Universidad Internacional de la Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Modelado y caracterización de jugadores de tenis para la predicción de resultados en eventos deportivos

Trabajo Fin de Máster

presentado por: Iván Pérez Fernández

Dirigido por: Dr. Juan Carlos Infante Moro

Ciudad: Madrid

Fecha: 24 de Junio de 2021

Resumen

Tomando como referencia portales web donde se pueden encontrar infinidad de datos deportivos, el objetivo del presente proyecto se basa en la explotación de dicha información para realizar predicciones deportivas a través de redes neuronales. En primer lugar se realiza una exportación masiva de información disponible sobre jugadores de tenis mediante técnicas de *web scraping* y consumición de APIs públicas. Posteriormente, se utilizarán dichos datos para alimentar una RNN *(Red Neuronal Recurrente)* para predecir la probabilidad de ganar un encuentro entre dos jugadores enfrentados. De esta manera se puede caracterizar de manera automática las prestaciones individuales de cada jugador. En segundo lugar, el siguiente objetivo sería crear un *bot* que pueda actuar como analista, proporcionando de forma automatizada sugerencias de apuestas deportivas y comparar los resultados obtenidos a largo plazo para conocer si el modelo de negocio obtiene beneficios a largo plazo. Dentro de todos los eventos deportivos este proyecto focaliza en el tenis porque tiene varias ventajas: deporte individual y muchos eventos diariamente para conseguir una base de datos consistente y que los resultados no tengan variación en sesgo.

Palabras Clave: redes neuronales, deportes, web scraping, predicción, apuestas

Abstract

Taking as a reference web portals where an infinite amount of sports data can be found, the objective of this project is based on the exploitation of this information to make sports predictions through neural networks. First of all, a massive export of available information about tennis players is carried out by means of web scraping techniques and consumption of public APIs. Subsequently, this data will be used to feed a RNN (Recurrent Neural Network) to predict the probability of winning a match between two players facing each other. In this way, the individual performances of each player can be automatically characterized. Secondly, the next objective would be to create a *bot* that can act as an analyst, providing in an automated way sports betting suggestions and compare the results obtained in the long term to know if the business model is profitable in the long term. Within all sporting events this project focuses on tennis because it has several advantages: individual sport and many events daily to achieve a consistent database and that the results have no variation in bias.

Keywords: neural networks, sports, web scraping, prediction, betting

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En la primera sección del documento se explica cuáles son los acontecimientos que han impulsado el desarrollo del proyecto así como los problemas o mejoras que plantea solventar. Además, se hace un breve resumen sobre cómo se ha enfocado la fase del desarrollo y la estructura que seguirán las siguientes secciones donde se documenta todo el proceso.

1.1. Motivación

Hoy en día existe un gran número de personas consideradas expertas en ciertos deportes, bien sea porque son fieles seguidores de determinados equipos o jugadores, o bien porque simplemente ese deporte es un *hobby* para ellos. Poseen un conocimiento muy actualizado del estado de los equipos a los que siguen y son capaces, en cierto modo, de predecir las tendencias en el resultado de un partido en función de qué equipos se enfrentan entre sí y de experiencias previas. Esta es la clave del presente proyecto, puesto que este grupo de personas, a las cuales a partir de ahora serán referidas como *tipsters*, se lucran de sus conocimientos compartiendo sus predicciones con las personas interesadas.

Además, el avance de la tecnología ha producido la aparición de ciertas empresas o casas de apuestas online a través de las cuales es muy sencillo 'apostar' sobre infinidad de parámetros estadísticos (entre ellos el resultado de un partido) y obtener un beneficio económico a cambio en caso de acierto.

Si bien es cierto que muchos de los *tipsters* más conocidos tienen un gran número de seguidores debido a su elevada tasa de acierto, sus predicciones se basan simplemente en el criterio personal y experiencia previa. Como es de esperar, es muy complicado que puedan tener en cuenta factores críticos como el estado de ánimo y salud de la gran infinidad de

jugadores que se enfrentan a diario o cada fin de semana, lo cual sería una tarea muy sencilla para un ordenador.

Esta barrera es la que se intenta romper con este proyecto, utilizando técnicas de *Machine Learning* para entrenar un modelo capaz de predecir el resultado de partidos de tenis, donde el factor humano sea eliminado de la ecuación y se tenga en cuenta cualquier característica disponible de cada jugador para modelar su estado. Como es obvio, esto es escalable a los múltiples partidos que se disputan diariamente, por lo que la idea es automatizar todo este proceso de forma que un **bot** actúe como *tipster* y envíe sus predicciones automáticamente a través de un canal informativo como una red social o una aplicación de mensajería móvil.

En función de los resultados obtenidos se valorará la opción de habilitar un modelo de suscripciones a dicho canal por un módico precio, de forma que se pueda obtener un beneficio añadido además de los obtenidos por las predicciones en casas de apuestas.

1.2. Planteamiento del trabajo

Como en cualquier problema de *Machine Learning* lo primordial es obtener una cantidad considerable de datos con la mayor calidad posible. En internet se pueden encontrar infinidad de webs que contienen un histórico de información y estadísticas sobre distintos deportes. La idea es utilizarlas para extraer los datos que sean necesarios para obtener el objetivo mencionado.

En este proyecto se explotará la información de la web SofaScore, ya que es un portal en el que se recogen múltiples datos de varios deportes, aunque en este caso se utilizarán solamente los de la sección de tenis. Para ello, el primer *approach* es utilizar técnicas de *web scraping*, sin embargo, tras analizar la web se ha visto que dispone de APIs REST públicas que devuelven toda la información necesaria de forma estructurada, por lo tanto ésta será la vía a explorar.

La primera parte es por lo tanto, conseguir una base de datos actualizada a partir de la información extraída de la web. En una primera fase nos centraremos en el ranking de jugadores de ATP, es decir, obtendremos toda la información disponible de los 500 mejores jugadores de tenis de esta categoría. Además, podremos obtener una buena cantidad de información sobre partidos que han jugado previamente (hasta varios meses atrás), lo que nos permitirá saber los resultados obtenidos y relacionar qué jugadores se han enfrentado

entre sí.

Esto permitirá utilizar aprendizaje supervisado, ya son conocidos los resultados de partidos previos y toda esta información será el *input* a una RNN (*Red Neuronal Recurrente*) que será entrenada y utilizada para validar eventos futuros en función de lo aprendido.

En la figura 1.1 se muestra un diagrama a muy alto nivel del flujo de tareas a realizar:

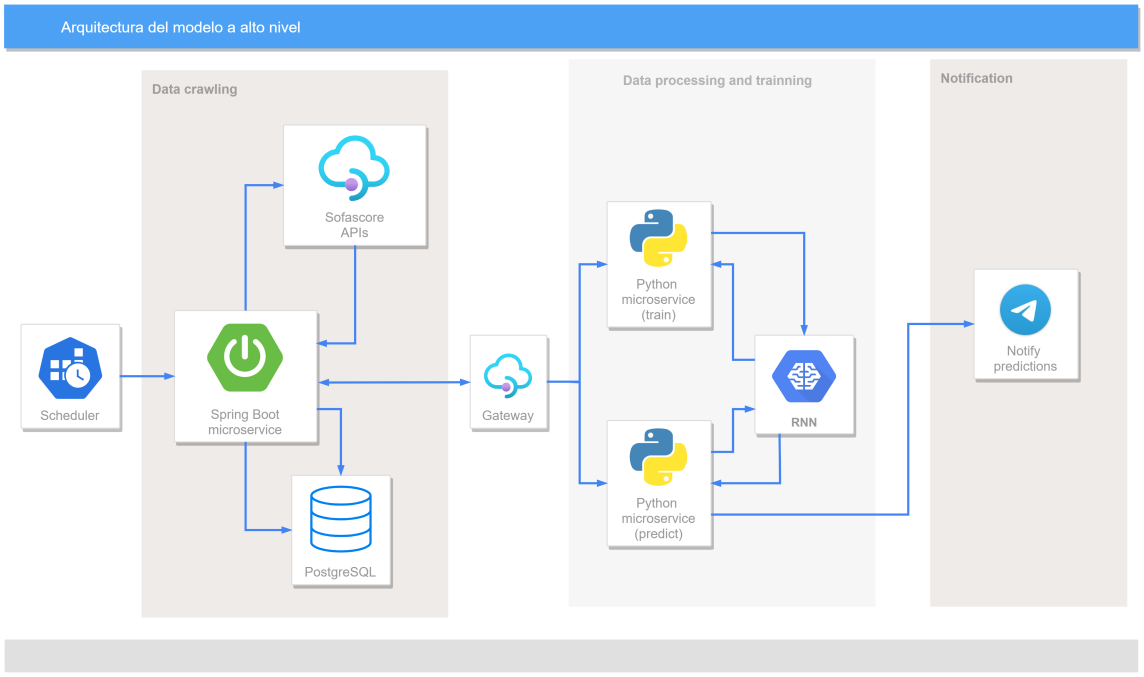


Figura 1.1: Arquitectura a alto nivel del sistema (elaboración propia)

1.3. Estructura de la memoria

El presente documento recoge paso a paso todas las tareas desarrolladas durante el proyecto, desde el planteamiento del problema hasta la finalización del desarrollo software y las conclusiones obtenidas.

En el primer apartado se realiza la introducción, donde se explica principalmente cómo surge la idea de este desarrollo y el problema que plantea solventar y/o mejorar. Además, se explican a muy alto nivel las distintas fases por las que pasa el proceso de desarrollo, así como qué técnicas o tecnologías serán utilizadas durante el mismo.

Seguidamente se hace un análisis del estado del arte en el ámbito del proyecto, donde se analizan *papers* que tratan temáticas similares y que pueden servir como fuente de inspiración para el desarrollo a acometer además de una comparativa de las fuentes de datos deportivas más importantes a día de hoy, explicando por qué se ha optado por una

y no por otra.

El siguiente apartado desarrolla qué objetivos buscan alcanzarse con este desarrollo, empezando por aquellos de carácter más general o a alto nivel, pero que luego son desglosados en objetivos mucho más concretos o específicos. Estos últimos serán la base para realizar una planificación del proyecto y así poder desglosar los distintos hitos o tareas en las que se dividirá el desarrollo.

Los capítulos 4, 5 y 6 son el grueso del proyecto. Como en cualquier proyecto de desarrollo software, se parte por una toma de requisitos y una estimación de esfuerzos para poder realizar posteriormente una planificación. Seguidamente, se explica detalladamente cada uno de los componentes desarrollados y su función así como las tecnologías que se han empleado. Una vez todos los componentes están finalizados, se debe comprobar el funcionamiento completo del sistema *end-to-end*, y será entonces cuando se obtengan los resultados y se analiza el funcionamiento y la viabilidad del proyecto.

Finalmente, se explican las conclusiones que se pueden obtener del trabajo realizado así como posibles mejoras o nuevas funcionalidades que pueden ser implementadas posteriormente para dotar al sistema de una mayor autonomía o rendimiento.

Capítulo 2

Contexto y Estado del Arte

En esta sección se realiza en primer lugar un repaso de los acontecimientos más importantes que ha sufrido la inteligencia artificial desde sus orígenes hasta la actualidad y lo que se espera de ella en los próximos años dada su evolución en la última década. Esto permitirá obtener una visión general de las capacidades que ofrece y la viabilidad de los objetivos que se quieren conseguir con este proyecto de acuerdo a los avances tecnológicos del momento.

Además, como debería hacerse en cualquier proyecto de estas características, antes de proceder con el desarrollo del proyecto se ha realizado una exploración en distintas bases de datos de investigación sobre artículos cuyo objetivo sea la predicción de eventos deportivos mediante técnicas de aprendizaje supervisado. Esto permitirá, por un lado identificar qué obstáculos o problemas se han encontrado otros investigadores durante el proceso y cómo se han solventado, es decir, servirán como base para evitar repetir los mismos problemas que ya otras personas se hayan encontrado. En este apartado se analizan varios *papers* cuyo enfoque es bastante similar al objetivo que se quiere conseguir en el presente proyecto y que podrán servir como base para enfocar el desarrollo de acuerdo a las conclusiones que proporcionan.

Finalmente, se realiza un análisis de las capacidades y características principales de las herramientas más extendidas en cuanto al entrenamiento de redes neuronales. Esto permitirá utilizar los últimos avances y explotar al máximo todas las mejoras que se han ido desarrollando con el paso de los años. Se explicarán las características fundamentales de cada una de ellas para poder obtener facilitar la selección de la más adecuada a las necesidades del proyecto.

2.1. Evolución de la inteligencia artificial hasta la actualidad

Aunque el concepto de inteligencia artificial nació en 1956 en la *Conferencia de Dartmouth* (Veisdal, 2019), la investigación y el desarrollo de sistemas con algún tipo de conocimiento *racional* ya comenzó varios años atrás. El interés por replicar el comportamiento humano en máquinas siempre ha sido un foco principal de investigación debido a las ventajas que podría suponer en muchos ámbitos.

Bien es cierto que la evolución en este campo no ha sido siempre tan rápida como en los últimos años. Uno de los motivos principales han sido los avances en la capacidad de computación que tienen los ordenadores hoy en día, lo cual ha posibilitado que, por ejemplo, el desarrollo de redes neuronales vuelva a estar en boca de todo el mundo debido a los avances que han surgido en la última década. Dicho esto, es importante recalcar que el concepto de inteligencia artificial muchas veces se utiliza de forma errónea, ya que es un concepto muy amplio que está compuesto por diversas áreas de la ciencia, tal y como se muestra en la figura 2.1.

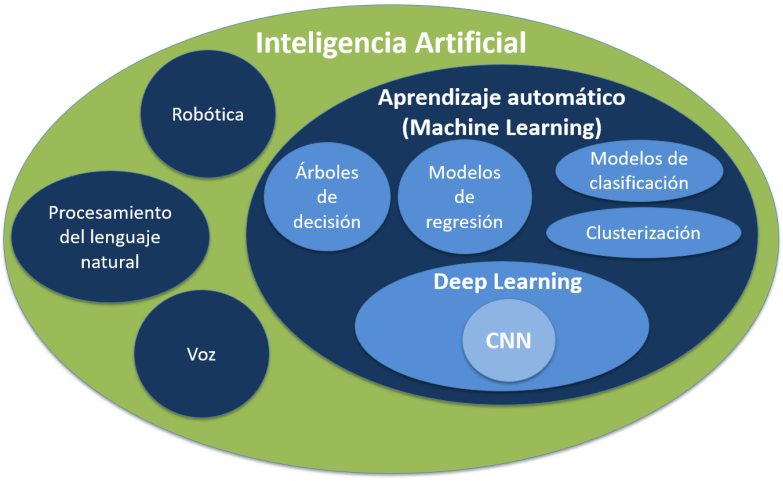


Figura 2.1: Diagrama de la Inteligencia Artificial (cienciacarbonica, 2019)

Actualmente, la inteligencia artificial está presente en prácticamente todos los sectores, como por ejemplo en medicina, donde ayuda a la detección prematura de enfermedades graves, o en el sector automovilístico, donde el coche completamente autónomo está cada vez más cerca de ser una realidad. Además, incluso en cualquier *smartphone* de última generación se incluyen agentes conversacionales que hacen uso de esta tecnología (como Siri o Alexa), lo cual hace que esté prácticamente al alcance de todo el mundo.

No hace falta hablar de robots o de ciencia ficción para referirse a la inteligencia

artificial. Muchas veces pasa desapercibida, pero por ejemplo las recomendaciones que ofrecen empresas como Amazon o Netflix hacen uso de técnicas de machine learning para proporcionar una experiencia mucho más personalizada e incitar al usuario al consumo de sus productos.

En la figura 2.2 se presenta un desglose temporal donde se hace un repaso de varios de los hitos más importantes que ha sufrido la inteligencia artificial desde prácticamente sus inicios hasta la actualidad:

Figura 2.2: Hitos principales de la Inteligencia Artificial (Unir, 2020)

La inteligencia artificial dista aún bastante de igualar las capacidades del cerebro humano, sin embargo, con los avances obtenidos en los últimos años ha quedado reflejado su potencial y la gran capacidad de resolver problemas que tiene.

2.2. Inteligencia Artificial en deportes

El concepto de Inteligencia Artificial está alterando positivamente el deporte y elevándolo a un nivel de éxito inaudito. Aunque los hechos confirman que las mediciones y la investigación cuantitativa han asumido un trabajo focal en los deportes durante bastante tiempo, la Inteligencia Artificial está afectando esencialmente a cómo se planifican los

partidos, cómo se juegan y cómo atraen al público.

La inteligencia artificial está desarrollando una forma más creativa de triunfar en el deporte, tanto para los deportistas como para los entrenadores, proporcionando conocimientos constantes del juego, previsión de estrategias de juego que ayudan a capacitar al jugador a elegir el procedimiento correcto e incluso alarmarle si se produce una posible disminución del rendimiento o una lesión. La innovación se ha convertido en algo inevitable en el deporte y en un apoyo clave para su desarrollo, tanto dentro como fuera del estadio, lo que permite a todos los jugadores y grupos ser simplemente los mejores.

En la actualidad, los datos son una materia prima de incalculable valor en muchos sectores ya que son la base fundamental de cualquier proyecto de inteligencia artificial. Además, con la evolución de internet y los ordenadores personales o smartphones, dichos datos son prácticamente accesibles por cualquier persona con solo un par de clicks en cuestión de segundos.

Tal y como se indica en el artículo *Deportes y Data Mining* (Univaso, s.f.), en el mundo de los deportes, se recolecta infinidad de información con el fin de construir una base de datos sólida. Dicha información puede ser explotada con infinidad de objetivos que son comunes a cualquier deporte.

2.2.1. Análisis del rendimiento de jugadores

Permite caracterizar el rendimiento de los jugadores y tomar decisiones que mejoren el mismo. La Inteligencia Artificial está cambiando la forma en que los entrenadores llevan a cabo su trabajo independientemente del tipo de deporte. Las decisiones sobre la alineación que se debe presentar contra los grupos rivales se ven actualmente afectadas tanto por los escenarios generados computacionalmente como por la experiencia del entrenador. La Inteligencia Artificial sería capaz de cuantificar la velocidad, el giro y la disposición del saque de tenis, de una bola curva, de un pase hacia adelante, de un tiro extra, y muchas otras actividades, así como la situación de los jugadores en el espacio. Esta información mejora la preparación de los mentores para preparar a los jugadores para la rivalidad. De manera igualmente significativa, la I.A. puede anticipar las probabilidades de éxito de diferentes estrategias de juego. Por ejemplo, algunos entrenadores de fútbol recurren actualmente a la I.A. para que les ayude a planificar a las jugadas correctas durante un juego.

Sin embargo, no todo tienen por qué ser ventajas. Más allá de la precisión de la pre-

dicción (la cual puede no ser tan buena como debería), otro problema notable del uso de algoritmos de aprendizaje automático es la perpetuación de posibles sesgos en los procesos de evaluación y toma de decisiones. Estos problemas potenciales en la evaluación, predicción y sesgo se aplican ampliamente a todo tipo de investigaciones en ciencias del deporte. Los datos utilizados pueden estar subrepresentando a ciertos grupos demográficos, y los análisis sobre las diferencias de los subgrupos pueden no ser tan sólidos. Además, existe la inevitable asociación de la inteligencia artificial con la falta de privacidad y la vigilancia digital. Cuando se recopilan datos a gran escala sin el consentimiento informado para una de investigación, se plantean cuestiones como cuándo y cómo se pueden utilizar estos tipos de datos, hasta qué punto es necesario el consentimiento informado y cómo se puede proteger la privacidad de las personas en el proceso de análisis e investigación. La aparición de este tipo de datos requiere que los científicos del deporte trabajen con los desafíos éticos de la confidencialidad y la privacidad relacionados con los grandes datos.

2.2.2. Previsión de venta de entradas

Tiene un objetivo más económico. Se centra en obtener información sobre cuál sería el precio óptimo de puesta en venta de entradas, así como la previsión de ventas. La inteligencia artificial sería capaz de comprar asientos determinados para los aficionados en función de sus intereses, les permitirá cambiar de asiento e incluso reservarles la asistencia a un partido determinado basándose en el comportamiento e intereses del usuario.

La predicción del volumen de ventas es decisiva para la mayoría de las empresas, pero también es un campo de gestión difícil. El artículo *Dynamic Pricing For Sports Event Tickets* (Huang y col., 2020) explica cómo se puede llegar a predecir el mejor precio mediante técnicas de machine learning. La mayoría de los investigadores y empresas utilizan métodos estadísticos como el análisis de regresión para predecir y analizar los volúmenes de ventas. Además, por regla general, sólo se utilizan cantidades muy pequeñas de datos para las previsiones de ventas. Para aumentar la calidad de las previsiones de ventas, la IA puede tener en cuenta otros muchos puntos de datos, tanto históricos como en tiempo real, datos internos y externos, datos económicos y del entorno (datos de almacenes, precios, ferias, precios de la competencia, etc.). Los algoritmos y la IA, por un lado, ayudan a capturar estos numerosos puntos de datos estructurados y no estructurados de forma sistemática y automatizada y, por otro lado, a analizarlos automáticamente para obtener una previsión precisa.

2.2.3. Anticipación de estrategias de juego

Esta característica va más enfocada a los entrenadores. Puede ser utilizada para gestionar mejor a un equipo en función del contrincante o de los jugadores que lo componen, logrando una mejor o más fructífera alineación. Esto no tiene por qué ser utilizando al equipo real, si no que existen herramientas de software que permiten simular estrategias mediante inteligencia artificial.

En el artículo *Diseño de Agentes experimentando con robots que juegan al Fútbol en ambientes reales y simulados* (Kogan y col., 2006) se hace alusión a este tipo de herramientas, donde se explica la aplicación del desarrollo de agentes inteligentes a la simulación de estrategias en partidos de fútbol. En él, incluyen agentes de comportamiento reactivo, adquisición de estrategias, aprendizaje, planificación en tiempo real, sistemas multiagentes y reconocimiento del ambiente, entre otros.

Para obtener resultados aceptables en cada uno de los apartados nombrados, es necesario obtener una cantidad de datos bastante considerable, además de garantizar la calidad de los mismos. La generación de esta información y su estructuración no es tarea sencilla, alguien debe recolectar todos los datos necesarios para que puedan ser explotados en un futuro cercano y facilitar la toma de ciertas decisiones.

Esto podría suponer un gran trabajo y conllevar una cantidad de tiempo considerable, sin embargo, según indica (Beetz y col., 2007) en su artículo, con los rápidos avances de la inteligencia artificial en tareas de percepción, los ordenadores son capaces de observar y analizar los partidos a un nivel de detalle que es prácticamente imposible de registrar y procesar para los humanos. Ahora pueden seguir a los jugadores y calcular métricas útiles sin la necesidad de un laborioso etiquetado humano. Incluso pueden estimar la eficacia de cada pase a lo largo de múltiples dimensiones, como la relación riesgo-recompensa potencial asociada a un pase, la 'presión' que aplica el equipo contrario antes del pase, etc. Un ordenador también es capaz de inferir cosas como la intención de forma automática analizando las posiciones y velocidades de todos los jugadores en el campo, o etiquetar automáticamente esos pases como 'completados', 'interceptados', 'marcados', etc.

En definitiva, se han desarrollado sistemas que hacen uso de la inteligencia artificial para facilitar la obtención de datos de calidad, los cuales serán explotados por otros sistemas que harán a su vez uso de la inteligencia artificial con un fin totalmente distinto. Algo que para un humano llevaría días puede ser sustituido por un ordenador cuyo rendimiento

es mucho mayor y el margen de error es bastante reducido.

2.3. Estudios de actualidad

Como se ha comentado en el apartado 2.1, la inteligencia artificial está presente en cualquier lugar. En este proyecto se va a tratar de darle una aplicación dentro del mundo de los deportes, donde el objetivo en concreto es conseguir la caracterización o modelización del estado de jugadores de tenis profesionales. En concreto, la idea es recopilar gran cantidad de información ya conocida sobre los 500 jugadores de la clasificación de ATP *(Asociación de Tenistas Profesionales)*, tanto sobre su estado de forma física como los datos estadísticos de los últimos años, para así poder predecir su probabilidad de ganar (o perder) un encuentro futuro.

Antes de comenzar, conviene realizar un análisis de estudios previos en este ámbito, lo cual permitirá obtener ideas o una base sobre la que poder partir para así mejorar los resultados obtenidos. Se han encontrado varios *papers* donde, aunque el enfoque sea un poco diferente, el objetivo final es la predicción de resultados en partidos, lo cual puede servir para el caso de uso que se va a desarrollar. A continuación se analizan aquellos que más se adaptan al caso de uso tratado en este proyecto y se extraen las ideas principales que servirán para obtener un punto de partida sólido.

Una de las partes más importantes a la hora de realizar cualquier proyecto de aprendizaje máquina es la selección de variables predictoras o *features*. Para el caso de la caracterización de jugadores, hay varios factores o parámetros que pueden influir directamente en sus resultados. En primer lugar, su estado físico es uno de los elementos clave que podrán ser buen candidato para entrenar un modelo, ya que por ejemplo, un determinado jugador que acaba de recuperarse de una lesión complicada es muy difícil que pueda rendir al 100 % desde el día en que se reincorpora a la pista.

2.3.1. Análisis del rendimiento de jugadores en base su forma física

En el artículo *Machine learning for the prediction of professional tennis matches* (Sipko y Knottenbelt, 2015) se realiza este tipo de aproximación. En él se basan en la selección de parámetros que definen la forma física de los jugadores, lo cual puede proporcionar indicadores de qué jugadores se encuentran mejor que otros, proporcionando así un indicador de quién podrá vencer un partido en función de si su estado de forma supera o no al del

contrincante. Además, tienen en cuenta variables de distintos tipos, es decir, algunas están relacionadas con la complexión corporal del jugador, otras con los resultados obtenidos en distintas pruebas de esfuerzo, coordinación, etc... En la tabla 2.1 se recogen algunos de los parámetros que utilizan como entrada para generar un modelo:

Categorías	**Morfología**	**Fuerza explosiva**
Variables	Altura	Prueba del sargento
	Peso	Lanzamiento balón medicinal (2kg)
	IMC	Prueba de salto
	Porcentaje de grasa	
	Porcentaje de tejido muscular	
	Porcentaje de tejido óseo	

Tabla 2.1: Parámetros para la caracterización de jugadores (Sipko y Knottenbelt, 2015)

En dicho artículo, se han recopilado este tipo de datos sobre 1002 tenistas eslovenos, tanto hombres como mujeres, que son preprocesados para posteriormente entrenar distintos tipos de algoritmos de aprendizaje máquina con el fin de elegir aquel que mejor resultados proporcione. En concreto, se hacen pruebas con *vectores de máquina soporte*, *naive Bayes*, *árboles de clasificación* y *k-nearest neighbour*. Según indican al final del artículo, tras analizar los resultados obtenidos por cada uno de estos modelos se ha comprobado que unos funcionan mejor que otros para este caso, donde al parecer el método *naive Bayes* es el que mejor se adapta o mejores resultados proporciona.

Si bien es cierto que la forma física de un jugador es clave y afecta directamente a su rendimiento, en este proyecto se considera información insuficiente como para poder predecir con exactitud la capacidad de ganar o perder un partido. Por ello es necesario añadir algún tipo de información extra al modelo que proporcione una mayor fiabilidad.

2.3.2. Análisis del rendimiento de jugadores en base a datos históricos

Otro de las aproximaciones a la hora de estimar la probabilidad de ganar o perder un partido es utilizar directamente técnicas de aprendizaje supervisado sobre datos estadísticos históricos, es decir, atender a información de un gran número de partidos ya disputados sobre los que ya conocemos el resultado y los jugadores que se enfrentaron entre sí.

Este tipo de enfoques no tiene en cuenta parámetros particulares del jugador, como su

altura, peso o estado físico general si no que únicamente se centra en resultados obtenidos durante el histórico de los partidos que ya ha disputado anteriormente.

En el artículo *Prediction of the successfulness of tennis players with machine learning methods* (Panjan y col., 2010) se hace un estudio aplicando esta técnica. En él, se recogen única y exclusivamente estadísticas sobre partidos que ya han tenido lugar, sin prestar atención a información sobre el estado físico de los jugadores. Por ejemplo, en la tabla 2.2 se muestran algunas de las variables predictoras que utiliza para entrenar el modelo:

Feature	**Explanation**	**Cleansing**
RANK	ATP rank	
POINTS	ATP points	
FS	First serve success percentage	P, E
W1SP	Winning on first serve percentage	P, E
W2SP	Winning on second serve percentage	P, E
WSP	Overall winning on serve percentage	
WRP	Winning on return percentage	P, E
TPW	Percentage of all points won	P, E
TMW	Percentage of all matches won	E, U
ACES	Average number of aces per game	
DF	Average number of double faults per game	
UE	Average number of unforced errors per game	
WIS	Average number of winners per game	
BP	Percentage of break points won	P, E, U
NA	Percentage of net approaches won	P, E, U
A1S	Average first serve speed	S
A2S	Average second serve speed	S
FATIGUE	Fatigue from matches in past 3 days	
RETIRED	Whether first match since retirement	
COMPLETE	Player completeness	
SERVEADV	Advantage when serving	
DIRECT	Head-to-head balance	U

Tabla 2.2: Parámetros utilizados para predicción de resultados (Panjan y col., 2010)

donde los parámetros P, E, U y S significan:

- P − prior to averaging, remove if value is not in range [0, 1]
- S − prior to averaging, remove if serve speed is below 120 / 100 km/h for first / second serves
- E − after averaging, remove if value is exactly 0 or 1
- U − after averaging, remove if the uncertainty is above threshold (e.g., 1.0)

Con esta información, proceden a entrenar distintos modelos de aprendizaje máquina, por un lado emplean regresión lineal y por otro redes neuronales. Según comentan en el artículo, tras evaluar ambos modelos los resultados obtenidos mediante la red neuronal proporcionan mejores resultados, lo cual será un dato a tener en cuenta para el desarrollo de este proyecto.

Como se ha podido observar en este capítulo, existen distintas teorías y metodologías para obtener el resultado deseado. La clave reside en escoger la información adecuada seleccionar los parámetros que puedan ser buenos candidatos o influyentes en la predicción del resultado de un partido y posteriormente entrenar distintos modelos y evaluarlos para poder seleccionar aquel que mejores resultados proporcione.

2.4. Herramientas de actualidad para entrenamiento de redes neuronales

Una vez se ha visitado la parte funcional del proyecto, es hora de realizar un análisis de las herramientas del mercado que actualmente se utilizan para este tipo de funcionalidades. En concreto, se analizarán aquellas más extendidas entre los desarrolladores de modelos de IA y que además son *Open Source* por razones de costes, pero que no por ello son menos eficientes.

2.4.1. Tensorflow

Tensorflow (figura 2.3) es una herramienta desarrollada por (Google, 2021) cuya finalidad es proporcionar a los desarrolladores un ecosistema compuesto por todas las herramientas y bibliotecas necesarias para el desarrollo de proyectos de aprendizaje automático.

La primera versión se publicó en el año 2015, y a día de hoy sigue siendo una de las herramientas más utilizadas por la comunidad, compuesta tanto por desarrolladores independientes, investigadores, o incluso empresas privadas que ya han empezado a introducir la inteligencia artificial en su modelo de negocio, como por ejemplo *Nvidia*, *Coca Cola* o *Airbus* entre otras muchas.

Uno de los puntos fuertes de Tensorflow es la infinidad de documentación y tutoriales que se pueden encontrar en internet. Es una herramienta que se ha expandido muy rápidamente y ha sido acogida con los brazos abiertos por la comunidad. Además, permite realizar la ejecución de los programas tanto sobre la CPU como sobre la GPU, lo cual aprovecha la capacidad de computación de cualquier ordenador al máximo. Otra de las ventajas es su versión *Lite*, la cual está enfocada a ser ejecutada en aplicaciones móviles, donde la capacidad de procesamiento es menor, pero que aun así permite hacer uso de redes neuronales complejas.

Para finalizar, comentar que con el fin de aumentar la eficiencia del código fuente, Tensorflow fue desarrollado en C++. Sin embargo, las APIs con las que se interactúa con las librerías están desarrolladas en Python, que es un lenguaje ampliamente utilizado en el mundo del aprendizaje automático y amado por la comunidad de desarrolladores por su sencillez y flexibilidad.

Figura 2.3: Logotipo de Tensorflow (Google, 2021)

2.4.2. Keras

Keras (figura 2.4) es una herramienta compuesta por una serie de librerías de código abierto escrita en Python por (Chollet, 2015). Está enfocada al desarrollo de redes neuronales, más en concreto al *deep learning*.

La mayor ventaja de Keras es su integración con el core de Tensorflow, es decir, los desarrolladores pueden explotar las capacidades que ofrece Tensorflow mediante unas librerías que son mucho más sencillas de utilizar y que ahorran una gran cantidad de trabajo, es decir, Keras facilita el acceso y desarrollo de redes neuronales utilizando el core de Tensorflow, entre otros.

La comunidad detrás de Keras es también muy extensa, siendo utilizada incluso por

empresas de gran renombre como la NASA.

Keras permite el desarrollo de redes neuronales de una forma rápida y sencilla, tanto redes neuronales estándar como las convolucionales y las recurrentes. Además, permite su ejecución y entrenamiento tanto sobre CPUs como GPUs, lo cual será un punto muy a tener en cuenta.

Debido a que está desarrollada sobre el core de Tensorflow 2.0, Keras también permite generar modelos de deep learning tanto en navegadores web como en dispositivos móviles.

Figura 2.4: Logotipo de Keras (Chollet, 2015)

2.4.3. PyTorch

Pytorch (figura 2.5) es otra de las herramientas que cada vez está ganando más popularidad. Fue desarrollada por (Facebook, 2016) y está enfocada principalmente al desarrollo de todo tipo de redes neuronales.

Ha sido desarrollada en Python, y está pensada para ser ejecutada directamente sobre GPUs. Sin embargo, hay librerías que han sido desarrollados por Nvidia que permiten conectar la CPU con la GPU, permitiendo así también su ejecución en ambos entornos.

Pese a que su lanzamiento fue en el año 2016, ha sido recientemente en Marzo de 2021 cuando se ha liberado una versión estable de la herramienta. A diferencia de Tensorflow, su interfaz es bastante sencilla, por lo que permite trabajar directamente con el core sin la necesidad de instalar o utilizar librerías de niveles superiores como Keras.

Actualmente hay varias empresas de gran calibre utilizando PyTorch, por ejemplo Tesla en su *autopilot*. Esta herramienta está siendo cada vez más aceptada por la comunidad, pero quizá aún no sea posible encontrar tanta documentación o tutoriales como con Tensorflow o Keras debido a su prematura edad.

Figura 2.5: Logotipo de Pytorch (Facebook, 2016)

2.5. Conclusiones

Una vez analizadas las capacidades clave de las herramientas más utilizadas en proyectos de inteligencia artificial, el paso final sería decidir cuál utilizar en el proyecto. Independientemente de qué características tenga un framework respecto a otro, desde el punto de vista del desarrollo software hay ciertos factores a tener en cuenta y que son fundamentales, como por ejemplo:

- Documentación: La documentación de un framework es fundamental, tanto si se tiene experiencia previa como si no. Un framework evoluciona constantemente y es necesario poder tener de forma accesible la documentación donde se explique cómo explotar toda la funcionalidad.

- Comunidad: Esta parte puede llegar a ser incluso más importante que la documentación, ya que en el mundo del desarrollo software surgen problemas en el día a día que pueden no ser contemplados por la comunidad, especialmente si se utilizan librerías externas. Estos problemas suelen ponerse en común a través de foros como Stackoverflow o incluso directamente en los repositorios de Github del software. Que haya una buena comunidad detrás de un framework supone que ante un problema basta con realizar una consulta a través de Google para encontrar una posible solución.

- Experiencia previa del desarrollador: A la hora de programar, que un desarrollador conozca una herramienta va a facilitar futuros desarrollos. Independientemente de que se utilice o no el mismo lenguaje de programación, la curva de aprendizaje ante una nueva herramienta puede dificultar o incluso comprometer el desarrollo de un proyecto, y más teniendo en cuenta que hay una fecha de entrega límite.

En base a los puntos anteriores, se ha realizado una pequeña investigación para comprobar en qué puntos se beneficia un framework sobre otro. En la figura 2.6 se observa el nivel de contribución y participación en los repositorios de Github de las herramientas analizadas:

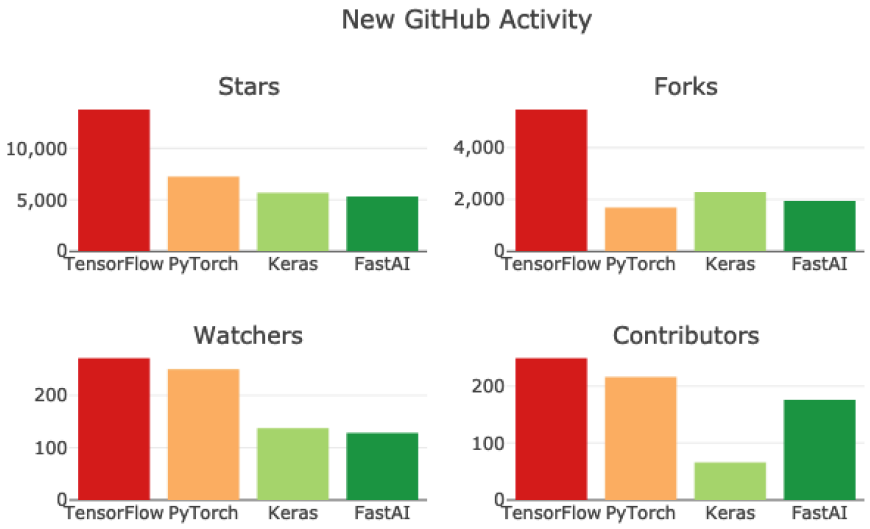


Figura 2.6: Contribución en Github sobre deep learning (kdnuggets, 2019)

Como era de esperar, tanto Tensorflow como PyTorch lideran las estadísticas, aunque cabe mencionar que al parecer Tensorflow (al menos hasta 2019) ha sido con creces el framework más utilizado y que más contribuciones por parte de la comunidad ha obtenido.

Atendiendo al nivel de crecimiento que han ido teniendo los frameworks, podemos obtener un resultado similar, tal y como se muestra en la figura 2.7.

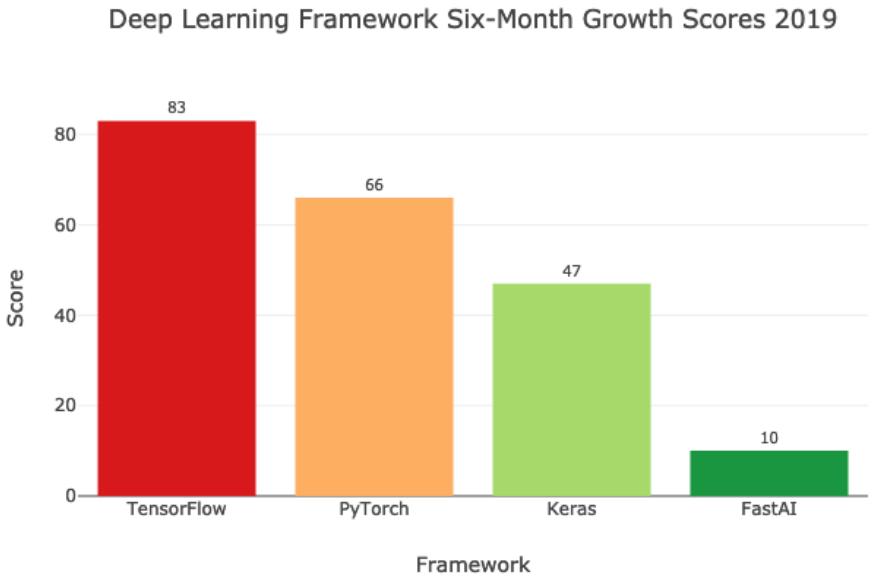


Figura 2.7: Tasa de crecimiento de frameworks de Deep Learning (kdnuggets, 2019)

En base a estos análisis, se deduce que la herramienta a utilizar en el proyecto será Tensorflow, pero que además, por su facilidad de interacción con el core, se empleará Keras, ya que facilitará en gran medida el desarrollo de la red neuronal.

Capítulo 3

Objetivos y metodologías de trabajo

Aunque la idea principal que se desea conseguir con este proyecto ya se ha comentado en las secciones anteriores, en esta sección se explica con mucho más detalle. Se parte de un objetivo general, que se irá desgranando en objetivos con menor alcance que deberán ser logrados en su totalidad para poder cubrir toda la funcionalidad del proyecto.

Una vez identificados los objetivos, cómo se van a acometer las tareas para conseguirlos es un elemento clave. Aunque el proyecto sea desarrollado por una sola persona, la forma de trabajar y la organización puede verse afectada si no se realiza una planificación previa y se define la metodología de trabajo, lo cual se explica en el apartado 3.3.

3.1. Objetivos generales

El presente proyecto tiene un objetivo global que es **crear un modelo predictivo en base a datos obtenidos y depurados de un portal de información deportiva**.

El factor humano en la toma de decisiones implica siempre ciertas limitaciones que deben ser tenidas en cuenta. Las personas por naturaleza son propensas a cometer errores que dependiendo de la tarea que se realiza pueden tener consecuencias catastróficas. Llevándolo al ámbito que se trata en este documento, un *tipster* puede obtener muy buenos resultados en sus predicciones durante un período de tiempo, pero de repente empezar a fallar sin saber por qué y sin haber cambiado su criterio.

Esto se debe a que bajo determinadas circunstancias, ciertos factores cognitivos como la atención pueden verse afectados, lo cual puede producir un error en un cálculo o

simplemente algo por lo que una persona ha pasado por alto sin darse cuenta.

A esto hay que añadirle que por muchos conocimientos que una persona tenga acerca de un deporte, la información con la que cuenta es limitada, no en cuanto a recursos, si no en cuanto a la cantidad de datos que puede utilizar para decidir que un determinado jugador ganará a otro. Además, teniendo en cuenta que esto conlleva pérdidas económicas provenientes de apuestas deportivas que no han sido acertadas, el problema es aún mayor.

Tanto la capacidad de cálculo como el tiempo del que puede disponer un tipster para solucionar un determinado problema (predicción) son bastante limitados, y es el problema principal del que parte el presente proyecto, en el que se intenta solventar esa limitación de procesamiento haciendo uso de la tecnología. Hoy en día cualquier ordenador casero es capaz de realizar operaciones en cuestión de segundos que hace pocos años serían impensables. Además, el acceso a la información está prácticamente al alcance de todo el mundo (lamentablemente no para todos), lo cual hace que todo el proceso de análisis sea replanteado.

Por otro lado hay que añadir que se juegan diariamente numerosos partidos de tenis. Analizarlos todos para poder ofrecer predicciones resulta inviable para un ser humano, tanto por el esfuerzo que ello conlleva como el tiempo que debe ser dedicado. Por ello hay varios tipos de tipsters, cada uno de ellos centrados en un abanico limitado de jugadores o ligas. Este es otro de los puntos que se busca solventar en este proyecto, donde la cantidad de información sobre la que se trabajará será muchísimo mayor, automatizando todo el proceso desde la extracción de datos actualizados, su preprocesado, entrenamiento de un modelo de aprendizaje automático y hasta finalmente la obtención de predicciones.

Dentro de todas las variantes que componen el aprendizaje automático, en este proyecto se va a hacer uso de aquellas en las que se parte de datos históricos previamente etiquetados, es decir, técnicas de aprendizaje supervisado.

El principal objetivo de este proyecto es en primer lugar **crear una base de datos sólida y actualizada** de forma periódica donde se registran datos de partidos ya finalizados cuyo resultado y jugadores son conocidos.

Tras analizar distintos portales web, se ha decidido explotar la información de *Sofascore* ya que contiene una gran cantidad de información actualizada y además puede ser obtenida mediante sus APIs REST de forma pública. Esto es uno de los grandes puntos a su favor, ya que el uso de técnicas de *web scraping* basadas en analizar el código fuente *Html* es más sensible a errores por posibles cambios que puedan darse en el portal. Sin embargo,

mediante el consumo de APIs REST la información es devuelta ya directamente de forma estructurada en formato *Json*, lo cual hace incluso más sencillo su manejo e interpretación. Además, en el caso en que se produjera una actualización de las apis por la inclusión de nuevos campos o endpoints, no supondría ningún problema para los clientes que la consumen, ya que la propia tecnología está construida de forma que no produzca los llamados *breaking changes* en el mundo del desarrollo software.

Una vez la base de datos esté poblada con información, el paso siguiente sería **entrenar un modelo mediante una red neuronal** que sea capaz de aprender y detectar ciertos patrones en dichos datos que serían prácticamente indetectables para un humano. Durante el entrenamiento de dicho modelo el resultado (victoria o pérdida) será la etiqueta y las entradas estarán compuesta por información sobre qué jugadores se enfrentan, si en tierra batida o hierba, número últimos partidos ganados o perdidos... etc.

Dentro de todos los modelos y técnicas disponibles, y en base a los estudios que han sido analizados en el capítulo 2.3, se emplearán redes neuronales, ya que se considera que son las más versátiles y que mayor flexibilidad ofrecen a la hora de realizar las pruebas para obtener el objetivo deseado.

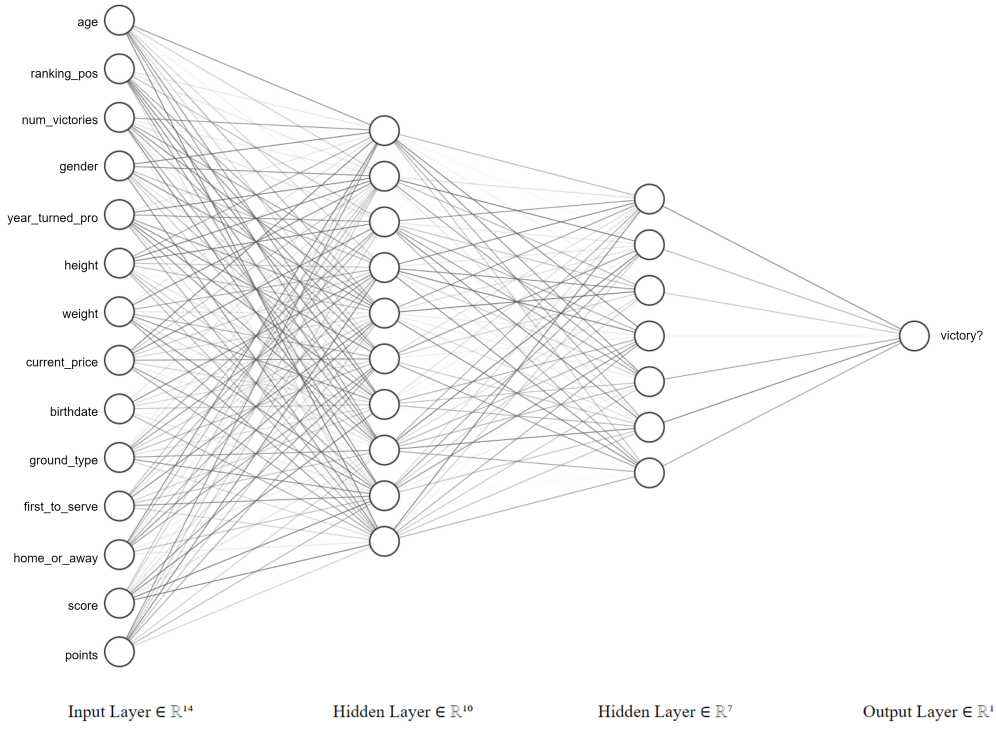


Figura 3.1: Ejemplo de red neuronal (elaboración propia)

Aunque sea extensible en futuras evoluciones del proyecto, en esta primera fase se utilizarán los datos del ránking de la categoría ATP, compuesta por los mejores jugadores del mundo profesional. Esto permitirá acotar en cierta medida la complejidad del desarrollo y poder evaluar su viabilidad antes de pueda extenderse demasiado temporalmente o en esfuerzos.

Finalmente, y con el fin de explotar las predicciones de la red neuronal, se desarrollará un bot sobre la aplicación de mensajería *Telegram* que se encargará de comunicarse con la red neuronal para obtener predicciones de partidos futuros e informarlas a través de un canal de comunicación. Esto permitirá a las personas suscritas a dicho canal realizar apuestas deportivas en cualquiera de las casas de apuestas online que existen para poder lucrarse de ello, tal y como se indica en el *manual de apuestas deportivas* (Sportytrader, 2020).

3.2. Objetivos específicos

Para poder cumplir los objetivos generales nombrados en el apartado anterior, es necesario cubrir los siguientes aspectos de carácter más específico:

- Crear un microservicio para obtener datos actuales de portal deportivo
- Persistir los datos en una base de datos externa
- Automatizar el proceso de obtención de datos y exponerlos datos mediante APIs Rest será otro requisito clave para mejorar la experiencia.
- Leer los datos obtenidos desde un programa en Python para crear un dataset que haga más manejable su uso
- Limpiar los datos, procesarlos para eliminar inconsistencias y convertir variables categóricas en numéricas
- Entrenar un modelo de aprendizaje automático basado en una red neuronal en base a los datos
- Evaluar las predicciones generadas por el modelo entrenado
- Desarrollar un bot para obtener las predicciones a través de una aplicación de mensajería

La primera parte de cualquier proyecto de desarrollo de modelos de aprendizaje profundo es la **obtención de datos**. Cuanta mayor sea la cantidad de información de la que se disponga, mejor funcionará el modelo y mayor capacidad tendrá de extraer patrones y realizar predicciones válidas.

Los datos pueden ser de distintos tipos en función del proyecto que se vaya a desarrollar, imágenes, texto... etc. Para el caso que se trata en este proyecto se necesitan datos estadísticos sobre jugadores de tenis, en concreto, datos relacionados con las características de los jugadores y estadísticas de partidos ya disputados en el pasado. Hay multitud de portales web donde encontrar este tipo de información, por lo tanto, el siguiente paso es evaluar la técnica mediante la cual se obtendrán los datos necesarios para poder almacenarlos en una base de datos.

La primera idea era utilizar técnicas de *web scraping* a través de las cuales un pequeño programa se conecte a ciertos portales y mediante parseo del código *html* sea capaz de extraer dicha información. El problema de esto es que es muy sensible a cambios, ya que una vez el robot obtiene la página que desea analizar hay que indicarle la posición exacta donde se encuentra el dato que queremos, bien sea mediante identificadores en los css o del propio html. Un pequeño cambio en la página podría hacer que el programa dejara de funcionar, o que en la posición del dato que se le indicó ahora hubiera otro tipo de dato totalmente distinto, lo cual provocaría un funcionamiento erróneo del sistema posteriormente.

Una opción más interesante sería el consumo de APIs REST para obtener este tipo de datos, donde la información es devuelta ya directamente de forma estructurada en formato *Json*, lo cual hace incluso más sencillo su manejo e interpretación. Además, en el caso en que se produjera una actualización de las apis por la inclusión de nuevos campos o endpoints, no supondría ningún problema para los clientes que la consumen, ya que la propia tecnología está construida de forma que no produzca los llamados *breaking changes* en el mundo del desarrollo software.

Por esto, se han analizado varios portales y finalmente se ha escogido *Sofascore*, ya que la información que se expone a través de su web es obtenida mediante APIs REST, las cuales son públicas y pueden ser consumidas por cualquier plataforma sin ni siquiera obtener un token. Explorando su página web con las herramientas de desarrollador que proporcionan los navegadores, se puede visualizar fácilmente el uso de apis para mostrar estos datos:

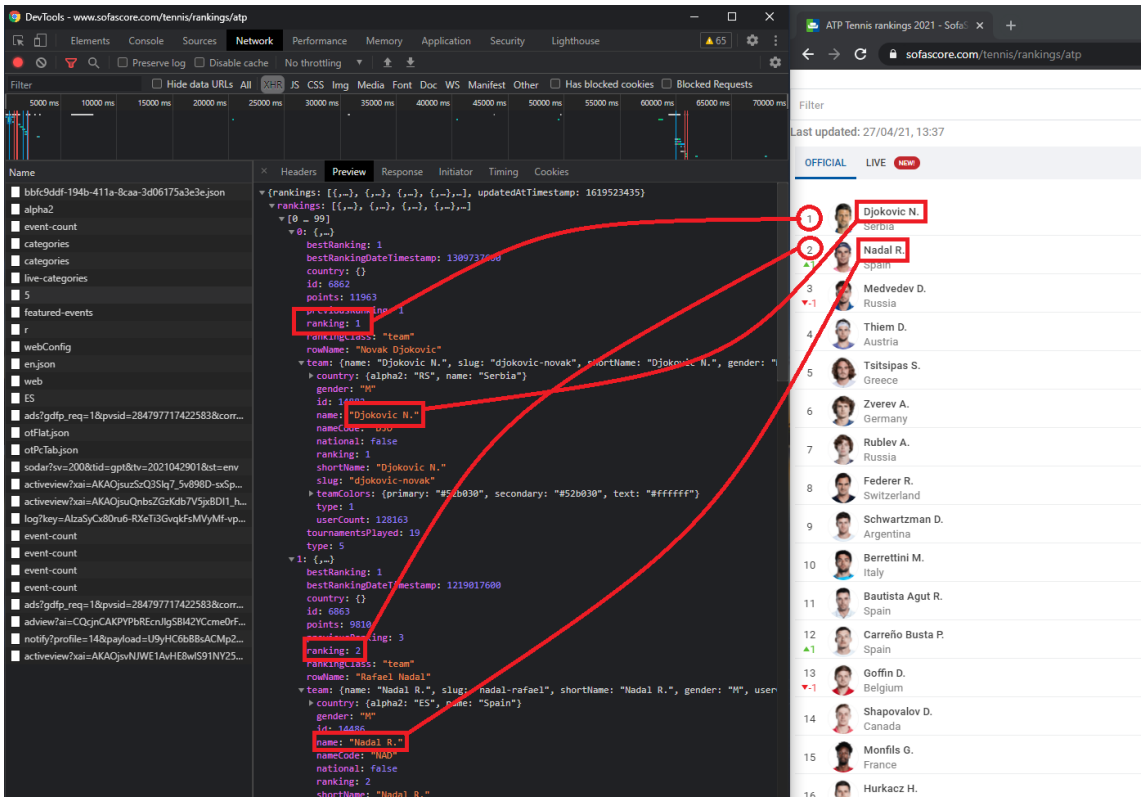


Figura 3.2: Mapeo entre APIs y web de Sofascore (elaboración propia)

De esta forma, es posible ir navegando por cada una de las secciones de la web buscando los datos relevantes que se deben extraer y analizando cada uno de los endpoints que devuelven dicha información. Para ello, será necesario un pequeño programa que haga las llamadas necesarias a los endpoints analizados y guardar la información que devuelvan de forma estructurada en una base de datos.

Esta base de datos será el punto de partida para la siguiente fase del desarrollo del proyecto, donde se extraerán estos datos para entrenar un modelo de aprendizaje profundo mediante redes neuronales.

Una vez se parte de una base de datos sólida y bien estructurada, el siguiente paso es utilizar esos datos para **entrenar modelos de aprendizaje profundo (redes neuronales)** que respondan a los requisitos planteados, en este caso el modelado y caracterización de jugadores de tenis para poder predecir su posibilidad de ganar o perder un partido que aún no ha sido disputado.

Antes de comenzar a explicar cada uno de los pasos, en la figura 3.3 se muestra un diagrama a muy alto nivel de lo que sería el ciclo de vida de un proyecto de estas características:

Figura 3.3: Ciclo de vida de un proyecto de data science (elaboración propia)

Como se observa en la figura 3.3, el primer paso siempre es la adquisición de los datos. La fuente puede tener distintos formatos, normalmente se importan ficheros '.csv' que contienen gran cantidad de entradas, pero esto puede variar. Según se ha planificado el proyecto, los datos podrán ser extraídos de la base de datos y expuestos a través de una simple llamada *http* a un microservicio, el cual devolverá la información en formato json. El modelado de la base de datos se ha realizado de tal manera que el preprocesado posterior sea lo más sencillo posible.

En segundo lugar, habría que analizar con qué tipo de información se va a trabajar en una fase que se denomina 'limpieza' de los datos. Esta fase se compone de una serie de operaciones sobre los datos donde se analizan si determinados valores vienen o no informados, el formato de los mismos, etc... El objetivo de esta 'limpieza' es realizar un filtrado donde a la salida se obtenga una versión de los datos consistente que pueda ser realizada para el entrenamiento de un modelo.

Además, antes de proceder con el entrenamiento, conviene realizar una exploración y análisis estadístico de los parámetros con los que se trabaja, atendiendo especialmente a la correlación de cada uno de ellos con la variable 'target' o a predecir. Esto puede suponer que varios de ellos puedan ser eliminados del conjunto de datos porque no aporten valor o puedan influir negativamente en el resultado.

Una vez los datos han sido procesados, se procede con el entrenamiento del modelo. En este caso se va a utilizar una red neuronal recurrente. La selección de hiperparámetros será una tarea necesaria para que la red se comporte de forma adecuada, por ejemplo la selección del número de capas ocultas, número de neuronas por capa, funciones de activación... Esta fase supone también una pequeña parte de prueba y error, donde se realizarán varios entrenamientos hasta conseguir los resultados óptimos. La validación de los resultados será el paso donde se tenga que tomar la decisión de si el modelo es lo suficientemente robusto como para darlo por válido.

Para finalizar, conviene realizar una visualización de los resultados obtenidos. Hay varias librerías con las que es muy sencillo desarrollar gráficas que proporcionan gran cantidad de información. En este tipo de gráficas se puede realizar una simulación de predicción por ejemplo, donde se analice si la red neuronal ha acertado o no con el resultado, y en caso negativo mostrar el margen de error para tener obtener una idea de cuán fiable puede llegar a ser.

El proceso de obtención de datos y entrenamiento de la red neuronal no son complicados de realizar pero para que funcione correctamente es necesario que siempre se encuentren actualizados. A medida que pasan los días se juegan nuevos partidos de tenis y los jugadores del ránking de ATP suben o bajan posiciones, lo cual podría afectar a las predicciones del modelo. Una primera opción es ejecutar cada uno de los programas manualmente cada cierto tiempo, pero esto incumpliría uno de los objetivos del proyecto, que es el evitar la intervención humana.

Para ello, otro de los objetivos es **automatizar el proceso de obtención de datos y entrenamiento de la red neuronal**. Para ello, la idea es desarrollar una serie de *cronjobs* que ejecuten el programa de extracción de información de *Sofascore* de forma automatizada. Esto permite decidir la frecuencia con la que los datos deben ser actualizados, y en una primera instancia se podría pensar que cuanto más frecuente sea, mejor, pero esto también supondría una mayor carga para el sistema, y en la primera versión del desarrollo tampoco se quiere invertir una gran cantidad de dinero en servidores y topologías de red que permitan balanceo y desplegar varias instancias.

Es necesario poner una balanza y acotar el margen de carga/beneficios. Como en el caso en el que se trabaja en un primer momento es el de obtener los datos de los jugadores del ránking, bastaría con realizar esta actualización una vez al día, donde se asegura que todos los partidos han finalizado.

Una vez finalice el proceso de actualización de la base de datos con la nueva información, el siguiente paso es 'avisar' al proceso de entrenamiento de la red neuronal de que hay nuevos datos y es necesario realizar un nuevo entrenamiento. En el momento en el que se escriben estas líneas, aún no se ha desarrollado la red neuronal, por lo que es imposible saber cuánto tardaría en ser entrenada, lo cual es un dato muy a tener en cuenta, pero la idea es que se produzca un nuevo entrenamiento cada cierto tiempo para que las predicciones siempre se hagan sobre datos lo más actuales posibles.

3.3. Metodologías de trabajo

En los proyectos de desarrollo software hay infinidad de formas de trabajar. No hay ninguna mejor o peor, si no que la selección de una u otra depende del tipo del proyecto o de sus requisitos, el tamaño del equipo, etc... Una de las metodologías más utilizadas actualmente en proyectos de desarrollo software es *Scrum*, la cual fomenta el trabajo de forma 'ágil' y define una serie de ceremonias que deben cumplirse durante el ciclo de vida del proyecto, como el *sprint refinement*, *sprint planning* o incluso reuniones diarias de unos quince minutos en las que se comenta el trabajo actual por cada miembro del equipo llamadas *dailies*. En el caso de desarrollo individual, no tiene mucho sentido aplicar metodología ágiles de este tipo, ya que se carece de personas con distintos roles como *Scrum master* o los *stakeholders*. En este caso, al ser un proyecto con un carácter más de investigación en el que solamente va a trabajar una sola persona como desarrolladora, carece de sentido realizar ciertas ceremonias como las *dailies* por ejemplo, pero no por ello hay que dejar de ser organizado.

Algo que comparten la mayoría de las metodologías es la generación de un tablero donde se identifican cada una de las tareas que deben ser realizadas durante la duración del proyecto. Este tablero está compuesto por varias columnas que identifican el estado en el que se encuentran las distintas tareas, de forma que con un simple vistazo se pueda obtener la información necesaria para identificar el estado general en el que se encuentra el desarrollo del proyecto.

Para el desarrollo de este proyecto, se ha creado un tablero en el que se darán de alta cada una de las tareas identificadas y que se irá actualizando conforme se van avanzando los distintos hitos del desarrollo. A continuación se muestra el estado en el que se muestra el proyecto en el momento en el que se escriben estas líneas:

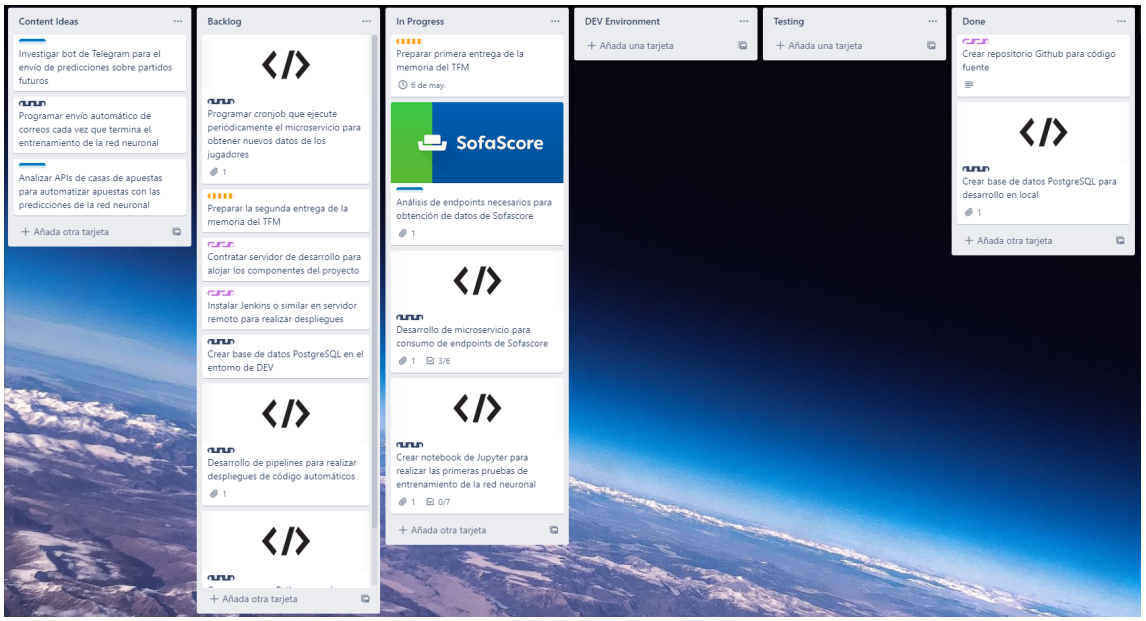


Figura 3.4: Tablero con el estado de las tareas del proyecto (elaboración propia)

Las columnas que se están empleando para definir el estado en el que se encuentra cada tarea son:

- Content ideas: Contiene ideas que no son prioritarias pero que en caso de que el desarrollo del proyecto se finalice a tiempo estaría bien implementar, bien porque aportan un valor añadido o simplemente por 'refinar' el desarrollo ya realizado.
- Backlog: Contiene cada una de las tareas que será necesario realizar para finalizar el proyecto pero que aún no han sido comenzadas.
- In Progress: Aquí se encuentran las tareas que se encuentran en desarrollo. Si una tarea contiene subtareas, no se dará por finalizada hasta que todas sean terminadas.
- DEV Environment: Esta columna solo aplica para aquellas tareas que suponen desarrollo software. Una vez el desarrollo se da por finalizado, deberá ser desplegado en un entorno diferente a local donde se ejecutará de forma continua e integrará con el resto de piezas.
- Testing: Como en cualquier proyecto de desarrollo software, es necesario realizar pruebas. Hay distintos tipos, pero en esta columna se almacenarán las pruebas de integración necesarias para que todos y cada uno de los microservicios a desarrollar funcionen y se integren correctamente en el entorno de desarrollo.

- Done: Una vez las tareas han pasado las pruebas necesarias (las que lo necesiten), se pueden dar por finalizadas y se moverán a esta columna. Por otro lado, aquellas que sean de gestión o de análisis, podrán ser movidas aquí una vez se terminen.

Algo que no se ve directamente reflejado en el tablero pero que es importante mencionar, es que una tarea puede contener varias subtareas. En el tablero se verá siempre la tarea global que en función de su tamaño puede o no ser dividida. Por ejemplo, si se abre el detalle de la tarea *'Desarrollo de microservicio para consumo de endpoints de Sofascore'*, se puede observar que está compuesta por las siguientes subtareas:

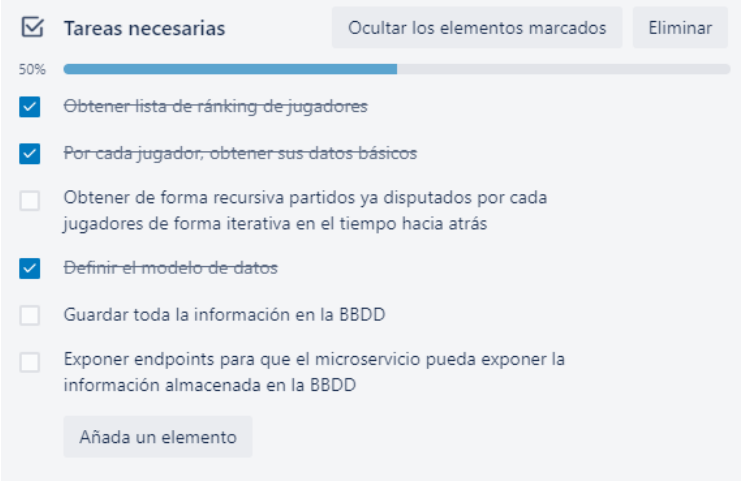


Figura 3.5: Subtareas que componen una tarea de desarrollo (elaboración propia)

En la figura 3.5 se puede observar que el 50 % de las subtareas ya han sido realizadas, y hasta que no se finalice el otro 50 % restante no se podrá desplegar el programa en el entorno de desarrollo.

Conviene también comentar que para evitar olvidar entregas, las tareas relativas a la documentación del proyecto o incluso a esta memoria están identificadas en el tablero. Cada tipo de tarea tiene un color asociado y en total se han definido las siguientes:

- Memoria: Tareas relativas a la documentación de este proyecto
- Gestión: Agrupará tareas relacionadas a gestión de recursos en el proyecto, como contratar un servidor remoto para implementar la infrastructura del entorno de desarrollo por ejemplo, o la creación de los repositorios necesarios en Github para alojar el código fuente.

- Análisis: Con esta etiqueta se identifican tareas que requieren un análisis previo o cuyo alcance aún no está claro como para empezar su desarrollo.
- Desarrollo: Esta etiqueta relaciona a todas las tareas que requieren de desarrollo software.

Capítulo 4

Identificación de Requisitos

Una vez identificados los objetivos, tanto a alto como a bajo nivel, el siguiente paso es identificar qué tareas serán necesarias acometer para cumplir dichos objetivos. En este apartado se explican en detalle cada uno de los requisitos que se han identificado. El orden de los requisitos será cronológico, es decir, se comienza explicando aquellos que deben ser acometidos en primer lugar y que son necesarios para poder continuar con los siguientes.

4.1. Preparación del entorno de desarrollo

Antes de comenzar a programar la solución, es necesario decidir qué entorno de desarrollo se va a utilizar para alojar los componentes software desarrollados y su ejecución. En una primera instancia, se pensó contratar un servidor VPS dedicado donde serán desplegados los microservicios, pero esto conllevaría un gran coste si se quiere utilizar una máquina cuyos recursos sean adecuados a la solución. Finalmente se ha optado por utilizar la suscripción que la universidad ofrece para estudiantes en la plataforma de Azure. Microsoft Azure se trata de una plataforma del tipo *PaaS (Platform as a Service* que ofrece todos los elementos infrastructurales que sean necesarios con un par de clicks. Ofrece infinidad de posibilidades en cuanto a configuración de los recursos de los componentes a utilizar y como es de esperar, cuanto más recursos, mayor es el coste, pero esto no es un problema ya que la suscripción de estudiantes ofrece un año con acceso a las herramientas básicas de la plataforma sin coste adicional.

Una vez decidida la plataforma de hosting del software, el resto de elementos a utilizar para el desarrollo es tarea más sencilla. Basta con un PC con un rendimiento medio, conexión a internet, y un repositorio de código donde poder llevar el versionado y control

de cambios del código fuente.

De este modo, la lista completa de herramientas de desarrollo quedaría como se indica a continuación:

- Suscripción de estudiantes Microsoft Azure
- Conexión a internet
- Ordenador de gama media con el que poder desarrollar de forma local
- Repositorio de código fuente para control de versiones (git)

Estos requisitos son la base para poder comenzar con el proyecto. Si no se cumplieran, no sería posible acometer el proyecto.

4.2. Extracción de información

La primera tarea en un proyecto de inteligencia artificial es recolectar los datos necesarios que serán posteriormente utilizados para entrenar un modelo mediante aprendizaje máquina. Como se ha comentado en apartados anteriores, se va a realizar una explotación de las APIs públicas de Sofascore para obtener toda la información necesaria sobre los jugadores de tenis que se quiere modelar.

En primer lugar, debe hacerse un análisis de qué información nos puede proporcionar el portal web y un breve esquema con los datos básicos que se quiere obtener y que aportarán valor a las predicciones de la red neuronal. En la figura 4.1 se observa la información básica que el portal aloja sobre los jugadores de tenis:

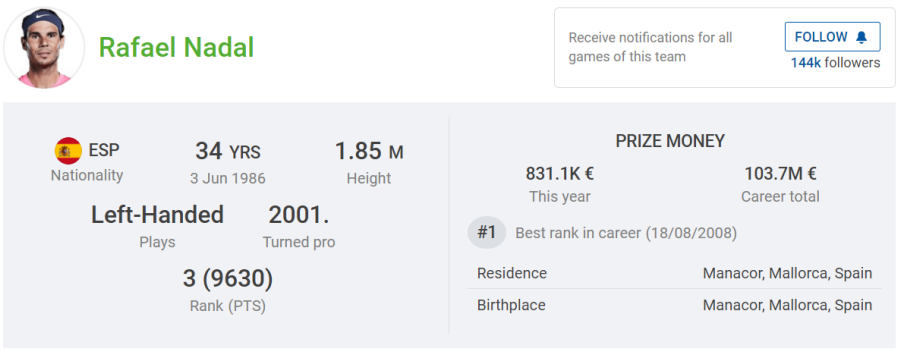


Figura 4.1: Tarjeta de presentación de jugador de tenis (Sofascore, 2021)

Estos datos son candidatos a ser obtenidos, ya que información como la posición actual en el ranking o el valor de cotización del jugador pueden ser muy influyentes a la hora de realizar una predicción sobre resultados.

Además, no solo nos interesan los datos del jugador, si no cómo ha sido su rendimiento en los últimos partidos. Esto también tendrá gran importancia para la red neuronal, ya que básicamente lo que se quiere obtener como salida es la capacidad de victoria o derrota en base a datos históricos de partidos ya disputados, por lo tanto, la siguiente tarea es descubrir cómo aloja el portal esta información. En la figura 4.2 se observa la información detallada de un partido en Sofascore:

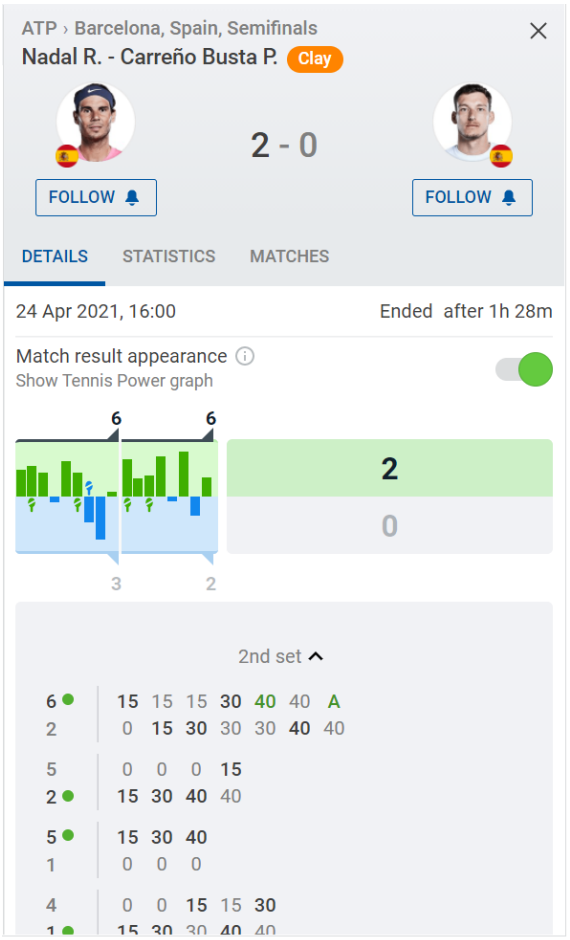


Figura 4.2: Estadísticas sobre partidos de tenis (Sofascore, 2021)

Como se puede observar en las imágenes anteriores, el portal proporciona lo que a priori es información suficiente para cubrir las necesidades del proyecto. Tal y como se comentaba en secciones anteriores, la idea es consumir las APIs rest del portal para obtener los datos

necesarios. Para poder cumplir este requisito, se debe realizar un análisis del tráfico web del portal y de cómo obtiene los datos para mostrarlos a nivel de *frontend*. Esta tarea es bastante sencilla si se hace uso de las herramientas para desarrolladores que los navegadores web suelen traer embebidas. En este proyecto se ha utilizado Google Chrome, y mediante la tecla F12 se puede abrir la consola donde es fácil identificar los recursos a los que llama el portal si se selecciona la pestaña *Network*. La clave es identificar qué recursos son los que devuelven la información que se ha comentado en esta sección.

En primer lugar, de todos los recursos a los que llama, hay que identificar cuál devuelve los datos que se buscan. En la figura 4.3 se muestra cómo funciona este análisis:

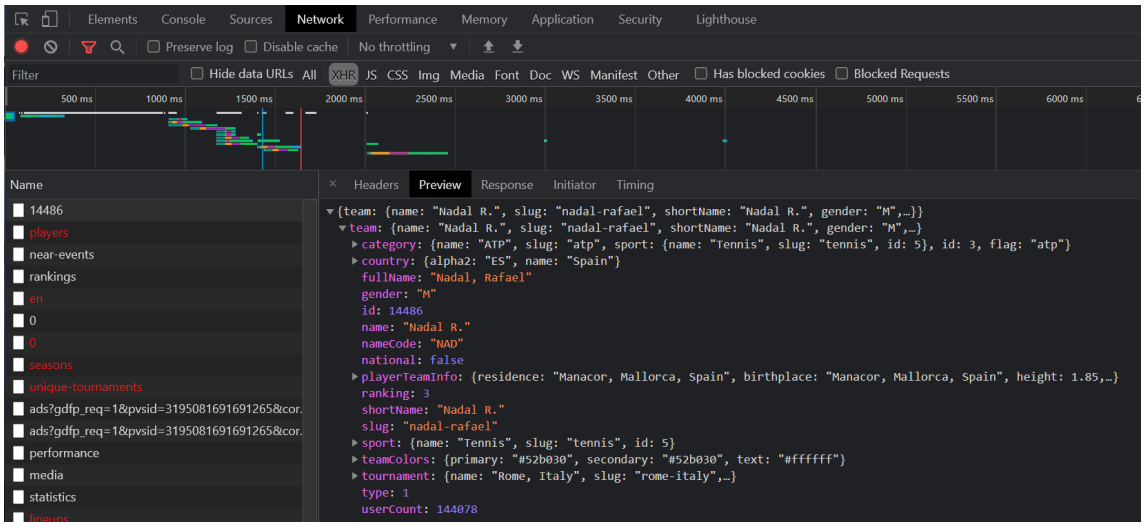


Figura 4.3: Ejemplo de respuesta de API (Sofascore, 2021)

Una vez identificado, para obtener el nombre del recurso basta con hacer click sobre la pestaña *Headers*. En ella se puede observar la llamada http que se ha realizado y todas las cabeceras que han sido enviadas y recibidas a través de dicha llamada. En la figura 4.4 se muestra la petición http que devuelve la información de la figura 4.3.

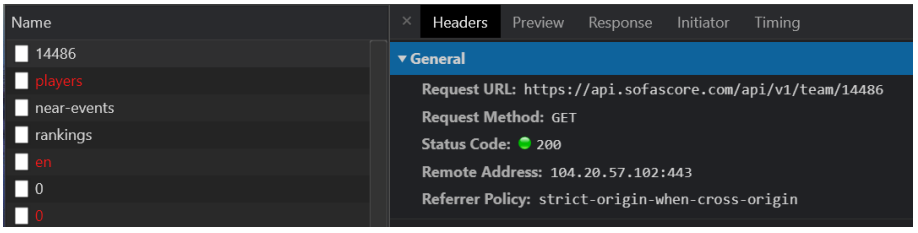


Figura 4.4: Ejemplo de llamada a API (Sofascore, 2021)

En la sección 5 de este documento se analizará todo este proceso mucho más en detalle, así como la solución técnica desarrollada para obtener esta información de forma recurrente y mantener una base de datos actualizada siempre.

4.3. Definición de la arquitectura del sistema

En este apartado se definen los requisitos de arquitectura que requiere la solución del proyecto. El objetivo de este requisito es proporcionar los esquemas o planos que servirán como referencia para el desarrollo.

En primer lugar, es preciso contar con que este proyecto es un piloto, que en función de los resultados que proporcione tendrá futuros desarrollos que hagan la solución más robusta y mucho más ambiciosa, por lo tanto, la arquitectura debe de ser escalable.

Para ello, se ha optado por una solución basada en microservicios donde la funcionalidad de cada uno de los componentes esté bien definida y pueda funcionar de manera desacoplada. En el artículo *Architectural Patterns for Microservices: A Systematic Mapping Study* (Taibi y col., 2018) se hace un análisis muy detallado de este tipo de arquitecturas y de cuáles son sus principales ventajas y desventajas.

4.3.1. Identificación de componentes a desarrollar

Una vez comentado el tipo de arquitectura falta por desglosar los componentes que proporcionarán la funcionalidad completa end-to-end. Como se ha comentado en la sección de objetivos específicos, podemos diferenciar el proyecto en tres grandes bloques, los cuales albergarán los distintos componentes software y otros elementos tales como bases de datos que en su conjunto ofrecerán la funcionalidad completa:

1. Obtención de información: Este bloque consiste en desarrollar una solución software que sea capaz de obtener información de un portal web y almacenarla en una base de datos. Esta obtención de información debe hacerse automáticamente sin ningún tipo de intervención humana y ejecutarse periódicamente una vez al día.

    - Sofascore Data Miner: Elemento encargado de obtener la información de las distintas APIs de Sofascore, parsearla, unificarla y almacenarla según la definición del modelo de datos.

- Base de datos PostgreSQL: Base de datos sobre en la que se persistirá la información obtenida por el microservicio.
- Cronjob: Será el encargado de ejecutar periódicamente el servicio encargado de obtener la última información de Sofascore.

2. Red neuronal: Constará de un programa escrito en Python que debe contener las siguientes funcionalidades:
   - Carga de datos: Mediante llamadas a APIs expuestas por el microservicio del apartado anterior, obtendrá los datos actualizados con información sobre infinidad de jugadores de tenis y los últimos partidos disputados
   - Tratamiento de los datos: Se realizará un análisis de los datos obtenidos (valores nulos, etc...) de forma que la entrada a la red neuronal contenga la información con el mayor valor posible
   - Entrenamiento de la red neuronal: En base a la información ya tratada, se dividirá en conjunto de test y conjunto de entrenamiento y se procederá al entrenamiento de una red neuronal.
   - Validación: Se validarán los resultados obtenidos por la red neuronal en base a las métricas típicas como la precisión.
   - Predicción: Será el objetivo final que se quiere conseguir con la red neuronal. En base a datos de entrada similares a los que se han utilizado para entrenarla, la idea es poder predecir resultados de partidos que aún no han sido disputados.
3. Bot de Telegram: Pequeño microservicio que será el encargado de realizar llamadas a la red neuronal con información de partidos futuros para así poder obtener las predicciones, las cuales serán compartidas a través de un canal seguro de comunicación a través de la aplicación de mensajería *Telegram*.

4.3.2. Despliegue del software en un entorno productivo

Una vez el desarrollo del componente software ha sido validado y finalizado, el siguiente paso es desplegarlo en un entorno productivo. Normalmente, como en cualquier proyecto de desarrollo, hay varios entornos intermedios entre el equipo local del desarrollador y el productivo. En la figura 4.5 (Wavemaker, 2020) se observan los distintos entornos por los

que pasa un artefacto software desde el momento en el que el desarrollador termina de escribir el código fuente hasta su llegada a un entorno productivo:

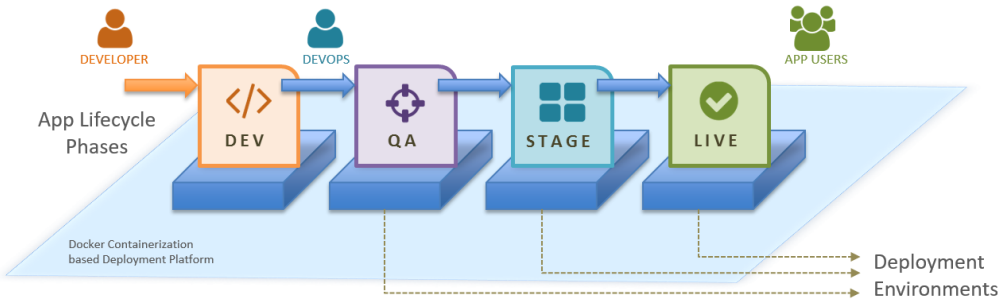


Figura 4.5: Gestión de releases entre entornos (Wavemaker, 2020)

En este modelo de entornos, el software debe ser validado en el entorno previo antes de poder pasar al siguiente. Las validaciones van desde pruebas unitarias y de integración, pruebas de carga, o hasta pruebas de seguridad (pentesting). Sin haber pasado por todas estas validaciones, el software no estará listo para alcanzar el entorno productivo.

- DEV: A pesar de que el software se ejecuta en un entorno distinto de la máquina local de los desarrolladores, se le denomina entorno de desarrollo ya que aquí se realizan las primeras pruebas. En el caso en el que un sistema esté compuesto por varios componentes software, se comprueba la integración entre todos ellos para asegurar el correcto funcionamiento.

- QA: Se le denomina entorno de pruebas. Normalmente los desarrolladores dejan de tener acceso a este entorno, y es el equipo de QA (en algunos casos incluso el cliente final) el que se encarga de realizar toda serie de pruebas funcionales, bien sea de forma automatizada o de forma manual.

- STAGE: Este entorno se encuentra justo en la fase previa a producción. Debe de ser a nivel de infraestructura y recursos exactamente igual que el entorno productivo. Esto es así para asegurar que el software funciona correctamente con esa configuración, de modo que cuando se decida pasarlo al entorno productivo el funcionamiento sea el esperado, ya que ha sido comprobado en un entorno igual previo. Además, en este entorno suelen tener lugar los tests de penetración, también denominados

como *hacking ético*, donde un equipo externo de ciberseguridad comprueba que no existan vulnerabilidades críticas, ya que en ese caso se bloquearía el pase al entorno productivo.

- LIVE: Entorno final productivo. Aquí es donde funcionará el software y será consumido por los usuarios finales. Para llegar a este entorno, deben de haberse pasado de forma satisfactoria todas las pruebas realizadas en los entornoa previos.

Sin embargo, dado que estamos en un ámbito donde los recursos son limitados (tanto por tiempo como económicos) y puesto que el objeto de este proyecto es en una primera fase estudiar la viabilidad del modelo de predicciones, se utilizará únicamente un entorno remoto distinto al entorno local de desarrollo, donde se realizarán las pruebas y validaciones. Las características de dicho entorno pueden ser de distinta índole, ya sea un servidor VPS remoto, un entorno cloud dockerizado, una máquina linux, windows...

Ya que la universidad UNIR ofrece a sus estudiantes cuentas de prueba para la infraestructura de Microsoft Azure, esta será la elección para el desarrollo y despliegue del proyecto. Aquí se crearán los componentes necesarios para que el software sea ejecutado correctamente y proporcione un nivel de servicio constante, y a ser posible de 24x7. En la figura 4.6 se observan los distintos componentes que han sido configurados para este proyecto:

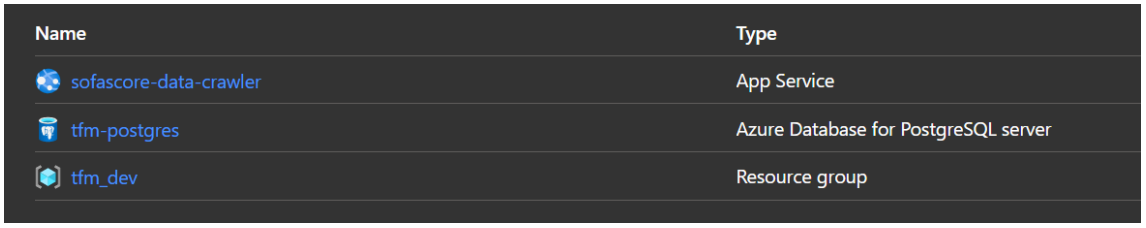


Figura 4.6: Recursos configurados en la infraestructura de Azure (elaboración propia, 2021)

Una vez todo el entorno está configurado, el siguiente paso es desplegar el software. Esto se puede hacer de distintas maneras, tradicionalmente se generaba el artefacto, y se copiaba manualmente accediendo a la máquina remota mediante consola de comandos. A día de hoy existen técnicas que automatizan todo este proceso y lo hacen mucho más fácil y ágil para los desarrolladores. En este proyecto se han implementado técnicas de CI/CD (*Continuous Integration & Continuous Delivery*) mediante pipelines que se encargan de realizar el despliegue de los artefactos software.

Las técnicas de CI/CD se pueden definir como un método para distribuir aplicaciones a los clientes con frecuencia mediante el uso de la automatización en las etapas del desarrollo de aplicaciones. Los principales conceptos que se atribuyen a la CI/CD son la integración continua, la distribución continua y la implementación continua, (Redhat, 2021).

Por otro lado, una pipeline no es mas que la definición de las acciones que ocurren durante el despliegue del artefacto software. Normalmente, antes de finalizar un despliegue una pipeline se encarga de ejecutar tests unitarios y de integración para validar que el software funciona correctamente. Si esto es así, se procede con el despliegue, de lo contrario será bloqueado y el desarrollador tendrá que corregir los errores antes de poder desplegar de nuevo. Existen numerosas herramientas para el desarrollo de pipelines, como el archiconocido *Jenkins*, sin embargo, requiere de una instalación propia SaaS (*Software As a Service*. Puesto que en este proyecto todo el código fuente se aloja en repositorios de *Github*, se explorarán las opciones que brindan las denominadas *Github Actions*, ya que son una gran novedad en el ecosistema y su misión es facilitar el desarrollo de pipelines para la automatización de procesos.

De este modo, el despliegue en Azure mediante pipelines de Github Actions quedaría como se indica en la figura 4.7:

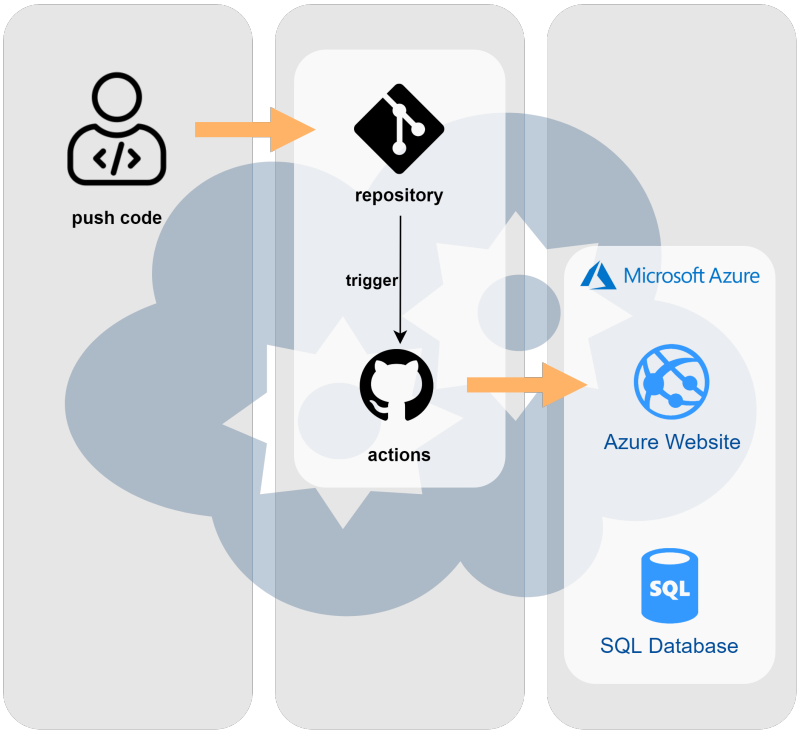


Figura 4.7: Proceso CI/CD (elaboración propia, 2021)

De este modo, cada vez que se realiza un *commit* sobre la rama *master* de git, la pipeline será lanzada automáticamente y la nueva versión del software será desplegada y estará disponible en Azure, proporcionando así una gran agilidad durante el proceso de desarrollo.

4.4. Creación de una red neuronal para predicciones futuras

Alcanzado este punto, ya se ha conseguido implementar una base de datos consistente con toda la información necesaria para poder entrenar una red neuronal. Para ello, se desarrollará un pequeño programa en Python que contendrá las siguientes funcionalidades.

4.4.1. Obtención de los datos

Como se ha comentado en el apartado 4.2, el microservicio encargado de la obtención de los datos expone un pequeño endpoint mediante una API Rest a través del cual estos pueden ser obtenidos por un sistema externo. Por lo tanto, el primer paso del script de Python será realizar una llamada http a dicho endpoint para recuperar los datos en formato Json.

A pesar de que el microservicio ya devuelve los datos de una forma estructurada, a la hora de trabajar con redes neuronales haremos uso de librerías como *pandas*, ya que facilitan enormemente el análisis de información. Por ello, el primer paso será convertir el formato de información que se obtiene mediante el microservicio a algo que se pueda manejar de forma sencilla, por ejemplo un dataframe sobre el cual se podrán hacer las operaciones básicas y típicas en un proyecto de análisis de datos.

4.4.2. Preparación de los datos

Este paso es necesario en cualquier proyecto de aprendizaje máquina. Bien es cierto que la información que se ha guardado en la base de datos ya está muy filtrada y estructurada, pero es posible varios valores vengan como nulos o vacíos, el formato de ciertas variables no sea el deseado, o simplemente que ciertos parámetros no sean buenos candidatos para entrenar la red neuronal, ya que no porporcionan gran información de valor de cara a las predicciones.

Debido a que se va a tratar el problema como un problema de regresión, el primer paso es convertir las variables categóricas a numéricas. Para ello habrá que establecer un

mapeo y asignar valores numéricos a cada una de las distintas categorías que pueda tomar un valor. Por otro lado, hay que examinar si algunas de las variables son prescindibles de cara a la predicción, ya posiblemente el valor que aporten sea nulo y eso podía provocar comportamientos erróneos o directamente restarle precisión a la red neuronal que se quiere construir. Por último, hay que realizar un análisis global de los valores con los quedamos, ya que posiblemente muchos de ellos no vengan informados. Aquí será importante decidir si esa entrada se elimina o el valor se sustituye por otro como la media entre todos los valores por ejemplo.

4.4.3. Proceso de entrenamiento

El entrenamiento de la red neuronal es la fase clave del proyecto. Para ello los datos deben estar lo más 'limpios' posible, ya que este será uno de los factores principales que determinen el funcionamiento de la red.

En primer lugar, debe realizarse la división entre el conjunto de entrenamiento y el conjunto de validación. Para este proyecto se emplean los valores estándar de un 80 % de los datos para el entrenamiento y el 20 % restante para validar.

Por otro lado, debe seleccionarse la técnica o el modelo a utilizar para la red neuronal. En este proyecto, se van a emplear diversos modelos para comprobar cuál es el que mejor funciona de acuerdo al problema que se quiere solucionar. Además, por cada modelo deberá realizarse una exploración de los hiperparámetros para escoger los más adecuados al caso que trata este proyecto, ya que la elección de unos u otros puede afectar en gran medida al rendimiento de la red neuronal.

4.4.4. Validación y pruebas

Una vez la red neuronal ha sido entrenada, el siguiente paso es utilizar el 20 % de los datos que se obtuvieron en el apartado anterior para validar su funcionamiento. Puesto que se el requisito a desarrollar es un problema de regresión, las métricas a las que ha de atenderse son las siguientes:

- MSE (*Mean Square Error*): Se define como la media entre el valor real y el valor predicho o estimado al cuadrado:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2$$

- MAE (*Mean Absolute Error*): Se define como la diferencia en valor absoluto entre el valor real y el valor predicho:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - y_i|$$

- RMSE (*Root Mean Square Error*): Se define como la raíz cuadrada de la media de la diferencia entre el valor real y el valor predicho o estimado al cuadrado:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2}$$

Estas medidas proporcionan una idea de la desviación de las predicciones de la red neuronal. No obstante, también se atenderá a métricas propias de este tipo de algoritmos como la precisión y se deberán mostrar gráficas para ver la evolución del entrenamiento y así poder detectar problemas como *overfitting*.

Capítulo 5

Descripción de la herramienta software desarrollada

En esta sección se explicará con detalle todo el proceso de desarrollo de cada uno de los componentes software del proyecto. Se comenzará presentando el diseño técnico, que permitirá obtener una visión completa de la arquitectura y funcionamiento del sistema, y seguidamente se irá haciendo hincapié en cada una de las partes describiendo desde un punto de vista muy técnico cómo ha sido desarrollada.

5.1. Diseño técnico

A continuación se explica a nivel técnico la arquitectura de cada uno de los componentes y la funcionalidad que brindan. Además, se explican las tecnologías empleadas para su implementación y el motivo de su elección.

5.1.1. Microservicio para obtención de datos

La obtención de los datos necesarios para entrenar la red neuronal es una de las partes más importantes de este proyecto, ya que la disposición de unos datos de calidad es la clave del funcionamiento correcto de cualquier modelo de predicciones en proyectos de *machine learning*.

Para la implementación de la lógica de negocio de este microservicio se han empleado tecnologías actuales como Spring Boot junto con la versión 11 de Java y Maven como gestor de dependencias. Se han elegido estas tecnologías por su facilidad a la hora de

incluir nuevas librerías para realizar llamadas http y la creación de recursos REST para exponer información a sistemas terceros.

Además, se han seguido un modelo de arquitectura escalable que permita incluir nuevas funcionalidades en el código fuente y que soporten retrocompatibilidad. El desarrollo parte por la implementación de los controladores, y continúa por la capa de servicio hasta llegar a la capa de negocio, donde se encuentra el acceso al modelo de datos, tal y como se detalla en el artículo *Servicio Web API REST sobre el Framework Spring, Hibernate, JSON Web Token y BBDD Oracle* (Monago Ruiz, 2019).

La figura 5.1 muestra la definición arquitectura empleada para el desarrollo de este microservicio:

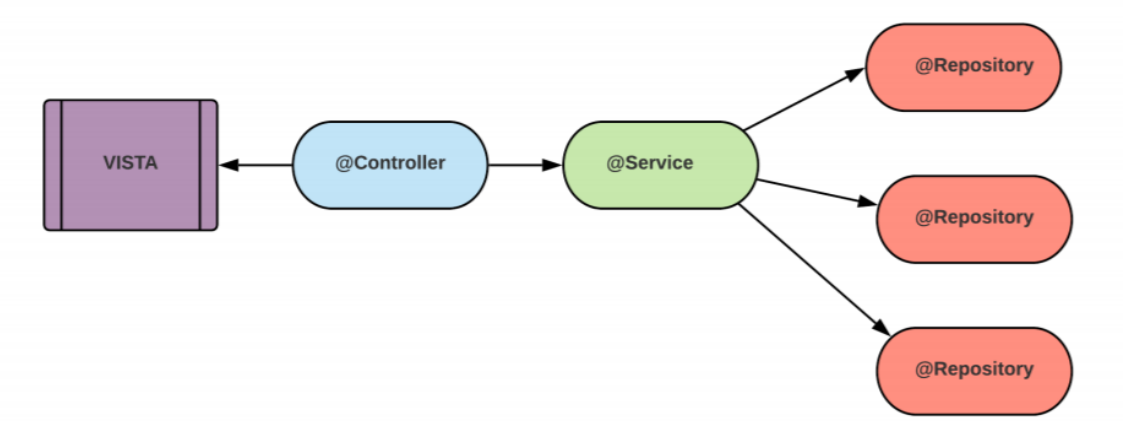


Figura 5.1: Patrón de desarrollo MVC (Monago Ruiz, 2019)

Este componente cubre básicamente dos funcionalidades:

1. Obtención de datos actualizados de Sofascore: En primer lugar, se encarga de consumir las APIs públicas que el portal de deportes expone para obtener la información necesaria de cara a entrenar la red neuronal posteriormente. Este proceso ocurre de forma automatizada mediante un *cronjob* que se ejecuta todas las noches a las 23:55 horas, ya que es cuando se puede asegurar que los partidos del día han finalizado y se pueden obtener los resultados de los mismos. Para ello, la primera tarea es identificar los endpoints de Sofascore que permiten obtener dicha información, que son los siguientes:

```
Url base: https://api.sofascore.com/api/v1
```

- `GET /category/3/scheduled-events/{FECHA-EVENTO}`: Esta petición devuelve la lista de eventos programados para un día en concreto que se informará como un parámetro en el*path* en formato yyyy/MM/dd, y puede ser tanto una fecha pasada como futura. Como es de esperar, los eventos de las fechas pasadas habrán finalizado e informarán los resultados de los partidos, mientras que los futuros estarán aún por disputar (estos serán los que se tendrán que predecir). Además, devuelven la información básica de los jugadores que se enfrentan en cada partido. La estructura de la información que es devuelta en la respuesta se indica en la tabla 5.1:

Variable	**Descripción**
firstToServe	Determina qué jugador saca
tournament	Información general sobre el torneo, localización... etc
roundInfo	Ronda que se disputa
customId	Identificador único del evento
status	Identifica si el partido ha finalizado, ha sido aplazado... etc
winnerCode	Determina cuál de los dos jugadores es el vencedor
homeTeam	Información general sobre el jugador local
awayTeam	Información general sobre el jugador visitante
homeScore	Puntuación por set del jugador local
awayScore	Puntuación por set del jugador visitante
time	Duración del partido
id	Identificador del partido
startTimestamp	Fecha y hora del comienzo del partido
slug	Descriptor que indica los jugadores que se enfrentan
groundType	Formato del terreno de juego, hierba, tierra batida... etc

Tabla 5.1: Información obtenida de Sofascore sobre eventos

- `GET /team/(ID-JUGADOR)`: Devuelve la información completa del jugador en base a su identificador, el cual va configurado de igual manera mediante una variable de tipo numérica en el *path*. Estos ids son conocidos ya que se han obtenido en la anterior llamada, de forma que se pueden reutilizar para consumir

este nuevo endpoint. Los datos que devuelve este recurso se describen en la tabla 5.2:

Variable	**Descripción**
name	Nombre de pila del jugador
slug	Nickname para identificar el jugador
shortName	Nombre corto del jugador
gender	Género del jugador
sport	Deporte que practica (siemore será tenis)
category	Categoría en la que juega el jugador (ATP)
tournament	Torneo en el que juega
playerTeamInfo	Información detallada (país de residencia, altura, peso, precio, mano de juego, cotización, fecha de nacimiento... etc)
nameCode	Iniciales del nombre
ranking	Posición actual en el ranking
id	Identificador único del jugador
country	País
fullName	Nombre completo
teamColors	Códigos de color del equipo

Tabla 5.2: Información detallada sobre jugadores

Este microservicio hará uso de los dos endpoints que se acaban de explicar para obtener toda la información detallada. La idea es, en primer lugar, obtener la lista de partidos pasados que ya han sido disputados. Esta lista devuelve información acerca del partido, pero falta por completar la información de los jugadores, de modo que se iterará sobre esa lista para completar la información de cada uno de los jugadores que juegan en ese partido mediante el segundo endpoint. En resumen, se hace una llamada para obtener la lista completa de partidos, y por cada partido se harán dos para completar la información de sus jugadores.

Una vez visto en detalle la información que se obtiene de Sofascore, la tarea es organizarla y modelarla, de forma que cuando se necesite para entrenar la red neuronal sea devuelta lo más limpia y organizada posible.

2. Exposición de la información almacenada: Una vez la base de datos ha sido poblada (independientemente de que vaya actualizándose a diario), la siguiente fase es exponer esa información a entidades externas para que puedan utilizarla según sus necesidades. En el caso de este proyecto, habrá un segundo componente escrito en Python donde será necesario acceder a estos datos para poder entrenar la red neuronal. La información será accesible a través de endpoints REST, que se han definido como se indica a continuación:

   - `GET /data-crawler/events?finished=date=:` Obtiene la lista de partidos almacenados en la base de datos con la información detallada de los jugadores que se enfrentan en cada uno de ellos. Este endpoint ofrece además dos opciones de filtrado a través de parámetros opcionales que pueden ser configurados en la *query*:

     - `finished:` Variable booleana que indica si se quieren obtener los partidos que ya hayan finalizado (y que por ende, se sepa su resultado) o que no hayan comenzado aún (partidos futuros).
     - `date:` Variable de tipo String a través de la cual se puede indicar la fecha para la cual se quieren consultar los eventos. Debe ir en formato yyyy/MM/dd.

     Con estos parámetros, se cubren dos necesidades del proyecto:

     a) Obtención de partidos ya disputados: Será la consulta que se realice para obtener la información necesaria para el entrenamiento de la red neuronal.

     b) Obtención de partidos futuros: Se realizará esta consulta para obtener partidos que aún no han comenzado. Esto es requerido para poder mandárselos a la red neuronal una vez ya esté entrenada y poder obtener las predicciones de resultados.

Por último, quedaría por comentar el modelo de datos que se ha definido. Puesto que para este proyecto de momento solamente se va a considerar información sobre los partidos y jugadores, se han definido básicamente dos tablas con la siguiente relación:

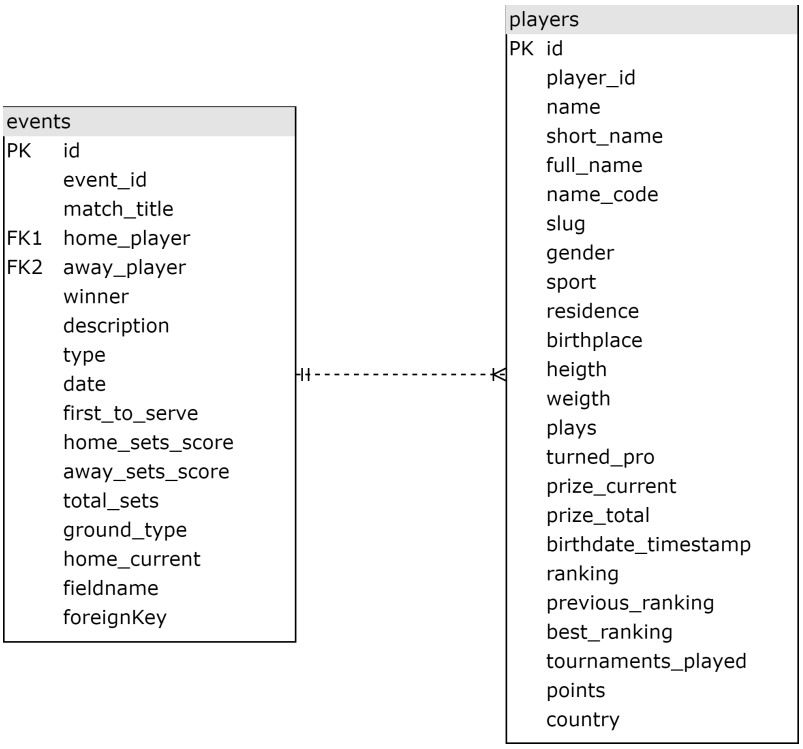


Figura 5.2: Modelo de datos del importador de información (elaboración propia, 2021)

De este modo, la información que será devuelta a través del endpoint REST descrito anteriormente estará bien estructurada y aportará información de valor a la red neuronal.

5.1.2. Red neuronal

La red neuronal es la parte crítica de este proyecto. Todo el desarrollo hasta ahora comentado está enfocado a la obtención de los datos que servirán como entrada para la red neuronal, cuyo objetivo es proporcionar predicciones acerca de la victoria o pérdida de partidos disputados por jugadores de tenis.

Cuando se desarrollan redes neuronales, es muy difícil acertar a la primera sobre qué arquitectura o modelo seguir. Debe ser recordado que las redes neuronales o modelos de *deep learning* se caracterizan por ser modelos muy versátiles capaces de adaptarse a gran variedad de problemas. Si bien esto es una ventaja, su implementación se puede realizar bastante tediosa, ya que las opciones de configuración que ofrecen son inmensas.

A grandes rasgos, los parámetros que se deben tener en cuenta son los siguientes:

1. Número de neuronas y capas ocultas: Cuando se tratan problemas de deep learning, quiere decir que las redes neuronales contienen capas más produndas u ocultas que

una simple capa de entrada y otra de salida. Esto ayuda a la red neuronal a identificar patrones ocultos en los datos que de otro modo sería impensable calcular. Hay numerosos estudios que han intentado definir la forma óptima para calcular el número de neuronas ocultas que una red debe tener de acuerdo al problema a resolver, como por ejemplo los citados en *Comparative analysis of methods for determining number of hidden neurons in artificial neural network* (Vujicic y col., 2016), pero en realidad no hay una fórmula exacta para ello. La experiencia de los desarrolladores de redes neuronales puede llegar a ser más valida, ya que el encontrar el número óptimo de neuronas distribuidas en capas ocultas es una tarea más de prueba y de error que el aplicar una ecuación directamente.

2. Función de activación: La función de activación devuelve una salida que será generada por la neurona dada una entrada o conjunto de entradas. Cada una de las capas que conforman la red neuronal tienen una función de activación que permitirá reconstruir o predecir. Además, se debe considerar que en la red neuronal se usará una función no lineal debido a que le permite al el modelo adaptarse para trabajar con la mayor cantidad de datos. Históricamente, se han propuesto muchas funciones de activación, pero en este documento se describirán las dos utilizadas para el desarrollo de la red neuronal: la sigmoide y ReLU.

    - ReLU: Permiten el paso de todos los valores positivos sin alterarlos, pero asigna todos los valores negativos a 0. Aunque existen funciones de activación aún más recientes, la mayoría de las redes neuronales de hoy utilizan ReLU o una de sus variantes.

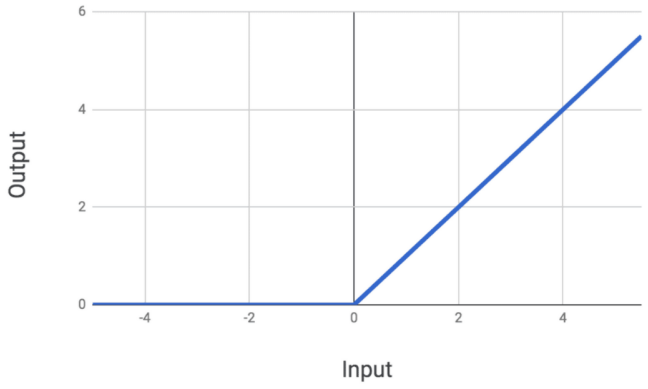


Figura 5.3: Función de activación ReLU (bootcampai, 2020)

- Lineal: Esta función también conocida como identidad, permite que lo de la entrada sea igual a la salida por lo que si tengo un red neuronal de varias capas y aplicó función lineal se dice que es una regresión lineal. Por lo tanto, esta función de activación lineal se usa si a la salida se requiere una regresión lineal y de esta manera a la red neuronal que se le aplica la función va a generar un valor único. Por ejemplo se usa cuando se solicita predecir el valor de un número de ventas.

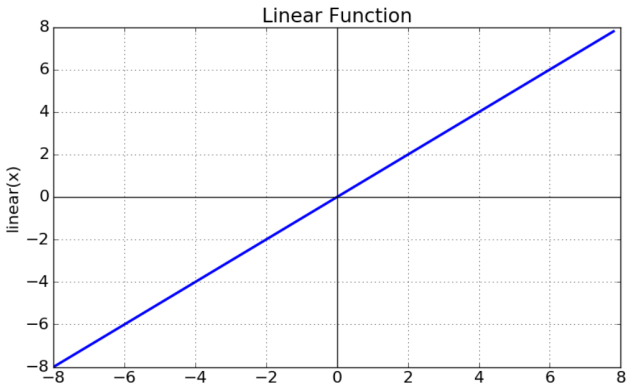


Figura 5.4: Función de activación Lineal (mihaileric, 2020)

3. Arquitectura de la red: En este caso se va a utilizar la opción *fully connected*, que quiere decir que todos las neuronas de la red están interconectadas unas con otras. Además, la información viajará en sentido *forward*, de lo contrario se hablaría de redes neuronales recurrentes.

4. Optimizador: En deep learning existe el concepto de pérdida (loss), que indica el rendimiento del modelo en un instante determinado. Esa pérdida debe ser utilizada para entrenar la red neuronal de forma que funcione mejor. Básicamente, lo que se busca es tomar la pérdida y tratar de minimizarla, ya que una pérdida menor significa que el modelo va a funcionar mejor. El proceso de minimizar (o maximizar) cualquier expresión matemática se llama optimización. Hay varios tipos de optimizadores, y el emplear uno u otro depende de muchos factores. Aquí vuelve a entrar en juego la prueba y error que se ha comentado anteriormente. En el caso de este proyecto, se va a utilizar el optimizador *Adam*. El algoritmo de optimización Adam *(Adaptive Moment Estimation)* es una extensión del descenso de gradiente estocástico que recientemente ha visto una mayor adopción para aplicaciones de aprendizaje profundo en la visión

por ordenador y el procesamiento del lenguaje natural. La intuición detrás del Adam es no ir tan rápido para saltar sobre el mínimo, sino disminuir la velocidad un poco para una búsqueda cuidadosa y así poder encontrar el mínimo más óptimo.

5. Learning rate: La tasa de aprendizaje es un hiperparámetro que controla cuánto cambiar el modelo en respuesta al error estimado cada vez que se actualizan los pesos del modelo durante la fase de entrenamiento. La elección de la tasa de aprendizaje es un reto, ya que un valor demasiado pequeño puede dar lugar a un largo proceso de entrenamiento que podría atascarse, mientras que un valor demasiado grande puede dar lugar al aprendizaje de un conjunto subóptimo de pesos demasiado rápido o a un proceso de entrenamiento inestable.

   La tasa de aprendizaje controla la rapidez con la que el modelo se adapta al problema. Las tasas de aprendizaje más pequeñas requieren más *epochs* de entrenamiento, dado que los cambios en los pesos son menores en cada actualización, mientras que las tasas de aprendizaje más grandes producen cambios rápidos y requieren menos *epochs* de entrenamiento.

   Una tasa de aprendizaje demasiado grande puede hacer que el modelo converja demasiado rápido a una solución subóptima, mientras que una tasa de aprendizaje demasiado pequeña puede hacer que el proceso se atasque.

Esto quiere decir que desarrollar red neuronales implica mucha prueba y error, ya que son modelos muy propensos a causar *overfitting* y eso debe ser evitado a toda costa.

Lo primero que debe identificarse cuando se va a desarrollar una red neuronal es el problema que se debe resolver. En el caso de este proyecto, se han obtenido una serie de variables o *features* que serán la entrada de la red, y una variable *target* que indicará si el jugador que juega en casa gana el partido o no, proporcionando como valor un 1 en caso positivo o un 0 en caso contrario.

Dicho esto, la primera capa de la red serán las features correspondientes a los datos, será una capa densa y estará compuesta por tantas neuronas como features haya, en este caso 24.

En cuanto al número de capas ocultas, se van a emplear dos, con 1000 y 200 neuronas, respectivamente. En ambos casos, la función de activación a utilizar será *relu*.

Finalmente, en la última capa se configurará una neurona, ya que en este caso solamente tenemos una variable target cuyo valor debe ser predicho. En este caso, la función de

activación será *linear*.

La figura 5.5 muestra el resumen generado por Keras con el modelo resultante de la red neuronal que se va a utilizar:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_840 (Dense)            (None, 24)                600
_________________________________________________________________
batch_normalization_163 (Bat (None, 24)                96
_________________________________________________________________
activation_380 (Activation)  (None, 24)                0
_________________________________________________________________
dropout_255 (Dropout)        (None, 24)                0
_________________________________________________________________
dense_841 (Dense)            (None, 1000)              25000
_________________________________________________________________
batch_normalization_164 (Bat (None, 1000)              4000
_________________________________________________________________
activation_381 (Activation)  (None, 1000)              0
_________________________________________________________________
dropout_256 (Dropout)        (None, 1000)              0
_________________________________________________________________
dense_842 (Dense)            (None, 200)               200200
_________________________________________________________________
batch_normalization_165 (Bat (None, 200)               800
_________________________________________________________________
activation_382 (Activation)  (None, 200)               0
_________________________________________________________________
dropout_257 (Dropout)        (None, 200)               0
_________________________________________________________________
dense_843 (Dense)            (None, 1)                 201
_________________________________________________________________
batch_normalization_166 (Bat (None, 1)                 4
_________________________________________________________________
activation_383 (Activation)  (None, 1)                 0
=================================================================
Total params: 230,901
Trainable params: 228,451
Non-trainable params: 2,450
```

Figura 5.5: Estructura de la red neuronal (elaboración propia, 2021)

Cabe hacer mención a las capas de *BatchNormalization* y *Dropout*, las cuales han tenido que ser configurada tras realizar distintas pruebas y comprobar que la red producía un ligero *overfitting*. Mediante estas capas de regularización se corrige este problema y se mejora la precisión de la red neuronal. En la sección 6.2 se profundizará más en estos aspectos.

5.1.3. Notificador de predicciones

Este componente ofrece la última funcionalidad del proyecto. El objetivo es obtener predicciones de partidos que aún no han sido disputados y que las envíe a través de un canal de comunicación. El componente como tal es muy simple, sin embargo, debe de comunicarse con los dos componentes que se han desarrollado anteriormente para poder realizar su función.

En la figura 5.6 se refleja el diagrama de flujo del envío de notificaciones:

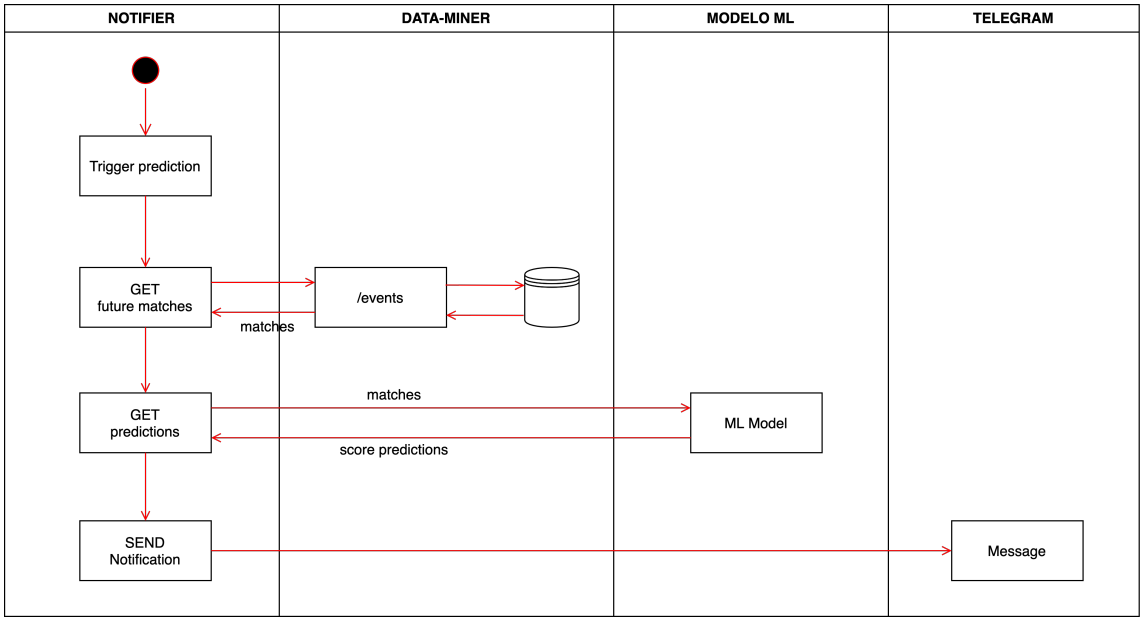


Figura 5.6: Diagrama de flujo del notificador (elaboración propia, 2021)

A continuación se explica cada uno de los pasos en detalle:

- Obtención de partidos futuros no disputados: Para ello, debe de consultar el microservicio de obtención de información mediante la siquiente petición:

  `GET /data-crawler/events?finished=true&date:` Como se puede observar, uno de los parámetros clave es el flag 'finished=true' lo que hará que el microservicio filtre la query hacia la base de datos y obtenga únicamente los partidos que aún no han comenzado. Por otro lado, en cuanto a la fecha, se le indicará una fecha posterior a la actual (por ejemplo el día siguiente) en formato yyyy/MM/dd.

- Obtención de predicciones de partidos no disputados: Una vez obtenidos los partidos aún no disputados con la petición anterior, se le enviarán al modelo de aprendizaje

automático ya entrenado para que calcule la predicción de los resultados, los cuales serán devueltos en un formato Json.

- Envío de notificaciones: Cuando el programa ya haya obtenido los partidos futuros con sus respectivas predicciones de resultados, el paso final es notificarlo. Para ello lo comunicará a través de un canal de Telegram donde la gente podrá suscribirse y hacer uso de la información.

5.1.4. Arquitectura completa

Una vez se han explicado los distintos componentes del sistema, se muestra a modo de esquema en la figura 5.7 cómo quedaría la arquitectura completa:

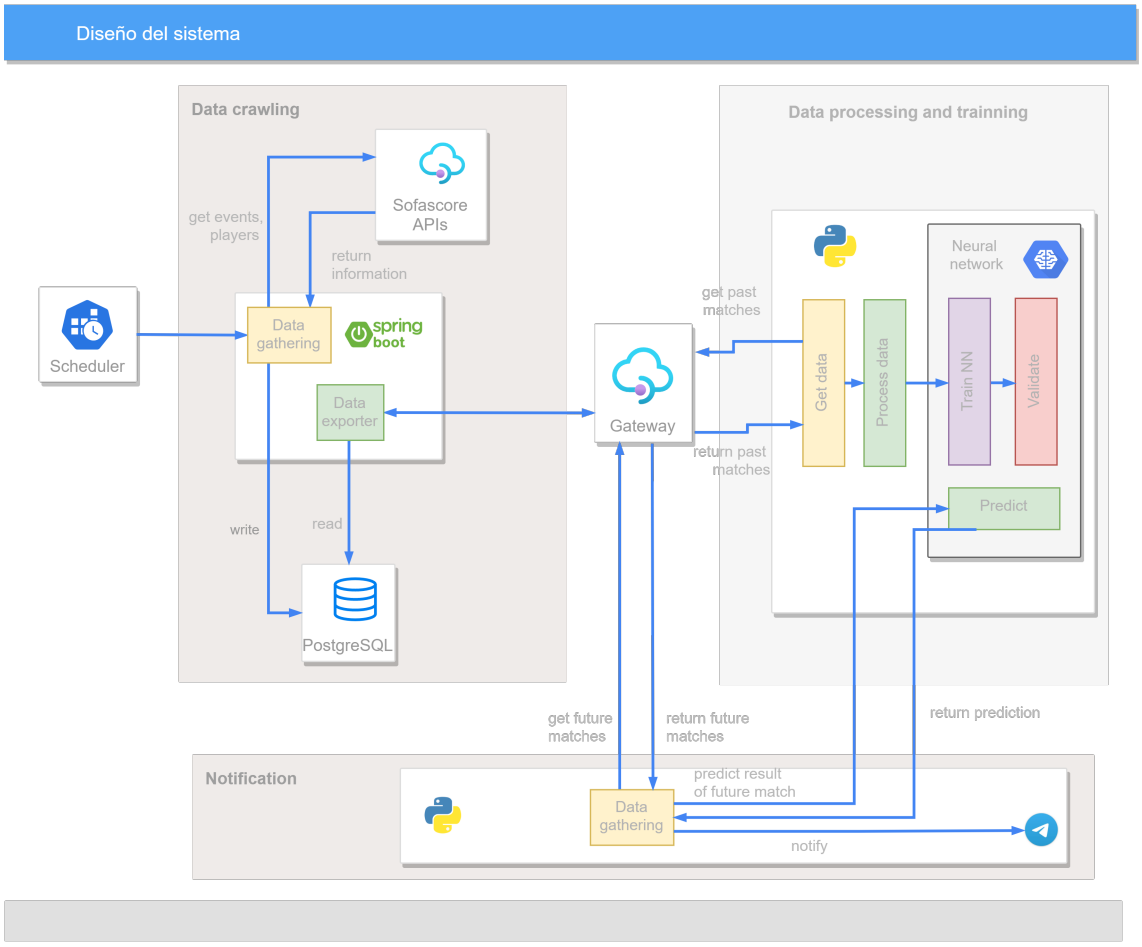


Figura 5.7: Diseño del sistema completo (elaboración propia, 2021)

5.2. Pipelines de automatización del proceso de despliegues

Con el fin de agilizar al máximo el proceso de desarrollo, el despliegue de los artefactos se ha automatizado mediante pipelines, tal y como se ha comentado anteriormente. De este modo, una vez el desarrollo de cualquier funcionalidad o corrección sea finalizado en el entorno local, se hará un commit a la rama *master* del repositorio de git que se ha creado para este proyecto.

Es en este momento donde entran en juego las funciones de *Github actions*, que detectarán automáticamente que se ha producido un nuevo cambio en el repositorio de código fuente y ejecutarán la pipeline que se ha desarrollado para ello. A continuación, se muestra dicha pipeline:

```
name: Build and deploy JAR app to Azure Web App - sofascore-data-crawler
on:
  push:
    branches:
      - master
  workflow_dispatch:
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up Java version
      uses: actions/setup-java@v1
      with:
        java-version: '11'
    - name: Build with Maven
      run: mvn clean install
    - name: Upload artifact for deployment job
      uses: actions/upload-artifact@v2
      with:
        name: java-app
        path: '${{ github.workspace }}/target/*.jar'
```

```
deploy:
  runs-on: ubuntu-latest
  needs: build
  environment:
    name: 'production'
    url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}
  steps:
  - name: Download artifact from build job
    uses: actions/download-artifact@v2
    with:
      name: java-app
  - name: Deploy to Azure Web App
    id: deploy-to-webapp
    uses: azure/webapps-deploy@v2
    with:
      app-name: 'sofascore-data-crawler'
      slot-name: 'production'
      publish-profile: {{ secrets.AzureAppService_PublishProfile_42d4315a }}
      package: '*.jar'
```

Se trata de un simple fichero en formato 'yaml' que es capaz de ejecutar la compilación y empaquetado del software, en este caso un fichero *.jar* ya que se está empleando Spring Boot, que posteriormente será publicado en el entorno de Azure que se ha creado para ello.

Es decir, en cuestión de 2-3 minutos los nuevos cambios realizados estarán disponibles y ejecutándose en el entorno remoto de la suscripción de Microsoft Azure.

Capítulo 6

Evaluación

Uno de los objetivos de este proyecto era evaluar la viabilidad del mismo, es decir, comprobar si las predicciones que se obtienen con la red neuronal son lo suficientemente precisas y fiables como para poder obtener un beneficio económico de ellas mediante apuestas deportivas.

En este apartado se evalúan tanto los datos que formarán la entrada de la red neuronal como el resultado obtenido de la misma. Además, se guiarán las distintas pruebas realizadas durante el proceso, indicando las modificaciones y/o mejoras que se han tenido que introducir para conseguir el resultado deseado.

6.1. Evaluación de los datos

En este apartado se detallan las validaciones y operaciones que se han tenido que realizar sobre los datos obtenidos para que la red neuronal pueda ser entrenada y trabajar correctamente. Si bien es cierto que el módulo de *data-crawling* fue pensado para que esta tarea sea lo mas liviana posible, es imprescindible comprobar que los datos obtenidos sean consistentes y tengan el formato correcto para realizar un entrenamiento de la red neuronal satisfactorio.

Puesto que la información se devuelve mediante el API expuesta por el microservicio explicado en apartados anteriores, lo primero es convertir la respuesta en formato JSON a un dataframe, que es la estructura con la que normalmente se trabaja en este tipo de proyectos. Esto no supone ningún problema ya que hay librerías como*pandas* que realizan esta función automáticamente:

```
BASE_PATH = 'http://sofascore-data-crawler.azurewebsites.net/data-miner'
```

```
# Generamos el endpoint del microservicio
req = requests.get(BASE_PATH + '/events?finished=true'

# Generamos un dataframe en base a la respuesta obtenida por el API
dataframe = pd.DataFrame.from_dict(req.json())
```

Una vez los datos se encuentran en el formato adecuado, será sencillo realizar operaciones sobre ellos para dejarlos listos para la fase de entrenamiento.

En primer lugar, se debe definir el tipo de problema a resolver. Puesto que la mayoría de las variables son numéricas y lo que interesa es obtener la probabilidad de que un jugador gane un encuentro, podemos modelar el problema como un problema de regresión. No obstante, hay muchas de las variables o *features* que tienen carácter categórico, pues solamente toman un cierto valor determinado. En estos casos, una práctica muy común es mapear esas categorías a variables numéricas, de modo que puedan ser tratadas de forma matemática. Las variables que necesitarán ser convertidas son las que se muestran en tabla 6.1:

Variable Categórica	**Categoría**	**Conversión numérica**
plays	right-handed	1
	left-handed	0
ground_type	Clay	1
	Grass	0
winner	home_player	1
	away_player	0

Tabla 6.1: Conversión de variables categóricas a numéricas

Cabe destacar que la columna *winner* nombrada en esta tabla es generada al vuelo y añadida al dataframe, es decir, en base a los parámetros que devuelve el API se puede extraer la información sobre si el jugador que gana el partido es el que juega en casa o el jugador visitante.

Una vez hecho esto, ya swe puede aplicar un problema de regresión, pero no sin antes comprobar los datos que se tienen. Hay *features* como por ejemplo *gender* que no aportan nada, ya que todos los jugadores que se van a tratar para este problema son hombres. Otro

como por ejemplo la fecha del encuentro tampoco aportará información de gran peso para las predicciones, por lo que este tipo de features serán eliminadas del dataframe.

Se puede ver qué variables tienen mayor o menor valor mediante la matriz de correlaciones, que puede ser calculada de forma muy sencilla. En la figura 6.1 se muestran los resultados obtenidos.

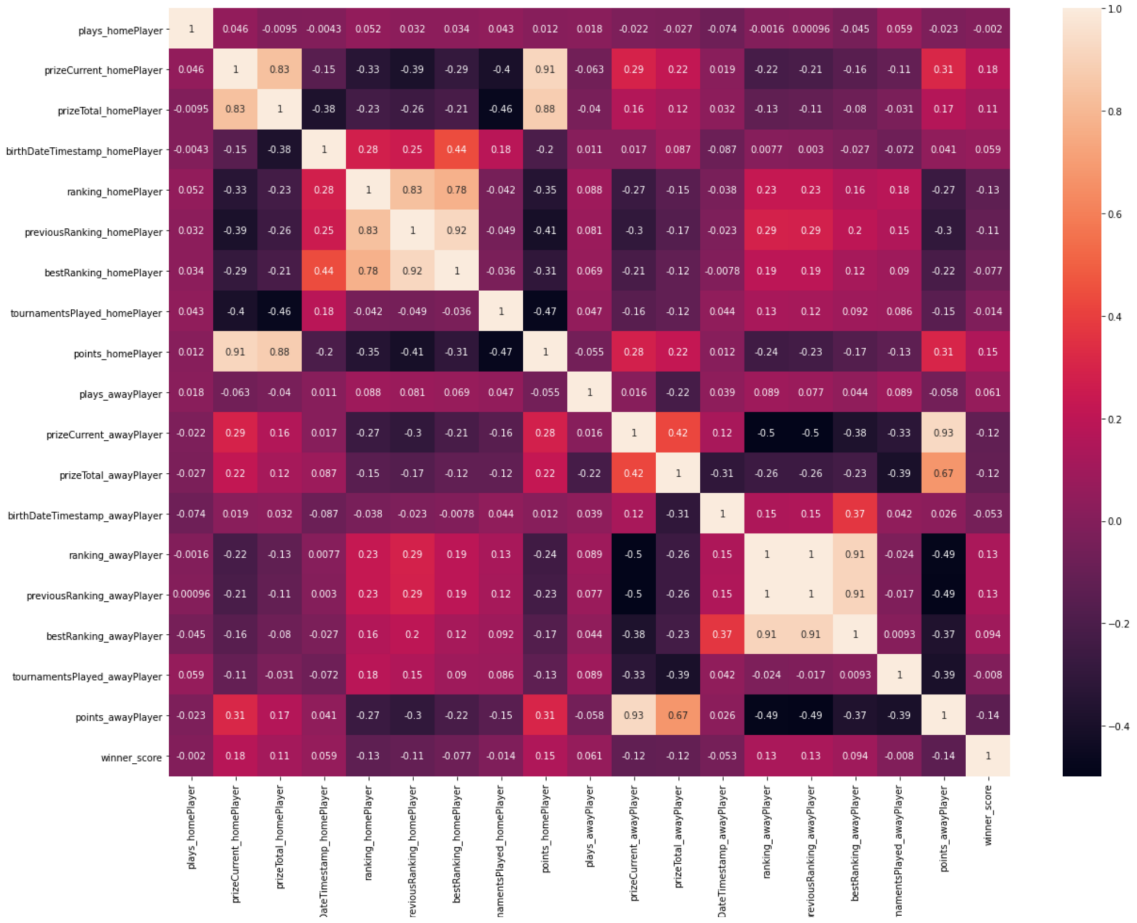


Figura 6.1: Matriz de correlación de las features (elaboración propia, 2021)

Como se puede observar, las variables relativas al ranking del jugador son las que más impacto tienen.

Otra tarea que se debe realizar es comprobar si del total de entradas que se han obtenido hay valores nulos o que no vengan informados. La comprobación es sencilla utilizando comandos de pandas, el problema radica en decidir qué hacer ante estos valores. Eliminar la fila directamente podría ser una opción, pero se perdería información valiosa que podría aportar eficiencia a la red neuronal. En nuestro caso, se realizan dos operaciones en función de la naturaleza de la variable faltante:

- Variable categórica: El valor será cumplimentado con **la moda** de dicha variable
- Variable numérica: El valor será cumplimentado con **la media** de todos los valores de dicha variable

De este modo, el dataset tendrá todos los valores informados mediante variables numéricas. El sustituir los valores faltantes por la media o la moda es una técnica muy común que se utiliza para evitar la pérdida de información y que además no altera la valoración de los datos.

Para terminar, una buena práctica es comprobar si el dataset con el que se trabaja está o no equilibrado, es decir, si tiene más o menos los mismos valores de las categorías o resultados de la variable *target* que se busca predecir.

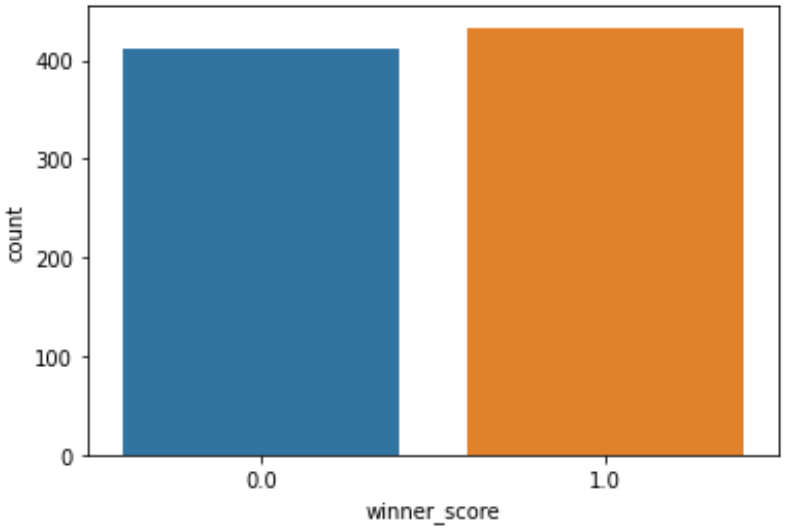


Figura 6.2: Comprobación equilibrado del dataset (elaboración propia, 2021)

En la figura 6.2 se observa que se obtienen más o menos el mismo número de victorias que de derrotas para los jugadores en casa, lo cual facilita el entrenamiento y el funcionamiento de la red neuronal, ya que si se obtuviera un número de victorias mayoritario, la red neuronal sería capaz de aprender esta característica, pero no las derrotas. Los datasets no equilibrados se utilizan por ejemplo para el desarrollo de modelos cuyo objetivo es la detección de anomalías, pero no es el caso que incumbe a este proyecto.

6.2. Evaluación de la red neuronal

A continuación se muestran los resultados obtenidos con la configuración de la red neuronal del apartado anterior. Tal y como se ha comentado, las opciones disponibles para configurar redes neuronales son muy amplias y es muy difícil obtener buenos resultados a la primera. Cuando no se tienen datos suficientes o la configuración con la que se entrena la red no es la adecuada, suele producirse el efecto llamado *overfitting*. Este fenómeno consiste en que la red aprende o se ajusta muy bien a los datos del conjunto de entrenamiento, pero es incapaz de generalizar, es decir, cuando obtenga un dato de entrada distinto al conjunto de entrenamiento no será capaz de proporcionar buenas predicciones.

El overfitting puede detectarse atendiendo a la evolución de los parámetros de precisión y pérdida durante el proceso de entrenamiento según se cita en el artículo *'The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial'* (Ghojogh y Crowley, 2019). En la figura 6.3 se muestra un ejemplo de overfitting:

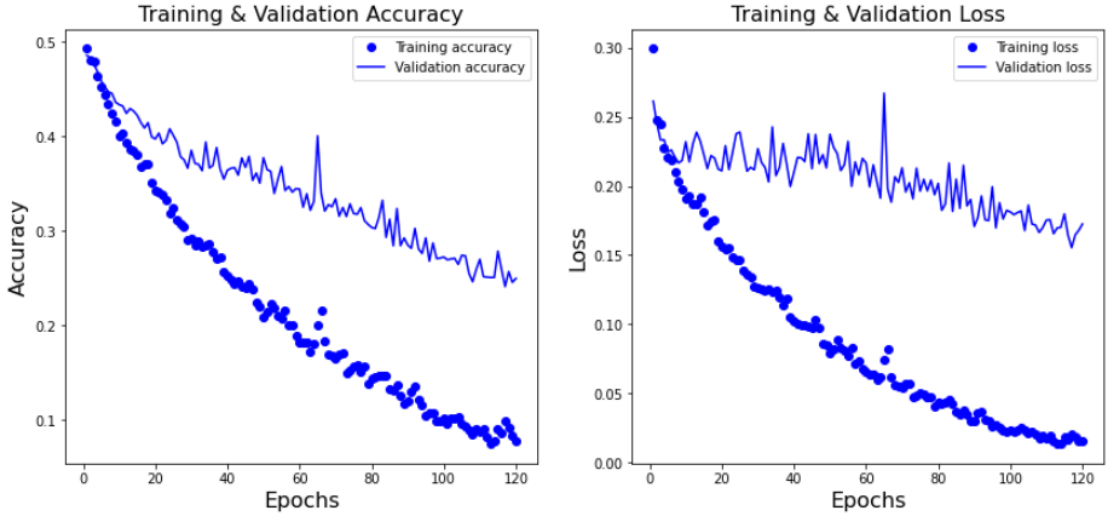


Figura 6.3: Ejemplo de detección de overfitting (elaboración propia, 2021)

Tal y como se observa, a medida que la red está siendo entrenada llega un momento en el que la precisión del conjunto de test se 'separa' de la precisión del conjunto de entrenamiento, lo cual quiere decir que la red no es capaz de generalizar y proporcionar buenos resultados para datos distintos a los de entrenamiento.

En este caso, añadiendo las capas de *batch normalization* y *dropout* se ha conseguido corregir este problema. En la figura 6.4 se observa cómo se comporta la red una vez corregido el problema de overfitting:

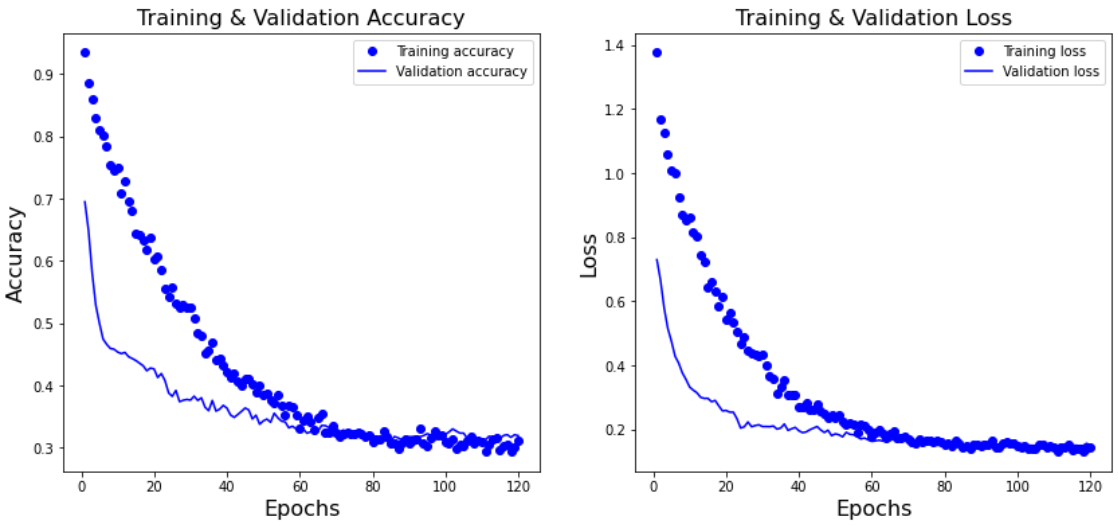


Figura 6.4: Comportamiento con overfitting corregido (elaboración propia, 2021)

En este caso, tanto la precisión del conjunto de entrenamiento como la precisión del conjunto de test van más o menos a la par y siguen incrementándose durante el entrenamiento de la red de igual forma.

A modo de resumen, a continuación se indica la configuración de todos los parámetros adaptados para el entrenamiento de la red neuronal:

- Nº epochs: 120
- Nº capas ocultas: 2
  - Densa con 1000 neuronas, función de activación 'Relu' y Dropout de 0.3
  - Densa con 200 neuronas, función de activación 'Relu' y Dropout de 0.3
- Learning rate: 1e-3
- Optimizador Adam
- Batch size: 40

Además de este tipo de gráficas para comprobar cómo ha ido el proceso de entrenamiento, existen otras métricas estadísticas que permiten analizar redes neuronales. Cabe mencionar que las métricas no son iguales si se trata de un problema de regresión o de un problema de clasificación. En este caso, siguiendo las recomendaciones y los estudios realizados en el artículo *On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression* (Qi y col., 2020) puesto que tratamos de resolver un problema mediante

regresión lineal, se ha analizado el MAE *(Mean Absolute Error)* de la red, un parámetro originado en una medida de error medio que se emplea a menudo para evaluar modelos de regresión vectorial (también conocidos como multivariantes).

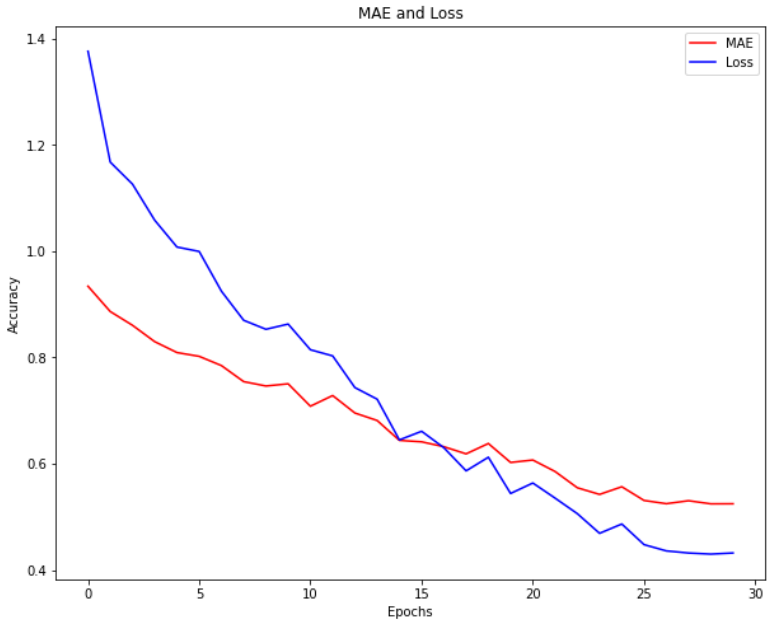


Figura 6.5: Mean Absolute Error (elaboración propia, 2021)

6.3. Resultados obtenidos

Una vez analizado el proceso de entrenamiento, el paso final es evaluar las predicciones de la red neuronal. Para ello, se emplea el 20 % de los datos del conjunto de test que se reservan para este propósito, es decir, el llamado conjunto de validación.

La forma de evaluar las predicciones es utilizar el conjunto de validación como entrada de la red neuronal y obtener las predicciones sobre el mismo. Comparando los resultados de dichas predicciones con los resultados de los partidos (los cuales son conocidos, ya que el conjunto de validación está etiquetado), se puede obtener una primera aproximación de cómo se comportaría la red en situaciones reales.

En la figura 6.6 se pueden observar los resultados obtenidos ante las treinta primeras entradas del conjunto de validación:

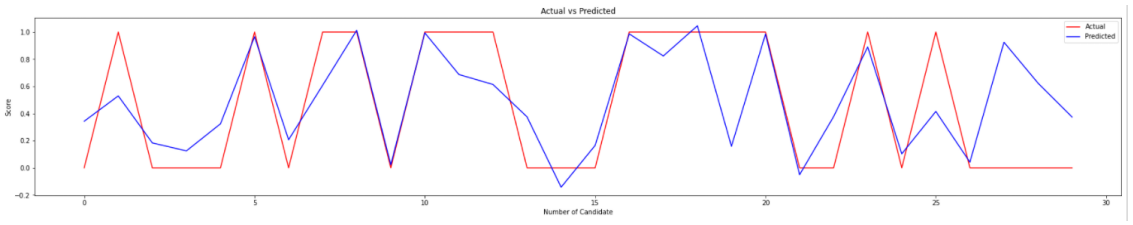


Figura 6.6: Predicciones obtenidas (elaboración propia, 2021)

La línea roja indica las victorias del jugador en casa, siendo valor 1 cuando gana y 0 cuando pierde, mientras que la línea azul indica las predicciones obtenidas por la red neuronal.

En términos generales y sobre el ejemplo de la figura anterior, se observa que la red neuronal produce más aciertos que errores, aunque no siempre es capaz de acercarse con gran precisión al valor que debería ser. Los casos en los que falla, lo hace con muchísimo error, como por ejemplo en las muestras que se muestran en la figura 6.7:

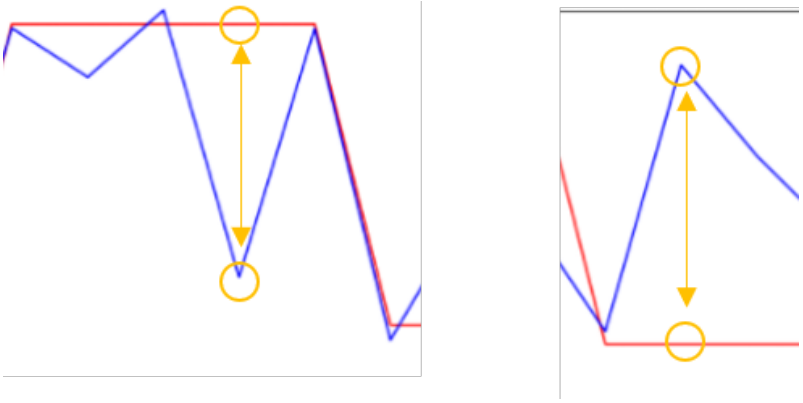


Figura 6.7: Errores de predicción (elaboración propia, 2021)

Este fenómeno sería un buen caso a analizar en evoluciones futuras del proyecto. Posiblemente, según se vayan obteniendo datos de más partidos y aumente el número de información con la que se entrena la red neuronal, este tipo de comportamientos se corrija. No obstante se puede concluir en rasgos generales que situando el rango de ganar/perder en 0.5, el modelo es capaz de generalizar y dar predicciones que se acercan bastante a lo esperado.

Para terminar, la implementación del bot predictivo ha sido finalizada y se comprueba que su funcionamiento es correcto. Tiene básicamente dos comandos, el primero sería /help que muestra una ayuda de cómo usarlo, y el segundo y fundamental es /predictions, que

se comunica con la red neuronal para obtener las predicciones y mostrárselas al usuario. La figura 6.8 muestra un ejemplo:

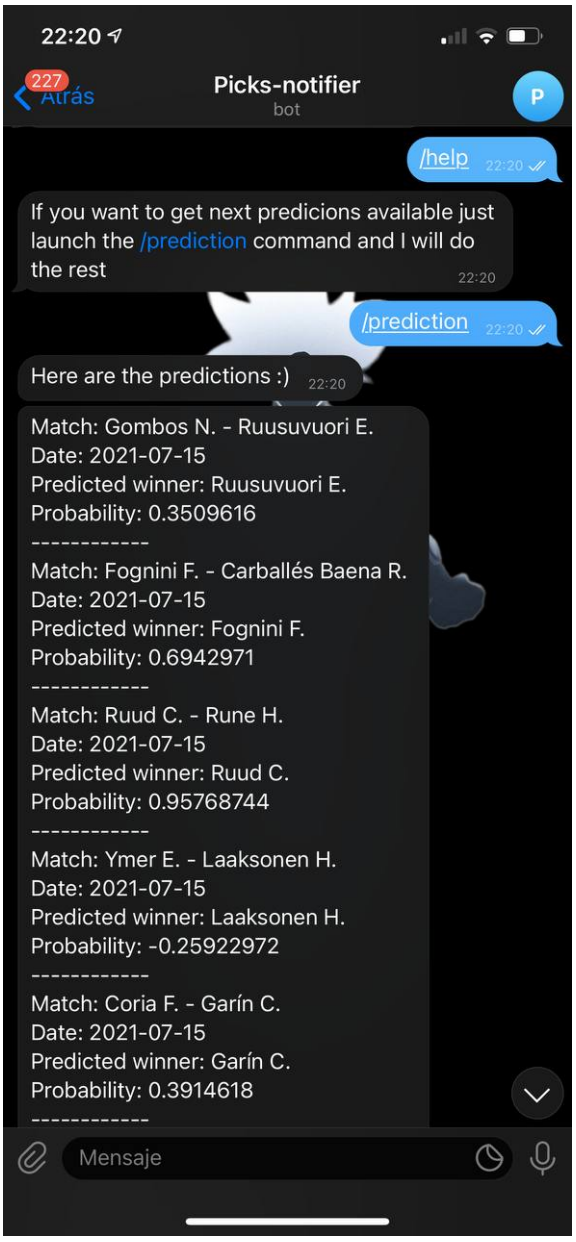


Figura 6.8: Ejemplo de mensaje con predicciones (elaboración propia, 2021)

Capítulo 7

Conclusiones y Trabajo Futuro

En esta última sección del documento, trataremos de resumir la experiencia obtenida durante el proceso de desarrollo, así como la viabilidad del mismo en cuanto a los resultados obtenidos. Además, se nombrarán los *gaps* identificados que pueden ser mejorados en un futuro, así como nuevas funcionalidades que pueden enriquecer el producto desarrollado.

7.1. Conclusiones

Recapitulando todo lo comentado en secciones anteriores y las capacidades de la herramienta desarrollada, se ha obtenido:

1. Motor de obtención de datos automático: Este componente es capaz de mantener una base de datos actualizada con los resultados de los últimos partidos, así como de proporcionar datos de partidos aún no disputados para poder obtener su predicción.

2. Red neuronal capaz de predecir si un jugador vencerá o no en un partido determinado: En base a lo aprendido durante el proceso de entrenamiento, la red neuronal será capaz de predecir la capacidad de victoria o derrota de un jugador frente a su contrincante en base a los parámetros de entrada.

3. Notificador de predicciones: Este componente se encarga de preguntar al motor de obtención de datos sobre partidos futuros (día siguiente por ejemplo), enviar dicha información a la red neuronal y notificar al usuario sobre las predicciones directamente desde un canal de Telegram.

La parte más crítica de todo este desarrollo se concentra en la red neuronal, ya que es la encargada de proporcionar la predicción sobre un determinado partido de tenis, que era

el objetivo principal de este proyecto. Considerando un pequeño margen de error, se ha conseguido inferir dicha información de una forma correcta, si bien es cierto que aún hay cierto margen de mejora.

Atendiendo a lo comentado en el apartado del desarrollo de la red neuronal y las validaciones de la misma, se han podido corregir los problemas de *overfitting* que se produjeron en una primera instancia y se han obtenido unos resultados aceptables.

En cuanto al motor de obtención de datos, ya se ha conseguido que funcione de forma autónoma, lo cual significa que siempre mantendrá la base de datos con la información más actualizada de cada día. En este sentido se ha conseguido completar el requisito inicial en su totalidad de forma más que satisfactoria. En el siguiente apartado se comentará alguna mejora que podrá introducirse en dicho microservicio de cara a enriquecer la información que almacena, pero para un primer desarrollo es más que suficiente lo conseguido hasta el momento.

Por último, el notificador de predicciones es el encargado de explotar la red neuronal, es decir, se ocupa de obtener la predicción de partidos aún no disputados, lo cual era el objetivo de cara a las apuestas deportivas. Esta función está cubierta también, aunque tenga que ser ejecutada de forma manual. En la sección que viene a continuación se consideran una serie de mejoras que afectan a este componente y que lo harán mucho más eficiente.

En definitiva, se considera que los objetivos propuestos inicialmente quedan cubiertos para esta primera fase del proyecto, aunque aún pueda ser mejorado en muchos aspectos.

7.2. Trabajo futuro

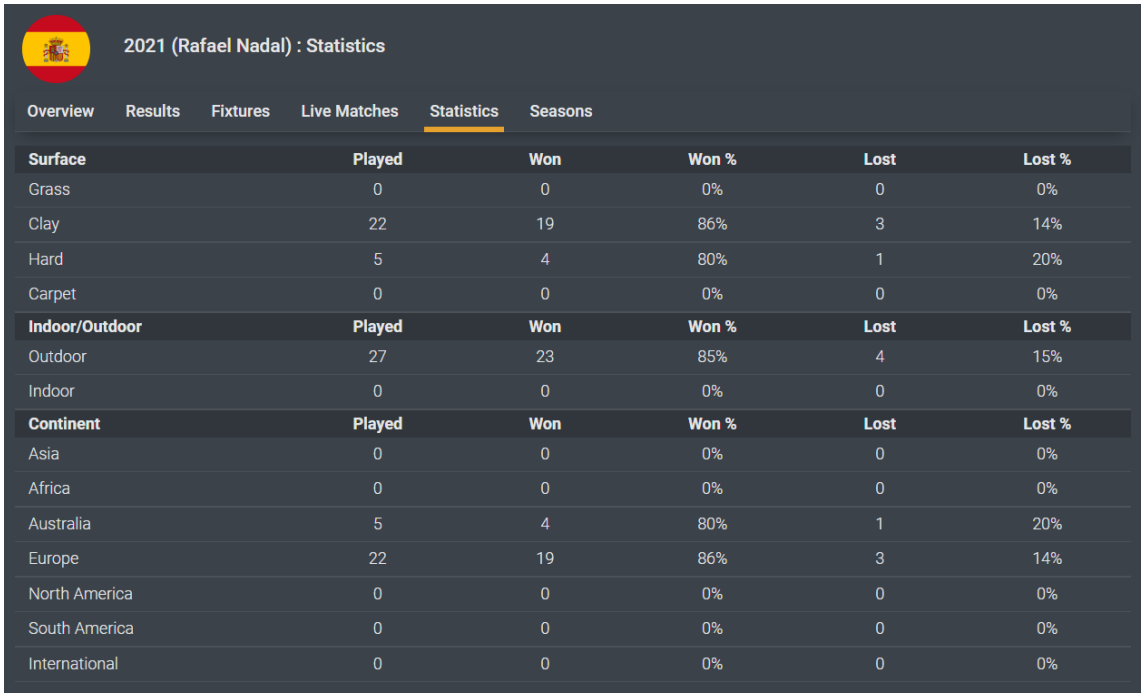
Este proyecto puede ser considerado como un MVP (*Minimum Viable Product)*, ya que cubre la funcionalidad básica de obtención de predicciones sobre los resultados de partidos de tenis. No obstante, tiene aún mucho margen de mejora y nuevas características que pueden ser implementadas como evolutivos del producto y que aportarán más valor en muchos sentidos.

Entre las más importantes y que deberían ser consideradas para próximas versiones, deberían incluirse:

- Aumento del número de features mediante web scraping de otros portales de deportes

  Como se ha observado en la sección 6.3, la red neuronal en ocasiones arroja predicciones erróneas con un margen de error muy alto. Una de las formas de tratar de

aumentar la precisión de las predicciones es aumentar el número de features que se han utilizado para caracterizar a los distintos jugadores. En este proyecto se han empleado los datos que están accesibles a través de Sofascore, pero existen muchos otros portales web que ofrecen también información actualizada sobre infinidad de deportes y que podrían ser considerados para futuros evolutivos para enriquecer los datos extraídos de Sofascore. Un portal con bastante información estadística sobre jugadores de tenis podría ser *ScoresPro*. La figura 7.1 muestra un ejemplo de ello:



2021 (Rafael Nadal) : Statistics

Overview | Results | Fixtures | Live Matches | Statistics | Seasons

Surface	Played	Won	Won %	Lost	Lost %
Grass	0	0	0%	0	0%
Clay	22	19	86%	3	14%
Hard	5	4	80%	1	20%
Carpet	0	0	0%	0	0%
Indoor/Outdoor	**Played**	**Won**	**Won %**	**Lost**	**Lost %**
Outdoor	27	23	85%	4	15%
Indoor	0	0	0%	0	0%
Continent	**Played**	**Won**	**Won %**	**Lost**	**Lost %**
Asia	0	0	0%	0	0%
Africa	0	0	0%	0	0%
Australia	5	4	80%	1	20%
Europe	22	19	86%	3	14%
North America	0	0	0%	0	0%
South America	0	0	0%	0	0%
International	0	0	0%	0	0%

Figura 7.1: Ejemplo de información en ScoresPro (elaboración propia, 2021)

- Implementación de mecanismos de seguridad (OAuth) en las apis del microservicio de importación de datos

  A día de hoy, el consumo de los endpoints implementados para la obtención de los datos puede hacerse simplemente poniendo la url en el navegador web. Es decir, no tienen ningún tipo de mecanismo de seguridad, por ello lo suyo sería incluir un servidor de OAuth mediante el cual se pueda obtener un token JWT dadas unas credenciales. Esto evitaría que terceras personas que puedan conocer la url puedan acceder a la información.

- Automatización del entrenamiento de la red neuronal

En esta primera implementación del proyecto, se ha automatizado el proceso de obtención de información, de modo que siempre se obtengan los últimos partidos disputados cada día y la información de los jugadores esté siempre actualizada. Sin embargo, el desarrollo e implementación de la red neuronal se ha realizado de forma manual, es decir, hay que lanzar el programa para que el script recupere los datos de los partidos, haga el preprocesamiento de los datos oportuno, y finalmente entrene la red neuronal.

Como se puede observar, no sirve de mucho tener la base de datos actualizada si la red neuronal no lo va a hacer igualmente de forma autónoma. La idea en este sentido es implementar un mecanismo de entrenamiento automatizado, por ejemplo semanal, de forma que la red neuronal se actualice cada cierto tiempo con los últimos datos obtenidos y siga proporcionando predicciones en base a los últimos acontecimientos ocurridos.

- Automatización del proceso de notificaciones de las predicciones

  Este punto va bastante ligado al anterior. Ahora se ha desarrollado un pequeño bot en Telegram que proporciona un comando a través del cual se puede preguntar a la red neuronal sobre las predicciones del día siguiente. Este proceso debería ser automatizado de igual forma, de modo que todos los días sea el propio bot el que por sí mismo recupere las predicciones y realice un broadcast con las mismas hacia todos los seguidores del canal.

- Aumento de la granularidad de las predicciones

  Otra mejora que implica bastante desarrollo y va ligada al primer punto de este apartado es incrementar la granularidad salida de la red neuronal en cuanto a predicciones. Es decir, en estos momentos la red es capaz de predecir si un jugador va a ganar o perder, pero existen muchos otros parámetros que pueden ser interesantes conocer si se quiere enfocar este proyecto a apuestas deportivas. El beneficio que se puede obtener apostando a que un jugador gana o pierde es muchísimo menor que si se apuesta a algo más arriesgado, como el número de sets que tendrá el partido o incluso el resultado final.

  En resumen, la mejora en este sentido sería que la red neuronal sea capaz no únicamente de proporcionar datos acerca de victoria o derrota, si no del número de sets o el resultado final, entre otros.

- Implementación de un modelo de suscripción a las predicciones de la red neuronal

  Este punto sería el menos prioritario y antes debería garantizarse que los mencionados anteriormente han sido implementados en su totalidad y proporcionan resultados aceptables. Una vez realizado, se podría implementar un mecanismo de suscripciones al canal de Telegram en distintas modalidades, una *free* que sea gratuita y proporcione acceso únicamente a las predicciones de victoria o derrota, y una *premium* que tenga un coste de suscripción mensual y que proporcione acceso al total de las predicciones desarrolladas, de modo que cualquier usuario suscrito pueda lucrarse de las mismas.

Capítulo 8

Bibliografía

Beetz, M., Buss, M. & Wollherr, D. (2007). Cognitive technical systems—what is the role of artificial intelligence? *Annual Conference on Artificial Intelligence*, 19-42.

bootcampai. (2020). Función ReLU. https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

Chollet, F. (2015). Keras. https://keras.io/

Facebook. (2016). PyTorch. https://pytorch.org/

Ghojogh, B. & Crowley, M. (2019). The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial.

Google. (2021). Tensorflow. https://www.tensorflow.org/

Huang, Z., Huang, W., Chen, W.-C. & Lanham, M. A. (2020). Dynamic Pricing For Sports Event Tickets.

Kogan, P., Parra, G. & Castillo, R. d. (2006). Diseño de agentes experimentando con robots que juegan al fútbol en ambientes reales y simulados. *VIII Workshop de Investigadores en Ciencias de la Computación*.

mihaileric. (2020). Función Lineal. https://www.mihaileric.com/posts/neural-network-grab-bag/

Monago Ruiz, A. (2019). Servicio Web API REST sobre el Framework Spring, Hibernate, JSON Web Token y BBDD Oracle. *Website*.

Panjan, A., Sarabon, N. & Filipčič, A. (2010). Prediction of the successfulness of tennis players with machine learning methods. *Kinesiology*, *42*(1), 98-106.

Qi, J., Du, J., Siniscalchi, S. M., Ma, X. & Lee, C.-H. (2020). On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression. *IEEE Signal Processing Letters*, *27*, 1485-1489. https://doi.org/10.1109/lsp.2020.3016837

Redhat. (2021). CI/CD. https://www.redhat.com/es/topics/devops/what-is-ci-cd

Sipko, M. & Knottenbelt, W. (2015). Machine learning for the prediction of professional tennis matches. *MEng computing-final year project, Imperial College London*.

Sportytrader. (2020). Manual de apuestas deportivas. https://www.sportytrader.es/apuestas-deportivas/manual/

Taibi, D., Lenarduzzi, V. & Pahl, C. (2018). Architectural Patterns for Microservices: A Systematic Mapping Study. *CLOSER*, 221-232.

Univaso, P. (s.f.). Deportes y Data Mining.

Veisdal, J. (2019). Conferencia de Dartmouth. https://www.cantorsparadise.com/the-birthplace-of-ai-9ab7d4e5fb00

Vujicic, T., Matijevic, T., Ljucovic, J., Balota, A. & Sevarac, Z. (2016). Comparative analysis of methods for determining number of hidden neurons in artificial neural network. *Central european conference on information and intelligent systems*, 219.

Wavemaker. (2020). Gestión de releases. https://www.wavemaker.com/8/learn/app-development/deployment/release-management/index.html

Apéndice A

Apendices

Los repositorios donde se aloja el código fuente desarrollado:

- Microservicio extracción de información

  https://github.com/ivan1405/sofascore-data-miner

- Red neuronal

  https://github.com/master-in-artificial-intelligence/tfm-neural-network

- Bot de applicación de mensajería

  https://github.com/master-in-artificial-intelligence/tfm-bot-predictor

Modelado y caracterización de jugadores de tenis para la predicción de resultados en eventos deportivos

Iván Pérez Fernández

Universidad Internacional de la Rioja, Logroño (España)

05 de Julio de 2021

RESUMEN

El objetivo del presente proyecto se basa en la explotación de información de portales deportivos para realizar predicciones a través de redes neuronales. En primer lugar, se realiza una exportación masiva de datos disponibles sobre jugadores de tenis mediante consumición de APIs públicas que serán utilizados para entrenar un modelo y predecir la probabilidad de ganar un encuentro entre dos jugadores. En segundo lugar, el siguiente objetivo sería crear un bot que que proporcione de forma automatizada sugerencias de apuestas deportivas para conocer si el modelo de negocio obtiene beneficios a largo plazo. Este proyecto se focaliza en el tenis porque tiene varias ventajas: deporte individual y muchos eventos diariamente para conseguir una base de datos consistente y que los resultados no tengan variación en sesgo.

PALABRAS CLAVE

apuestas, deportes, predicción, redes neuronales, web scraping

I. INTRODUCCIÓN

En la primera sección del documento se explica cuáles son los acontecimientos que han impulsado el desarrollo del proyecto así como los problemas o mejoras que plantea solventar.

A. Motivación

Hoy en día existe un gran número de personas consideradas expertas en ciertos deportes. Poseen un conocimiento muy actualizado del estado de los equipos a los que siguen y son capaces, en cierto modo, de predecir las tendencias en el resultado de un partido en función de qué equipos se enfrentan entre sí y de experiencias previas. Esta es la clave del presente proyecto, puesto que este grupo de personas, a las cuales a partir de ahora serán referidas como *tipsters*, se lucran de sus conocimientos compartiendo sus predicciones con las personas interesadas.

Si bien es cierto que muchos de los *tipsters* más conocidos tienen un gran número de seguidores debido a su elevada tasa de acierto, sus predicciones se basan simplemente en el criterio personal y experiencia previa. Como es de esperar, es muy complicado que puedan tener en cuenta factores críticos como el estado de ánimo y salud de la gran infinidad de jugadores que se enfrentan a diario, lo cual sería una tarea muy sencilla para un ordenador.

Esta barrera es la que se intenta romper con este proyecto, utilizando técnicas de *Machine Learning* para entrenar un modelo capaz de predecir el resultado de partidos de tenis, donde el factor humano sea eliminado de la ecuación y se tenga en cuenta cualquier característica disponible de cada jugador para modelar su estado.

B. Planteamiento del trabajo

Como en cualquier problema de *Machine Learning* lo primordial es obtener una cantidad considerable de datos con la mayor calidad posible. En internet se pueden encontrar infinidad de webs que contienen información y estadísticas sobre distintos deportes. La idea es utilizarlas para extraer los datos que sean necesarios para obtener el objetivo mencionado.

En este proyecto se explotará la información de la web SofaScore, ya que es un portal en el que se recogen múltiples datos de varios deportes, aunque este proyecto se centrará sección de tenis. Para ello, el primer *approach* es utilizar técnicas de *web scraping*, sin embargo, tras analizar la web se ha visto que dispone de APIs REST públicas que devuelven toda la información necesaria de forma estructurada, por lo tanto ésta será la vía a explorar.

La primera parte es por lo tanto, conseguir una base de datos actualizada a partir de la información extraída de la web. En una primera fase nos centraremos en el ranking de jugadores de ATP, es decir, obtendremos toda la información disponible de los 500 mejores jugadores de tenis de esta categoría. Además, se obtendrá gran cantidad de información sobre partidos que han jugado previamente, lo que permitirá saber sus resultados y relacionar qué jugadores se han enfrentado entre sí.

Esto permitirá utilizar aprendizaje supervisado, ya que son conocidos los resultados de partidos previos y toda esta información será el *input* a una RNN (*Red Neuronal Recurrente*) que será entrenada y utilizada para validar eventos futuros en función de lo aprendido.

II. ESTADO DEL ARTE

En esta sección se realiza un repaso de los acontecimientos más importantes que ha sufrido la inteligencia artificial desde sus orígenes hasta la actualidad y lo que se espera de ella en los próximos años dada su evolución.

Además, como debería hacerse en cualquier proyecto de estas características, antes de proceder con el desarrollo del proyecto se ha realizado una exploración en distintas bases de datos de investigación sobre artículos cuyo objetivo sea la predicción de eventos deportivos mediante técnicas de aprendizaje supervisado. Esto permitirá, por un lado identificar qué obstáculos o problemas se han encontrado otros investigadores durante el proceso y cómo se han solventado, es decir, servirán como base para evitar repetir los mismos problemas que ya otras personas se hayan encontrado.

Finalmente, se realiza un análisis de las capacidades y características principales de las herramientas más extendidas en cuanto al entrenamiento de redes neuronales.

A. Evolución de la inteligencia artificial hasta la actualidad

Aunque el concepto de inteligencia artificial nació en 1956 en la *Conferencia de Dartmouth* ([1]), la investigación y el desarrollo de sistemas con algún tipo de conocimiento *racional* ya comenzó varios años atrás.

La evolución en este campo no ha sido siempre tan rápida como en los últimos años. Uno de los motivos principales han sido los avances en la capacidad de computación que tienen los ordenadores hoy en día, lo cual ha posibilitado que el desarrollo de redes neuronales vuelva a estar en auge debido a los avances que han surgido en la última década.

Bien es cierto que dista aún bastante de igualar las capacidades del cerebro humano, sin embargo, con los avances obtenidos en los últimos años ha quedado reflejado su potencial y su gran capacidad de resolver problemas.

B. Inteligencia Artificial en deportes

El concepto de Inteligencia Artificial está alterando positivamente el deporte y elevándolo a un nivel de éxito inaudito. Aunque los hechos confirman que las mediciones y la investigación han asumido un trabajo focal en los deportes durante bastante tiempo, la Inteligencia Artificial está afectando a cómo se planifican los partidos, cómo se juegan y cómo atraen al público. Está desarrollando una forma más creativa de triunfar en el deporte, tanto para los deportistas como para los entrenadores, proporcionando conocimientos constantes del juego,

previsión de estrategias de juego que ayudan a capacitar al jugador a elegir el procedimiento correcto e incluso alarmarle si se produce una disminución del rendimiento o una lesión.

Tal y como se indica en el artículo *Deportes y Data Mining* ([2]), en el mundo de los deportes se recolecta infinidad de información con el fin de construir una base de datos sólida. Dicha información puede ser explotada con infinidad de objetivos que son comunes a cualquier deporte.

Sin embargo, no todo tienen por qué ser ventajas. Más allá de la precisión de la predicción (la cual puede no ser tan buena como debería), otro problema notable del uso de algoritmos de aprendizaje automático es la perpetuación de posibles sesgos en los procesos de evaluación y toma de decisiones ya que los datos utilizados pueden estar subrepresentando a ciertos grupos demográficos, y los análisis sobre las diferencias de los subgrupos pueden no ser tan sólidos.

C. Estudios de actualidad

Como se ha comentado en el apartado A, la IA está presente en cualquier lugar. En este proyecto se va a aplicar dentro del mundo de los deportes, donde el objetivo en concreto es conseguir la caracterización del estado de jugadores de tenis profesionales para predecir resultados de partidos futuros.

Una de las partes más importantes a la hora de realizar cualquier proyecto de aprendizaje máquina es la selección de variables predictoras o *features*. Para el caso de la caracterización de jugadores, hay varios factores o parámetros que pueden influir directamente en sus resultados, por ejemplo, su estado físico es uno de los elementos clave que podrán ser buen candidato para entrenar un modelo.

C.1. Análisis del rendimiento de jugadores en base su forma física

En el artículo *Machine learning for the prediction of professional tennis matches* [3] se realiza este tipo de aproximación. En él se basan en la selección de parámetros que definen la forma física de los jugadores, lo cual puede proporcionar indicadores de qué jugadores se encuentran mejor que otros, determinando así quién podrá vencer un partido en función de si su estado de forma supera al del contrincante. Además, tienen en cuenta variables de distintos tipos, algunas están relacionadas con la complexión corporal del jugador, otras con pruebas de esfuerzo, coordinación, etc...

En dicho artículo, se han recopilado este tipo de datos sobre 1002 tenistas eslovenos, tanto hombres como mujeres, que son preprocesados para posteriormente entrenar distintos tipos de algoritmos de aprendizaje máquina con el fin de elegir aquel que mejor resultados proporcione.

C.2. Análisis del rendimiento de jugadores en base a datos históricos

Otra de las aproximaciones a la hora de estimar la probabilidad de ganar o perder un partido es utilizar directamente técnicas de aprendizaje supervisado sobre datos estadísticos históricos, es decir, atender a información de un gran número de partidos ya disputados sobre los que ya se conoce el resultado y los jugadores que se enfrentaron entre sí.

Este tipo de enfoques no tiene en cuenta parámetros particulares del jugador, como su altura, peso o estado físico general si no que únicamente se centra en resultados obtenidos durante el histórico de los partidos que ya ha disputado anteriormente.

En el artículo *Prediction of the successfulness of tennis players with machine learning methods* [4] se hace un estudio aplicando esta técnica. En él, se recogen única y exclusivamente estadísticas sobre partidos que ya han tenido lugar, sin prestar atención a información sobre el estado físico de los jugadores.

En definitiva, existen distintas teorías y metodologías para obtener el resultado deseado. La clave reside en escoger la información adecuada seleccionar los parámetros que puedan ser buenos candidatos o influyentes en la predicción del resultado de un partido y posteriormente entrenar distintos modelos y evaluarlos para poder seleccionar aquel que mejores resultados proporcione.

D. Herramientas para el entrenamiento de redes neuronales

Una vez se ha visitado la parte funcional del proyecto, es hora de realizar un análisis de las herramientas del mercado que actualmente se utilizan para este tipo de funcionalidades.Se analizarán aquellas más extendidas entre los desarrolladores de modelos de IA y que además son *Open Source* por razones de costes, pero que no por ello son menos eficientes.

- **Tensorflow**: Desarrollada por Google [5]. Su finalidad es proporcionar a los desarrolladores un ecosistema compuesto por todas las herramientas necesarias para el desarrollo de proyectos de aprendizaje automático.
- **Keras**: Está compuesta por una serie de librerías de código abierto escrita en Python por François Chollet [6]. Está enfocada al desarrollo de redes neuronales, más en concreto al *deep learning*. La mayor ventaja de Keras es su integración con el core de Tensorflow, es decir, los desarrolladores pueden explotar las capacidades que ofrece Tensorflow mediante unas librerías más sencillas de utilizar.
- **PyTorch**: Desarrollada por Facebook [7] y está enfocada al desarrollo de redes neuronales. Está pensada para ser ejecutada directamente sobre GPUs. A diferencia de Tensorflow, su interfaz es bastante sencilla, por lo que permite trabajar directamente con el core sin la necesidad de instalar o utilizar librerías de niveles superiores como Keras.

III. OBJETIVOS Y METODOLOGÍA

El presente proyecto tiene un objetivo global que es **crear un modelo predictivo en base a datos obtenidos y depurados de un portal de información deportiva**.

El factor humano en la toma de decisiones implica siempre ciertas limitaciones que deben ser tenidas en cuenta. Las personas por naturaleza son propensas a cometer errores que dependiendo de la tarea que se realiza pueden tener consecuencias catastróficas. Esto se debe a que bajo determinadas circunstancias, ciertos factores cognitivos como la atención pueden verse afectados, lo cual puede producir un error en un cálculo o simplemente algo que una persona ha pasado por alto sin darse cuenta. Estas limitaciones son las que se intentan solventar haciendo uso de la tecnología. Además, se juegan diariamente numerosos partidos de tenis. Analizarlos todos para poder ofrecer predicciones resulta inviable para un ser humano, tanto por el esfuerzo que ello conlleva como el tiempo que debe ser dedicado. Este es otro de los puntos que se busca solventar en este proyecto, donde la cantidad de información sobre la que se trabajará será muchísimo mayor, automatizando todo el proceso desde la extracción de datos actualizados, su preprocesado, entrenamiento de un modelo y hasta finalmente la obtención de predicciones.

Tras analizar distintos portales web, se ha decidido explotar la información de *Sofascore* ya que contiene una gran cantidad de información actualizada y además puede ser obtenida mediante sus APIs REST de forma pública. Una vez se haya conseguido tener una base de datos poblada con información, el paso siguiente sería **entrenar un modelo mediante una red neuronal** que sea capaz de aprender y detectar ciertos patrones en dichos datos que serían prácticamente indetectables para un humano.

Desglosando todo esto en objetivos más a bajo nivel, se podría concluir:

- Crear un microservicio para obtener datos actuales de portal deportivo y persistir los datos en una base de datos
- Automatizar el proceso de obtención de datos y exponerlos mediante APIs Rest
- Leer los datos obtenidos desde un programa en Python para crear un dataset que haga más manejable su uso
- Limpiar los datos, procesarlos para eliminar inconsistencias y convertir variables

categóricas en numéricas

- Entrenar un modelo basado en una red neuronal en base a los datos
- Evaluar las predicciones generadas por el modelo entrenado
- Desarrollar un bot para obtener predicciones a través de una app de mensajería

IV. CONTRIBUCIÓN

En esta sección se describen cada uno de los componentes desarrollados y la función que estos aportan a la solución final.

Desde el punto de vista de arquitectura, se han definido tres componentes que brindarán la solución final.

El primer componente se trata de un **microservicio para obtener datos**, que se encargará de consumir las APIs del portal de deportes para obtener la información necesaria para entrenar la red neuronal. Este proceso ocurre de forma automática mediante un cronjob que se ejecuta todas las noches a las 23:55 h, ya que es cuando los partidos del día han finalizado y se pueden obtener los resultados de los mismos. La idea es obtener la lista departidos pasados que ya han sido disputados e información característica de los jugadores para almacenarla en una base de datos.

Otro de los componentes clave será la **red neuronal**. Cuando se desarrollan redes neuronales, es muy difícil acertar a la primera sobre qué arquitectura o modelo seguir. Debe ser recordado que las redes neuronales o modelos de *deep learning* se caracterizan por ser modelos muy versátiles capaces de adaptarse a gran variedad de problemas. Si bien esto es una ventaja, su implementación se puede realizar bastante tediosa, ya que las opciones de configuración que ofrecen son inmensas.

A grandes rasgos, los parámetros que se deben tener en cuenta son los siguientes:

1. Número de neuronas y capas ocultas
2. Función de activación: Devuelve una salida que será generada por la neurona dada una entrada o conjunto de entradas. Históricamente, se han propuesto muchas funciones de activación, pero en este documento se describirán las dos utilizadas para el desarrollo de la red neuronal: la sigmoide y ReLU.

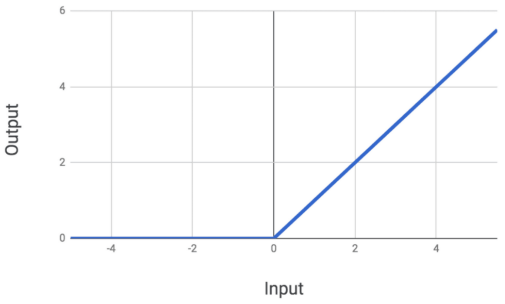


Figura 1: Función de activación ReLU ([8])

Por otro lado, ya que estamos ante un problema de regresión lineal, la salida de la red estará compuesta por la función *linear*, también conocida como identidad, permite que lo de la entrada sea igual a la salida.

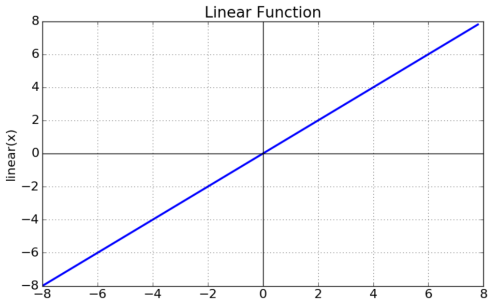


Figura 2: Función de activación Lineal ([9])

3. Arquitectura de la red: Se va a utilizar la opción *fully connected*, quiere decir que todos las neuronas de la red están interconectadas unas con otras.
4. Optimizador: En deep learning existe el concepto de pérdida (loss), que indica el rendimiento del modelo en un instante determinado. Básicamente, lo que se busca es tratar de minimizarla, ya que una pérdida menor significa que el modelo va a

funcionar mejor. Hay varios tipos de optimizadores y el emplear uno u otro depende de muchos factores. Aquí vuelve a entrar en juego la prueba y error que se ha comentado anteriormente. En este proyecto se va a utilizar el optimizador *Adam*.

5. Learning rate: La tasa de aprendizaje es un hiperparámetro que controla cuánto cambiar el modelo en respuesta al error estimado cada vez que se actualizan los pesos del modelo durante la fase de entrenamiento. La elección de la tasa de aprendizaje es un reto, ya que un valor demasiado pequeño puede dar lugar a un largo proceso de entrenamiento, mientras que un valor demasiado grande puede dar lugar al aprendizaje subóptimo de pesos o a un proceso de entrenamiento inestable.

Finalmente, el último de los componentes será un **bot predictor**. El componente como tal es muy simple, sin embargo, debe de comunicarse con los dos componentes que se han desarrollado anteriormente para poder realizar su función. Por un lado obtener la lista de partidos futuros aún no disputados del microservicio que obtiene los datos, y por otro enviarlos a la red neuronal para que prediga los resultados.

V. RESULTADOS O EVALUACIÓN

Tal y como se ha comentado, las opciones disponibles para configurar redes neuronales son muy amplias y es muy difícil obtener buenos resultados a la primera. Cuando no se tienen datos suficientes o la configuración con la que se entrena la red no es la adecuada, suele producirse el efecto llamado *overfitting*. Este fenómeno consiste en que la red aprende o se ajusta muy bien a los datos del conjunto de entrenamiento, pero es incapaz de generalizar, es decir, cuando obtenga un dato de entrada distinto al conjunto de entrenamiento no será capaz de proporcionar buenas predicciones.

El overfitting puede detectarse atendiendo a la evolución de los parámetros de precisión y pérdida durante el proceso de entrenamiento según se cita en el artículo *'The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial'* ([10]). En la figura 3 se muestra un ejemplo de overfitting:

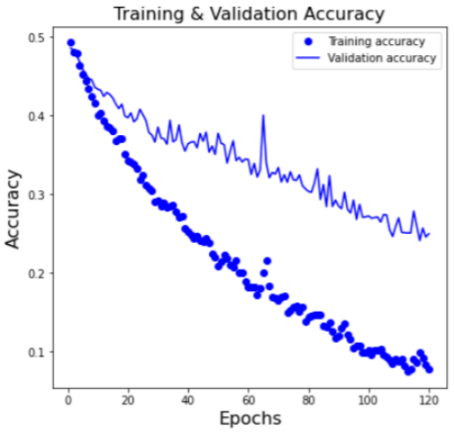


Figura 3: Ejemplo de detección de overfitting (elaboración propia, 2021)

Tal y como se observa, a medida que la red está siendo entrenada llega un momento en el que la precisión del conjunto de test se 'separa' de la precisión del conjunto de entrenamiento, lo cual quiere decir que la red no es capaz de generalizar y proporcionar buenos resultados para datos distintos a los de entrenamiento. En este caso, añadiendo las capas de batch normalization y dropout se ha conseguido corregir este problema, tal y como muestra la figura 4:

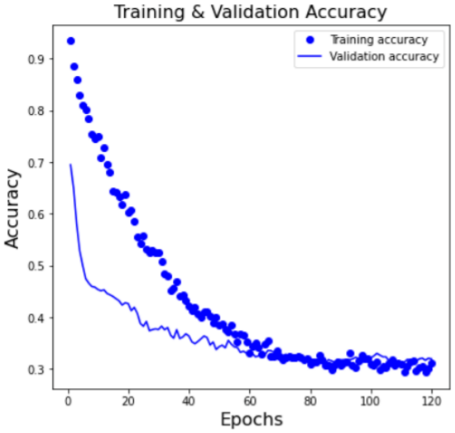


Figura 4: Comportamiento con overfitting corregido (elaboración propia, 2021)

En este caso, tanto la precisión del conjunto de entrenamiento como la precisión del conjunto de test van más o menos a la par y siguen incrementándose durante el entrenamiento de la red de igual forma.

A modo de resumen, la configuración de todos los parámetros adaptados para el entrenamiento de la red neuronal ha sido de 120 epochs, 2 capas ocultas (1000 neuronas con ReLU y Dropout de 0,3 y 200 neuronas con ReLU y Dropout de 0,3), learning rate de 1e-3, optimizador Adam, y batch size 40.

VI. DISCUSIÓN O ANÁLISIS DE RESULTADOS

Una vez analizado el proceso de entrenamiento, el paso final es evaluar las predicciones de la red neuronal.

En la figura 5 se pueden observar los resultados obtenidos ante las treinta primeras entradas del conjunto de validación:

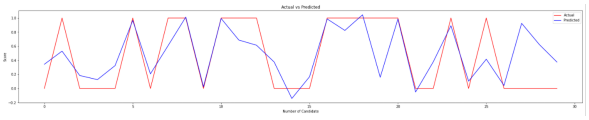


Figura 5: Predicciones obtenidas (elaboración propia, 2021)

La línea roja indica las victorias del jugador en casa, siendo valor 1 cuando gana y 0 cuando pierde, mientras que la línea azul indica las predicciones obtenidas por la red neuronal.

En términos generales y sobre el ejemplo de la figura anterior, se observa que la red neuronal produce más aciertos que errores, aunque no siempre es capaz de acercarse con gran precisión al valor que debería ser. Los casos en los que falla, lo hace con muchísimo error, como por ejemplo en las muestras que se muestran en la figura 6:

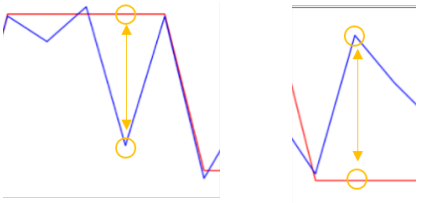


Figura 6: Errores de predicción (elaboración propia, 2021)

Este fenómeno sería un buen caso a analizar en evoluciones futuras del proyecto. Posiblemente, según se vayan obteniendo datos de más partidos y aumente el número de información con la que se entrena la red neuronal, este tipo de comportamientos se corrija. No obstante se puede concluir en rasgos generales que situando el rango de ganar/perder en 0.5, el modelo es capaz de generalizar y dar predicciones que se acercan bastante a lo esperado.

VII. CONCLUSIONES

En esta última sección del documento, trataremos de resumir la experiencia obtenida durante el proceso de desarrollo, así como la viabilidad del mismo en cuanto a los resultados obtenidos. Además, se nombrarán los *gaps* identificados que pueden ser mejorados en un futuro, así como nuevas funcionalidades que pueden enriquecer el producto desarrollado.

La parte más crítica de todo este desarrollo se concentra en la red neuronal, ya que es la encargada de proporcionar la predicción sobre un determinado partido de tenis, que era el objetivo principal de este proyecto. Considerando un pequeño margen de error, se ha conseguido inferir dicha información de una forma correcta, si bien es cierto que aún hay cierto margen de mejora.

En cuanto al motor de obtención de datos, ya se ha conseguido que funcione de forma autónoma, lo cual significa que siempre mantendrá la base de datos con la información más actualizada de cada día. En este sentido se ha conseguido completar el requisito inicial en su totalidad de forma más que satisfactoria.

Por último, el notificador de predicciones es el encargado de explotar la red neuronal, es de-

cir, se ocupa de obtener la predicción de partidos aún no disputados, lo cual era el objetivo de cara a las apuestas deportivas. Esta función está cubierta también, aunque tenga que ser ejecutada de forma manual.

En definitiva, se considera que los objetivos propuestos inicialmente quedan cubiertos para esta primera fase del proyecto, aunque aún pueda ser mejorado en muchos aspectos en un trabajo futuro, como por ejemplo:

- Aumento del número de features mediante web scraping de otros portales de deportes
- Implementación de mecanismos de seguridad (OAuth) en las apis del microservicio de importación de datos
- Automatización del entrenamiento de la red neuronal
- Automatización del proceso de notificaciones de las predicciones
- Aumento de la granularidad de las predicciones

A. APÉNDICES

Los repositorios donde se aloja el código fuente desarrollado:

- Microservicio extracción de información

  `https://github.com/ivan1405/sofascore-data-miner`

- Red neuronal

  `https://github.com/master-in-artificial-intelligence/tfm-neural-network`

- Bot de applicación de mensajería

  `https://github.com/master-in-artificial-intelligence/tfm-bot-predictor`

Referencias

[1] Jørgen Veisdal. Conferencia de Dartmouth, 2019.

[2] Pedro Univaso. Deportes y data mining.

[3] Michal Sipko and William Knottenbelt. Machine learning for the prediction of professional tennis matches. *MEng computing-final year project, Imperial College London*, 2015.

[4] Andrej Panjan, Nejc Sarabon, and Aleš Filipčič. Prediction of the successfulness of tennis players with machine learning methods. *Kinesiology*, 42(1):98–106, 2010.

[5] Google. Tensorflow, 2021.

[6] François Chollet. Keras, 2015.

[7] Facebook. PyTorch, 2016.

[8] bootcampai. Función ReLU, 2020.

[9] mihaileric. Función Lineal, 2020.

[10] Benyamin Ghojogh and Mark Crowley. The theory behind overfitting, cross validation, regularization, bagging, and boosting: Tutorial. 05 2019.