



TRABAJO FIN DE MÁSTER

Máster en Ciencia de Datos e Ingeniería de Datos en la Nube

Análisis y Predicción de Resultados en Partidas de Clash Royale

Autor: Iván Fernández García

Tutores: Juan Carlos Alfaro Jiménez , Juan Ángel Aledo Sánchez

Junio, 2025

A mis padres,
por su apoyo incondicional

Resumen

Este Trabajo de Fin de Máster aborda el análisis y la predicción de resultados en partidas de *Clash Royale* mediante el uso de técnicas de aprendizaje automático. En un contexto donde los videojuegos competitivos generan grandes volúmenes de datos en tiempo real, analizar esta información y aprender de ella puede aportar un valor significativo a la comunidad y a los propios desarrolladores.

Para llevar a cabo este proyecto, se ha creado un conjunto de datos a partir de información extraída a través de la API oficial de *Clash Royale*, que permite recopilar información clave sobre jugadores y partidas reales. A partir de estos datos, se han creado distintos modelos de aprendizaje automático con el objetivo de predecir qué jugador tiene más probabilidades de ganar una nueva partida antes de que esta comience.

Uno de los principales desafíos en la aplicación de modelos predictivos es la interpretabilidad de los resultados. Por ello, además de la implementación de modelos, se han incorporado técnicas de explicabilidad para comprender la influencia de las variables en la toma de decisiones y en el resultado final de las partidas. Estas técnicas permiten ofrecer una visión más clara sobre cómo los diferentes elementos del juego pueden impactar en el desempeño de los jugadores.

Como parte del trabajo, también se ha desarrollado una aplicación web que integra el mejor modelo obtenido y las técnicas de explicabilidad, brindando a los usuarios la posibilidad de interactuar con una herramienta capaz de predecir los resultados de las partidas y mejorar su comprensión sobre las dinámicas del juego.

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres y a mi familia por su constante apoyo, paciencia y confianza. Gracias por estar siempre a mi lado, por creer en mí en todo momento y por hacerme la vida más fácil en todos los sentidos.

Quiero expresar mi agradecimiento a mis tutores, Juan Carlos Alfaro y Juan Ángel Aledo, por guiarme durante todo este proceso de la mejor forma posible. Mi gratitud se extiende a José Antonio Gámez, un profesor que ha estado siempre presente para mí. Sin la pasión y el entusiasmo con los que me transmitió sus conocimientos, probablemente no habría tomado la decisión de continuar formándome en este campo.

Agradezco también a todas y cada una de las personas que, de una forma u otra, han formado parte de mi vida durante este último año. Gracias por los momentos compartidos y por las valiosas lecciones que me han ayudado a crecer. Del mismo modo, quiero reconocer el trabajo de todos los profesores del máster, quienes se han esforzado por brindarnos una formación de calidad y abrirnos puertas hacia un futuro mejor.

Por último, quiero agradecerme a mí mismo por haber afrontado cada desafío con determinación, por no rendirme ante las dificultades y por seguir apostando por mi crecimiento personal y profesional como siempre lo he hecho.

Índice general

Glosario **1**

1 Introducción **3**

1.1 Motivación 4

1.2 Objetivos 4

1.3 Estructura 5

2 Estado del arte **7**

2.1 Clash Royale 7

2.2 RoyaleAPI 10

2.3 Aprendizaje Automático: Clasificación 12

2.3.1 Regresión Logística *12*

2.3.2 Árboles de Decisión *12*

2.3.3 K-Nearest Neighbors (KNN) *13*

2.3.4 Ensembles *13*

2.3.5 Redes Neuronales *14*

3 Desarrollo del proyecto **15**

3.1 Adquisición de datos 15

3.2 Creación del conjunto de datos 17

3.3 Análisis exploratorio de datos 19

3.4 Preprocesamiento de datos 23

3.5 Creación de modelos 26

3.5.1 Validación y selección de modelos *27*

3.5.2 Evaluación final de modelos *28*

3.6 Explicabilidad 32
3.7 Desarrollo de la aplicación 34

4 Conclusiones del proyecto 37

4.1 Conclusiones y futuras mejoras 37
4.2 Competencias desarrolladas 39

A Conjuntos de datos 41

B Metodología 51

C Planificación del proyecto 53

D Recursos utilizados 55

Referencia bibliográfica 58

Índice de figuras

2.1 Partida de *Clash Royale* . . . 8
2.2 Menú principal y selección del mazo en *Clash Royale* . . . 9
2.3 Interfaz de *RoyaleAPI* . . . 11
2.4 Matchup en *RoyaleAPI* . . . 11

3.1 Distribución de la variable objetivo . . . 19
3.2 Número de partidas por arena . . . 20
3.3 Cartas más utilizadas . . . 20
3.4 Distribución del número de *counters* . . . 21
3.5 Nivel medio de las cartas en función de la arena . . . 21
3.6 Densidad del nivel estelar respecto a la variable objetivo . . . 22
3.7 Importancia de las variables . . . 22
3.8 Ejemplos de *baselines* . . . 27
3.9 Rendimiento medio de los modelos seleccionados por *pipeline* de preprocesamiento . . . 29
3.10 Rendimiento medio de los modelos seleccionados por algoritmo de aprendizaje . . . 29
3.11 Matriz de confusión y Curva ROC del modelo final sobre el conjunto de prueba . . . 30
3.12 Variables más influyentes (SHAP) . . . 32
3.13 Dependencia de las variables más influyentes (SHAP) . . . 33
3.14 Dependencia parcial entre pares de variables (PDP) . . . 33
3.15 Interpretación de una predicción individual (SHAP) . . . 34
3.16 Interfaz principal de la aplicación . . . 35
3.17 Carga de jugadores en la aplicación . . . 35
3.18 Selección de mazos en la aplicación . . . 36
3.19 Predicción de resultados en la aplicación . . . 36

B.1 Metodología CRISP-DM . . . 51

C.1 Diagrama de planificación del proyecto . . . 53

Índice de tablas

3.1 Evaluación final de modelos 28

A.1 Descripción del conjunto de datos de cartas 41
A.2 Descripción del conjunto de datos de cartas de soporte 42
A.3 Descripción del conjunto de datos de partidas 42

Glosario

Ajuste de hiperparámetros: Proceso de selección de los parámetros óptimos para un modelo de aprendizaje automático, con el fin de mejorar su rendimiento.

API (Interfaz de Programación de Aplicaciones): Conjunto de definiciones y protocolos que permiten la comunicación entre diferentes aplicaciones de software.

Aprendizaje Automático: Disciplina de la inteligencia artificial que se centra en el desarrollo de algoritmos capaces de aprender y mejorar a partir de datos, sin ser programados explícitamente para cada tarea.

Aprendizaje Supervisado: Tipo de aprendizaje automático donde el modelo se entrena utilizando un conjunto de datos etiquetados, es decir, con entradas y salidas conocidas, para luego poder predecir resultados en datos nuevos.

Ciencia de Datos: Campo interdisciplinario que utiliza métodos, procesos y sistemas científicos para extraer conocimiento y generar valor a partir de los datos.

Clasificación: Tarea de aprendizaje automático supervisado que consiste en asignar una etiqueta o categoría a una instancia basada en sus características.

Clash Royale: Videojuego de estrategia en tiempo real desarrollado por Supercell, donde los jugadores compiten en batallas uno contra uno utilizando mazos de cartas que representan tropas, hechizos y estructuras.

CRL (Clash Royale League): Competición oficial de eSports organizada por Supercell, donde equipos profesionales compiten en torneos de Clash Royale.

CRISP-DM (Cross Industry Standard Process for Data Mining): Metodología estándar de minería de datos que describe el proceso comúnmente utilizado.

EDA (Análisis Exploratorio de Datos): Proceso de análisis inicial de datos para resumir sus principales características, a menudo utilizando técnicas de visualización y estudiando las relaciones entre las variables.

eSports: Competencias de videojuegos organizadas a nivel profesional, donde jugadores y equipos compiten en diversos títulos en torneos y ligas.

Explicabilidad: Capacidad de un modelo de aprendizaje automático para proporcionar explicaciones claras y comprensibles sobre sus decisiones y predicciones.

Feedback: Retroalimentación o información recibida sobre un proceso, desempeño o resultado, utilizada para realizar mejoras o ajustes.

Framework. Entorno de desarrollo que proporciona un conjunto de herramientas, librerías y estructuras predefinidas para facilitar la creación de aplicaciones.

Freemium: Modelo de negocio que ofrece productos o servicios básicos de forma gratuita, mientras que se cobra por características o contenidos adicionales.

Inteligencia Artificial: Rama de la informática que se dedica a crear sistemas capaces de realizar tareas que requieren inteligencia humana.

JSON (JavaScript Object Notation). Formato ligero de intercambio de datos basado en texto, ampliamente utilizado por las APIs para representar estructuras de datos de forma clara y legible.

Matchmaking: Proceso en los videojuegos en línea que empareja a jugadores de habilidades similares para competir entre sí, garantizando partidas equilibradas.

Metajuego: Estrategias y tendencias dominantes en un juego en un momento determinado, que los jugadores adoptan para maximizar sus posibilidades de éxito.

Modelo Predictivo: Modelo de aprendizaje automático diseñado para predecir resultados futuros basándose en datos históricos y patrones identificados.

Outlier: Dato que se encuentra significativamente alejado de los valores promedio del conjunto de datos, pudiendo indicar errores o fenómenos inusuales.

Pipeline: Secuencia de procesos o pasos en un flujo de trabajo de ciencia de datos, desde la limpieza de datos hasta el entrenamiento y evaluación de modelos.

RoyaleAPI: Plataforma que proporciona análisis detallados, perfiles e información sobre mazos, jugadores y equipos en Clash Royale.

Sobreajuste: Situación en la que un modelo de aprendizaje automático se ajusta demasiado a los datos de entrenamiento, capturando ruido y patrones irrelevantes, lo que reduce su capacidad de generalización a nuevos datos.

Valor perdido: Dato ausente en un conjunto de datos, generalmente causado por errores de recolección o falta de información.

1. Introducción

En los últimos años, los videojuegos han experimentado un auge significativo, no solo como una forma de entretenimiento, sino también como un campo de estudio en ciencia de datos e inteligencia artificial. Desde su lanzamiento en 2016 para dispositivos móviles, *Clash Royale* se ha consolidado como uno de los títulos más populares en la categoría de estrategia en tiempo real. Con una gran cantidad de jugadores activos, el análisis de datos de partidas puede proporcionar información valiosa sobre cuáles son los factores más determinantes para conseguir la victoria.

El uso de técnicas de aprendizaje automático para predecir el resultado de partidas en *Clash Royale* abre nuevas posibilidades para la comunidad e incluso para los propios desarrolladores. A partir de un histórico de partidas obtenido a través de la API oficial del juego, es posible extraer patrones y desarrollar modelos que aprendan de los datos para estimar la probabilidad de victoria en función de diferentes variables.

Se llevará a cabo un proceso que incluye la recopilación de datos a través la API del juego, la limpieza y transformación de estos, la exploración de relaciones entre variables y la construcción de modelos predictivos mediante técnicas de aprendizaje automático. Finalmente, el mejor modelo obtenido se integrará en una aplicación interactiva que permitirá a los usuarios introducir información sobre una partida y obtener una estimación de su posible desenlace. Además, se incorporarán técnicas de explicabilidad para comprender cómo los diferentes elementos del juego influyen en las predicciones.

Las aplicación del aprendizaje automático en distintos campos ha demostrado que aprender de grandes cantidades de datos permite tomar mejores decisiones. En el caso de *Clash Royale*, este enfoque permite extraer conocimiento a partir de partidas para identificar patrones de éxito y optimizar estrategias. Así como en otros sectores estas técnicas ayudan a reducir la incertidumbre y mejorar el rendimiento, en este contexto pueden convertirse en una herramienta clave para jugadores que buscan apoyarse en datos para progresar y definir sus prioridades dentro del juego.

1.1. Motivación

La toma de decisiones basada en datos puede proporcionar ventajas competitivas significativas en juegos como *Clash Royale*. Sin embargo, a pesar de la gran cantidad de información disponible, esta no siempre se aprovecha de manera óptima. La aplicación del aprendizaje automático ofrece un enfoque innovador para extraer conocimiento útil de partidas pasadas y mejorar la toma de decisiones estratégicas en el juego.

La motivación de este trabajo se basa en la oportunidad de aplicar metodologías y técnicas avanzadas en un contexto real y dinámico, proporcionando una herramienta que puede beneficiar tanto a jugadores casuales como a competidores de alto nivel. Además, la extracción y procesamiento de datos a través de la API del juego, así como el desarrollo de una aplicación que integre el modelo predictivo, son desafíos técnicos adicionales que permiten explorar el ciclo completo de un proyecto de ciencia de datos, desde la obtención de información hasta la implementación de una solución práctica.

1.2. Objetivos

El objetivo principal de este Trabajo Fin de Máster es desarrollar un modelo de aprendizaje automático capaz de predecir el resultado en partidas de *Clash Royale*, un famoso juego de estrategia cuyos fundamentos se detallan en la sección 2.1. Para ello se utilizarán datos históricos de partidas extraídas a través de la API oficial del juego.

Por lo tanto, se consideran los siguientes objetivos:

- Utilizar la API oficial de *Clash Royale* para extraer datos de partidas, diseñando un proceso automatizado que permita adquirir la información. Realizar una limpieza y transformación de los datos crudos para crear un conjunto de datos estructurado a partir del cual podamos comenzar a resolver el problema.
- Llevar a cabo un análisis exploratorio de los datos, analizando las variables disponibles y las relaciones entre ellas. En base a lo observado, aplicar las técnicas de preprocesamiento necesarias para que los algoritmos de aprendizaje automático puedan trabajar correctamente con los datos y aprender de ellos.
- Aplicar técnicas de aprendizaje supervisado para desarrollar modelos capaces de predecir resultados de forma eficiente, ajustando los hiperparámetros y evaluando de manera honesta el rendimiento de distintos clasificadores.
- Desarrollar una aplicación que integre el mejor modelo obtenido e incorpore técnicas de explicabilidad, de modo que los usuarios puedan predecir nuevas partidas y comprender cómo los distintos factores del juego influyen en el resultado.
- Analizar qué variables son más determinantes y sacar conclusiones sobre el rendimiento del modelo: ¿El *matchmaking* está amañado?

1.3. Estructura

El presente Trabajo Fin de Máster está formado por las siguientes partes:

1. **Introducción.** Se presenta el trabajo a desarrollar y se describen aspectos como la motivación de su realización y los objetivos del proyecto (Capítulo 1).
2. **Estado del arte.** Se realiza un análisis de *RoyaleAPI*, una de las herramientas más conocidas en relación con los objetivos del proyecto, destacando su importancia en la recopilación y visualización de datos de *Clash Royale* y su influencia en la toma de decisiones estratégicas de la comunidad. Además, se explican los aspectos clave del juego y los fundamentos del aprendizaje automático, con un enfoque específico en problemas de clasificación (Capítulo 2).
3. **Desarrollo del proyecto.** Se lleva a cabo el proceso de ciencia de datos, desde la adquisición de la información hasta el despliegue, pasando por las fases habituales de análisis exploratorio de datos, preprocesamiento y modelado (Capítulo 3).
4. **Conclusiones del proyecto.** Se extraen las conclusiones y las futuras mejoras del proyecto (Capítulo 4).
5. **Apéndice.** Parte complementaria que incluye información adicional sobre los conjuntos de datos utilizados (Capítulo A), la metodología (Capítulo B), la planificación del proyecto (Capítulo C) y los recursos utilizados (Capítulo D).

2. Estado del arte

A lo largo de este capítulo se realizará un análisis de *RoyaleAPI*, una de las herramientas más relevantes relacionadas con este proyecto, destacando su papel en la recopilación y visualización de datos de *Clash Royale*. También se explicarán brevemente los aspectos más importantes del juego para comprender su funcionamiento y los fundamentos del aprendizaje automático, con un enfoque particular en clasificación.

2.1. Clash Royale

Clash Royale es un juego de estrategia en tiempo real desarrollado por Supercell [9], empresa creadora de otros títulos conocidos como *Clash of Clans* y *Brawl Stars*. Desde su lanzamiento para dispositivos móviles a principios de 2016 [1], el juego ha mantenido una base de jugadores activa y ha evolucionado a través de constantes actualizaciones con nuevos modos de juego y cambios de balance.

Con el paso de los años se ha convertido en un éxito tanto en el ámbito casual como en el competitivo, alcanzando más de 500 millones de descargas en dispositivos Android [3] y generando ingresos significativos a través de compras dentro del juego, fruto del modelo de negocio *freemium* [5]. Gracias a su accesibilidad y jugabilidad, ha desarrollado una gran comunidad y una amplia presencia en plataformas de streaming.

En *Clash Royale*, dos jugadores se enfrentan durante un tiempo inicial de tres minutos en un campo de batalla dividido en dos mitades, cada una con tres torres: dos torres de coronas y una torre del rey (Figura 2.1). El objetivo es destruir más torres que el oponente antes de que termine el tiempo o, idealmente, derribar la torre del rey para lograr una victoria inmediata de tres coronas. En caso de empate, la partida se prolonga durante un tiempo adicional de muerte súbita. Si ningún jugador consigue dar el golpe final, gana el que deje con menos vida alguna de las torres del rival.



Figura 2.1: Partida de *Clash Royale*.

El juego se basa en el uso de cartas, que representan tropas, estructuras y hechizos. Antes de la partida, cada jugador debe construir un mazo seleccionando ocho de sus cartas disponibles (Figura 2.2), y durante el enfrentamiento estas se pueden desplegar en el campo a cambio de elixir, un recurso que va regenerando con el tiempo. Además de las ocho cartas del mazo, también se debe seleccionar un tipo de tropa especial para defender las dos torres de coronas. Cada carta tiene una rareza (común, especial, épica, legendaria o campeón), habilidades y funciones concretas, un nivel y un coste de elixir. Por lo tanto, administrar correctamente el elixir es clave para evitar quedar en desventaja ante el oponente. En el último minuto la velocidad de generación se duplica, lo que intensifica la acción y obliga a los jugadores a tomar decisiones rápidas y estratégicas. Para poder mejorar las cartas hay que obtener unidades de las mismas en la tienda o a través de diferentes elementos del juego, y habitualmente la diferencia en el nivel de las cartas de ambos jugadores es un factor clave en el resultado.

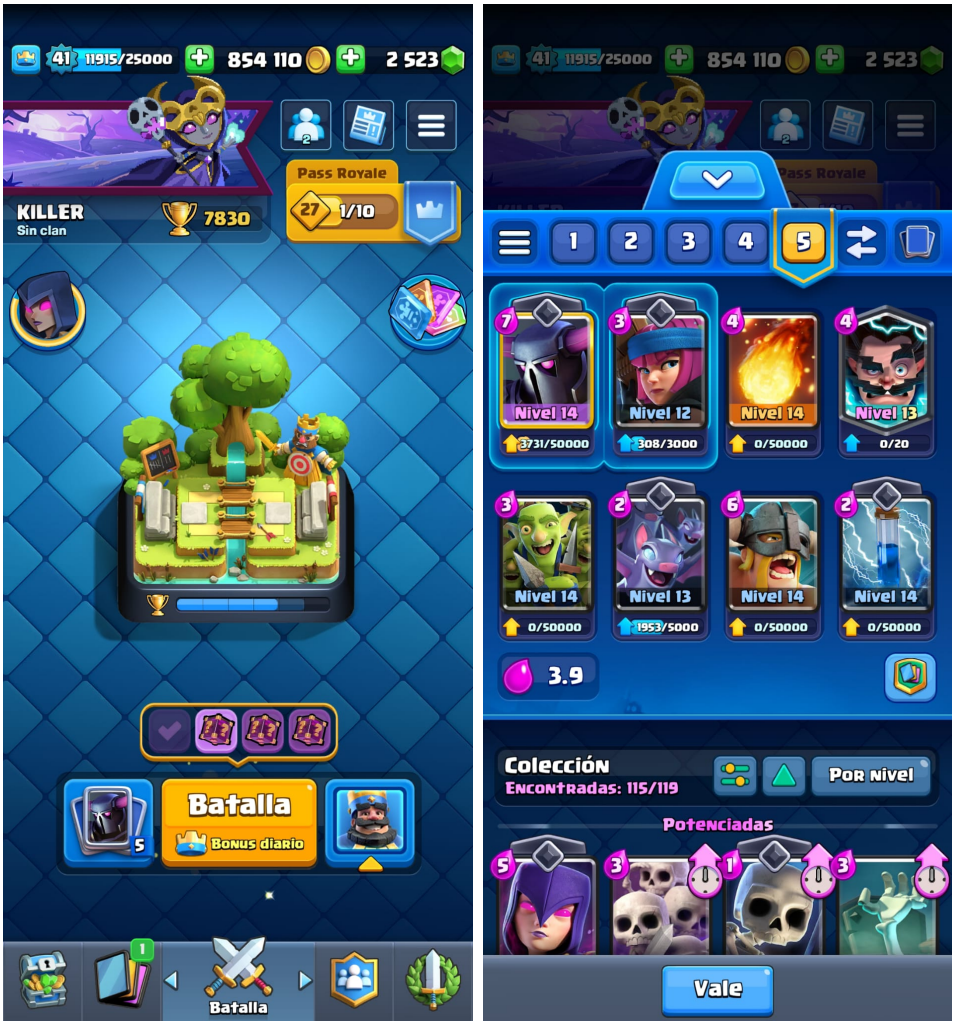

Figura 2.2: Menú principal y selección del mazo en *Clash Royale*.

El sistema de progresión en *Clash Royale* se conoce como *Camino de Trofeos* y se basa en arenas que los jugadores van desbloqueando al acumular trofeos ganando partidas. Cada nueva arena otorga acceso a cartas adicionales y plantea desafíos estratégicos más complejos. A partir de los 5000 trofeos, los jugadores también tienen acceso a un sistema de ligas conocido como *Camino de Leyendas*, donde compiten en un entorno más competitivo con reglas de emparejamiento más exigentes. Además de las batallas tradicionales uno contra uno, existe una gran variedad de modos de juego como desafíos, torneos o guerras de clanes, entre otros. *Clash Royale* ha desarrollado un ecosistema competitivo y de *eSports* [11], con torneos organizados por Supercell y terceros. La *Clash Royale League* (CRL) ha sido el evento estrella del juego, atrayendo a jugadores de élite y ofreciendo grandes premios en sus finales mundiales [6].

Uno de los aspectos clave del juego ha sido su constante evolución mediante actualizaciones regulares. Estas incluyen:

- Ajustes de balance: Modificaciones en las estadísticas de las cartas para mantener un entorno de juego equilibrado.
- Nuevas cartas y mecánicas: Introducción de tropas con habilidades únicas que ofrecen nuevas posibilidades y diversifican las estrategias.
- Mejoras en la jugabilidad y emparejamiento: Cambios en el algoritmo de *matchmaking* y en el sistema de progresión para mejorar la experiencia de los jugadores.

El estudio de datos provenientes de la API oficial de *Clash Royale* [4] puede proporcionar información valiosa sobre patrones de juego, variables importantes en las partidas y predicción de resultados, lo que tiene aplicaciones tanto en la mejora individual del jugador como en el análisis de tendencias a nivel casual y competitivo.

2.2. RoyaleAPI

RoyaleAPI es una de las herramientas más reconocidas y utilizadas dentro de la comunidad de *Clash Royale*, destacándose como un recurso esencial para la recopilación, visualización y análisis de datos relacionados con el juego [8]. Desde su lanzamiento, esta plataforma ha permitido a los jugadores optimizar sus estrategias, tomar decisiones informadas y comprender mejor la dinámica del juego mediante el acceso a información detallada y en tiempo real (Figura 2.3). Nació en 2017 como una API para desarrolladores y posteriormente se convirtió en una página web para jugadores, pero no debe confundirse con la API oficial del juego que utilizaremos para adquirir los datos en este proyecto. En 2020 se deshabilitaron los servicios de la API debido a los altos costes y a las políticas de Supercell, aunque la página web no se vio afectada por estos cambios [2].

Se caracteriza por cubrir una amplia gama de funcionalidades diseñadas para abordar diferentes necesidades dentro de la comunidad de *Clash Royale*:

- **Estadísticas de jugadores:** Permite a los usuarios consultar estadísticas detalladas de cualquier jugador, como el historial de partidas, mazos utilizados, rendimiento frente a las cartas o progreso en ligas y trofeos.
- **Análisis de clanes:** Proporciona herramientas para la gestión de clanes, como las contribuciones individuales de los miembros y el progreso en las guerras.
- **Meta del juego y popularidad de cartas:** Recopila y analiza datos a gran escala en tiempo real sobre la popularidad y efectividad de las cartas en el metajuego actual para los distintos modos de juego. Esto incluye información sobre las combinaciones de mazos más exitosas, tendencias emergentes y counters más efectivos. Por lo tanto, es ampliamente utilizada en el ámbito competitivo.

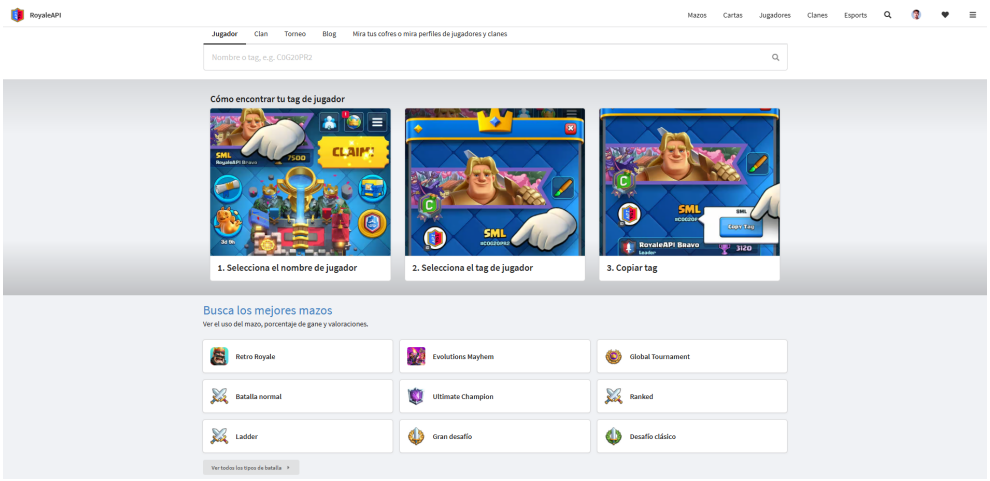

Figura 2.3: Interfaz de *RoyaleAPI* [8].

RoyaleAPI también ha incorporado una funcionalidad que permite analizar los enfrentamientos entre mazos (*matchups*), proporcionando información sobre las tasas de victoria (Figura 2.4). Esta herramienta ofrece datos precisos sobre cómo un mazo se desempeña contra otro, incluso cuando las coincidencias no son exactas. Sin embargo, este proyecto tiene como objetivo ir un paso más allá. Mientras que *RoyaleAPI* se centra principalmente en la visualización y análisis de los datos en tiempo real, nuestro enfoque se orientará hacia el uso de técnicas de aprendizaje automático para extraer patrones de enfrentamientos pasados y predecir el resultado de nuevas partidas.

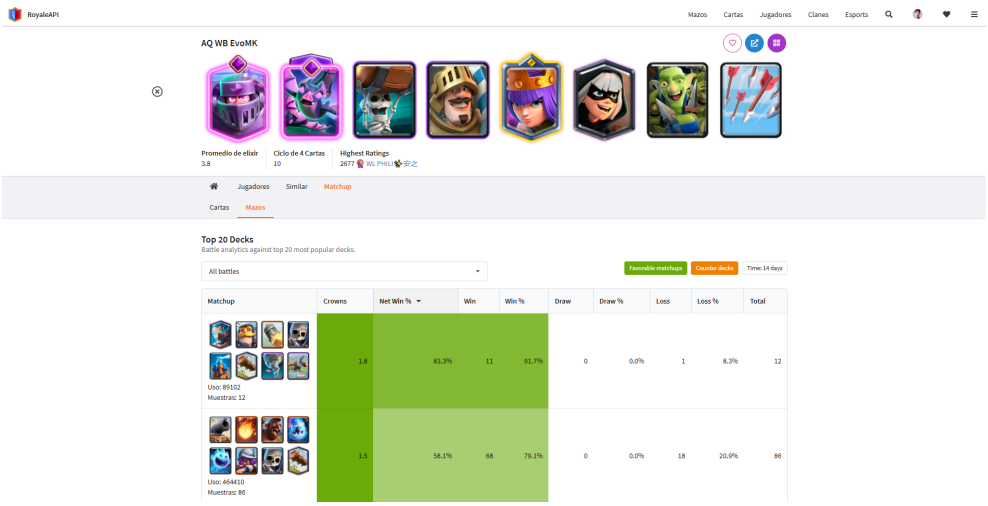

Figura 2.4: Matchup en *RoyaleAPI* [7].

2.3. Aprendizaje Automático: Clasificación

El aprendizaje automático, también conocido como *Machine Learning*, es una rama de la inteligencia artificial que permite a los sistemas identificar patrones en los datos para tomar decisiones de forma autónoma sin necesidad de ser programados [18].

Dentro de esta disciplina, los problemas se clasifican en tres categorías principales:

- **Aprendizaje supervisado:** El modelo se entrena con datos etiquetados, es decir, ejemplos donde conocemos la respuesta correcta.
- **Aprendizaje no supervisado:** Se trabaja con datos sin etiquetas, buscando patrones subyacentes y agrupaciones sin conocimiento previo.
- **Aprendizaje por refuerzo:** Los agentes aprenden a tomar decisiones interactuando el entorno a través de un sistema basado en recompensas y penalizaciones.

En este proyecto nos enfocaremos en el aprendizaje supervisado, y más específicamente en un problema de clasificación, donde la variable objetivo es categórica. En el contexto de *Clash Royale*, esto se traduce en un problema de clasificación binaria en el que, a partir de un histórico de partidas de las que conocemos el ganador, queremos predecir el resultado de nuevas partidas. A continuación se van a explicar brevemente algunos algoritmos tradicionales de aprendizaje automático para clasificación.

2.3.1. Regresión Logística

La regresión logística es un modelo estadístico utilizado para problemas de clasificación binaria, aunque también se puede ajustar a problemas con más de dos clases. A diferencia de la regresión lineal, que predice valores continuos, la regresión logística modela la probabilidad de pertenencia mediante la función sigmoide, donde z es una combinación lineal de las variables de entrada:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \qquad\qquad z = w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n$$

El resultado es un valor entre 0 y 1, que se interpreta como la probabilidad de pertenecer a la clase positiva [16]. Para clasificar se suele establecer un umbral de 0.5.

2.3.2. Árboles de Decisión

Los árboles de decisión son modelos que organizan la toma de decisiones en una estructura jerárquica de nodos que puede interpretarse como un conjunto de reglas [14]. Cada nodo representa una condición sobre una característica del conjunto de datos, y cada rama corresponde a una posible categoría en el caso de las variables discretas y a un valor binario (superar o no un umbral) en el caso de las variables numéricas.

El criterio de partición suele basarse en la ganancia de información mediante la entropía o el índice de *Gini*, que mide la pureza de los subconjuntos generados tras una división. En problemas de clasificación, el valor de un nodo hoja coincide con la clase mayoritaria en la partición que lo alcanza.

Los árboles de decisión son fáciles de interpretar, pero pueden ser propensos al sobreajuste si no se limita su profundidad o el número de nodos. Esto significa que el modelo encuentra dificultades para clasificar correctamente datos nuevos pese a obtener un buen rendimiento sobre el conjunto de entrenamiento [12].

2.3.3. K-Nearest Neighbors (KNN)

El algoritmo K-Nearest Neighbors clasifica o predice una muestra en función de sus vecinos más cercanos [13]. El procedimiento es el siguiente:

1. Se elige un valor de k (número de vecinos a considerar).
2. Se calcula la distancia entre la nueva muestra y las muestras de entrenamiento.
3. Se seleccionan los k vecinos más cercanos.
4. En clasificación, se asigna la clase mayoritaria entre los vecinos.

KNN es un modelo que suele ofrecer buenos resultados pese a su sencillez, aunque puede volverse ineficiente con grandes conjuntos de datos debido al aumento de la dimensionalidad. Antes de utilizarlo se recomienda estandarizar los datos para que las distancias entre ejemplos se calculen de forma más precisa, ya que las variables numéricas pueden estar en diferentes escalas.

2.3.4. Ensembles

Los métodos de ensemble o multiclasificadores combinan varios modelos para mejorar la precisión y reducir el sobreajuste. La idea principal es que un conjunto de modelos puede ser más robusto y generalizar mejor que uno solo [15].

Bagging (*Bootstrap Aggregating*)

Es una técnica que reduce la varianza de los modelos base mediante la creación de múltiples subconjuntos aleatorios del conjunto de datos, entrenando un modelo independiente en cada uno y combinando sus predicciones:

1. Se generan subconjuntos del conjunto original mediante muestreo con reemplazo.
2. Se entrena un modelo independiente en cada subconjunto.
3. En clasificación, la predicción final se obtiene mediante votación por la mayoría.

Random Forest es una extensión de *Bagging* aplicada a árboles de decisión y un algoritmo ampliamente utilizado en problemas de aprendizaje supervisado. Tiene algunas particularidades respecto a *Bagging*:

- Los modelos base son siempre árboles de decisión.
- No solo selecciona subconjuntos de datos, sino que también selecciona aleatoriamente un subconjunto de características en cada división del árbol. Esto reduce la correlación entre los clasificadores, mejorando la generalización del modelo.

Boosting

Se basa en la idea de entrenar modelos de manera secuencial, donde cada modelo trata de corregir los errores del anterior. Los modelos más recientes reciben más peso en las observaciones que fueron mal clasificadas en iteraciones anteriores:

1. Se entrena un modelo débil (*weak learner*) en el conjunto de datos.
2. Se calculan los errores y se ajustan los pesos de las muestras mal clasificadas.
3. Se entrena un nuevo modelo con mayor énfasis en las muestras difíciles.
4. Tras varias iteraciones, se combinan los modelos para la predicción final.

Los algoritmos más conocidos son:

- ***AdaBoost* (*Adaptive Boosting*):** Ajusta los pesos de las muestras en cada iteración de modo que los ejemplos mal clasificados reciban más importancia. La predicción final se obtiene mediante una combinación ponderada.
- ***Gradient Boosting*:** Optimiza una función de pérdida mediante descenso de gradiente. La idea es minimizar el error residual en cada iteración ajustando modelos adicionales para corregir los errores previos.

2.3.5. Redes Neuronales

Las redes neuronales artificiales están inspiradas en el funcionamiento del cerebro humano y consisten en capas de neuronas interconectadas. Una red neuronal estándar tiene una capa de entrada, una capa de salida y capas ocultas que procesan la información mediante combinaciones de pesos y funciones de activación. Al entrenar, los pesos se van actualizando mediante el algoritmo del gradiente descendente para minimizar la función de pérdida e ir mejorando el rendimiento [17].

Las redes neuronales son potentes para detectar patrones complejos y suelen ofrecer mejores resultados que otros modelos tradicionales, pero requieren grandes cantidades de datos y mayores recursos computacionales.

3. Desarrollo del proyecto

En este capítulo se presenta una visión general del desarrollo del proyecto, desde la adquisición inicial de los datos hasta la implementación de una solución práctica que integra el modelo desarrollado. Cada fase del proceso ha sido documentada y explicada en la memoria, pero para una comprensión más profunda, se recomienda consultar el resto de material que acompaña el proyecto. Esto incluye el código completo y análisis detallados que complementan y amplían lo expuesto en este documento.

3.1. Adquisición de datos

Esta primera fase se ha centrado exclusivamente en la adquisición y recopilación de datos relevantes para resolver el problema que queremos abordar. Para ello, se ha utilizado principalmente la API oficial de Clash Royale, la cual proporciona acceso a información detallada sobre jugadores, clanes, partidas recientes, estadísticas de cartas, etc. Además, se ha explorado la posibilidad de complementar esta información con datos provenientes de otras fuentes. El resultado de esta etapa ha sido una gran cantidad de datos en crudo que han servido como base para fases posteriores.

Debemos tener en cuenta que *Clash Royale* cuenta con varios modos de juego en los que las reglas son diferentes, por lo que no es adecuado desarrollar un modelo que abarque todo tipo de partidas. Este proyecto se ha enfocado en partidas 1vs1 del modo *Camino de Trofeos*. De este modo, podremos predecir partidas entre jugadores sin aplicar restricciones adicionales de otros modos de juego más recientes y menos comunes. Hay que tener en cuenta que también se pueden simular partidas amistosas entre jugadores dispares, pero no debemos olvidar que aprenderemos de partidas entre jugadores con copas muy similares fruto del propio emparejamiento del juego. Como futura mejora se propone desarrollar modelos para modos de juego competitivos como *Camino de Leyendas*, en el que se iguala el nivel de las cartas.

La única forma de extraer información de partidas a través de la API es mediante el registro de batalla de los jugadores, por lo que se definió una metodología que permitiera obtener un conjunto de datos lo suficientemente grande y variado que cumpliera las restricciones del problema a resolver:

1. Obtener tags únicos de clanes aleatorios con muchos miembros.
2. Obtener tags de jugadores a través de los miembros de los clanes anteriores.
3. Obtener la última partida de los jugadores anteriores considerando que:
    - Es posible que haya jugadores sin partidas recientes.
    - Sólo se tendrán en cuenta partidas del modo que nos interesa.
    - Se van a complementar con información de otras fuentes como el propio juego y plataformas como RoyaleAPI o similares (tipos de cartas, *win conditions*, tropas aéreas, cartas con daño aéreo, *counters*...)
    - En la siguiente etapa se limpiarán y transformarán los datos.

Esta metodología se eligió principalmente para obtener una cantidad de datos manejable y que a la misma vez permitiera crear modelos capaces de generalizar. Hubiera sido posible obtener todo el registro de batalla de los jugadores, pero la cantidad de información crecería enormemente. Si tenemos en cuenta que el conjunto de datos tampoco puede ser excesivamente grande, se ha considerado que es preferible quedarnos solamente con la última partida de cada jugador. Dispuestos a tener un número de ejemplos similar en nuestro conjunto de datos, una única partida por jugador podría proporcionar una mayor variedad, ya que un mismo jugador no suele cambiar frecuentemente de mazo y tiene una forma determinada de jugar. Por lo tanto, la estrategia elegida tiene como objetivo adquirir información de partidas de modo que esta sea variada y apta para los recursos disponibles.

Con el objetivo de enriquecer el conjunto de datos, se evaluó la posibilidad de complementar los registros de partidas con información adicional de los perfiles de los jugadores que pudiera aportar valor (nivel de experiencia, tasa de victorias, récord de trofeos o maestría de cartas). Esta información es accesible, pero presenta una limitación importante: la API solo devuelve el estado actual del perfil del jugador, sin conservar versiones anteriores que permitan reconstruir su estado exacto en el momento de cada partida. Dado que muchas de estas variables del perfil pueden haber cambiado desde el momento en que se jugó la partida (por ejemplo, un jugador podría haber ganado más puntos de experiencia o podría haber alcanzado un nuevo récord de trofeos), estaríamos entrenando el modelo con información generada después del evento que se quiere predecir. Aunque en un escenario de predicción real el perfil del jugador sí estará disponible antes de que ocurra una nueva partida, el problema radica en el entrenamiento. Al usar partidas antiguas complementadas con perfiles actuales, se asume erróneamente que esa información es contemporánea al evento. Por ello, se ha optado por utilizar exclusivamente los datos disponibles en el registro de la partida.

3.2. Creación del conjunto de datos

Esta fase ha consistido en transformar los datos crudos (en formato JSON con estructuras anidadas) en un conjunto de datos tabular de modo que cada fila corresponda a una observación. Para ello, se ha comprendido la información recopilada y posteriormente se han realizado diferentes tareas de limpieza y transformación.

Para las cartas (Tabla A.1) y las tropas de las torres (Tabla A.2):

- Normalizar la información de las tropas de las torres obtenida a través de la API.
- Normalizar la información de las cartas obtenida a través de la API.
- Fusionar la información normalizada de las cartas con el resto de propiedades obtenidas a través de otras fuentes.

Para las partidas (Tabla A.3):

- Normalizar la información de las partidas obtenida a través de la API.
- Descartar variables irrelevantes comunes a todas las partidas del modo de juego.
- Al ser partidas 1vs1, extraer la información del único jugador de cada equipo.
- Descartar variables de los jugadores de las no dispondremos en el momento de la predicción, ya que son estadísticas al final de la partida (p.e. el daño a las torres).
- Crear la variable objetivo, utilizando el cambio de trofeos de cada jugador para determinar si el ganador es el primero o el segundo.
- Descartar las propiedades irrelevantes para la tropa de las torres de cada jugador, conservando solamente el nombre, el nivel y la rareza.
- Descartar las propiedades irrelevantes para las cartas del mazo de cada jugador y fusionar el resto con las demás propiedades.
- A partir de las propiedades las cartas del mazo de cada jugador, realizar agregaciones y diferentes transformaciones para crear variables que lo describan (nivel medio, número de cartas de cada rareza, número de cartas de cada tipo...)
- Incluir el número de *counters* de cada jugador mediante las combinaciones de sus cartas con las del rival, así como el número de cartas del mazo que quedan sin contrarrestar y pueden suponer una ventaja.
- Crear una variable binaria por cada carta del juego para cada jugador y asignar unos a las cartas de su mazo. De este modo el mazo se puede interpretar como un subconjunto de cartas en el que el orden no importa.

Hay que tener en cuenta que se dispone de una variable que indica el instante en el que se jugó la partida. Esto no supone un problema siempre que las condiciones dentro del juego sean las mismas (podría considerarse que se han jugado en el mismo periodo), pero sí cuando tenemos partidas que se han jugado antes y después de un evento que modifica la jugabilidad. Esto ocurre en los siguientes casos:

- Cambios de balance: No podemos aprender patrones de partidas en las que las cartas tienen determinadas propiedades (por ejemplo, daño) para predecir otras anteriores en las que esta información era diferente.
- Nueva carta: No podemos aprender patrones de partidas en las que se ha utilizado una nueva carta para predecir otras anteriores en las que esta no estaba disponible.

Para evitar tener que gestionar este problema reservando las últimas partidas en el conjunto de prueba y utilizando tipos de validación cruzada específicos para respetar el orden cronológico, se han filtrado las partidas que se jugaron después del último de estos eventos (en este caso los cambios de balance del 9 de abril) y se ha considerado que todas ellas pertenecen a un mismo periodo donde las condiciones dentro del juego no cambian independientemente del instante temporal de cada una. Cada vez que ocurra uno de estos cambios los modelos tendrían que entrenarse con datos posteriores y podrían ser utilizados de manera eficiente hasta que se produzca otro cambio. Los modelos podrían utilizarse incluso de forma indefinida si así se considera, aunque podrían ser menos eficientes. En cualquier caso, nunca se deberían predecir partidas anteriores al último de estos eventos con información actual.

Para resolver el problema, lo ideal sería que el conjunto de datos estuviera completamente balanceado. De este modo, si los modelos tienden a predecir uno de los jugadores se deberá únicamente a los patrones detectados, pero nunca a una desigualdad en el número de ejemplos de cada clase. Por lo tanto, a pesar de tener una diferencia muy reducida, se han descartado aleatoriamente algunos ejemplos de la clase mayoritaria con el objetivo de conseguir un equilibrio total.

Finalmente, se ha divido el conjunto de datos en un subconjunto de entrenamiento con el 80% de los ejemplos y otro de prueba con el 20% restante. Se ha reservado el segundo para la evaluación final de modelos, de modo que la información de sus partidas no se utilice hasta entonces. Tampoco durante el análisis exploratorio, pues estaríamos tomando decisiones en base a estos datos.

3.3. Análisis exploratorio de datos

El objetivo de esta fase es comprender mejor los datos de los que partimos para resolver el problema, estudiando las relaciones entre variables y sacando conclusiones que nos permitan tomar mejores decisiones de cara al preprocesamiento para que los modelos ofrezcan buenos resultados y generalicen lo suficiente. La descripción del conjunto de datos se encuentra en la tabla A.3.

Tras realizar una breve exploración inicial, se detectaron algunos valores extremos en determinadas variables como mazos compuestos exclusivamente por hechizos, edificios, cartas aéreas o cartas con la misma rareza. Por otra parte, la distribución de la variable objetivo se puede apreciar en la Figura 3.1.

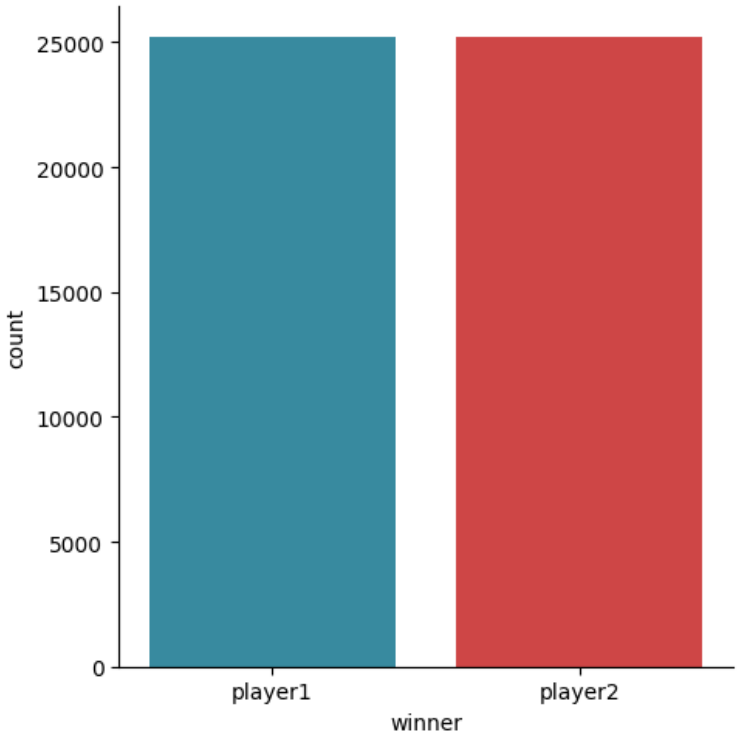

Figura 3.1: Distribución de la variable objetivo.

A lo largo del análisis, se ha utilizado el color azul para mostrar información del primer jugador y el color rojo para mostrar información del segundo jugador.

La metodología empleada durante la adquisición de datos ha permitido recopilar partidas de diferentes arenas, si bien es cierto que no se distribuyen igual (Figura 3.2).

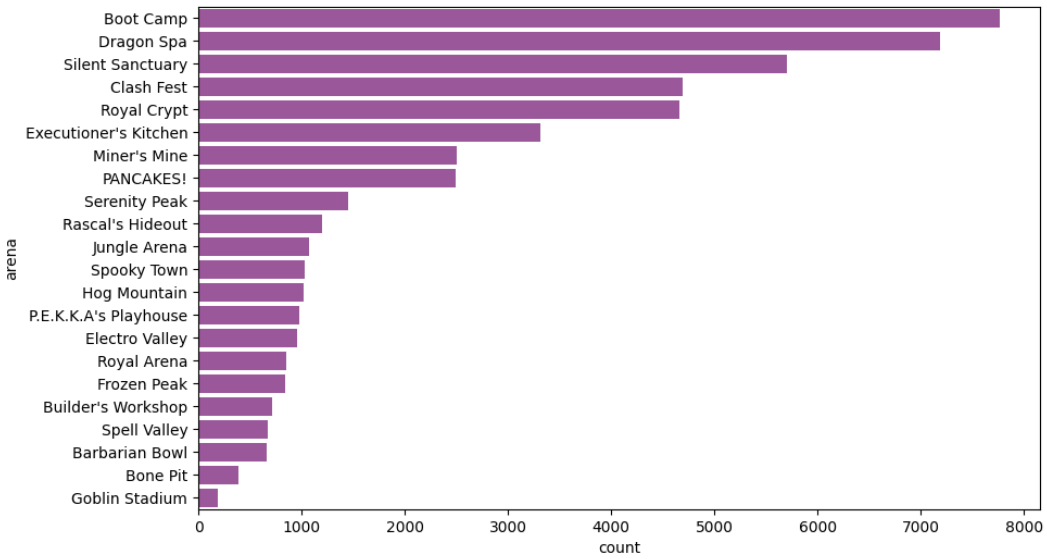

Figura 3.2: Número de partidas por arena.

En la Figura 3.3 se muestran las cartas más utilizadas por cada jugador.

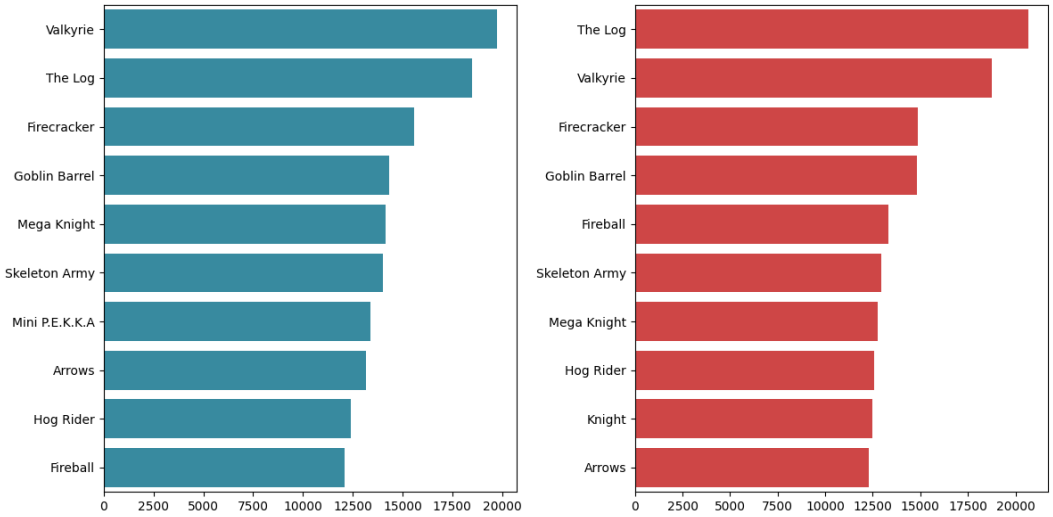

Figura 3.3: Cartas más utilizadas.

Al detectar que la mayoría de variables eran las mismas para ambos jugadores, se ha decidido explorar la posibilidad de calcular la diferencia para medir la ventaja. Si el resultado es positivo, significa que el valor es mayor para el primer jugador y viceversa. También se ha estudiado crear otras variables a través de conocimiento experto que puedan aportar valor adicional, como un puntaje que mida el equilibrio del mazo o la ventaja que puede suponer utilizar más cartas *Win Condition* que edificios el rival. Además, mostrar las distribuciones de la diferencias también nos permite evaluar cómo de sensible es el sistema de *matchmaking* del juego frente a distintos factores. Por ejemplo, en la Figura 3.4 se aprecian los histogramas para el número de *counters*.

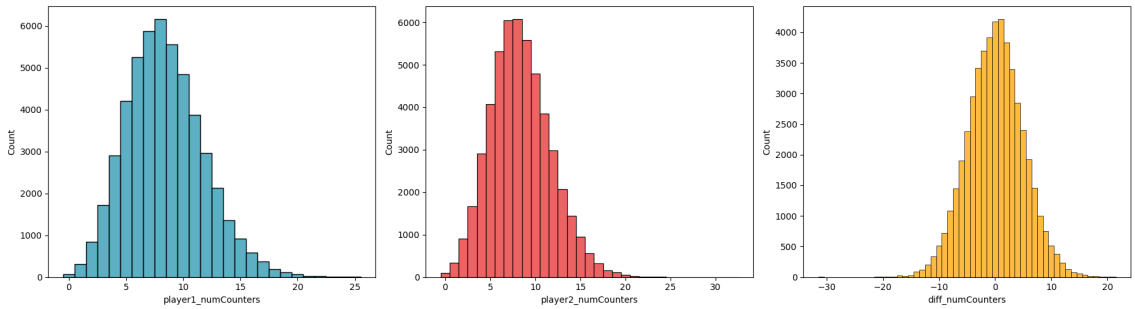

Figura 3.4: Distribución del número de *counters*.

También puede resultar de interés observar cómo varía el nivel medio de las cartas en función de la arena. La Figura 3.5 muestra que este es creciente, aunque no se observa una diferencia clara entre ambos jugadores para ninguna arena en concreto.

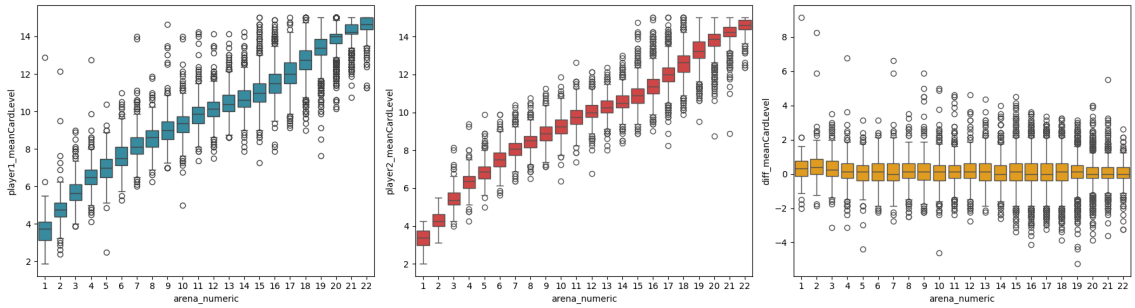

Figura 3.5: Nivel medio de las cartas en función de la arena.

Para estudiar la relación de las variables predictoras con la variable objetivo, se ha decidido mostrar la distribución de densidad para ambos jugadores y para la diferencia. En la Figura 3.6 se aprecia que el nivel estelar parece ser, por sí mismo, un factor bastante determinante en el resultado.

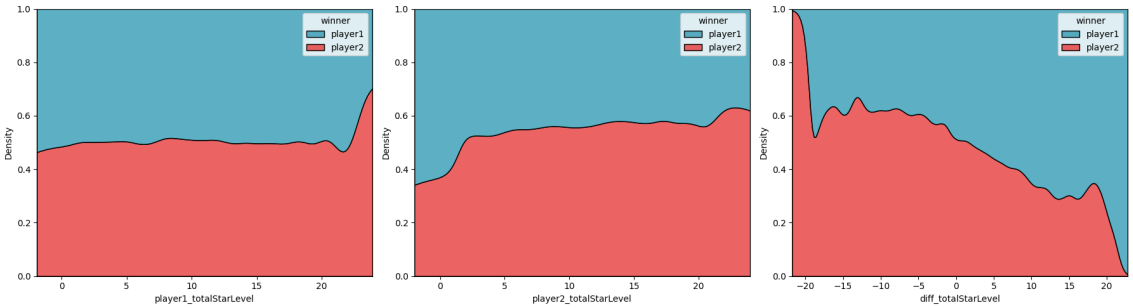

Figura 3.6: Densidad del nivel estelar respecto a la variable objetivo.

También se han obtenido matrices de correlación para las variables numéricas de cada jugador y sus diferencias, con el objetivo de comprender mejor los datos y de descartar posteriormente alguna característica altamente correlacionada con otras que pudiera empeorar la capacidad de los modelos para aprender patrones. Así mismo, se ha calculado la varianza de las variables y se ha evaluado su importancia entrenando, a modo informativo, clasificadores que permiten obtenerla. La Figura 3.7 muestra la importancia de las variables tras entrenar un *Random Forest* con las diferencias.

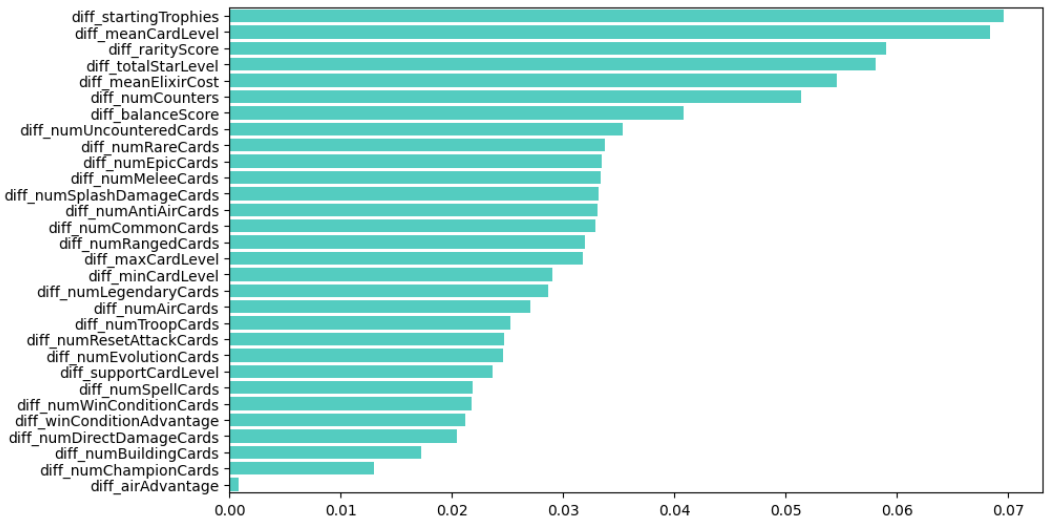

Figura 3.7: Importancia de las variables.

3.4. Preprocesamiento de datos

Durante esta fase se ha utilizado toda la información y las conclusiones obtenidas durante el análisis para implementar diferentes pasos de preprocesamiento que permitan a los algoritmos aprender correctamente y puedan mejorar el rendimiento de los modelos (creación y selección de características, imputación de valores perdidos, codificación de variables categóricas...). Para realizar este proceso de forma adecuada, encapsularemos todas estas transformaciones en *pipelines* de preprocesamiento que nos permitan realizar el proceso de manera lineal. Esto evita fugas de datos y hace que los cambios se apliquen automáticamente de manera sencilla.

Comprender cómo se pueden agrupar las variables predictoras ha sido fundamental para llevar a cabo una mejor toma de decisiones:

- Variables propias de la partida, como el instante temporal y la arena.
- Variables propias de cada jugador:
    - Información propia como el tag, el nombre y los trofeos.
    - Cartas del mazo (binarias).
    - Propiedades del mazo como el nivel medio de las cartas (numéricas).
    - Información de la tropa de las torres como el nombre, la rareza y el nivel.

Se ha decidido crear 5 *pipelines* de preprocesamiento, destinados a ser utilizados por diferentes algoritmos y cuya diferencia principal es la selección de características.

- El primer *pipeline* utiliza tanto las variables binarias como las variables numéricas de cada jugador, por lo que su salida es bastante parecida al conjunto de datos original en cuanto a columnas.
- El segundo *pipeline* también conserva las variables binarias, pero transforma las variables numéricas de cada jugador en las diferencias. Se añaden también nuevas variables como la diferencia en los puntajes.
- El tercer *pipeline* combina los dos anteriores, conservando las variables binarias, las numéricas de cada jugador y añadiendo también las diferencias. Esto permite aprovechar el valor añadido de estas últimas sin perder la información que nos pueden aportar las columnas individuales. Se añaden también las nuevas variables, tanto para cada jugador como las diferencias.
- El cuarto *pipeline* conserva solamente las diferencias (incluyendo las de las nuevas características) y se descartan todas las variables binarias.
- El quinto *pipeline* mantiene únicamente las diferencias entre variables numéricas, y se selecciona solamente un número reducido de ellas en base a conocimiento experto y a lo observado durante el análisis (importancia, correlaciones...).

Esto ha permitido probar diferentes opciones para encontrar un equilibrio entre el rendimiento del modelo y su complejidad. Estos *pipelines* se utilizarán posteriormente en la fase de modelado, donde cada modelo será un nuevo *pipeline* que solamente tendrá dos pasos: [[PIPELINE DE PREPROCESAMIENTO] => ESTIMADOR]

Imputación de valores perdidos

Si bien es cierto que no contamos con valores perdidos en el conjunto de entrenamiento, se ha definido cómo gestionarlos para evitar problemas frente a ejemplos nuevos que pudieran contener información faltante por cualquier motivo. Se ha decidido incluir la imputación como primer paso porque se van a utilizar variables individuales de cada jugador durante la creación de nuevas características. Estas diferencias no pueden calcularse si hay valores perdidos en las variables originales, por lo que se debe imputar antes de crearlas y no después. Por lo tanto, debemos imputar:

- Las variables numéricas necesarias para crear las diferencias.
- Del resto de variables, las que no vayan a ser descartadas posteriormente.

Concretamente, se imputará de la siguiente manera:

- Variables numéricas: Imputación por la media.
- Variables binarias: Imputación por 0 (Se asume que la carta no se utiliza).
- Variables categóricas: Imputación por la moda.

No es necesario imputar el resto, ya que las descartaremos y nunca serán utilizadas.

Creación de atributos

Tras apreciar una simetría en gran parte de los datos durante el análisis exploratorio, se decidió explorar la posibilidad de calcular las diferencias entre las variables numéricas de ambos jugadores. Además, se estudió crear otras variables como los puntos de equilibrio de los mazos, calculando a su vez las diferencias.

Se ha tomado la decisión de crear una nueva variable por cada uno de estos pares a excepción de la ventaja en tropas aéreas frente a cartas con daño aéreo, que ofreció unos resultados extremadamente pobres en cuanto a importancia y una varianza muy baja. Para ello, se han definido las funciones de preprocesamiento necesarias para transformar las columnas.

En esta primera aproximación se ha decidido mantener las variables binarias de las cartas de ambos jugadores tal y como están. Otra opción es realizar las diferencias, aunque no distinguiríamos cuando ningún jugador utiliza una carta y cuando la usan los dos. En futuras iteraciones, podríamos estudiar la posibilidad de combinarlas para que tomen los cuatro posibles valores y distinguir todos los casos.

Selección de variables

Se han eliminado en todos los *pipelines* las siguientes variables:

- Los tags y los nombres de ambos jugadores: Son completamente irrelevantes para el problema que queremos resolver.
- El instante de tiempo de la partida: Todas pertenecen al periodo posterior a los cambios de balance del 9 de abril, por lo que la jugabilidad no cambia.
- La arena y los trofeos (por jugador o diferencia según el *pipeline*): El conjunto de datos contiene partidas en las que dos jugadores han sido emparejados, con una diferencia de trofeos mínima y en la misma arena (a menos que estén justo en el límite y en ese caso se jugaría en la del jugador con más copas). A pesar de ello, nuestro modelo también se quiere utilizar para predecir partidas amistosas entre dos jugadores cualesquiera sin que estos tengan que cumplir estrictamente las condiciones necesarias para ser emparejados por el juego. Si bien es cierto que estas variables pueden afectar al resultado, conservarlas podría empeorar el funcionamiento real del modelo. Al crear un nuevo registro, la diferencia de trofeos podría ser demasiado grande. Lo ideal para mantenerlas sería contar también con partidas amistosas (sin restricciones de nivel ni de modos de juego especiales) en los datos adquiridos en la primera fase del proyecto. Como no es el caso, debemos adaptarnos al problema y tomar las decisiones adecuadas.
- El número de hechizos (por jugador o diferencia según el *pipeline*): Se ha decidido eliminar debido a su alta correlación con el número de cartas de daño directo a torre y con el número de tropas.
- La rareza de la tropa de las torres (para ambos jugadores): Al haber solamente cuatro cartas de soporte, se ha considerado insuficiente el valor que nos puede aportar conocer su rareza además del nombre.

Además, del segundo *pipeline* se han eliminado también las variables numéricas de cada jugador al ser reemplazadas por las diferencias. Lo mismo ocurre con el cuarto pipeline, en el que también se han eliminado todas las variables binarias y el nombre de la tropa de las torres para ambos jugadores.

Por último, del quinto pipeline también se han eliminado algunas diferencias adicionales en base a conocimiento experto o a lo observado durante el análisis, principalmente por alta correlación o baja importancia, para reducir la dimensionalidad.

Discretización

En esta primera aproximación, no se ha discretizado ninguna variable numérica para ningún *pipeline* de preprocesamiento.

Codificación

Debemos codificar las variables categóricas. Para ello se ha aplicado codificación *One-Hot* al nombre de la tropa de las torres para ambos jugadores. Este paso no se ha aplicado en los dos últimos *pipelines*, ya que estas variables no se conservan.

Escalado

Por último, se han estandarizado las variables numéricas correspondientes. Como se van a probar algunos algoritmos que funcionan mejor con escalado y no afecta negativamente el rendimiento de otros para los que no es necesario, lo añadiremos siempre para poder combinar los modelos con todos los *pipelines*.

3.5. Creación de modelos

Durante la fase de modelado, se han entrenado diferentes modelos de aprendizaje automático. Concretamente, se han probado varios algoritmos de clasificación que reciben los datos transformados mediante los *pipelines* de preprocesamiento.

La metodología utilizada ha sido la siguiente:

- En primer lugar, por cada *pipeline* de preprocesamiento se ha creado un modelo extremadamente sencillo que pueda utilizarse como *baseline* para comparar los resultados. Se ha entrenado un árbol con un solo nivel de profundidad, por lo que los modelos más básicos clasificarán utilizando únicamente el criterio que más ganancia de información aporte (Figura 3.8).
- Se han seleccionado diferentes algoritmos de aprendizaje y se han probado con cada uno de los *pipelines* de preprocesamiento.
- Para cada par *algoritmo-pipeline*, se han optimizado los hiperparámetros mediante búsqueda en malla con una validación cruzada de 5 *folds* (optimizando *accuracy* por ser una métrica adecuada para nuestro problema) y se ha utilizado la mejor configuración para entrenar un modelo con todos los datos de entrenamiento. Se ha optado por un tipo de validación que nos permita obtener una buena configuración sin requerir un tiempo de ejecución excesivo, ya que el número de combinaciones es elevado.
- Se ha evaluado el rendimiento de los *baselines* y de los modelos seleccionados sobre entrenamiento, validación y prueba. Para obtener un rendimiento aún más honesto, en lugar de utilizar el rendimiento obtenido con la *5-CV* durante la optimización de hiperparámetros para validar, se ha utilizado una *10x5-CV* como alternativa más robusta. Esto nos permite evaluar de manera más precisa el rendimiento real de nuestros modelos, seleccionando los más eficientes y comprobando también que el rendimiento sobre el conjunto de prueba se mantiene. Finalmente, se ha elegido el mejor modelo teniendo en cuenta factores como el resultado sobre validación y el sobreajuste.

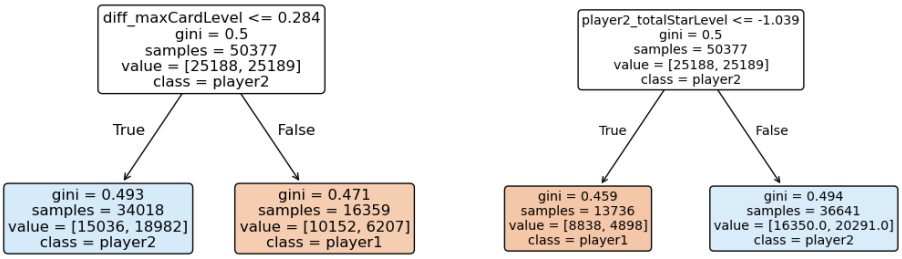

Figura 3.8: Ejemplos de *baselines*.

3.5.1. Validación y selección de modelos

Como se ha mencionado anteriormente, se han probado diferentes algoritmos de clasificación con cada uno de los pipelines de *preprocesamiento*. Para cada una de estas configuraciones, se han optimizado los hiperparámetros mediante búsqueda en malla y se ha entrenado un modelo con la mejor configuración sobre validación cruzada. En este proceso se han seleccionado algunos de los hiperparámetros más relevantes para cada algoritmo (coeficientes para las regresiones, profundidad para los árboles, número de estimadores para los ensembles, etc), aunque no se han probado variantes en los pasos de preprocesamiento. Todo ello ha sumado un total de 1330 modelos, de los que se han seleccionado 45 (la mejor configuración para cada par *algoritmo-pipeline*).

Algunas conclusiones sobre los resultados obtenidos son las siguientes:

- Modelos lineales como Regresión Logística y Regresión Ridge adaptada a clasificación han obtenido un rendimiento superior a otros modelos que utilizan *KNN* o Árboles de decisión, e incluso que algunas configuraciones de ensembles con determinados *pipelines* de preprocesamiento.
- Los árboles con mejor rendimiento sobre validación han sido poco profundos, ya que una mayor expansión sobreajusta.
- La Regresión Logística es el algoritmo de aprendizaje que mejor ha funcionado como estimador base para *Bagging*.
- *Random Forest* mejora ligeramente el rendimiento del resto de modelos, pero no destaca por encima de ellos. Al contrario que a nivel individual, funciona mejor con árboles más profundos.
- Los algoritmos de *Boosting* funcionan mejor con árboles poco profundos.
- El pipeline de preprocesamiento que tiende a ofrecer mejores resultados incluye tanto las diferencias como las variables binarias y numéricas de ambos jugadores.

3.5.2. Evaluación final de modelos

Para realizar la evaluación final y seleccionar el mejor modelo, se ha medido el rendimiento utilizando el *accuracy* como métrica sobre:

- Entrenamiento: El propio conjunto de *train*.
- Validación: Para validar y seleccionar los mejores modelos, se ha utilizado una *10x5-CV* en lugar del valor obtenido en la optimización de hiperparámetros con la *5-CV*. Utilizar una validación cruzada más robusta nos permite validar de manera aún más honesta una vez que ya no contamos con tantas configuraciones. En el caso de los *baselines*, esta ha sido la primera validación.
- Prueba: El conjunto de *test* con ejemplos no observados previamente que habíamos reservado, el cual se ha utilizado a modo informativo.

En esta fase se han evaluado un total de 50 modelos, los 45 compuestos por la mejor configuración para cada par *algoritmo-pipeline* y los 5 *baselines*. En la Tabla 3.1 se pueden apreciar los que mejores y peores resultados han ofrecido sobre validación.

Model	Train Accuracy	Val Accuracy	Test Accuracy
Random Forest (Bin + Num + Diff)	0.994978	0.630036	0.630965
Gradient Boosting (Bin + Num + Diff)	0.654525	0.629686	0.628503
Histogram Gradient Boosting (Bin + Num + Diff)	0.672628	0.629631	0.628980
AdaBoost (Bin + Num + Diff)	0.711475	0.628160	0.625724
Gradient Boosting (Bin + Num)	0.719475	0.628160	0.630171
Histogram Gradient Boosting (Bin + Num)	0.673939	0.626544	0.624375
AdaBoost (Bin + Num)	0.729857	0.625246	0.625645
Random Forest (Bin + Diff)	0.989380	0.624475	0.625963
Logistic Regression (Bin + Num + Diff)	0.629077	0.623354	0.621199
Ridge Classifier (Bin + Num + Diff)	0.628700	0.623149	0.622787
...	...	...	...
KNN (Selected Diff)	0.999742	0.596860	0.592934
Decision Tree (Only Diff)	0.601564	0.595746	0.593251
Decision Tree (Binary + Diff)	0.599202	0.595383	0.592934
Decision Tree (Selected Diff)	0.593406	0.589049	0.591187
Baseline (Binary + Diff)	0.578319	0.578319	0.571417
Baseline (Only Diff)	0.578319	0.578319	0.571417
Baseline (Bin + Num)	0.578220	0.578220	0.579198
Baseline (Bin + Num + Diff)	0.578220	0.577875	0.579198
Baseline (Selected Diff)	0.576255	0.576094	0.572052
KNN (Bin + Num)	1.000000	0.575937	0.580627

Tabla 3.1: Evaluación final de modelos.

En las Figuras 3.9 y 3.10 se muestra el rendimiento medio de los 50 modelos resultantes por *pipeline* de preprocesamiento y algoritmo respectivamente.

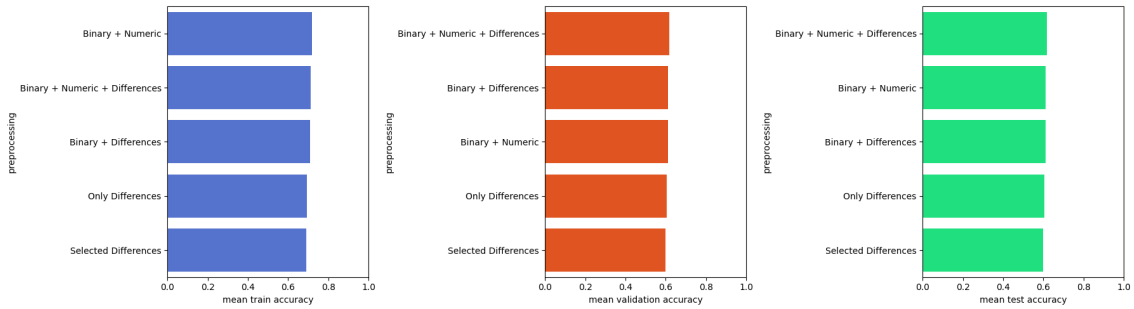

Figura 3.9: Rendimiento medio de los modelos seleccionados por *pipeline* de preprocesamiento.

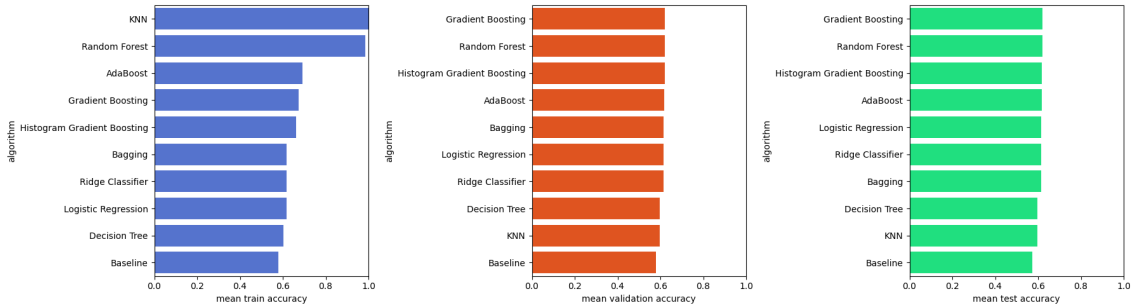

Figura 3.10: Rendimiento medio de los modelos seleccionados por algoritmo de aprendizaje.

Vemos que en el caso de los *pipelines* la diferencia en el rendimiento medio es muy reducida, aunque tendemos a obtener mejores resultados cuando conservamos más características. Por otra parte, de los resultados por algoritmo podemos sacar algunas conclusiones más para cada uno de los gráficos.

Sobre el conjunto de entrenamiento:

- Los modelos que más se ajustan a los datos son *KNN* (que realmente no aprende de ellos) y *Random Forest* (las mejores configuraciones utilizaban árboles sin límite de profundidad).
- Los ensembles de *Boosting* ofrecen resultados similares entre sí.

- Los árboles de decisión no se ajustan tanto a los datos porque las mejores configuraciones no eran árboles profundos.
- Los árboles con un solo nivel de profundidad utilizados como base son capaces de explicar cerca del 58% de los datos de entrenamiento.

Tras validar con una *10x5-CV*:

- *Random Forest* sobreajusta.
- La diferencia en el rendimiento entre algoritmos es muy pequeña, aunque el orden es similar (a excepción de *KNN*, que es basado en instancias y simplemente memoriza los datos de entrenamiento).
- Los ensembles están ligeramente por encima de los clasificadores individuales.
- Tampoco existe una gran diferencia respecto a los modelos base, lo que nos indica que estamos ante un problema complejo con un margen de mejora bastante reducido (al menos con los datos actuales).

Sobre el conjunto de prueba:

- Los resultados son muy similares a los obtenidos mediante la validación.

El modelo elegido ha sido el que aparece segundo en la Tabla 3.1, ya que obtiene un rendimiento elevado sobre validación sin sobreajustar en exceso. La matriz de confusión y la curva ROC sobre el conjunto de prueba se puede apreciar en la Figura 3.11.

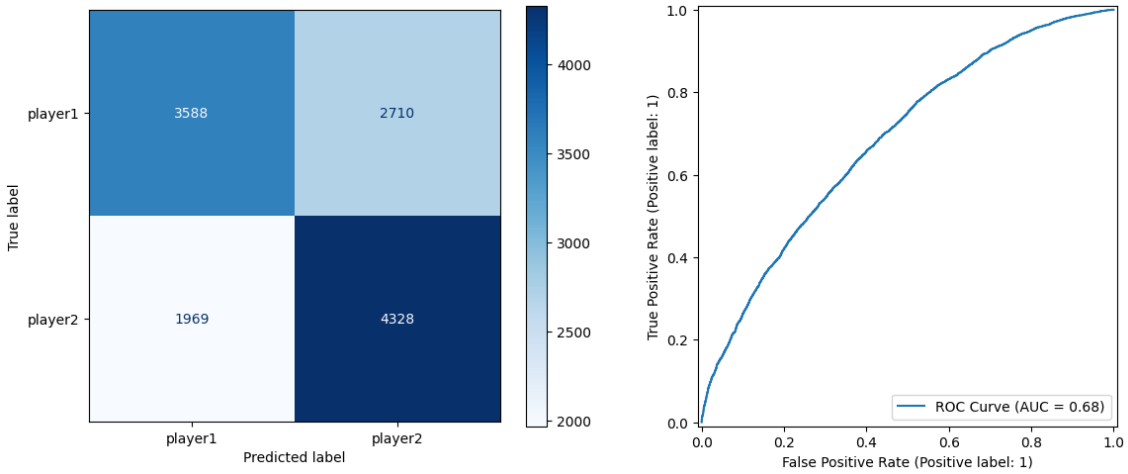

Figura 3.11: Matriz de confusión y Curva ROC del modelo final sobre el conjunto de prueba.

Además, se ha observado que la diferencia en las probabilidades de victoria de los jugadores tiende a ser reducida, lo que indica que el modelo encuentra dificultades para clasificar con claridad gran parte de los enfrentamientos y sugiere que estos están muy empatados. Concretamente, la diferencia es menor al 4% en 1 de cada 10 partidas del conjunto de prueba (el 10% de las predicciones están más ajustadas que un [0.48, 0.52] o viceversa). Esta nunca alcanza el 90%, y es menor que el 72% en el 99% de los casos. La media es 0.24 y la mediana 0.20, lo que indica que en la mitad de los casos se obtienen probabilidades más ajustadas que un [0.4, 0.6] o viceversa. Esto nos da a entender que en este caso existe bastante incertidumbre en la predicción y que las clases están muy equilibradas por lo que es complejo encontrar patrones claros en los datos. El rendimiento obtenido y la cercanía en las probabilidades son señales de que nos encontramos ante un problema difícil de resolver, al menos cuando solamente utilizamos la información de los mazos. Esto tiene sentido, ya que todos los casos de nuestro conjunto de datos surgen de un emparejamiento dentro del juego para enfrentar a dos jugadores que a priori disputarán una partida sin un claro ganador.

Respecto a los resultados, podemos concluir lo siguiente:

- Nos encontramos ante un problema complejo, en que el hemos logrado alcanzar una tasa de acierto alrededor del 63% sobre validación y *test* sin un gran margen de mejora respecto a los *baselines*.
- Las variables más importantes parecen ser las relacionadas con los niveles.
- Los ensembles han ofrecido mejores resultados, aunque la diferencia con los clasificadores individuales es muy reducida. Sorprende que los modelos lineales hayan tenido un rendimiento tan cercano a ellos.
- Se ha producido sobreajuste en algunos casos, aunque para *Random Forest* ha sido excesivo.
- Existe mucha incertidumbre en las predicciones, cuyas probabilidades son cercanas para una gran cantidad de ejemplos del conjunto de prueba.
- Si no trabajamos con probabilidades cuando utilicemos el modelo, podemos incrementar ligeramente el umbral de clasificación para nivelar el número de aciertos por jugador (sobre *test* se predice algo más el segundo jugador por los patrones encontrados en los datos).

La principal conclusión que sacamos tras esta primera aproximación es que el sistema de *matchmaking* del juego es correcto, ya que todas las partidas con las que hemos trabajado representan un enfrentamiento entre dos jugadores emparejados por el juego sin un ganador claro. La incertidumbre y la dificultad para encontrar patrones sólidos puede deberse a que hay tantos factores a tener en cuenta que al final la partida termina nivelándose. Si hubiéramos logrado obtener un mejor modelo, esto podría llegar a significar que este sistema está, en cierto modo, amañado.

3.6. Explicabilidad

La interpretabilidad de las predicciones es un componente esencial en proyectos de ciencia de datos, ya que en muchos casos tomamos decisiones importantes a través de un modelo "de caja negra" y necesitamos comprender el por qué de los resultados obtenidos para generar confianza en el sistema. Se han explorado diferentes técnicas de explicabilidad aplicadas al modelo seleccionado tras la fase de modelado, tanto desde una perspectiva global (que busca comprender el comportamiento general del modelo) como local (centrada en cómo se generan predicciones para casos concretos).

En el contexto del proyecto, podemos preguntarnos:

- ¿Cómo influyen los diferentes factores en el desenlace de una partida?
- ¿Qué variables favorecen la victoria de un jugador? ¿Y la del otro?
- ¿Cuáles de estas pesan más y hacia qué lado se inclina la balanza?
- ¿Qué cambios serían necesarios para que, según el modelo, gane el otro jugador?

Para entender mejor el comportamiento del modelo a nivel global, utilizaremos el conjunto de prueba que habíamos reservado para realizar la evaluación final. En la Figura 3.12 podemos observar un resumen de las variables más influyentes obtenido mediante *SHapley Additive exPlanations* (SHAP).

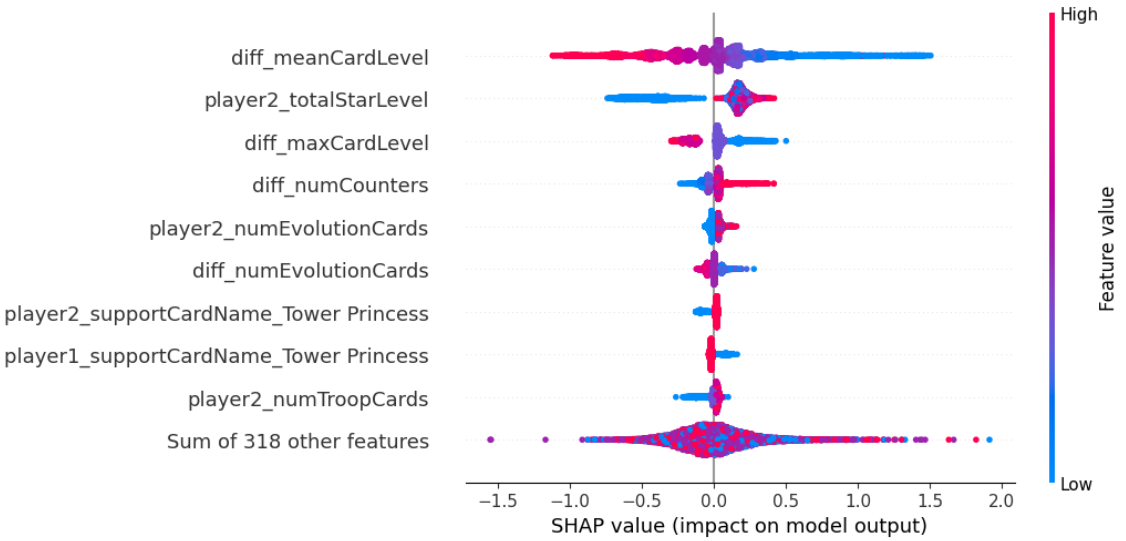

Figura 3.12: Variables más influyentes (SHAP).

Vemos que claramente la diferencia en el nivel de las cartas es el factor más determinante. Además, observamos una tendencia que se repite en el resto de diferencias. Lo que ocurre es que cuando la diferencia es menor (azul), significa que el segundo jugador tiene mayor nivel de cartas que el rival y tiende a ganar (valor SHAP positivo porque la clase positiva es *player2*). Cuando la diferencia es mayor (fucsia), significa que el primer jugador tiene mayor nivel de cartas que el rival y tiende a ganar (valor SHAP negativo). Con los *counters* ocurre lo mismo pero al revés.

También se han obtenido gráficos de dependencia con SHAP para las variables más influyentes (Figura 3.13). En ellos se puede apreciar claramente que son muy discriminativas, especialmente la diferencia en el nivel medio de las cartas.

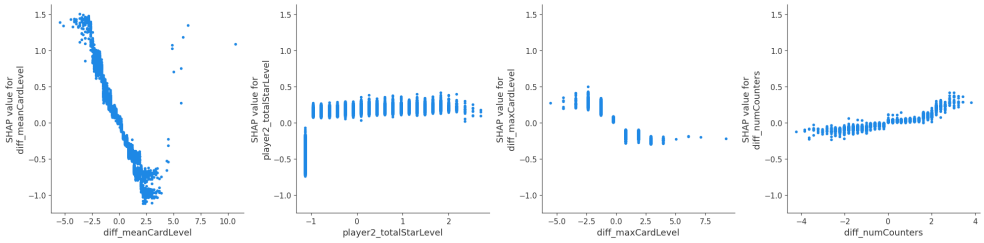

Figura 3.13: Dependencia de las variables más influyentes (SHAP).

Los gráficos de dependencia parcial (PDP) para estos pares de variables (Figura 3.14) nos permiten comprender mejor cómo las relaciones influyen en las predicciones. Por ejemplo, observamos que el hecho de que el modelo tienda a predecir como ganador un jugador con un nivel medio de las cartas muy superior es todavía más habitual cuando el nivel máximo también le favorece o cuando el número de *counters* es menor.

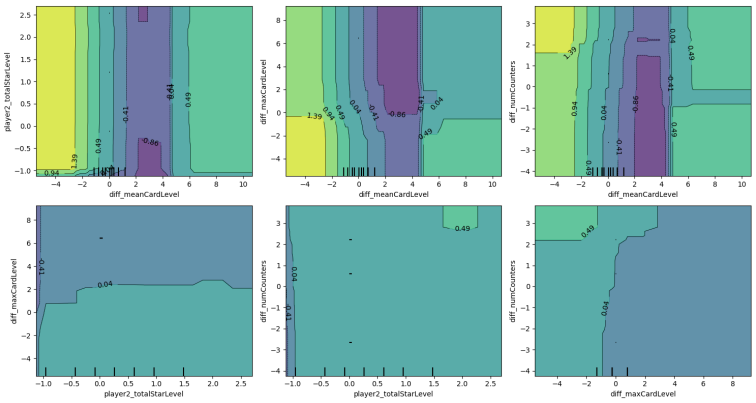

Figura 3.14: Dependencia parcial entre pares de variables (PDP).

Por último, se han explicado predicciones sobre muestras individuales del conjunto de prueba mediante gráficos de cascada en los que podemos ver la fuerza que ejercen las variables a favor de un jugador u otro. En el caso de la Figura 3.15, el modelo predice que gana el segundo jugador con una probabilidad de 0.5075, por lo que los valores están muy ajustados en ambos sentidos.

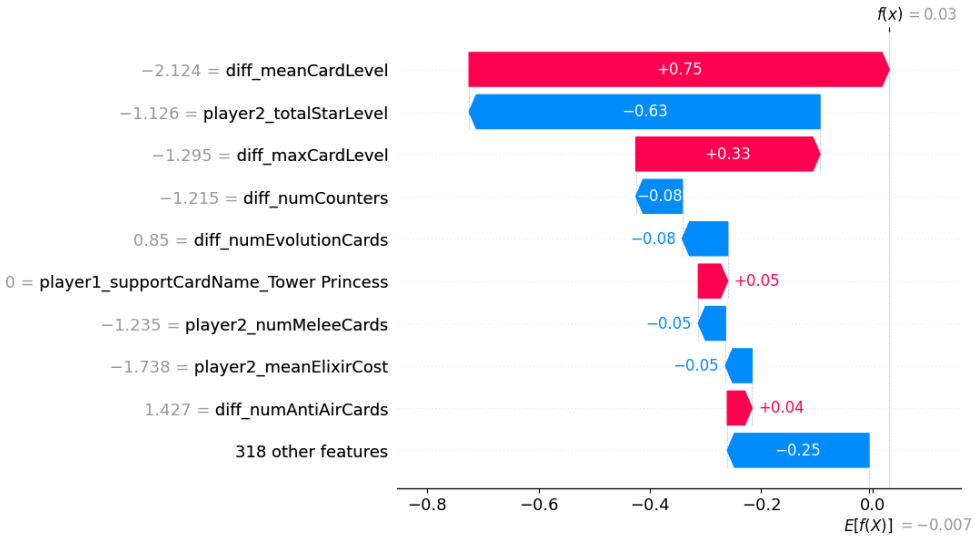

Figura 3.15: Interpretación de una predicción individual (SHAP).

Como futuras mejoras, se podría realizar un análisis más profundo o incluir otras técnicas como los contrafactuales, que nos permiten conocer qué cambios tendrían que producirse para que el resultado fuese distinto.

3.7. Desarrollo de la aplicación

Como fase de despliegue, se ha desarrollado una aplicación que nos permite simular partidas entre dos jugadores cualesquiera (Figura 3.16). El usuario introduce el tag de ambos jugadores y se cargan sus perfiles a través de la API, mostrando información como el nivel de experiencia o los trofeos actuales (Figura 3.17). El usuario debe seleccionar las cartas que va a utilizar cada jugador, tanto las ocho del mazo como la tropa de las torres. La colección de cartas de cada jugador puede ser diferente, ya que esta información se obtiene con la API y por lo tanto es posible distinguir niveles, evoluciones, cartas sin desbloquear... (Figura 3.18). Una vez seleccionadas, se crea un nuevo registro con el formato adecuado a partir de los perfiles y los mazos para predecir la probabilidad de victoria de cada jugador y los factores que más la favorecen mediante la integración del modelo y las técnicas de explicabilidad (Figura 3.19).

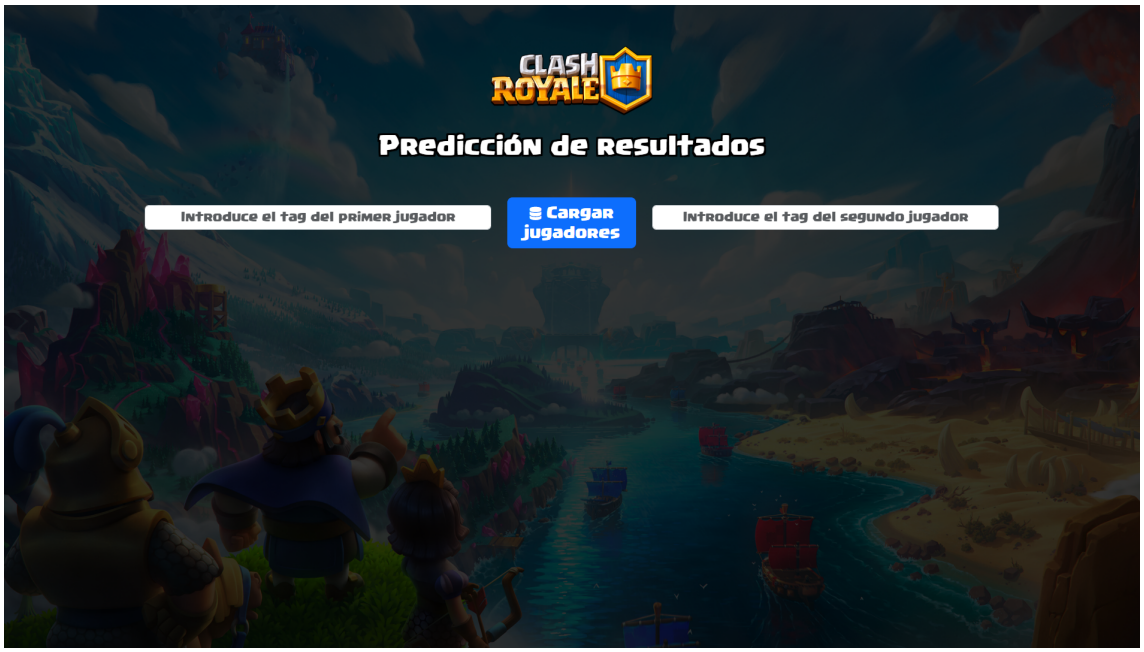

Figura 3.16: Interfaz principal de la aplicación.



Figura 3.17: Carga de jugadores en la aplicación.

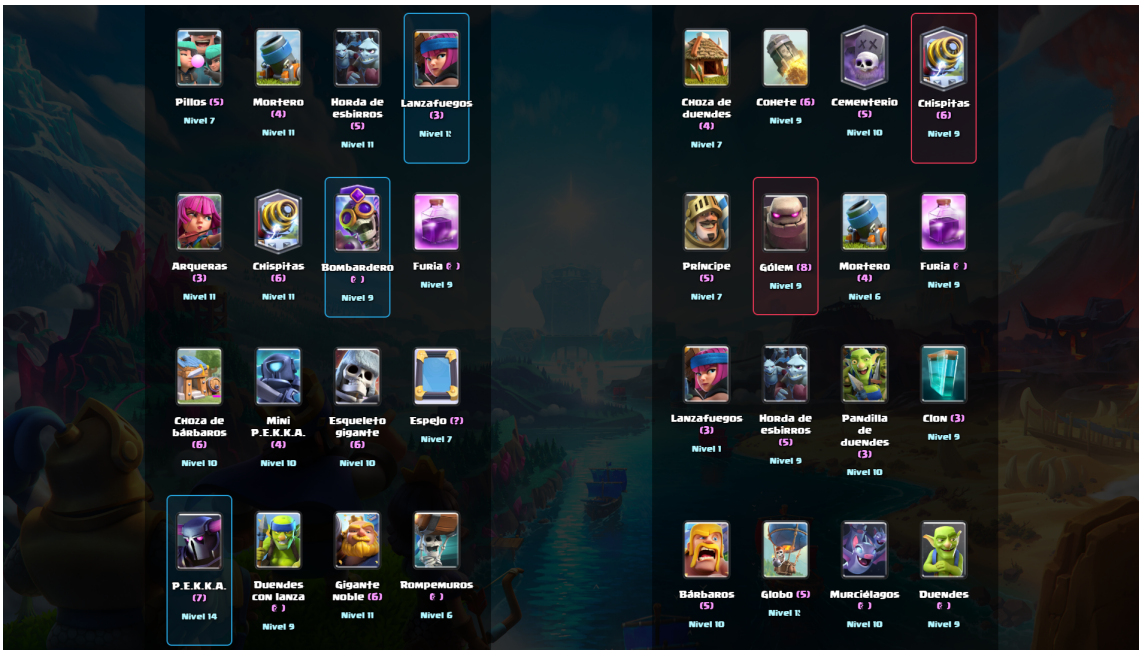

Figura 3.18: Selección de mazos en la aplicación.



Figura 3.19: Predicción de resultados en la aplicación.

4. Conclusiones del proyecto

Tras la realización del proyecto se han extraído las conclusiones sobre el mismo, así como las competencias desarrolladas y una serie de posibles mejoras de cara al futuro.

4.1. Conclusiones y futuras mejoras

Las conclusiones y las lecciones aprendidas sobre el proyecto son las siguientes:

- Se ha obtenido información de cartas y partidas a través de la API oficial de Clash Royale. En este proceso ha sido necesario definir claramente el problema a resolver, comprender profundamente los datos, complementarlos con información obtenida de fuentes externas y tener en cuenta aspectos como las posibles fugas de datos debidas a desfases temporales.
- A partir de los datos crudos, se ha creado un conjunto de datos alineado con el objetivo del proyecto en el que cada fila corresponde a una observación (información sobre la partida). Para ello, se han llevado a cabo tareas de limpieza y transformación sobre las cartas y las partidas. Se ha dividido el conjunto de datos resultante en entrenamiento y prueba.
- Se ha realizado un análisis exploratorio de los datos de entrenamiento para estudiar la distribución de las variables y las relaciones entre ellas. También se ha realizado un proceso de ingeniería de características en el que se ha explorado la posibilidad de crear nuevos atributos y se han aplicado diferentes técnicas como medir la varianza y la importancia de las variables para tomar mejores decisiones de cara al preprocesamiento de los datos.
- Se han construido diferentes *pipelines* de preprocesamiento cuya diferencia principal es la selección de características. Esto permite evaluar el rendimiento de un mismo algoritmo de aprendizaje con distintas variantes del conjunto de datos.

- Se ha evaluado de manera honesta el rendimiento de una gran cantidad de modelos de aprendizaje automático, optimizando sus hiperparámetros y seleccionando las mejores configuraciones. El modelo final se ha integrado en una aplicación web que permite predecir nuevas partidas y comprender qué factores favorecen la victoria de cada jugador mediante técnicas de explicabilidad.
- El proceso se ha diseñado para ser en cierto modo reproducible, escalable y fácilmente adaptable a nuevos datos o escenarios, permitiendo iterar rápidamente sobre nuevas hipótesis o mejoras.
- Las conclusiones sobre los resultados mencionadas al final de la Subsección 3.5.2, destacando la dificultad para encontrar patrones sólidos en los datos y la capacidad del sistema de *matchmaking* del juego para crear enfrentamientos igualados.

Algunas futuras mejoras son las siguientes:

- Realizar iteraciones adicionales de la metodología CRISP-DM para mejorar el rendimiento de los modelos obtenidos.
- Introducir nuevas variantes en el preprocesamiento (discretización, combinación de las variables binarias, creación de otras características...).
- Aumentar el tamaño del conjunto de datos de entrenamiento. Esto podría hacer que las variables comunes a ambos jugadores tuvieran un comportamiento más parecido entre ellas y las diferencias ganaran importancia.
- Complementar la información de los mazos con otros datos sobre los jugadores (nivel de experiencia, récord de trofeos, tasa de victorias, maestría de las cartas utilizadas, etc.). Esta opción se planteó al inicio del proyecto, pero no se llevó a cabo porque existe un desfase temporal entre el momento en el que se jugó la partida y el perfil actual de los jugadores obtenido a través de la API. Para ello, sería necesario asumir un margen de error o recopilar información en tiempo real.
- Diseñar un flujo automatizado que permita extraer y procesar información en tiempo real para facilitar el mantenimiento de los modelos de cara a nuevas actualizaciones, reduciendo el impacto de nuevas cartas y cambios de balance.
- Considerar el uso de otros modelos como *XGBoost* o redes neuronales.
- Ampliar el problema e implementar modelos capaces de resolver tareas más complejas como predicción de resultados en partidas del *Camino de Leyendas* y otros modos de juego competitivos donde se iguala el nivel de las cartas.
- Utilizar los datos disponibles para desarrollar otras funcionalidades similares a las que ofrece *RoyaleAPI* que sean independientes de la herramienta predictiva.
- Incorporar un sistema para obtener *feedback* y sugerencias de los usuarios.

4.2. Competencias desarrolladas

A lo largo del trabajo se han desarrollado diferentes competencias relacionadas con la ciencia de datos y, en general, con las ciencias de la computación:

- Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.
- 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 de forma computable para la resolución de problemas mediante un sistema informático en cualquier ámbito de aplicación, particularmente los relacionados con aspectos de computación y actuación en 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.

Si nos enfocamos específicamente en este proyecto, podemos añadir las competencias particulares que aparecen a continuación.

- Aprender a adquirir, consultar y procesar datos de manera eficiente utilizando la API oficial de *Clash Royale*. Emplear el conocimiento del dominio para extraer la información relevante de los datos crudos, realizando tareas de limpieza y transformación para poder utilizar los datos en etapas posteriores del proyecto.
- Llevar a cabo un análisis exploratorio de los datos para visualizar las distribuciones de las variables y las relaciones entre ellas. Estudiar también la existencia de *outliers* y valores perdidos para su posterior manejo, así como la posibilidad de crear nuevas características que puedan aportar valor adicional.
- Utilizar *Scikit-learn* para implementar modelos de aprendizaje automático y aplicar técnicas de preprocesamiento como codificación, creación y selección de características, empleando buenas prácticas y evitando fugas de datos.
- Estudiar las métricas más adecuadas para el problema en cuestión, optimizando los hiperparámetros de los modelos y evaluando su rendimiento de manera honesta mediante técnicas de selección como validación cruzada.
- Desarrollar una aplicación interactiva que integre el mejor modelo y permita predecir nuevas partidas a partir de los perfiles y los mazos de los jugadores.
- Comprender el funcionamiento y la importancia de las técnicas de explicabilidad en aprendizaje automático, incorporándolas a la aplicación para proporcionar información sobre cómo los distintos elementos del juego afectan a las predicciones.

A. Conjuntos de datos

En este capítulo se describen los conjuntos de datos utilizados durante el proyecto.

Tabla A.1: Descripción del conjunto de datos de cartas.

Variable	Descripción
id	Identificador único de la carta
name	Nombre de la carta
rarity	Rareza de la carta
type	Tipo de carta (tropa, hechizo o edificio)
elixirCost	Coste de elixir para jugar la carta
maxLevel	Nivel máximo alcanzable de la carta
hasEvolution	Indica si la carta tiene evolución disponible
winCondition	Indica si la carta es una condición de victoria
melee	Indica si la carta tiene ataque cuerpo a cuerpo
ranged	Indica si la carta tiene ataque a distancia
air	Indica si la carta es una tropa aérea
antiAir	Indica si la carta causa daño a unidades aéreas
directDamage	Indica si la carta causa daño directo a torre
splashDamage	Indica si la carta causa daño de salpicadura
resetAttack	Indica si la carta reinicia el ataque de cartas rivales
counters	Nombres de las cartas que contrarrestan la carta
iconUrls_medium	URL del icono de la carta
iconUrls_evolutionMedium	URL del icono de la evolución de la carta (si aplica)

Tabla A.2: Descripción del conjunto de datos de cartas de soporte.

Variable	Descripción
id	Identificador único de la carta de soporte
name	Nombre de la carta de soporte
maxLevel	Nivel máximo alcanzable de la carta soporte
rarity	Rareza de la carta de soporte
iconUrls_medium	URL del icono de la carta de soporte

Tabla A.3: Descripción del conjunto de datos de partidas.

Variable	Descripción
winner	Indica el ganador de la partida mediante la etiqueta `player1` o `player2`
battleTime	Instante de tiempo en el que se disputó la partida
arena	Nombre de la arena en la que se disputó la partida
player1_tag	Tag único del primer jugador
player1_name	Nombre del primer jugador
player1_startingTrophies	Número de trofeos con los que el primer jugador comenzó la partida
player1_hasKnight	El primer jugador utiliza Caballero
player1_hasArchers	El primer jugador utiliza Arqueras
player1_hasGoblins	El primer jugador utiliza Duendes
player1_hasGiant	El primer jugador utiliza Gigante
player1_hasP.E.K.K.A	El primer jugador utiliza P.E.K.K.A.
player1_hasMinions	El primer jugador utiliza Esbirros
player1_hasBalloon	El primer jugador utiliza Globo bombástico
player1_hasWitch	El primer jugador utiliza Bruja
player1_hasBarbarians	El primer jugador utiliza Bárbaros
player1_hasGolem	El primer jugador utiliza Gólem
player1_hasSkeletons	El primer jugador utiliza Esqueletos
player1_hasValkyrie	El primer jugador utiliza Valquiria
player1_hasSkeletonArmy	El primer jugador utiliza Ejército de esqueletos
player1_hasBomber	El primer jugador utiliza Bombardero
player1_hasMusketeer	El primer jugador utiliza Mosquetera
player1_hasBabyDragon	El primer jugador utiliza Bebé dragón
player1_hasPrince	El primer jugador utiliza Príncipe
player1_hasWizard	El primer jugador utiliza Mago
player1_hasMiniP.E.K.K.A	El primer jugador utiliza Mini P.E.K.K.A.
player1_hasSpearGoblins	El primer jugador utiliza Duendes con lanza
player1_hasGiantSkeleton	El primer jugador utiliza Esqueleto gigante
player1_hasHogRider	El primer jugador utiliza Montapuercos

Continúa en la siguiente página

Variable	Descripción
player1_hasMinionHorde	El primer jugador utiliza Horda de esbirros
player1_hasIceWizard	El primer jugador utiliza Mago de hielo
player1_hasRoyalGiant	El primer jugador utiliza Gigante noble
player1_hasGuards	El primer jugador utiliza Guardias
player1_hasPrincess	El primer jugador utiliza Princesa
player1_hasDarkPrince	El primer jugador utiliza Príncipe oscuro
player1_hasThreeMusketeers	El primer jugador utiliza Trío de mosqueteras
player1_hasLavaHound	El primer jugador utiliza Sabueso de lava
player1_hasIceSpirit	El primer jugador utiliza Espíritu de hielo
player1_hasFireSpirit	El primer jugador utiliza Espíritu de fuego
player1_hasMiner	El primer jugador utiliza Minero
player1_hasSparky	El primer jugador utiliza Chispitas
player1_hasBowler	El primer jugador utiliza Montacarneros
player1_hasLumberjack	El primer jugador utiliza Leñador
player1_hasBattleRam	El primer jugador utiliza Ariete de batalla
player1_hasInfernoDragon	El primer jugador utiliza Dragón infernal
player1_hasIceGolem	El primer jugador utiliza Gólem de hielo
player1_hasMegaMinion	El primer jugador utiliza Megaesbirro
player1_hasDartGoblin	El primer jugador utiliza Duende lanzadardos
player1_hasGoblinGang	El primer jugador utiliza Pandilla de duendes
player1_hasElectroWizard	El primer jugador utiliza Mago eléctrico
player1_hasEliteBarbarians	El primer jugador utiliza Bárbaros de élite
player1_hasHunter	El primer jugador utiliza Cazador
player1_hasExecutioner	El primer jugador utiliza Verdugo
player1_hasBandit	El primer jugador utiliza Bandida
player1_hasRoyalRecruits	El primer jugador utiliza Reclutas reales
player1_hasNightWitch	El primer jugador utiliza Bruja nocturna
player1_hasBats	El primer jugador utiliza Murciélagos
player1_hasRoyalGhost	El primer jugador utiliza Fantasma real
player1_hasRamRider	El primer jugador utiliza Montacarneros
player1_hasZappies	El primer jugador utiliza Electrocutadores
player1_hasRascals	El primer jugador utiliza Pillos
player1_hasCannonCart	El primer jugador utiliza Cañón con ruedas
player1_hasMegaKnight	El primer jugador utiliza Megacaballero
player1_hasSkeletonBarrel	El primer jugador utiliza Barril de esqueletos
player1_hasFlyingMachine	El primer jugador utiliza Máquina voladora
player1_hasWallBreakers	El primer jugador utiliza Rompemuros
player1_hasRoyalHogs	El primer jugador utiliza Puercos reales
player1_hasGoblinGiant	El primer jugador utiliza Duende gigante
player1_hasFisherman	El primer jugador utiliza Pescador
player1_hasMagicArcher	El primer jugador utiliza Arquero mágico
player1_hasElectroDragon	El primer jugador utiliza Dragón eléctrico

Continúa en la siguiente página

Variable	Descripción
player1_hasFirecracker	El primer jugador utiliza Lanzafuegos
player1_hasMightyMiner	El primer jugador utiliza Gran minero
player1_hasBattleHealer	El primer jugador utiliza Curandera guerrera
player1_hasSkeletonKing	El primer jugador utiliza Rey esqueleto
player1_hasArcherQueen	El primer jugador utiliza Reina arquera
player1_hasGoldenKnight	El primer jugador utiliza Caballero dorado
player1_hasMonk	El primer jugador utiliza Monje
player1_hasSkeletonDragons	El primer jugador utiliza Dragones esqueleto
player1_hasMotherWitch	El primer jugador utiliza Bruja madre
player1_hasElectroSpirit	El primer jugador utiliza Espíritu eléctrico
player1_hasElectroGiant	El primer jugador utiliza Gigante eléctrico
player1_hasPhoenix	El primer jugador utiliza Fénix
player1_hasLittlePrince	El primer jugador utiliza Principito
player1_hasGoblinDemolisher	El primer jugador utiliza Demoledor duende
player1_hasGoblinMachine	El primer jugador utiliza Máquina duende
player1_hasSuspiciousBush	El primer jugador utiliza Arbusto sospechoso
player1_hasGoblinstein	El primer jugador utiliza Goblinstein
player1_hasRuneGiant	El primer jugador utiliza Gigante rúnica
player1_hasBerserker	El primer jugador utiliza Berserker
player1_hasBossBandit	El primer jugador utiliza Bandida líder
player1_hasCannon	El primer jugador utiliza Cañón
player1_hasGoblinHut	El primer jugador utiliza Choza de duendes
player1_hasMortar	El primer jugador utiliza Mortero
player1_hasInfernoTower	El primer jugador utiliza Torre infernal
player1_hasBombTower	El primer jugador utiliza Torre bombardera
player1_hasBarbarianHut	El primer jugador utiliza Choza de bárbaros
player1_hasTesla	El primer jugador utiliza Torre tesla
player1_hasElixirCollector	El primer jugador utiliza Recolector de elixir
player1_hasX-Bow	El primer jugador utiliza Ballesta
player1_hasTombstone	El primer jugador utiliza Lápida
player1_hasFurnace	El primer jugador utiliza Horno
player1_hasGoblinCage	El primer jugador utiliza Jaula del forzudo
player1_hasGoblinDrill	El primer jugador utiliza Excavadora de duendes
player1_hasFireball	El primer jugador utiliza Bola de fuego
player1_hasArrows	El primer jugador utiliza Flechas
player1_hasRage	El primer jugador utiliza Furia
player1_hasRocket	El primer jugador utiliza Cohete
player1_hasGoblinBarrel	El primer jugador utiliza Barril de duendes
player1_hasFreeze	El primer jugador utiliza Hielo
player1_hasMirror	El primer jugador utiliza Espejo
player1_hasLightning	El primer jugador utiliza Rayo
player1_hasZap	El primer jugador utiliza Descarga

Continúa en la siguiente página

Variable	Descripción
player1_hasPoison	El primer jugador utiliza Veneno
player1_hasGraveyard	El primer jugador utiliza Cementerio
player1_hasTheLog	El primer jugador utiliza El Tronco
player1_hasTornado	El primer jugador utiliza Tornado
player1_hasClone	El primer jugador utiliza Clon
player1_hasEarthquake	El primer jugador utiliza Terremoto
player1_hasBarbarianBarrel	El primer jugador utiliza Barril de bárbaro
player1_hasHealSpirit	El primer jugador utiliza Espíritu sanador
player1_hasGiantSnowball	El primer jugador utiliza Bola de nieve
player1_hasRoyalDelivery	El primer jugador utiliza Paquete real
player1_hasVoid	El primer jugador utiliza Vacío
player1_hasGoblinCurse	El primer jugador utiliza Maldición duende
player1_meanCardLevel	Nivel medio de las cartas del mazo del primer jugador
player1_minCardLevel	Nivel mínimo de las cartas del mazo del primer jugador
player1_maxCardLevel	Nivel máximo de las cartas del mazo del primer jugador
player1_totalStarLevel	Nivel estelar total de las cartas del mazo del primer jugador
player1_meanElixirCost	Coste de elixir del mazo del primer jugador
player1_numEvolutionCards	Número de cartas con evolución en el mazo del primer jugador (debe tener la evolución)
player1_numWinConditionCards	Número de cartas consideradas *Win Condition* en el mazo del primer jugador
player1_numMeleeCards	Número de cartas cuerpo a cuerpo en el mazo del primer jugador
player1_numRangedCards	Número de cartas a distancia en el mazo del primer jugador
player1_numAirCards	Número de unidades aéreas en el mazo del primer jugador
player1_numAntiAirCards	Número de cartas con daño a unidades aéreas en el mazo del primer jugador
player1_numDirectDamageCards	Número de cartas con daño directo a torre en el mazo del primer jugador
player1_numSplashDamageCards	Número de cartas con daño de salpicadura en el mazo del primer jugador
player1_numResetAttackCards	Número de cartas con reseteo de ataque en el mazo del primer jugador
player1_numCommonCards	Número de cartas comunes en el mazo del primer jugador

Continúa en la siguiente página

Variable	Descripción
player1_numRareCards	Número de cartas raras en el mazo del primer jugador
player1_numEpicCards	Número de cartas épicas en el mazo del primer jugador
player1_numLegendaryCards	Número de cartas legendarias en el mazo del primer jugador
player1_numChampionCards	Número de campeones en el mazo del primer jugador
player1_numTroopCards	Número de tropas en el mazo del primer jugador
player1_numBuildingCards	Número de edificios en el mazo del primer jugador
player1_numSpellCards	Número de hechizos en el mazo del primer jugador
player1_numCounters	Número de *counters* altamente efectivos en el mazo rival del primer jugador
player1_numUncounteredCards	Número de cartas del mazo del primer jugador sin *counters* efectivos en el mazo rival
player1_supportCardName	Nombre de la tropa de las torres de coronas utilizada por el primer jugador
player1_supportCardLevel	Nivel de la tropa de las torres de coronas utilizada por el primer jugador
player1_supportCardRarity	Rareza de la tropa de las torres de coronas utilizada por el primer jugador
player2_tag	Tag único del segundo jugador
player2_name	Nombre del segundo jugador
player2_startingTrophies	Número de trofeos con los que el segundo jugador comenzó la partida
player2_hasKnight	El segundo jugador utiliza Caballero
player2_hasArchers	El segundo jugador utiliza Arqueras
player2_hasGoblins	El segundo jugador utiliza Duendes
player2_hasGiant	El segundo jugador utiliza Gigante
player2_hasP.E.K.K.A	El segundo jugador utiliza P.E.K.K.A.
player2_hasMinions	El segundo jugador utiliza Esbirros
player2_hasBalloon	El segundo jugador utiliza Globo bombástico
player2_hasWitch	El segundo jugador utiliza Bruja
player2_hasBarbarians	El segundo jugador utiliza Bárbaros
player2_hasGolem	El segundo jugador utiliza Gólem
player2_hasSkeletons	El segundo jugador utiliza Esqueletos
player2_hasValkyrie	El segundo jugador utiliza Valquiria
player2_hasSkeletonArmy	El segundo jugador utiliza Ejército de esqueletos
player2_hasBomber	El segundo jugador utiliza Bombardero
player2_hasMusketeer	El segundo jugador utiliza Mosquetera
player2_hasBabyDragon	El segundo jugador utiliza Bebé dragón
player2_hasPrince	El segundo jugador utiliza Príncipe

Continúa en la siguiente página

Variable	Descripción
player2_hasWizard	El segundo jugador utiliza Mago
player2_hasMiniP.E.K.K.A	El segundo jugador utiliza Mini P.E.K.K.A.
player2_hasSpearGoblins	El segundo jugador utiliza Duendes con lanza
player2_hasGiantSkeleton	El segundo jugador utiliza Esqueleto gigante
player2_hasHogRider	El segundo jugador utiliza Montapuercos
player2_hasMinionHorde	El segundo jugador utiliza Horda de esbirros
player2_hasIceWizard	El segundo jugador utiliza Mago de hielo
player2_hasRoyalGiant	El segundo jugador utiliza Gigante noble
player2_hasGuards	El segundo jugador utiliza Guardias
player2_hasPrincess	El segundo jugador utiliza Princesa
player2_hasDarkPrince	El segundo jugador utiliza Príncipe oscuro
player2_hasThreeMusketeers	El segundo jugador utiliza Trío de mosqueteras
player2_hasLavaHound	El segundo jugador utiliza Sabueso de lava
player2_hasIceSpirit	El segundo jugador utiliza Espíritu de hielo
player2_hasFireSpirit	El segundo jugador utiliza Espíritu de fuego
player2_hasMiner	El segundo jugador utiliza Minero
player2_hasSparky	El segundo jugador utiliza Chispitas
player2_hasBowler	El segundo jugador utiliza Montacarneros
player2_hasLumberjack	El segundo jugador utiliza Leñador
player2_hasBattleRam	El segundo jugador utiliza Ariete de batalla
player2_hasInfernoDragon	El segundo jugador utiliza Dragón infernal
player2_hasIceGolem	El segundo jugador utiliza Gólem de hielo
player2_hasMegaMinion	El segundo jugador utiliza Megaesbirro
player2_hasDartGoblin	El segundo jugador utiliza Duende lanzadardos
player2_hasGoblinGang	El segundo jugador utiliza Pandilla de duendes
player2_hasElectroWizard	El segundo jugador utiliza Mago eléctrico
player2_hasEliteBarbarians	El segundo jugador utiliza Bárbaros de élite
player2_hasHunter	El segundo jugador utiliza Cazador
player2_hasExecutioner	El segundo jugador utiliza Verdugo
player2_hasBandit	El segundo jugador utiliza Bandida
player2_hasRoyalRecruits	El segundo jugador utiliza Reclutas reales
player2_hasNightWitch	El segundo jugador utiliza Bruja nocturna
player2_hasBats	El segundo jugador utiliza Murciélagos
player2_hasRoyalGhost	El segundo jugador utiliza Fantasma real
player2_hasRamRider	El segundo jugador utiliza Montacarneros
player2_hasZappies	El segundo jugador utiliza Electrocutadores
player2_hasRascals	El segundo jugador utiliza Pillos
player2_hasCannonCart	El segundo jugador utiliza Cañón con ruedas
player2_hasMegaKnight	El segundo jugador utiliza Megacaballero
player2_hasSkeletonBarrel	El segundo jugador utiliza Barril de esqueletos
player2_hasFlyingMachine	El segundo jugador utiliza Máquina voladora
player2_hasWallBreakers	El segundo jugador utiliza Rompemuros

Continúa en la siguiente página

Variable	Descripción
player2_hasRoyalHogs	El segundo jugador utiliza Puercos reales
player2_hasGoblinGiant	El segundo jugador utiliza Duende gigante
player2_hasFisherman	El segundo jugador utiliza Pescador
player2_hasMagicArcher	El segundo jugador utiliza Arquero mágico
player2_hasElectroDragon	El segundo jugador utiliza Dragón eléctrico
player2_hasFirecracker	El segundo jugador utiliza Lanzafuegos
player2_hasMightyMiner	El segundo jugador utiliza Gran minero
player2_hasBattleHealer	El segundo jugador utiliza Curandera guerrera
player2_hasSkeletonKing	El segundo jugador utiliza Rey esqueleto
player2_hasArcherQueen	El segundo jugador utiliza Reina arquera
player2_hasGoldenKnight	El segundo jugador utiliza Caballero dorado
player2_hasMonk	El segundo jugador utiliza Monje
player2_hasSkeletonDragons	El segundo jugador utiliza Dragones esqueleto
player2_hasMotherWitch	El segundo jugador utiliza Bruja madre
player2_hasElectroSpirit	El segundo jugador utiliza Espíritu eléctrico
player2_hasElectroGiant	El segundo jugador utiliza Gigante eléctrico
player2_hasPhoenix	El segundo jugador utiliza Fénix
player2_hasLittlePrince	El segundo jugador utiliza Principito
player2_hasGoblinDemolisher	El segundo jugador utiliza Demoledor duende
player2_hasGoblinMachine	El segundo jugador utiliza Máquina duende
player2_hasSuspiciousBush	El segundo jugador utiliza Arbusto sospechoso
player2_hasGoblinstein	El segundo jugador utiliza Goblinstein
player2_hasRuneGiant	El segundo jugador utiliza Gigante rúnica
player2_hasBerserker	El segundo jugador utiliza Berserker
player2_hasBossBandit	El segundo jugador utiliza Bandida líder
player2_hasCannon	El segundo jugador utiliza Cañón
player2_hasGoblinHut	El segundo jugador utiliza Choza de duendes
player2_hasMortar	El segundo jugador utiliza Mortero
player2_hasInfernoTower	El segundo jugador utiliza Torre infernal
player2_hasBombTower	El segundo jugador utiliza Torre bombardera
player2_hasBarbarianHut	El segundo jugador utiliza Choza de bárbaros
player2_hasTesla	El segundo jugador utiliza Torre tesla
player2_hasElixirCollector	El segundo jugador utiliza Recolector de elixir
player2_hasX-Bow	El segundo jugador utiliza Ballesta
player2_hasTombstone	El segundo jugador utiliza Lápida
player2_hasFurnace	El segundo jugador utiliza Horno
player2_hasGoblinCage	El segundo jugador utiliza Jaula del forzudo
player2_hasGoblinDrill	El segundo jugador utiliza Excavadora de duendes
player2_hasFireball	El segundo jugador utiliza Bola de fuego
player2_hasArrows	El segundo jugador utiliza Flechas
player2_hasRage	El segundo jugador utiliza Furia
player2_hasRocket	El segundo jugador utiliza Cohete

Continúa en la siguiente página

Variable	Descripción
player2_hasGoblinBarrel	El segundo jugador utiliza Barril de duendes
player2_hasFreeze	El segundo jugador utiliza Hielo
player2_hasMirror	El segundo jugador utiliza Espejo
player2_hasLightning	El segundo jugador utiliza Rayo
player2_hasZap	El segundo jugador utiliza Descarga
player2_hasPoison	El segundo jugador utiliza Veneno
player2_hasGraveyard	El segundo jugador utiliza Cementerio
player2_hasTheLog	El segundo jugador utiliza El Tronco
player2_hasTornado	El segundo jugador utiliza Tornado
player2_hasClone	El segundo jugador utiliza Clon
player2_hasEarthquake	El segundo jugador utiliza Terremoto
player2_hasBarbarianBarrel	El segundo jugador utiliza Barril de bárbaro
player2_hasHealSpirit	El segundo jugador utiliza Espíritu sanador
player2_hasGiantSnowball	El segundo jugador utiliza Bola de nieve
player2_hasRoyalDelivery	El segundo jugador utiliza Paquete real
player2_hasVoid	El segundo jugador utiliza Vacío
player2_hasGoblinCurse	El segundo jugador utiliza Maldición duende
player2_meanCardLevel	Nivel medio de las cartas del mazo del segundo jugador
player2_minCardLevel	Nivel mínimo de las cartas del mazo del segundo jugador
player2_maxCardLevel	Nivel máximo de las cartas del mazo del segundo jugador
player2_totalStarLevel	Nivel estelar total de las cartas del mazo del segundo jugador
player2_meanElixirCost	Coste de elixir del mazo del segundo jugador
player2_numEvolutionCards	Número de cartas con evolución en el mazo del segundo jugador (debe tener la evolución)
player2_numWinConditionCards	Número de cartas consideradas *Win Condition* en el mazo del segundo jugador
player2_numMeleeCards	Número de cartas cuerpo a cuerpo en el mazo del segundo jugador
player2_numRangedCards	Número de cartas a distancia en el mazo del segundo jugador
player2_numAirCards	Número de unidades aéreas en el mazo del segundo jugador
player2_numAntiAirCards	Número de cartas con daño a unidades aéreas en el mazo del segundo jugador
player2_numDirectDamageCards	Número de cartas con daño directo a torre en el mazo del segundo jugador
player2_numSplashDamageCards	Número de cartas con daño de salpicadura en el mazo del segundo jugador

Continúa en la siguiente página

Variable	Descripción
player2_numResetAttackCards	Número de cartas con reseteo de ataque en el mazo del segundo jugador
player2_numCommonCards	Número de cartas comunes en el mazo del segundo jugador
player2_numRareCards	Número de cartas raras en el mazo del segundo jugador
player2_numEpicCards	Número de cartas épicas en el mazo del segundo jugador
player2_numLegendaryCards	Número de cartas legendarias en el mazo del segundo jugador
player2_numChampionCards	Número de campeones en el mazo del segundo jugador
player2_numTroopCards	Número de tropas en el mazo del segundo jugador
player2_numBuildingCards	Número de edificios en el mazo del segundo jugador
player2_numSpellCards	Número de hechizos en el mazo del segundo jugador
player2_numCounters	Número de *counters* altamente efectivos en el mazo rival del segundo jugador
player2_numUncounteredCards	Número de cartas del mazo del segundo jugador sin *counters* efectivos en el mazo rival
player2_supportCardName	Nombre de la tropa de las torres de coronas utilizada por el segundo jugador
player2_supportCardLevel	Nivel de la tropa de las torres de coronas utilizada por el segundo jugador
player2_supportCardRarity	Rareza de la tropa de las torres de coronas utilizada por el segundo jugador

B. Metodología

En este proyecto se ha adoptado la metodología CRISP-DM (*Cross Industry Standard Process for Data Mining*), un estándar ampliamente reconocido en el ámbito de la ciencia de datos por su flexibilidad. Esta metodología proporciona un marco claro para gestionar las diferentes etapas de un proyecto de minería de datos, desde la comprensión inicial de los objetivos hasta la implementación final de un modelo (Figura B.1). Aunque CRISP-DM está principalmente diseñada para proyectos que parten de un conjunto de datos ya existente, en este caso se ha adaptado para incorporar un paso esencial: la extracción de información desde la API oficial de *Clash Royale* y su posterior limpieza y transformación. Este ajuste es crucial para garantizar que los datos crudos extraídos, provenientes de partidas, puedan ser procesados y utilizados de manera efectiva en las etapas posteriores. Es por ello que a pesar de haber utilizado esta metodología, las distintos apartados correspondientes al desarrollo del proyecto no coinciden exactamente con las fases de esta, sino que se ha optado por distinguir entre la adquisición de datos con sus posteriores transformaciones y el análisis exploratorio.

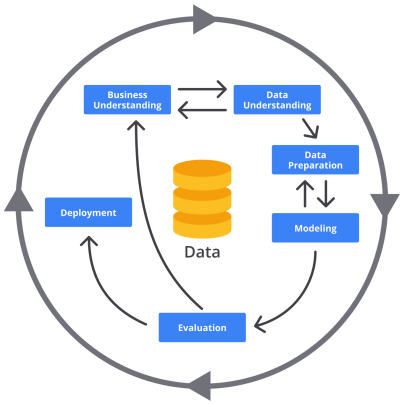

Figura B.1: Metodología CRISP-DM [10].

1. **Comprensión del negocio:** Busca definir los objetivos del proyecto desde la perspectiva organizacional o de los usuarios. Se identifican las metas a alcanzar y se delimita el problema que se desea resolver. La comprensión del negocio es fundamental para garantizar que el proyecto se alinee con los intereses de las partes involucradas y que los resultados sean relevantes y aplicables. En este proyecto, ha sido importante conocer las mecánicas del juego. Además, se ha estudiado profundamente la API oficial para comprender qué tipo de información se puede obtener, cómo está estructurada y qué datos nos interesan.

2. **Comprensión de los datos:** Es la etapa en la que se exploran las fuentes de información disponibles. Esta fase implica recopilar datos, analizar sus características principales y evaluar su calidad. Se busca identificar relaciones entre variables y problemas como valores extremos, datos inconsistentes o información incompleta. En proyectos como este, donde los datos provienen de fuentes externas como una API, este paso también incluye diseñar las estrategias de extracción y transformación necesarias para convertir datos crudos en formatos útiles para ser explorados y utilizados en las etapas posteriores.

3. **Preparación de los datos:** Precede al modelado y es la fase donde se suele realizar la mayor parte del trabajo técnico. Aquí se seleccionan las variables más relevantes y se transforman los datos para adaptarlos a los algoritmos. Esta etapa incluye técnicas como la codificación de variables categóricas, la normalización de datos numéricos y la creación de nuevas variables derivadas. Aunque a menudo es un proceso laborioso, es esencial para garantizar que los datos sean adecuados y útiles para los modelos de aprendizaje automático.

4. **Modelado:** En esta etapa, se eligen y aplican algoritmos para resolver el problema. Los modelos se ajustan y optimizan para maximizar su rendimiento, un proceso que a menudo incluye el ajuste de hiperparámetros y el uso de validación cruzada para realizar estimaciones honestas. En este proyecto se han utilizado los principales algoritmos para resolver problemas de clasificación.

5. **Evaluación:** Es una etapa crítica en la que se verifica la calidad y la utilidad de los modelos obtenidos. Esto incluye analizar las métricas adecuadas e interpretar el rendimiento en términos del problema planteado inicialmente.

6. **Despliegue:** Se implementan las soluciones desarrolladas en un entorno real. En este proyecto, el despliegue incluye el desarrollo de una aplicación que utiliza el mejor modelo para predecir el resultado de nuevas partidas a través de los perfiles y los mazos de los jugadores. También incorpora técnicas de explicabilidad para comprender cómo los diferentes factores influyen en la predicción.

C. Planificación del proyecto

Para llevar a cabo el proyecto se ha utilizado la metodología CRISP-DM explicada a lo largo del Capítulo B, mientras que este documento ha sido redactado de forma paralela. En el diagrama de la Figura C.1 se puede apreciar la duración y distribución de cada una de sus fases.

Debemos tener en cuenta que en este trabajo no se ha partido de un conjunto de datos ya existente, sino que ha sido necesario extraer la información de la API oficial de *Clash Royale* y transformar los datos crudos para poder utilizarlos en fases posteriores. Este proceso se ha considerado parte de la comprensión de los datos, ya que esta etapa incluye la recopilación de información a través de diversas fuentes y teóricamente al utilizar la API estamos recopilando datos que posteriormente debemos limpiar y transformar para comenzar a realizar la exploración. Si bien es cierto que en este tipo de proyectos la mayor parte del trabajo se concentra en la fase de preparación de los datos o preprocesamiento, en este caso también lo hace en la fase de comprensión de los datos, ya que recopilar los datos crudos y convertirlos al formato adecuado ha sido una tarea altamente demandante.

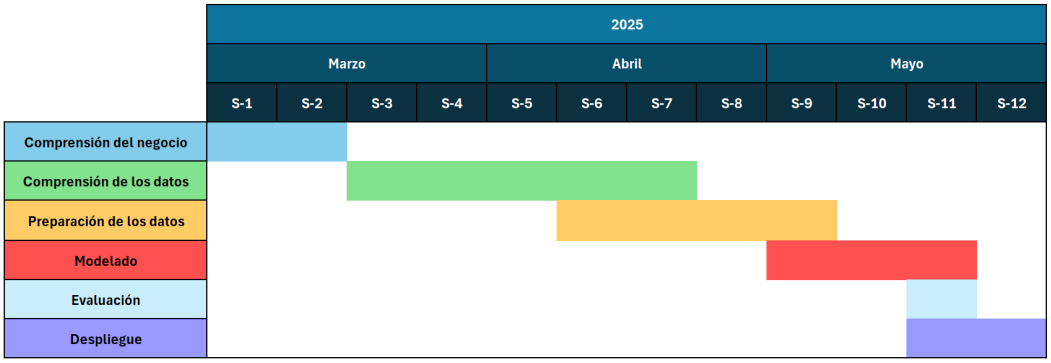

Figura C.1: Diagrama de planificación del proyecto.

D. Recursos utilizados

En este capítulo se van a listar los recursos software y hardware que han sido utilizados durante la realización del trabajo.

Recursos software

Los recursos software que se han utilizado son los citados a continuación.

- **API oficial de *Clash Royale*.** Interfaz de Programación de Aplicaciones proporcionada por Supercell que permite acceder a datos oficiales del juego, como información sobre perfiles de jugadores o partidas pasadas.
- **ChatGPT.** Chatbot de inteligencia artificial desarrollado por OpenAI, utilizado para aumentar la productividad durante la realización del proyecto.
- **CSS (Hoja de Estilos en Cascada):** Lenguaje utilizado para definir el diseño, colores y fuentes de las páginas web. Da estilo a la aplicación.
- **Flask.** *Framework* escrito en Python, utilizado para desarrollar la aplicación.
- **HTML.** Lenguaje de marcado utilizado para estructurar la interfaz de la aplicación, garantizando que el contenido esté organizado y sea accesible.
- **JavaScript.** Lenguaje de programación utilizado para mejorar la interacción en la aplicación a través de la selección de cartas.
- **Jupyter Notebooks.** Herramienta de desarrollo interactivo utilizada para desarrollar las diferentes fases del proyecto.
- **Matplotlib.** Biblioteca de Python empleada durante en análisis exploratorio de datos para visualizar la distribución de las variables y las relaciones entre ellas.
- **Microsoft Excel.** Software de hojas de cálculo utilizado para realizar el diagrama de planificación del proyecto.
- **NumPy.** Biblioteca de Python utilizada para realizar cálculos numéricos de alta eficiencia, utilizada para el manejo de estructuras de datos y cálculos relacionados con los modelos predictivos.

- **Overleaf.** Editor en línea de LaTeX, utilizado para redactar este documento.
- **Pandas.** Biblioteca de Python que facilita la manipulación y análisis de datos.
- **Python.** Lenguaje de programación de propósito general, utilizado durante el desarrollo del proyecto.
- **Requests.** Biblioteca de Python utilizada para realizar peticiones, esencial para la extracción de datos desde la API oficial de *Clash Royale*.
- **Scikit-learn.** Biblioteca de Python que ofrece herramientas de aprendizaje automático, utilizada para preprocesar los datos y entrenar los modelos.
- **Seaborn.** Biblioteca complementaria a Matplotlib, empleada para crear gráficos más detallados y estilizados durante el análisis exploratorio de datos.
- **SHAP.** Biblioteca de Python utilizada para interpretar modelos de aprendizaje automático mediante la asignación de valores de importancia a las características.
- **Visual Studio Code.** Entorno de desarrollo integrado utilizado para gestionar los archivos de código relacionados el proyecto.
- **Windows 10 Pro.** Sistema operativo utilizado durante el desarrollo y ejecución de las distintas fases del proyecto.

Recursos hardware

Los recursos hardware que se han utilizado son los citados a continuación.

- **CPU.** AMD Ryzen 5 4600H con Radeon Graphics a 3.00 GHz.
- **RAM.** 8,00 GB.
- **Tarjeta gráfica.** NVIDIA GeForce GTX 1650.

Referencia bibliográfica

[1] (111) clash royale: Gameplay first look - youtube. https://www.youtube.com/watch?v=_hNxfiXmeAE, 2025. Accedido: 2025-03-18.

[2] About us - royaleapi. https://royaleapi.com/about, 2025. Accedido: 2025-03-21.

[3] Clash royale - aplicaciones en google play. https://play.google.com/store/apps/details?id=com.supercell.clashroyale, 2025. Accedido: 2025-03-18.

[4] Clash royale api. https://developer.clashroyale.com/#/, 2025. Accedido: 2025-03-18.

[5] Freemium: Its business model, explained (with examples) | built in. https://builtin.com/articles/freemium, 2025. Accedido: 2025-03-18.

[6] Home | clash royale esports. https://esports.clashroyale.com/es, 2025. Accedido: 2025-03-18.

[7] Matchup - deck statistics - royaleapi. https://royaleapi.com/decks/stats/archer-queen,arrows,bandit,bats-ev1,goblin-gang,mega-knight-ev1,prince,wall-breakers/matchup/decks, 2025. Accedido: 2025-03-21.

[8] Royaleapi - clash royale analytics, profiles and insights. https://royaleapi.com/, 2025. Accedido: 2025-03-21.

[9] Supercell. https://supercell.com, 2025. Accedido: 2025-03-18.

[10] Muhammad Nazar Alwi. Crisp-dm: Tahapan, studi kasus, kelebihan, dan kekurangan. https://www.dicoding.com/blog/crisp-dm-tahapan-studi-kasus-kelebihan-dan-kekurangan/, 2025. Accedido: 2025-03-22.

[11] Baker College. The rise of esports: Exploring the competitive gaming phenomenon. https://www.baker.edu/about/get-to-know-us/blog/exploring-esports-industry-opportunities-in-competitive-gaming/, 2025. Accedido: 2025-03-18.

[12] Luis a De la Ossa and José Antonio Gámez. *Data Mining - Lesson 3: Model Validation and Evaluation*. Grado en Ingeniería Informática. Escuela Superior de Ingeniería Informática de Albacete. Universidad de Castilla-La Mancha, 2023. Accedido: 2024-02-20.

[13] Luis a De la Ossa and José Antonio Gámez. *Data Mining - Lesson 4: Instance-based methods (kNN)*. Grado en Ingeniería Informática. Escuela Superior de Ingeniería Informática de Albacete. Universidad de Castilla-La Mancha, 2023. Accedido: 2024-02-20.

[14] Luis a De la Ossa and José Antonio Gámez. *Data Mining - Lesson 5: Decision trees*. Grado en Ingeniería Informática. Escuela Superior de Ingeniería Informática de Albacete. Universidad de Castilla-La Mancha, 2023. Accedido: 2024-02-20.

[15] Luis a De la Ossa and José Antonio Gámez. *Data Mining - Lesson 6: Combining classifiers - Ensembles*. Grado en Ingeniería Informática. Escuela Superior de Ingeniería Informática de Albacete. Universidad de Castilla-La Mancha, 2023. Accedido: 2024-02-20.

[16] Luis a De la Ossa and José Antonio Gámez. *Data Mining - Lesson 8: Naive Bayes and Logistic Regression*. Grado en Ingeniería Informática. Escuela Superior de Ingeniería Informática de Albacete. Universidad de Castilla-La Mancha, 2023. Accedido: 2024-02-20.

[17] Luis a De la Ossa and José Antonio Gámez. *Data Mining - Lesson 9: Neural networks (and deep learning)*. Grado en Ingeniería Informática. Escuela Superior de Ingeniería Informática de Albacete. Universidad de Castilla-La Mancha, 2023. Accedido: 2024-02-20.

[18] A. Jung. Machine learning: The basics. https://arxiv.org/pdf/1805.05052, Springer, Singapore, 2022. Accedido: 2025-03-19.