o aplicación web.

4.6. MLXtend

[MLXtend](#)⁶ es una biblioteca de aprendizaje automático que incluye herramientas útiles para tareas de ciencia de datos. Se ha escogido para este proyecto por la facilidad que ofrece a la hora de preparar los datos y realizar el entrenamiento. Además incluye implementaciones de los algoritmos considerados, Apriori y FP-Growth para la resolución del problema.

4.7. Cassiopeia

[Cassiopeia](#)⁷ es una biblioteca que sirve de *wrapper* para acceder a la API de Riot Games. Permite acceder a los datos mediante creación de objetos. Usa *lazy loading* para evitar peticiones innecesarias y también incluye una

⁴<https://jupyter.org/>

⁵<https://www.djangoproject.com/>

⁶<http://rasbt.github.io/mlxtend/>

⁷<https://github.com/meraki-analytics/cassiopeia>

cache propia para evitar repetir peticiones recientes. Por último, incluye control de peticiones para no sobrepasar los límites de la clave.

La razón principal para escogerla sobre otros *wrappers* es su buena integración con *Django* (ver sección 4.5).

4.8. MongoDB

[MongoDB⁸](#) es una base de datos NoSQL de tipo documental. Permite almacenar datos cuya estructura sea similar a JSON. En el ámbito de *Big Data* es muy útil por su escalabilidad y velocidad. Adicionalmente proporciona herramientas para facilitar el proceso de transformación de los datos.

Se ha escogido esta plataforma porque da la posibilidad de guardar las respuestas en bruto de la API y, mediante agregaciones, poder realizar las transformaciones de datos necesarias de forma rápida y eficiente, además de poder cargarlos en la base de datos final.

⁸<https://www.mongodb.com/>

Aspectos relevantes del desarrollo del proyecto

5.1. Problema encontrado y propuesta de solución

Como se ha podido observar en [3.3](#), para cualquier persona que quiera empezar a jugar, existe una cantidad de información abrumadora que puede impedir disfrutar de la experiencia y crear frustraciones.

En primer lugar el gran número de campeones y sus habilidades hace que se tarde tiempo en aprender cómo manejar a los personajes. La siguiente dificultad es encontrar la mejor combinación de objetos para cada campeón de forma que su utilidad dentro de la partida se maximice.

En segundo lugar, al disponer de un espacio limitado en el inventario, la toma de decisión de qué objetos comprar se complica.

Para terminar, el juego está en constante evolución debido que cada dos semanas se realizan actualizaciones que modifican las estadísticas, los valores de las habilidades y la modificación de estadísticas de los objetos. Estos motivos pueden provocar que los objetos que han sido óptimos para un campeón dejen de ser viables con la actualización.

Por estas razones, y sabiendo qué jugadores son los que mayor habilidad demuestran en el juego, gracias al sistema de ligas, se puede analizar el comportamiento de estos jugadores para obtener el conocimiento que poseen sobre el juego y ponerlo a disposición de los recién llegados.

Para ello se propone un sistema que sea capaz de analizar las partidas de los jugadores de más alto nivel y extraer el conocimiento que poseen. Este

conocimiento será puesto a disposición de otros jugadores por medio de una aplicación web.

Para realizar el análisis de partidas se va a usar aprendizaje no supervisado, en concreto algoritmos para la obtención de conjuntos de elementos frecuentes. A partir de los datos de los jugadores de mayor habilidad, de forma indirecta, se debería obtener el conjunto de objetos más útil para cada campeón.

5.2. Desarrollo de *notebooks* y web

El desarrollo del proyecto ha tenido dos fases claramente diferenciadas, desarrollo de *notebooks* y desarrollo de la aplicación web. A pesar de que pueda parecer que se ha hecho el mismo trabajo dos veces, esta metodología ha aportado grandes ventajas a lo largo del trabajo.

En primer lugar, el uso de *notebooks* ha permitido realizar pruebas de forma rápida con la API y poder comprobar de forma ágil que los datos necesarios para el desarrollo estaban accesibles, al igual que, ya teniendo muestras de los datos, realizar varias ejecuciones de los algoritmos candidatos y evaluar sus resultados.

En segundo lugar, al poder ejecutar fragmentos de código aislados se han podido identificar y solucionar varios problemas en la *pipeline* (ver secciones 5.3 y 5.4) de forma rápida y sin que afecte al resto del desarrollo.

Por último, esta forma de gestionar el desarrollo ha permitido que el desarrollo de la *pipeline* integrada en la aplicación haya sido muy ágil. Además de que, como la mayoría de problemas se habían identificado en los *notebooks*, el proceso final dentro la aplicación ha estado libre, casi al completo, de errores.

5.3. Control de velocidad de peticiones por ratios de la API

Al depender de una API externa para realizar la recopilación de datos, las fases que dependen de la misma tienen un límite máximo de peticiones por tiempo⁹ que hay que respetar, ya que demasiadas violaciones de los límites pueden provocar que la desarrolladora restrinja el acceso a los datos.

⁹https://developer.riotgames.com/docs/portal#web-apis_api-keys

Para el tipo de clave de acceso que se tiene actualmente (clave personal) los límites son:

- 20 peticiones por segundo
- 100 peticiones cada dos minutos

Al inicio del desarrollo se decidió usar retroceso exponencial. Esto es ir aumentando un tiempo de retardo entre peticiones introducido de forma manual cuando una petición falla, e ir disminuyéndolo poco a poco cuando el servidor vuelve a responder correctamente. Sin embargo, el tiempo de espera que este método introducía disminuía notablemente el rendimiento de la extracción de datos, alargando el proceso más de lo necesario, además de añadir complejidad al proceso.

Más adelante se localizó en la documentación de la API que, cuando una petición se rechaza por haber sobrepasado el límite, la respuesta de esa petición fallida retornaba en una cabecera el tiempo de espera necesario para volver a obtener respuestas correctas.

Se procedió a cambiar el retroceso exponencial por una espera única del tiempo indicado en la respuesta, haciendo que el proceso de recopilación fuera más limpio y minimizando las esperas, de tal forma que se pudo aumentar el ratio de obtención de datos.

5.4. Tolerancia y recuperación de fallos

Como se ha explicado en la sección 5.3, la recopilación de datos ha sido temporalmente costoso. En el caso de que ocurriera un error mientras se ejecutase, el proceso empezaría de cero y se realizarían peticiones para datos que ya estuvieran guardados, perdiendo tiempo valioso.

Por ello, desde el inicio del proyecto se decidió que el sistema tendría que contar con medios para que, en caso de fallos imprevistos, el proceso debería ser capaz de continuar la ejecución como si no hubiera ocurrido ningún fallo.

Para lograr el objetivo propuesto se guarda de forma periódica el estado de ejecución, serializando múltiples variables de control. En caso de que el proceso se detenga, el estado previo de ejecución está guardado. Ahora, una vez reiniciada la ejecución, las variables que han sido guardadas previamente no se declaran como variables normales, si no que se comprueba si existe una serialización para ellas, en caso afirmativo se cargan y la ejecución continúa a partir del punto cargado. En caso de que no exista un estado guardado, se asigna un valor inicial a las variables y el proceso empezaría de cero.

5.5. Proceso de recopilación y entrenamiento

En esta sección se van a explicar las fases del proceso de recopilación de datos y cómo se ha llevado a cabo el proceso, además de su transformación y entrenamiento final.

5.5.1. Extracción jugadores

Con las peticiones que están disponibles en el API no existen formas de obtener jugadores de forma directa, hay que hacerlo a través de las ligas. Esta forma de obtener jugadores es ventajosa para este caso en concreto, ya que el objetivo final es la extracción del conocimiento de los jugadores más expertos.

Las ligas de las que se van a obtener los jugadores son Aspirante, Gran Maestro, Maestro y Diamante I. La respuesta que se obtiene contiene un listado de los jugadores presentes en esas ligas, sin embargo, para poder obtener partidas, se necesita usar el identificador de la cuenta, que no está presente en esta respuesta.

El siguiente paso de esta fase es obtener ese identificador. A partir del nombre de jugador es posible obtener el identificador de cuenta en otra petición. Por ello, antes de guardar los jugadores, se obtiene el identificador de cuenta para cada uno, se junta con los datos previos y, finalmente, se guarda todo en una colección de *MongoDB*.

Este proceso tardó entre dos y tres horas y recopiló 15.572 jugadores.

5.5.2. Extracción de partidas

Partiendo de los jugadores obtenidos en la fase anterior, se puede obtener sus partidas, el tipo de dato principal sobre el que se va a trabajar en las siguientes fases.

Para obtenerlas, el primer paso es traer desde la base de datos los jugadores guardados en la fase anterior. Para cada uno, se consultan en la fuente de datos las 20 partidas más recientes que ha jugado. Se ha escogido este número para obtener partidas de todos los jugadores teniendo en cuenta el límite de peticiones que se pueden realizar, y para asegurar que en la información final presentada a los usuarios hay datos recientes.

Cada partida se guarda en la base de datos comprobando que no está guardada ya. Al haber diez jugadores por partida, es muy probable que varios de los jugadores hayan coincidido y aparezca la misma partida en varios historiales. Es muy importante para asegurar la calidad final que no aparezcan partidas repetidas, ya que el porcentaje de aparición de cada objeto estaría adulterado.

Debido al número elevado de jugadores el proceso se paró manualmente antes de completarse, se mantuvo en ejecución durante cuatro días y se almacenaron 142.438 partidas.

5.5.3. Añadir posición

Esta sección podría aparecer como una subsección en el apartado anterior, pero, por como se introdujo en el desarrollo, se estima que es mejor que aparezca independientemente.

Inicialmente, solo se generaban transacciones en base al campeón, sin tener en cuenta la posición en la que se jugaba. Aunque para un gran número de campeones no es relevante la posición en la que se juega, hay varios en los que los objetos sufren variaciones notables. Por ello se optó por agrupar usando tanto campeón como posición.

En los datos en bruto de cada partida no existe un valor que indique la posición exacta, pero sí hay varios atributos que pueden ayudar a identificarla. Para ello se usan unos atributos que representan el rol del campeón y la línea en la que han jugado.

Para las líneas en las que hay un único jugador, esta se convierte en la posición. Para la línea de *bot* (en esta van dos jugadores) se mira el rol para obtener la posición correcta.

5.5.4. Generar transacciones

En esta fase se lleva a cabo la transformación de los datos en bruto, en datos que poder suministrar al algoritmo para la obtención de conocimiento. Esta etapa del procesamiento se realiza íntegramente usando agregaciones de MongoDB.

En los datos en bruto de partidas, los atributos que interesan para el problema son los objetos con los que se ha terminado la partida. El primer paso para generar las transacciones (conjunto de objetos comprados) es juntar los objetos en un conjunto.

A continuación, usando el campeón y la posición como claves, se agrupan los conjuntos de objetos del paso en anterior en una lista de conjuntos, de esta forma ya están localizados en un mismo documento el campeón, la posición y las transacciones. Por último, el resultado se guarda en otra colección en la base de datos. La salida de esta fase son los datos de entrada para el algoritmo de aprendizaje no supervisado.

Al realizarse completamente con operaciones de la base de datos esta fase es rápida en comparación a las anteriores. Es capaz de realizar la transformación de todas las partidas en un tiempo ~~localizado~~ entre 15 y 20 segundos.

5.5.5. Generar conjuntos frecuentes

Para terminar con el proceso, se tiene la generación de conjuntos frecuentes. Para ello se obtienen las transacciones generadas en el paso anterior y se ejecuta el algoritmo para cada grupo (campeón y posición). Una vez obtenido el resultado, cada conjunto de objetos se ordena en función de su coste de oro (descendiente) dentro del juego, y se carga en la base de datos que va a consultar la aplicación.

Se ha optado por ordenar usando el coste del objeto porque de forma general, cuanto más caro sea un objeto, más aporta a la estadísticas del campeón y más ventaja proporciona dentro de la partida.

La generación de los conjuntos frecuentes tarda 10 segundos aproximadamente.

5.6. Elección del algoritmo

Como se ha comentado anteriormente, para la resolución del problema se va a usar un algoritmo para obtención de conjuntos de elementos frecuentes.

El listado de algoritmos [5] que son capaces de resolver este problema es extenso, sin embargo, por tener conocimiento previo tanto del Apriori, como del FP-Growth, al haber sido tratados durante el máster, se ha optado por acotar la decisión a estos. Adicionalmente, estos son los que generalmente suelen recomendarse y los que se encuentran implementados en varias bibliotecas de aprendizaje automático.

En base a artículos existentes [6, 7] se decidió usar FP-Growth en vez del Apriori por ser más rápido y requerir menos memoria. Por lo que, durante

el desarrollo del proyecto, se usó ese algoritmo en las pruebas iniciales y después de realizar la recopilación de datos.

Sin embargo, con los datos finales ocurría que el Apriori finalizaba su ejecución más rápido que usando FP-Growth (ver tabla 5.2), por lo que finalmente se ha decidido dejar este algoritmo como el definitivo para la generación de conjuntos frecuentes.

Apriori	FP-Growth
9,88s	13,14s
10,37s	13,55s
9,98s	13,38s
10,74s	13,43s
10,19s	13,47s

Tabla 5.2: Tiempo de ejecución de los algoritmos

Trabajos relacionados

6.1. LoL Data Solution [12]

LoL Data Solution es un proyecto desarrollado dentro del equipo MAD Lions. Su objetivo principal es la recopilación y agregación de partidas de jugadores del propio equipo y sus adversarios, para así sacar estadísticas sobre el rendimiento de los jugadores y usarlo para tomar mejores decisiones en el entrenamiento del equipo y en la preparación contra otros equipos de la competición.

Conclusiones y ~~Líneas~~ de trabajo futuras

7.1. Conclusiones

Habiendo terminado el desarrollo del proyecto, me siento satisfecho con el producto conseguido y considero que se han cumplido los objetivos propuestos al inicio. Como resultado se ha obtenido un proceso capaz de recopilar y analizar partidas, capaz de obtener información útil que mostrar de una forma intuitiva en una aplicación web.

He conseguido aplicar varias técnicas aprendidas durante el desarrollo de máster y me he dado cuenta de muchas consideraciones a tener en mente siempre que me encuentre en el desarrollo de proyectos que estén relacionados con obtención masiva de datos.

Para terminar, me siento contento con el trabajo que he realizado estos meses, ha sido una experiencia positiva y enriquecedora de la que estoy seguro voy a ser capaz de aplicar lo aprendido en mi futuro.

7.2. Líneas de trabajo futuras

En el estado actual del proyecto, la aplicación puede proporcionar una gran ayuda a nuevos jugadores, a la vez que puede permitir a los profesionales analizar el estado actual del juego. Pero hay varios puntos que pueden mejorarse para ofrecer una herramienta más completa, algunos de los cuales serían:

- Tener en cuenta el orden de compra de cada objeto además del estado final.
- Se podrían usar algoritmos de *clustering* para obtener grupos de objetos similares y proporcionar alternativas a los usuario sobre los conjuntos existentes.
- Hay objetos que a pesar de ser útiles, podrían no aparecer por ser muy dependientes de la situación, por ejemplo, un campeón enemigo concreto, se podría incorporar una sección con todos los objetos y mostrar situaciones concretas donde conviene comprarlos.
- A la hora de generar las transacciones no se tiene en cuenta la versión del juego en la que han sido jugadas las partidas, lo que puede derivar en que las transacciones mezclen partidas que se hayan jugado en distintas actualizaciones. En futuras versiones de la aplicación habría que controlarlo para asegurar la información más correcta.
- Actualmente solo se recopilan datos del servidor de Europa, una buena forma de ampliar la cantidad de datos sería expandir la recopilación al resto de servidores (América y Asia).

Apéndices

Apéndice A

~~Plan de Proyecto Software~~

A.1. Introducción

En este apartado se va a exponer como se ha llevado a cabo la planificación del proyecto, además de los avances logrados entre cada iteración y reunión.

A.2. Planificación temporal

Durante el desarrollo del proyecto se ha seguido una versión simplificada de *Scrum*. Se han usado *sprints* para revisar los avances, con duración de una semana. Al final de cada *sprint* se realiza una reunión por videoconferencia para ver los avances logrados durante el mismo y planificar las tareas del siguiente.

El inicio oficial del proyecto fue el día 19 de mayo, con la reunión del Sprint 0 [A.2.1](#), habiendo realizado las tareas de ese sprint previamente.

A.2.1. Sprint 0

Sprint dedicado a tareas logísticas y preparación del inicio del proyecto.

- Crear el repositorio de código.
- Solicitar una clave permanente para la API a la desarrolladora del juego.
- Comprobaren la API la existencia de los datos necesarios.

- Formalizar el inicio del desarrollo del proyecto mediante una reunión con los tutores.

A.2.2. Sprint 1

Primer sprint de desarrollo. Dedicado a la extracción de jugadores. Desde 20/05/2021 hasta el 26/05/2021.

- Documentar el Sprint 0.
- Echar un ojo a *wrappers* existentes de la API para comprobar su viabilidad de uso.
- Desarrollar un *notebook* (cuaderno Jupyter) para la extracción de jugadores, teniendo en cuenta el límite de peticiones y la tolerancia a fallos.

A.2.3. Sprint 2

Desde 27/05/2021 hasta el 02/06/2021.

- Documentar el Sprint 1.
- Modificar la extracción de jugadores para añadir un identificador de cuenta, necesario para el siguiente paso de recuperación de partidas.
- Desarrollar un *notebook* para extraer las partidas de los jugadores recuperados previamente.
- Crear una biblioteca con funciones comunes para realizar peticiones y guardar estados de ejecución.

A.2.4. Sprint 3

Desde 03/06/2021 hasta el 09/06/2021.

- Documentar el Sprint 2.
- Partiendo de los datos de partidas del sprint anterior, localizar y extraer los objetos que se han comprado para cada campeón dentro de la partida.
- Crear el formato final con las entradas para el algoritmo Apriori.

A.2.5. Sprint 4

Desde 10/06/2021 hasta el 16/06/2021. Por conflicto con otras asignaturas durante esta semana, se realizaron menos tareas.

- Documentar Sprint 3.
- Escribir la introducción de la memoria.
- Realizar una presentación sobre el funcionamiento y conceptos del juego a los tutores.

A.2.6. Sprint 5

Desde 17/06/2021 hasta el 30/06/2021. La duración de este sprint se alargó a dos semanas por un problema de salud que impidió realizar avances significativos durante la primera.

- Realizar correcciones en la memoria.
- Escribir objetivos del proyecto.
- Investigar alternativas del algoritmo Apriori.
- Buscar y escoger una biblioteca con las implementaciones del algoritmo a utilizar y sus alternativas.
- Empezar con el aprendizaje de Django.
- Iniciar el desarrollo de la aplicación web.

A.2.7. Sprint 6

Desde 01/06/2021 hasta el 07/07/2021. Por conflicto con otras asignaturas durante esta semana, se realizaron menos tareas.

- Desarrollar un *notebook* en el que probar los algoritmos de conjuntos frecuentes.
- Seleccionar el algoritmo a usar para obtener los resultados que mostrar en la aplicación.

A.2.8. Sprint 7

Desde 08/07/2021 hasta 14/07/2021.

- Instalar una base de datos MongoDB.
- Guardar ejecución del algoritmo en MongoDB.
- Crear una vista con el listado de campeones en la aplicación web.
- Crear una vista con en la que mostrar los conjuntos de objetos frecuentes de un campeón seleccionado.

A.2.9. Sprint 8

Desde 15/07/2021 hasta 21/07/2021.

- Documentar los sprints 4, 5, 6 y 7.
- Incorporar el *notebook* de extracción de jugadores a la aplicación web.
- Incorporar el *notebook* de extracción de partidas a la aplicación web.
- Añadir buscador de campeones.
- Añadir la ficha del campeón junto a sus objetos frecuentes.
- Mostrar un mensaje cuando no existen datos para un campeón.

A.2.10. Sprint 9

Desde 22/07/2021 hasta 28/07/2021.

- Documentar los sprints 8 y 9.
- Extraer partidas de forma masiva.
- Incorporar el *notebook* de extracción de detalles de partidas a la aplicación web.
- Incorporar el *notebook* de extracción de ejecución de algoritmos a la aplicación web.
- Escribir los conceptos teóricos sobre el juego.

A.2.11. Sprint 10

Desde 29/07/2021 hasta 04/08/2021.

- Modificar la generación de transacciones para agrupar también por posición.
- Añadir pestañas a la vista de conjuntos frecuentes, una por posición.
- Cambiar el porcentaje de elección en texto por una barra de progreso.

A.2.12. Sprint 11

Desde 05/08/2021 hasta 18/08/2021.

- Realizar correcciones en la memoria.
- Intentar añadir panel de administrador para lanzar las tareas de la *pipeline* desde la interfaz.

A.2.13. Sprint 12

Desde 19/08/2021 hasta 25/08/2021.

- Realizar correcciones en la versión de la memoria del *sprint* anterior.
- Escribir capítulos de la memoria todavía sin completar.

A.2.14. Sprint 13

Desde 26/08/2021 hasta 03/09/2021.

- Realizar correcciones en la versión de la memoria del *sprint* anterior.
- Terminar la escritura de la memoria.

A.3. Estudio de viabilidad

A.3.1. Viabilidad económica

Costes de personal

El proyecto se ha llevado a cabo por un desarrollador contratado a tiempo parcial durante 3 meses. Se considera un salario neto de 1.000€ mensuales (ver tabla A.1).



La cotización a la seguridad social se ha calculado como horas comunes, según el régimen general de 2018 (28,30 %) [4].

Concepto	Coste
Salario neto	1000€
Retención IRPF (19 %)	360,53€
Seguridad social (28,30 %)	537,00€
Salario bruto (mensual)	1897,53€
Total 3 meses	5692,69€

Tabla A.1: Costes de personal

A.3.2. Viabilidad legal

A lo hora de añadir una licencia al proyecto hay que tener en cuenta a que licencias están sujetas las dependencias. En la tabla A.2 se listan las dependencias del proyecto y su licencia.

Dependencia	Licencia
django	BSD
cassiopeia	MIT
ipython	BSD 3-Clause
django-cassiopeia	MIT
django-extensions	MIT
djongo	AGPL-3.0
role-identification	MIT

Tabla A.2: Tabla de dependencias

De las dependencias usadas, todas son licencias bastante permisivas que permiten el uso con libertad, por lo que no tenemos que preocuparnos de incompatibilidades con la licencia que se escoja.

Con ayuda de las recomendaciones GNU [2], se ha escogido la licencia GPL-3.0 [1]. Esta licencia permite la modificación, uso y distribución del software, siempre que esto se haga bajo la misma licencia, se indiquen los cambios y se mencione al autor original.

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este apéndice se describen los objetivos generales de la aplicación y se detallan sus requisitos, tanto funcionales como no funcionales.

B.2. Objetivos generales

- Desarrollar un proceso ETL que sea capaz de recopilar los datos necesarios usando la API oficial de Riot Games¹.
- Aplicar técnicas de aprendizaje no supervisado sobre los datos recopilados, en este caso algoritmos para la obtención de conjuntos frecuentes de objetos.
- Ser capaz de extraer conocimiento útil a partir de los datos obtenidos.
- Desarrollar una aplicación en la que se pueda consultar el conocimiento extraído.
- Que el producto final sea capaz de ayudar a los nuevos jugadores.

¹Riot Games es la desarrolladora de League of Legends

B.3. Catálogo de requisitos

B.3.1. Requisitos funcionales

- **RF-1 Proceso ETL:** la aplicación debe ser capaz de recopilar, transformar y almacenar los datos para sus posterior uso.
 - **RF-1.1 Extracción de jugadores:** la aplicación debe poder extraer los jugadores de las ligas más altas.
 - **RF-1.2 Extracción de partidas:** la aplicación debe extraer las partidas de los jugadores obtenidos previamente.
 - **RF-1.3 Generación de transacciones:** la aplicación debe transformar los datos de partidas en listados de objetos agrupados por campeón.
 - **RF-1.4 Generación de conjuntos frecuentes:** la aplicación debe generar conjuntos frecuentes de objetos usando las transacciones obtenidas anteriormente.
- **RF-2 Consulta de información:** la aplicación debe ser capaz de recopilar, transformar y almacenar los datos para su posterior uso.
 - **RF-2.1 Búsqueda por campeón:** el usuario debe poder buscar conjuntos frecuentes filtrando por un campeón concreto.
 - **RF-2.2 Búsqueda en partida activa:** el usuario debe poder buscar conjuntos de objetos usando su nombre dentro del juego, de tal forma que se muestre la información adecuada para el campeón usado en ese momento.

B.3.2. Requisitos no funcionales

- **RNF-1 Usabilidad:** la aplicación debe ser intuitiva y fácil de usar.
- **RNF-2 Mantenibilidad:** debe ser sencillo añadir funcionalidad nueva a la aplicación.
- **RNF-3 Compatibilidad:** la aplicación debe poder funcionar en los principales navegadores.
- **RNF-4 Responsividad:** la aplicación debe adaptarse al tamaño de la pantalla.

Apéndice C

Especificación de diseño

C.1. Introducción

En este anexo se explican los diseños que se han llevado a cabo para realizar los objetivos anteriores al igual que se describen los usados utilizados para conseguirlo.

C.2. Diccionario de datos

C.2.1. Summoners

Summoners representa a la entidad del jugador.

Variable	Descripción	Tipo
accountId	Identificador de la cuenta a nivel de región	String
name	Nombre del jugador dentro del juego	String
region	Identificador de la región	String
puuid	Identificador global de la cuenta	String
summonerId	Identificador del jugador a nivel de región	String

Tabla C.1: Entidad del jugador

C.2.2. Matches

Matches es la entidad referida a las partidas jugadas. Se muestran solo los atributos relevantes para este problema ya el bruto contiene demasiada información que no aporta valor a este problema.

Variable	Descripción
creation	Fecha y hora en la que comienza la partida
region	Identificador de la región
version	Identificador de la versión del juego en la que se ha jugado
gameId	Identificador de la partida
teams	Lista que contiene a los dos equipos, siempre tiene tamaño 2
teams.participants	Lista que contiene a los jugadores del equipo, siempre tiene tamaño 10
teams.participants.stats	Varias estadísticas del jugador al terminar la partida
teams.participants.stats.item0	Identificador del primer objeto
teams.participants.stats.item1	Identificador del segundo objeto
teams.participants.stats.item2	Identificador del tercer objeto
teams.participants.stats.item3	Identificador del cuarto objeto
teams.participants.stats.item4	Identificador del quinto objeto
teams.participants.stats.item5	Identificador del sexto objeto
teams.participants.championId	Identificador del campeón seleccionado por el jugador
teams.participants.position	Posición en la que se ha jugado el campeón seleccionado

Tabla C.2: Entidad de la partida

C.2.3. Transactions

Transactions es la entidad que representa los datos de entrada que se van a suministrar al algoritmo Apriori. Es el resultado de la fase de transformación de la *pipeline*.

C.2.4. Frequent builds

Frequent builds son los datos que se muestran en la aplicación web al usuario final. Contienen la salida del algoritmo Apriori y atributos para identificar al campeón y posición.

Variable	Descripción	Tipo
champion	Identificador del jugador	Integer
position	Posición en la que se ha jugado al campeón	String
transactions	Listado de conjuntos de objetos que ha tenido el campeón en la posición	Array<
transactions.*	Conjunto de objetos con los que el campeón ha terminado la partida	Array<

Tabla C.3: Transacciones

Variable	Descripción
champion	Identificador del jugador
games	Cantidad de partidas jugadas con el campeón llevado en una posición
position	Posición en la que se ha jugado al campeón
builds	Listado de conjuntos frecuentes de objetos
builds.support	Porcentaje de veces que aparece el conjunto de objetos en el listado de transacciones
builds.itemset	Conjunto de objetos frecuentes encontrado por el algoritmo

Tabla C.4: Conjuntos frecuentes de objetos

C.3. Diseño procedimental

En esta sección se incluirán varios diagramas para comprender mejor cada una de las fases por las que se ha pasado hasta conseguir los conjuntos frecuentes que se van a mostrar al usuario.

C.3.1. Extracción de jugadores

La primera fase es la extracción de jugadores, se obtienen desde la API, se añade el identificador de cuenta y se guardan.

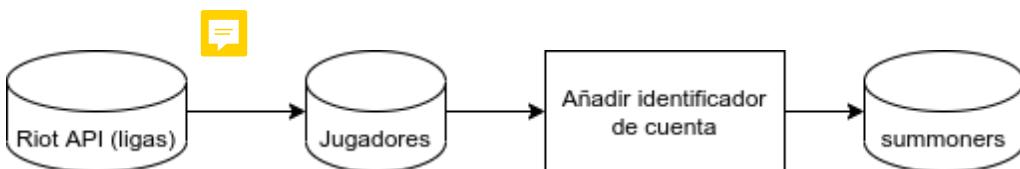


Figura C.1: Extracción de jugadores

C.3.2. Extracción de partidas

A continuación se obtienen las partidas de cada jugador, se añade la posición y se guardan.

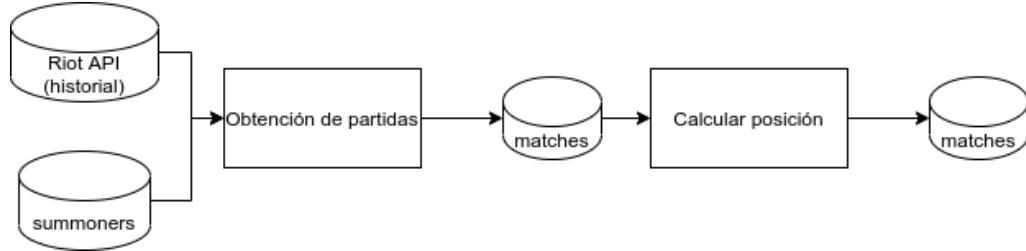


Figura C.2: Extracción de partidas

C.3.3. Generación de transacciones y conjuntos frecuentes

A partir de las partidas, se ejecuta la agregación en MongoDB y se guardan. A continuación se usan las transacciones como entrada para el Apriori y se guarda la salida en la colección final.

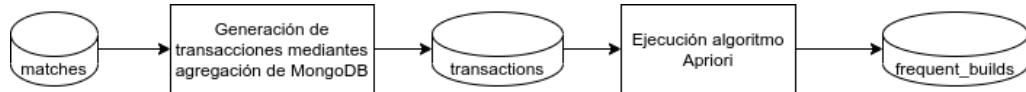


Figura C.3: Generación de transacciones y conjuntos frecuentes

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este apéndice se presenta todo lo que tiene que conocer un desarrollador para poder continuar con el desarrollo de la aplicación. Se describe la estructura de directorios del proyecto, cómo instalar la aplicación, etc.

El proyecto tiene dos partes diferenciadas en cuanto a desarrollo. La primera es una colección de *notebooks* en la que se realizar pruebas con la API y algoritmos. La segunda es una aplicación web Django estándar.

D.2. Estructura de directorios

```
/ Directorio raíz
  └── memoria/ - Documentación del proyecto
      ├── img/ - Imágenes de la memoria
      ├── tex/ - Secciones de la memoria
      ├── memoria.tex - Código fuente de la memoria
      ├── memoria.pdf - Memoria del proyecto
      └── bibliografia.bib - Fuentes bibliográficas
  └── scripts and notebooks/ - Colección de notebooks y scripts
      ├── outputs/ - Directorio donde se almacenan las salidas de
      │   los notebooks
      └── 0_extract_players.ipynb - Notebook para extraer jugadores
          └── 1_extract_matches.ipynb - Notebook para extraer partidas
```

```
└── 2_download_matches_details.ipynb - Notebook para obtener
    detalles de cada partida
└── 3_algorithms.ipynb - Notebook para probar algoritmos
└── utils.py - Colección de funciones comunes para los notebooks
└── api_key.txt - Fichero que contiene la clave de la API
└── requirements.txt - Fichero que lista las dependencias
web/ - Directorio del proyecto en Django
└── betterbuilds/ - App de Django que contiene la web
    ├── migrations/
    ├── static/
    ├── templates/
    ├── __init__.py__
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── tests.py
    ├── urls.py
    └── views.py
etl/ - App de Django que contiene el proceso ETL
└── management/
    └── commands/
└── migrations/
└── __init__.py__
└── admin.py
└── apps.py
└── models.py
└── tests.py
└── views.py
web/ - Directorio de configuración del proyecto Django
└── __init__.py__
└── asgi.py
└── settings.py
└── urls.py
└── utils.py
└── wsgi.py
└── .env - Fichero con variables de entorno
└── manage.py - Script para interactuar con el proyecto
└── requirements.txt - Fichero que lista las dependencias
```

D.3. Manual del programador

D.3.1. Notebooks

Como los *notebooks* han sido usados para realizar pruebas y probar la *pipeline*, se recomienda probar en los mismos cualquier modificación a realizar en la recopilación y transformación de los datos.

También es recomendable usarlos para explorar los parámetros en las diversas peticiones a la API y sus respuestas.

D.3.2. Aplicación web

Al ser una ~~una~~ aplicación estándar de *Django*, se necesita tener conocimiento sobre el *framework*. Para cualquier duda consultar en su [documentación¹](#).

En este apartado se explican de forma básica los conceptos del *framework* y cómo se han aplicado en este desarrollo.

Los proyectos escritos en *Django* se organizan en los que llaman *apps*. Cada *app* representa cada módulo o servicio del proyecto, se componen de vistas y modelos propios e independientes del resto de *apps*, aunque pueden integrarse para el correcto funcionamiento del proyecto. El producto desarrollado se compone de dos *apps*, **betterbuils** y **etl**.

La *app* betterbuils contiene las vistas y modelos que se encargan de pedir y mostrar los datos necesarios en el navegador. Esto incluye la pantalla de inicio, el listado de campeones y la ficha del campeón. Sería la aplicación web comúnmente hablando.

Dentro de la *app* etl se encuentran las distintas fases de la *pipeline* codificadas como comandos de gestión del proyecto. Los comandos de gestión se ejecutan desde el terminal y funcionan como si se tratase de la ejecución de un *script*. Para cada fase de la *pipeline* existe un comando que la realiza de principio a fin. La relación entre los distintos comandos y la fase es la siguiente:

Extracción de jugadores: `import_summoners.py`

Extracción de partidas: `import_matches.py`

Añadir posición: `add_position.py`

¹<https://docs.djangoproject.com/en/3.2/>

Generar transacciones: generate_transactions.py

Generar conjuntos frecuentes: import_frequent_builds.py

D.4. Instalación y ejecución del proyecto

Dado que el proyecto se ha desarrollado usando el sistema operativo Ubuntu, las instrucciones de instalación están orientadas a ese sistema.

D.4.1. Requisitos previos

Antes de comenzar con la instalación del proyecto, el sistema tiene que tener instaladas las siguientes utilidades:

- Anaconda²
- MongoDB³
- Git⁴

También es necesario tener una clave de acceso a la API de Riot Games para poder hacer uso de la misma. Para ello hay que crearse una cuenta de desarrollo en <https://developer.riotgames.com/> y seguir las instrucciones de <https://developer.riotgames.com/docs/portal>.



D.4.2. Instalación

La forma más cómoda de obtener el código del proyecto es mediante *git*, para ello usar el siguiente comando:

```
$ git clone <url_del_repositorio>
```

Siendo <https://github.com/IvanBeke/TFM> la URL del repositorio.

Ya con el proyecto descargado, el siguiente paso es instalar las dependencias. Para ello hay que ejecutar el comando:

```
$ pip install -r scripts and notebooks/requirementes.txt
```

²<https://docs.anaconda.com/anaconda/install/index.html>

³<https://docs.mongodb.com/manual/administration/install-community/>

⁴<https://git-scm.com/downloads>

Para la aplicación se recomienda usar un entorno virtual para tener mejor control sobre las dependencias exactas. Para crearlo y activarlo:

```
$ conda create --name django
$ conda activate django
```

Una vez está el entorno en uso se instalan las dependencias con el comando:

```
$ pip install -r web/requirementes.txt
```

Variables de entorno

Para el uso de *notebooks* se necesita crear un fichero `api_key.txt`, en el directorio `scripts and notebooks` que contiene la clave de acceso a la API.

Para la aplicación web, hay que crear un fichero `.env`, en el directorio `web` que contenga las variables de entorno. También se pueden definir en el propio sistema. El listado de variables a las que hay que dar un valor es el siguiente:

- RIOT_API_KEY: clave de acceso a la API.
- RIOT_API_REGION: región sobre la que se van a ejecutar las peticiones.
- SECRET_KEY: clave que usa *Django* para seguridad interna.
- DEBUG: define si está activado el modo depuración de la aplicación.

D.4.3. Ejecución

Partiendo del directorio raíz se muestra como ejecutar los *notebooks* o la aplicación web.

Notebooks

```
$ cd scripts and notebooks
$ jupyter notebook
```

Aplicación web

```
$ cd web
$ python manage.py runserver
```

Con eso se lanzaría la aplicación web. Para la ejecución de las distintas fases del proceso de recopilación y entrenamiento se ejecutarían los siguientes comandos:

```
$ cd web
$ python manage.py import_summoners
$ python manage.py import_matches
$ python manage.py add_position
$ python manage.py generate_transaction
$ python manage.py generate_frequent_builds
```

Apéndice E

Documentación de usuario

E.1. Introducción

En este apéndice se explican los requisitos que debe cumplir el usuario para ejecutar la aplicación, cómo lanzarla y cómo usarla.

E.2. Requisitos de usuarios

Al tratarse de una aplicación web, los requisitos que debe cumplir el usuario son los siguientes:

- Navegador web instalado, la aplicación se ha probado en *Firefox* y *Chrome*.
- JavaScript activo en el navegador.

E.3. Instalación

Debido a que se proporciona una aplicación web, no es necesario instalarla para poder usarla. Sin embargo, si se quiere proceder a la instalación, se pueden seguir las instrucciones en la sección [D.4.2](#).

E.4. Manual del usuario

En esta sección se enseña al usuario como manejar la aplicación.

E.4.1. Inicio

Nada más entrar a la aplicación se muestra la pantalla de inicio (figura E.1). En ella se puede ver un texto de presentación de la aplicación web y se da acceso al resto de funcionalidades.

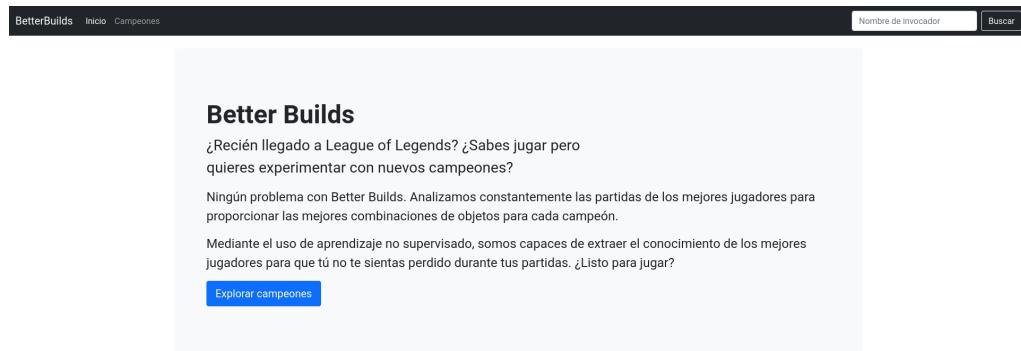


Figura E.1: Pantalla de inicio

Desde aquí se puede navegar al listado de campeones (sección E.4.2) y se puede buscar una partida en curso (sección E.4.4). Al estar localizados los enlaces en la cabecera, también se puede acceder desde el resto de la aplicación.

E.4.2. Listado de campeones

En esta pantalla se presenta un listado con todos los campeones actuales del juego (figura E.2). También se encuentra un buscador con el que se puede filtrar el listado (figura E.3), para encontrar de forma más rápida a un campeón concreto.

Una vez localizado el campeón se puede pinchar en él para navegar hacia su ficha (sección E.4.3) y consultar los conjuntos de objetos.

E.4.3. Ficha del campeón

La ficha del campeón (figura E.4 y E.5) se divide en una sección de presentación y en otra con los conjuntos de objetos. En la parte de presentación

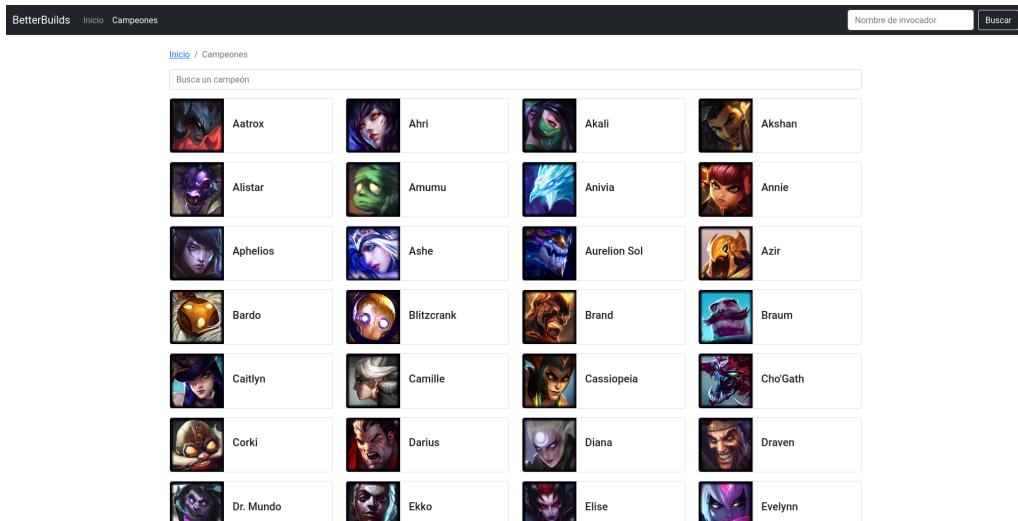


Figura E.2: Listado de campeones

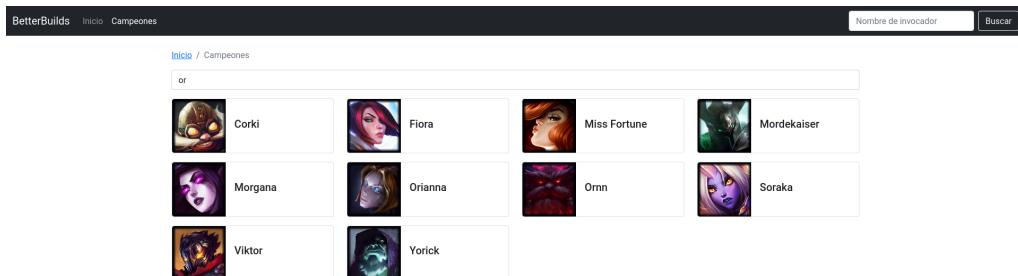


Figura E.3: Buscador

se muestra la foto del campeón, el número de partidas analizadas, un texto sobre la historia del mismo y consejos para su manejo en partida.

La parte de los objetos está dividida en pestañas, una para cada posición y una extra para las partidas en las que no ha sido posible identificarla. Se encuentran ordenadas por el porcentaje de partidas jugadas en esa posición.

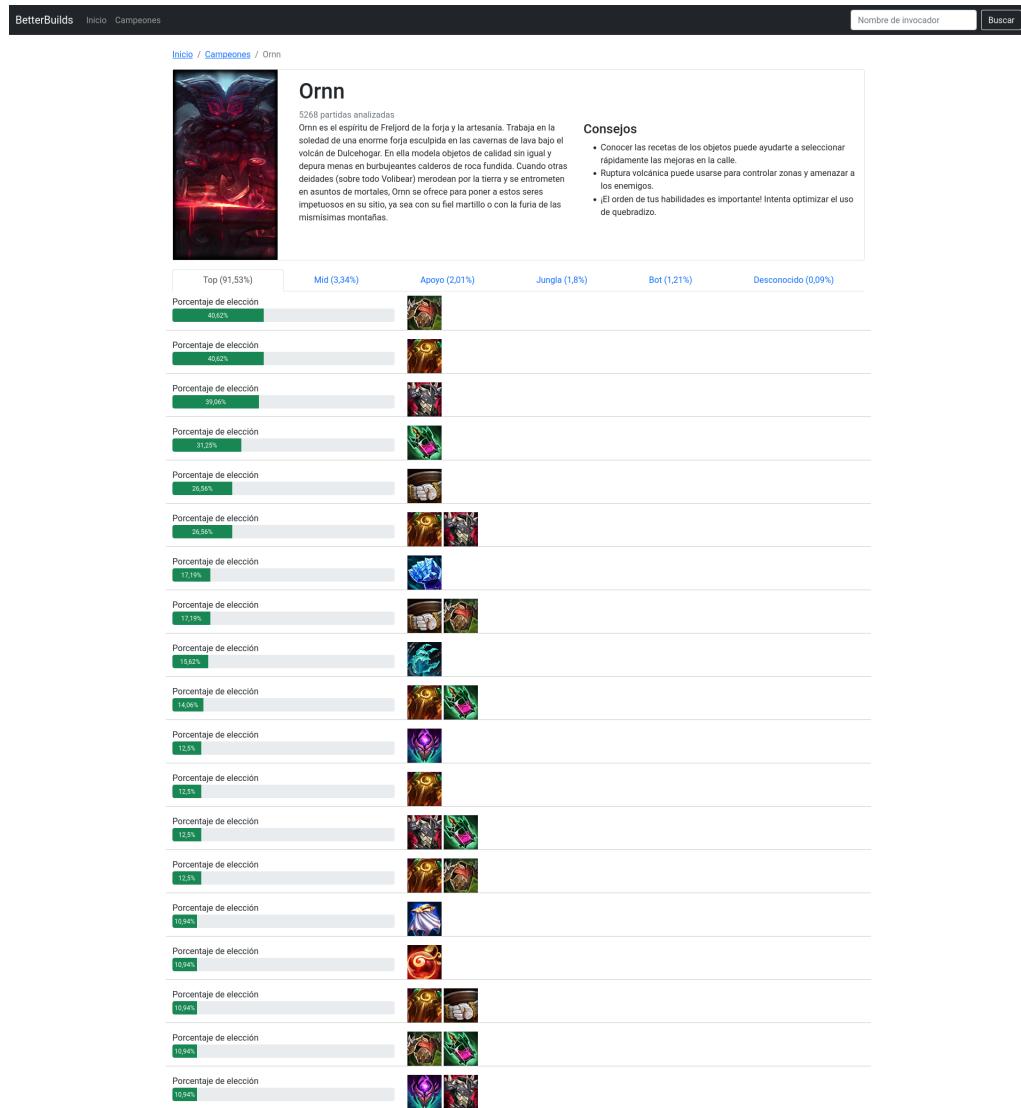


Figura E.4: Ficha del campeón

Dentro de cada pestaña la estructura es similar. Se presenta un listado de con los conjuntos de objetos que se han obtenido del análisis de partidas, junto con el porcentaje de elección de esa combinación, representado por una barra de progreso para una mejor usabilidad. Los conjuntos se encuentran ordenados por este porcentaje.

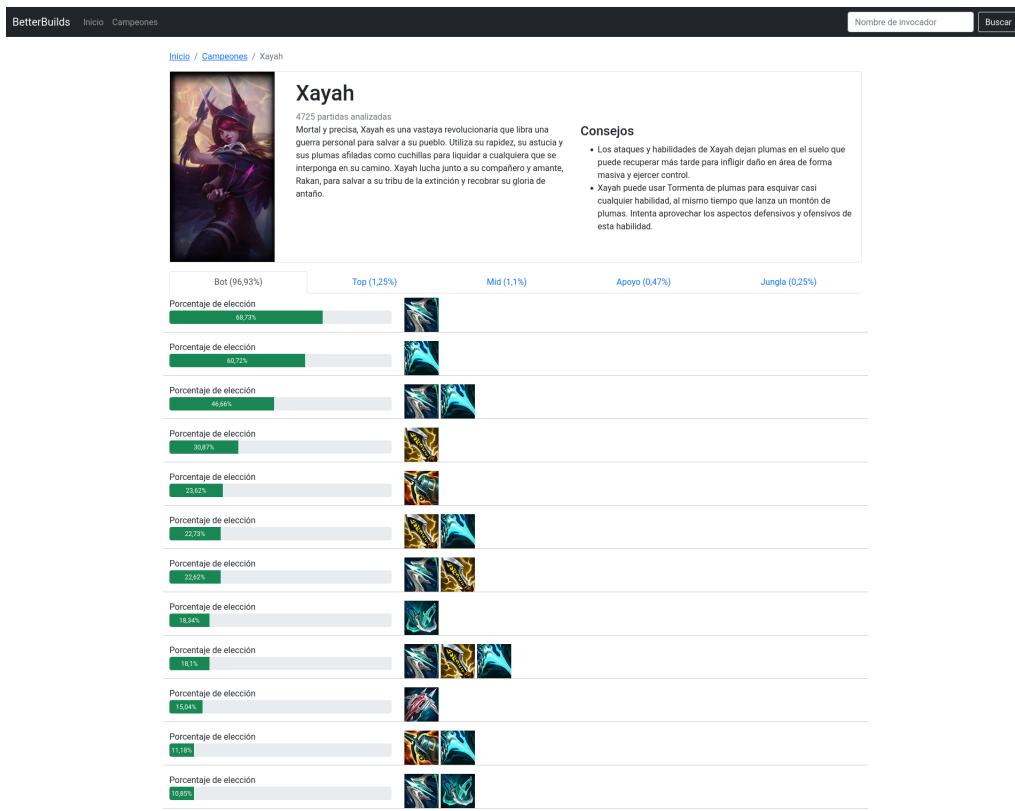


Figura E.5: Ficha de otro campeón

E.4.4. Partida actual

Para los jugadores que estén en partida y quieran realizar una consulta sobre el campeón que están jugando actualmente se da la opción de poder buscarse a si mismos mediante el apodo dentro del juego. Esta búsqueda lleva directamente al jugador a la ficha del personaje con el que se encuentra jugando (figura E.6).

En el caso de que el jugador no se encuentre en partida para el jugador buscado (figura E.7) o no exista el jugador (figura E.8), se muestra un mensaje de error avisando al usuario.

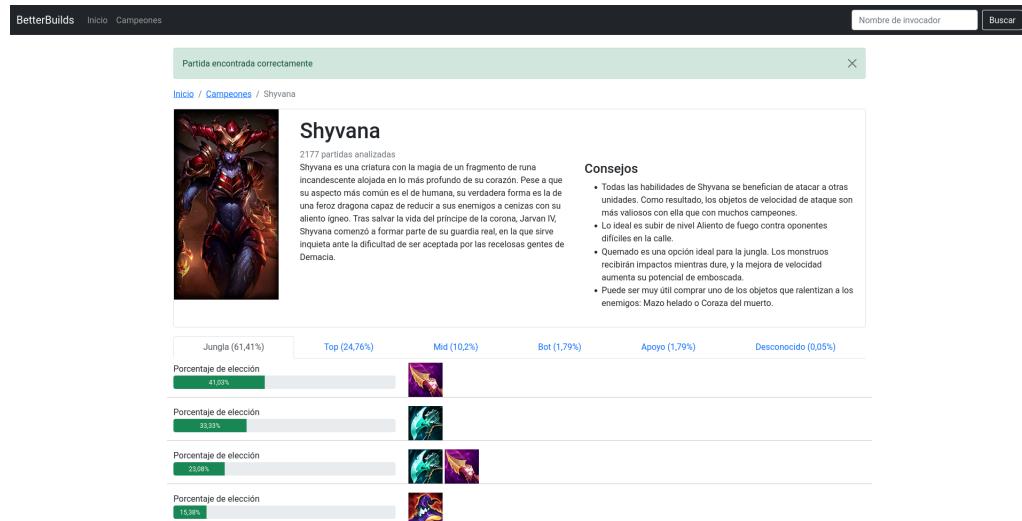


Figura E.6: Búsqueda por jugador

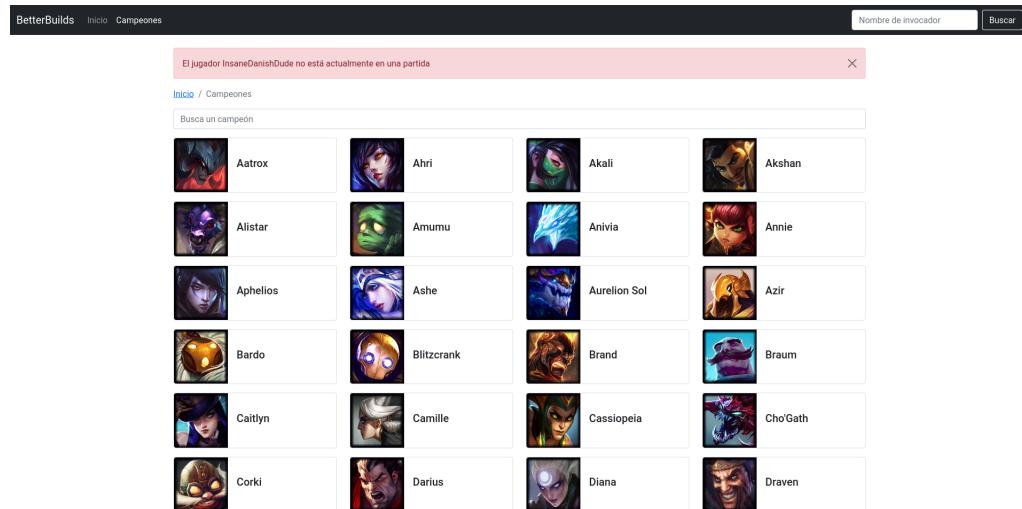


Figura E.7: El jugador no está en partida

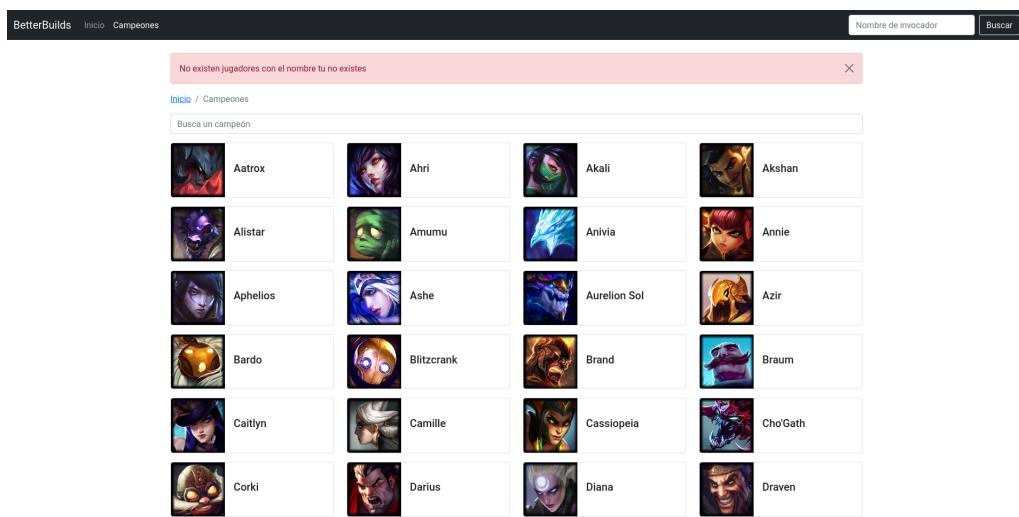


Figura E.8: No existe el jugador con el nombre introducido

Bibliografía

- [1] Gnu general public license v3.0. <https://www.gnu.org/licenses/gpl-3.0.en.html>.
- [2] How to choose a license for your own work. <https://www.gnu.org/licenses/license-recommendations.en.html>.
- [3] Rank (league of legends). [https://leagueoflegends.fandom.com/wiki/Rank_\(League_of_Legends\)](https://leagueoflegends.fandom.com/wiki/Rank_(League_of_Legends)). [Accedido 30/08/2021].
- [4] Seguridad Social: Bases y tipos de cotización 2021.
- [5] Chin-Hoong Chee, Jafreezal Jaafar, Izzatdin Abdul Aziz, Mohd Hilmie Hasan, and William Yeoh. Algorithms for frequent itemset mining: a literature review. *Artificial Intelligence Review*, 52(4):2603–2621, 2018.
- [6] Chonnyy. Apriori: Association rule mining in-depth explanation and python implementation. <https://towardsdatascience.com/apriori-association-rule-mining-explanation-and-python-implementation-290b42afdfc6>, Octubre 2020. [Accedido 29/08/2021].
- [7] Chonnyy. Fp growth: Frequent pattern generation in data mining with python implementation. <https://towardsdatascience.com/fp-growth-frequent-pattern-generation-in-data-mining-with-python-implementation-244e561ab1c3>, Noviembre 2020. [Accedido 29/08/2021].
- [8] Gonzalo Cortizo. Inditex cierra 2020 con 1.100 millones de beneficio apoyado en el crecimiento de ventas on line. <https://www.eldiario.es/galicia/politica/inditex-cierra->

- 1-100-millones-beneficio-apoyado-crecimiento-ventas-on-line_1_7291752.html, Marzo 2021. [Accedido 01/09/2021].
- [9] Riot Games. Cómo jugar - league of legends. <https://euw.leagueoflegends.com/es-es/how-to-play/>.
 - [10] Cale Michael. League of legends reportedly generated \$1.75 billion in revenue in 2020. <https://dotesports.com/league-of-legends/news/league-of-legends-reportedly-generated-1-75-billion-in-revenue-in-2020>, Enero 2021. [Accedido 01/09/2021].
 - [11] Vincenzo «Skulz» Milella. League of legends rank distribution in solo queue - june 2021. <https://www.esportstales.com/league-of-legends/rank-distribution-percentage-of-players-by-tier>.
 - [12] David Heras Pino. Lol data solution. <https://medium.com/mad-lions-e-c/lol-data-solution-2b4bead4a51>, Julio 2018. [Accedido 31/08/2021].
 - [13] Spezzy. How many people play league of legends? - updated 2021. <https://leaguefeed.net/did-you-know-total-league-of-legends-player-count-updated/>, Julio 2021. [Accedido 01/09/2021].