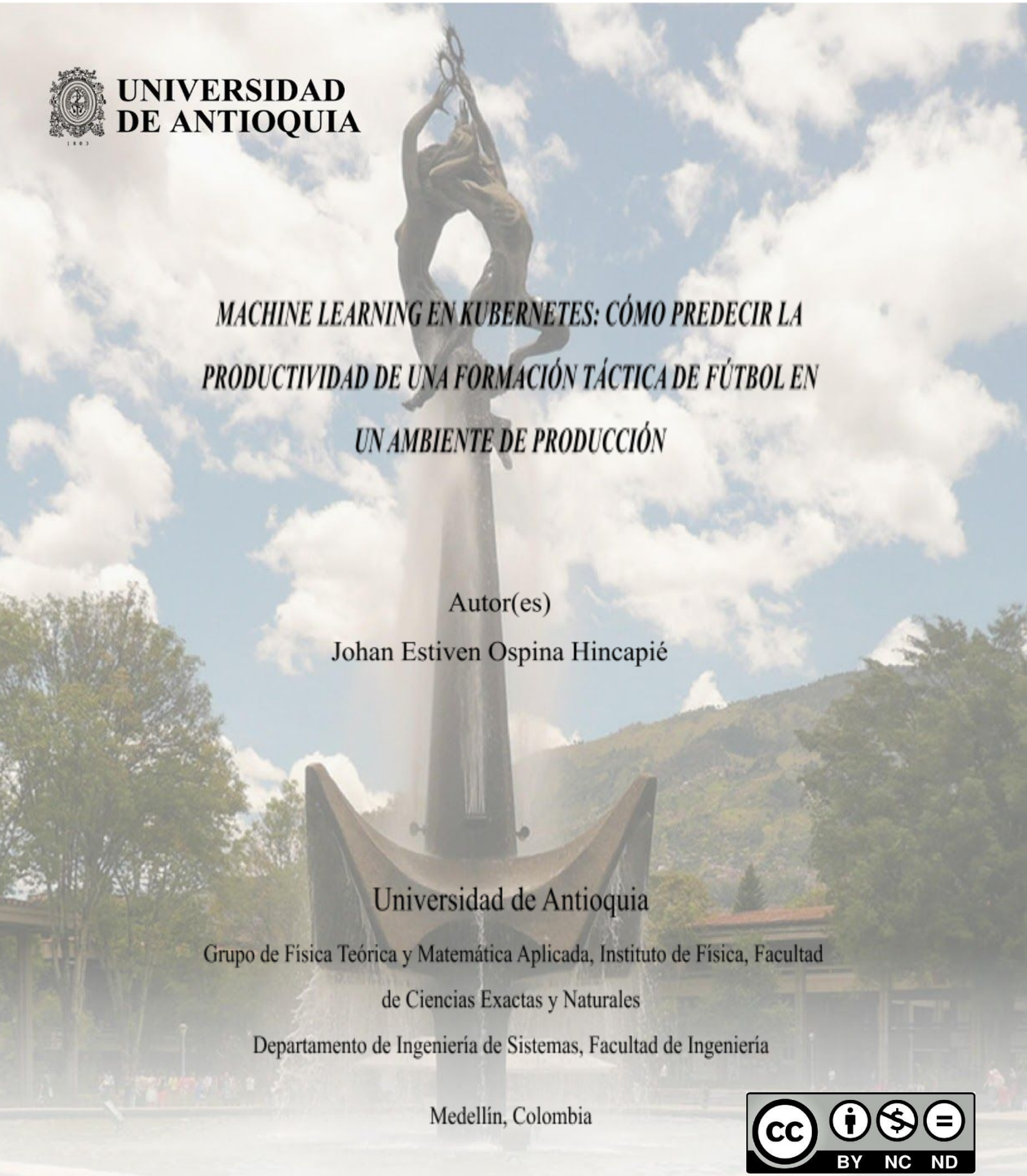

UNIVERSIDAD DE ANTIOQUIA

MACHINE LEARNING EN KUBERNETES: CÓMO PREDECIR LA PRODUCTIVIDAD DE UNA FORMACIÓN TÁCTICA DE FÚTBOL EN UN AMBIENTE DE PRODUCCIÓN

Autor(es)

Johan Estiven Ospina Hincapié

Universidad de Antioquia

Grupo de Física Teórica y Matemática Aplicada, Instituto de Física, Facultad de Ciencias Exactas y Naturales

Departamento de Ingeniería de Sistemas, Facultad de Ingeniería

Medellín, Colombia

2020

MACHINE LEARNING EN KUBERNETES: CÓMO PREDECIR LA PRODUCTIVIDAD DE UNA FORMACIÓN TÁCTICA DE FÚTBOL EN UN AMBIENTE DE PRODUCCIÓN

Johan Estiven Ospina Hincapié

Informe final para la modalidad de práctica empresarial presentado como requisito para optar al título de:

Ingeniero de sistemas

ASESORES

Ph.D. Leonardo Augusto Pachon Contreras

M.Sc. Joseph Fabricio Vergel Becerra

Grupo de Física Teórica y Matemática Aplicada, Instituto de Física, Facultad de Ciencias Exactas y Naturales, Universidad de Antioquia
Departamento de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de Antioquia

Medellín, Colombia

2020

ML En K8S: PREDECIR LA PRODUCTIVIDAD DE UNA FORMACIÓN TÁCTICA DE FÚTBOL

Nota del autor

- Practica empresarial desarrollada en *guane Enterprises S.A.S,* Medellin, Antioquia, Colombia.
- Departamento de Ingeniería de Sistemas, Facultad de Ingeniería, Universidad de Antioquia, Medellín, Colombia.
- Práctica asesorada de manera externa por Joseph Fabricio Vergel Becerra y de manera interna por Leonardo Augusto Pachón Contreras.
- Práctica desarrollada por Johan Estiven Ospina Hincapié

ML En K8S: PREDECIR LA PRODUCTIVIDAD DE UNA FORMACIÓN TÁCTICA DE FÚTBOL

Resumen

En la actualidad, la "inteligencia artificial" es capaz de proponer soluciones a problemas cada vez más complejos, de una manera mucho más versátil y flexible en comparación a las soluciones tradicionales. No obstante, llevar a producción estos sistemas construidos sobre algoritmos de *machine learning* y *deep learning*, es un reto constante que requiere de un gran abanico de funcionalidades como lo son la auto escalabilidad horizontal, autocuración, lanzamientos automáticos, atomicidad, entre otros. Por consiguiente, el objetivo principal de este proyecto consistió en diseñar e implementar una arquitectura de *software* sólida que sirviese como infraestructura base para el despliegue en producción de soluciones predictivas. Para ello, se retomó el problema de la estimación de la productividad de la formación tactical inicial para equipos del fútbol Europeo. Una vez replicado el desarrollo desde la perspectiva de ciencia de datos y *machine learning* se diseñó y construyó una plataforma web predictiva basada en una arquitectura de microservicios. Esta arquitectura utiliza tecnologías modernas como *Docker* y *Kubernetes* que permiten un despliegue eficiente en producción, permitiendo así incluso el entrenamiento de algoritmos por demanda del usuario según las configuraciones y parámetros elegidos por este. Finalmente, como perspectivas para la continuación del trabajo se plantea la incorporación de *service mesh* con *Istio* para el descubrimiento de servicios, *Kubeflow* como operador de *Kubernetes* para el desarrollo de modelos de *machine learning* y *rancher* para automatizar la construcción del cluster.

Palabras clave: Machine learning, microservicios*,* productividad, fútbol, *Docker*, *Kubernetes.*

ML En K8S: PREDECIR LA PRODUCTIVIDAD DE UNA FORMACIÓN TÁCTICA DE FÚTBOL

1. Introducción

El *machine learning* y el *deep learning* ponen a disposición un sinfín de herramientas capaces de modelar la dinámica de sistemas complejos. Sin embargo, el despliegue en un ambiente de producción de cualquier solución basada en estas tecnologías es todo un nuevo reto que requiere la combinación de buenas prácticas de desarrollo de *software*, patrones arquitectónicos robustos, protocolos de comunicación flexibles, entornos de virtualización eficientes y de alto rendimiento, entre muchas otras funcionalidades. En conjunto, todas estas características permiten el correcto flujo de las operaciones de una plataforma que suministre cualquier servicio predictivo.

Predecir es tan solo uno de la gran cantidad de engranajes que componen las plataformas predictivas y aunque fundamental, no comprende la mayor trascendencia dentro de un producto "cognitivo" o basado en "inteligencia artificial", que es como comúnmente se le conoce desde la perspectiva comercial. Por tanto, el desarrollo de arquitecturas modulares y atómicas, con escalado horizontal y balanceo de carga para hacerlas auto-escalables, que cuenten además con capacidades *self-healing* o de autocuración, lanzamientos automatizados y *rollback,* programación automatizada, entre otras características, fijan de entrada requisitos obligatorios para el desarrollo de cualquier producto en cualquier industria. Cumplir con esta alta demanda de funcionalidades es un reto a solucionar y al que le apuntan una gran cantidad de tecnologías como *Asynchronous* WSGI, *FastAPI*, *Docker*, *Swarm*, *Kubernetes* y muchas otras más.

Por consiguiente, en este trabajo se desarrolló e implementó una arquitectura basada en microservicios, escrita esencialmente en *Python,* servida de forma asíncrona con *FastAPI*, contenerizada en *Docker* y orquestada con *Kubernetes* para el entrenamiento de algunos algoritmos de *machine learning* y el despliegue de sus predictores. El objetivo principal fue generar la infraestructura necesaria para una plataforma web predictiva, que para este caso en específico fuese capaz de realizar predicciones sobre la productividad de formaciones tacticas de futbol, como un ejemplo de ejercicio predictivo. Aquí se presentan las principales características

y los resultados más relevantes, en pro de mantener la continuidad del proyecto dentro de la empresa en la que fue realizada esta práctica empresarial.

2. Objetivos

Dentro del marco de desarrollo de esta práctica se fijó un objetivo general y tres objetivos específicos que se describen a continuación.

2.1 Objetivo general

Construir una plataforma web, en una arquitectura de microservicios, capaz de estimar la productividad de una formación táctica de un equipo de fútbol a través del entrenamiento de algoritmos de *machine learning* y del despliegue de sus respectivos predictores,.

2.2 Objetivos específicos

- Realizar un análisis exploratorio de los datos disponibles y proponer un esquema de datos unificado, en una base de datos no relacional, de encuentros deportivos y de habilidades de los jugadores..
- Modelar la productividad de una determinada formación táctica a partir de algoritmos de *machine learning* clásico.
- Implementar una arquitectura basada en microservicios, para el despliegue de la plataforma web, a partir de las tecnologías de contenedores (*Docker*) y orquestadores (*Kubernetes*).

3. Marco Teórico

Python es un lenguaje de programación interpretado, de alto nivel y de propósito general, cuya filosofía hace hincapié en la legibilidad del código. Se trata de un lenguaje de programación multiparadigma, es decir, que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional[1]. A su vez, *Python* ofrece todo un set de herramientas para el desarrollo de *software* de calidad como es el caso del marco de estilo *Python Enhancement Proposal* N° 8[1] (PEP8). El PEP8 establece un conjunto de normas en pro de facilitar la estructuración del código, mejorar la legibilidad y al mismo tiempo la organización, estandarizando el código de manera eficiente y óptima. Por consiguiente, el uso de guías de estilo es considerado como elemento clave para desarrollar *software* de calidad[2, 3].

Por otra parte, el PEP-3107[2] introdujo el concepto de anotaciones. La idea básica de esta propuesta es sugerir a los lectores del código acerca de qué esperar como valores de argumentos en funciones. El uso de la palabra pista o sugerencia (*hint*) no es casual; las anotaciones permiten sugerencias de tipo (*type hints*) explícitas, siendo otra recurrente buena práctica para un desarrollo más eficiente [3]. Las anotaciones permiten especificar el tipo esperado de algunas variables que se han definido. En realidad, no se trata solo de los tipos, sino de cualquier tipo de metadatos que ayudan a tener una mejor idea de lo que representa realmente una variable, función, o clase. En el caso de *Python*, la biblioteca nativa *typing*[3] estructura de forma simple las sugerencias de tipo para listas, diccionarios, tuplas, entre otras estructuras computacionales.

De manera complementaria, la validación de datos y gestión de configuraciones mediante anotaciones "puede" , aplicar las sugerencias de tipo en tiempo de ejecución y proporcionar errores fáciles de usar cuando los datos no son válidos. Sin embargo, como lo estipula el PEP-484[4], esta validación y gestión es un elemento adicional. Este es el objetivo de bibliotecas

[1] PEP 8 -- Style Guide for Python Code: https://www.python.org/dev/peps/pep-0008/
[2] PEP 3107 -- Function Annotations: https://www.python.org/dev/peps/pep-3107/
[3] typing — Support for type hints: https://docs.python.org/3/library/typing.html
[4] PEP 484 -- Type Hints: https://www.python.org/dev/peps/pep-0484/

de terceros como *Pydantic*[5], que junto a *typing* proporciona la creación, validación y personalización de anotaciones para estructuras complejas que ingresan al sistema, se transforman dentro del *workflow* de las reglas lógicas, se retornan a las diferentes capas de presentación y que de manera simultánea se almacenan en la persistencia del sistema mediante el uso de ORMs (*Object-Relational Mapping*).

3.1 Análisis estadístico de los datos

En el uso de herramientas estadísticas sobre el conjunto de datos a analizar permite extraer el máximo conocimiento de la información proporcionada y hacer explicaciones acerca de variables conocidas y desconocidas dentro de la naturaleza del problema. La interpretación de los datos puede convertirse rápidamente en una falla masiva sin un análisis estadístico previo. Por esta razón, la elección del procedimiento estadístico apropiado depende del tipo de dato. Los datos pueden ser categóricos o numéricos. Si las variables son numéricas, al mismo tiempo pueden clasificarse entre continuos y discretos, con estrategias estadísticas bien definidas para cada subtipo. Por el contrario, si las variables representan categorizaciones cualitativas, entonces existen diferentes subtipos que se definen a partir de la cardinalidad y ordinalidad de las mismas[4].

Una vez definido el tipo de dato, también se distingue entre datos univariados que son dados por una sola variable, e.g., el peso de un jugador de fútbol. Los datos bivariados tienen dos parámetros o variables, por ejemplo, la posición (x, y) de un jugador en el campo de juego. Y finalmente, los datos multivariados que dependen de tres o más variables, como es el caso de la transferencia de un jugador con unas determinadas características, de un club a otro por una determinada suma de dinero[4]. Los conjuntos de datos pueden contener cientos o miles de estas

[5] pydantic — Overview: https://pydantic-docs.helpmanual.io/

variables, por lo que es necesario analizarlos, interpretarlos, filtrarlos y seleccionarlos, para lo que las estrategias de visualización de datos brindan la posibilidad de definir estos criterios.

La tarea dominante de la corteza cerebral es extraer información visual de los patrones de actividad en la retina, por lo que las técnicas de visualización de datos facilitan la detección patrones y la formulación de criterios[4]. Es entonces la estadística descriptiva quien proporciona como herramienta el gráfico, una representación visual de los datos que de manera clara aumenta el nivel de comprensión, especialmente en casos de patrones, tendencias e interconexiones complejas y subyacentes en el dominio de los datos. El diagrama de construcción de visualización de datos de la **Fig. 1**, es parte integral del análisis de datos y ayuda a decidir el tipo de gráfico más adecuado, facilitando así cómo analizarlos, cómo se distribuyen, como seleccionarlos, etc, acorde a los diferentes tipos y características de los mismos[5].

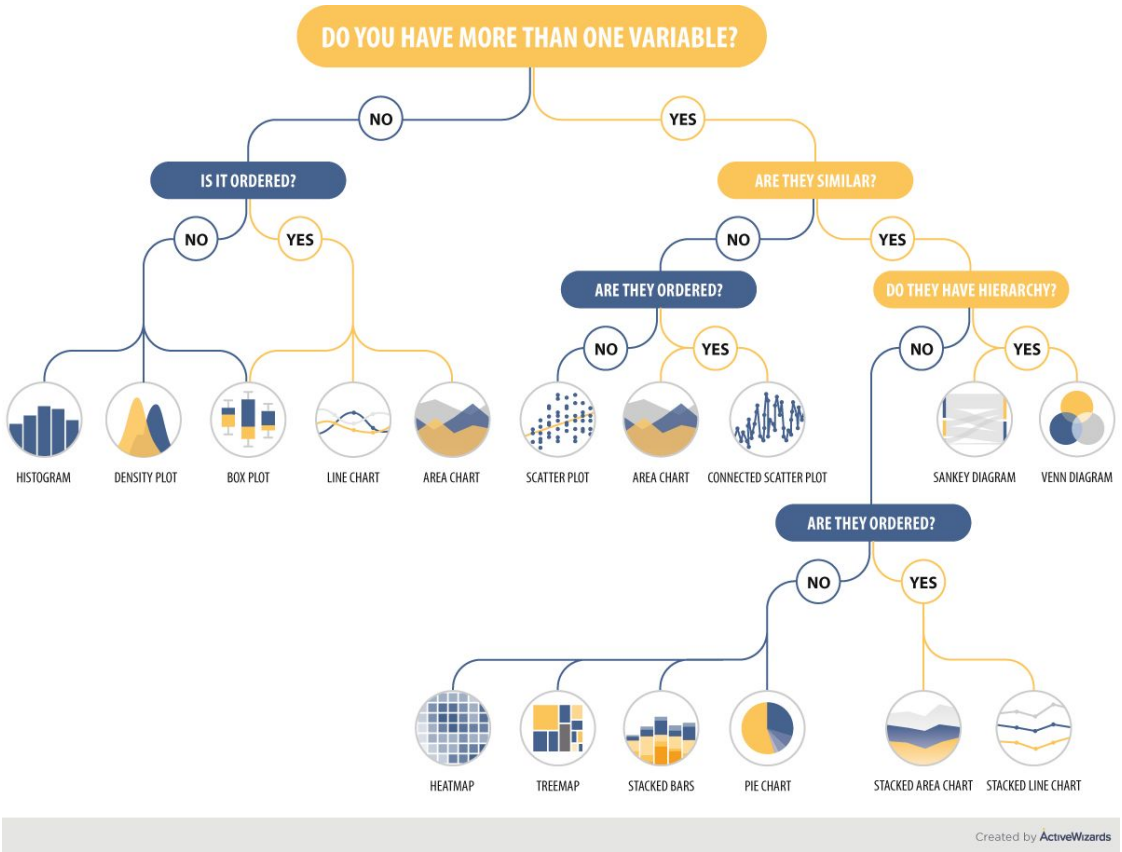

Figura 1. Diagrama de construcción de la visualización de datos. Fuente: [5].

3.2 Almacenamiento de datos en persistencias no relacionales

Es bastante común que los datos se utilicen como estructuras complejas dando lugar a representaciones de las entidades del contexto de los problemas que se estén estudiando. De esa forma se dividen los datos en dos familias categorizadas como datos estructurados y no estructurados. Los datos estructurados son aquellos que tienen bien definida su forma, es decir, conocemos su longitud, formato y el tamaño de datos. Comúnmente, los datos estructurados están asociados con patrones fáciles de asociar y donde la búsqueda sobre ellos es relativamente sencilla. Un ejemplo claro son las tablas de una base de datos relacional. Por otra parte, los datos no estructurados, si bien pueden tener una organización interna, no tienen un esquema de datos definido por lo que pueden contener tipos de datos no comunes o composiciones complejas de modelos de datos por ejemplo imágenes, ficheros de audio y video. Las bases de datos NoSQL están capacitadas para almacenar este tipo de datos.

3.3 Modelamiento como proceso de aprendizaje de los datos

El procesos de modelado o generación de modelos es herramienta cuantitativa utilizada en diversas áreas del conocimiento. Esto ha traído consigo que la definición de modelo se diversifique y aparezcan diferentes definiciones del mismo. En términos generales, un modelo es el artefacto capaz de capturar la relación intrínseca que existe entre variables exógenas (independientes) y endógenas (dependientes)[6] mediante una determinada metodología y obtener valores endógenos "predichos" a partir de información *a priori* de nuevos valores exógenos de interés.

Debido a la diversificación discutida arriba, existen modelos estadísticos capaces de capturar las correlaciones y significancias estadísticas de las variables exógenas y realizar inferencias sobre

[6] Wikipedia - Exogenous and endogenous variables:
https://en.wikipedia.org/wiki/Exogenous_and_endogenous_variables

nuevas muestras, por ejemplo,el caso de los enfoques econométricos para el pronóstico de series de tiempo en la bolsa de valores. Otro tipo de modelo son los matemáticos no estocásticos o determinísticos, como por ejemplo, las ecuaciones diferenciales ampliamente utilizadas para el modelamiento de poblaciones en epidemiología. Por último, se tienen los modelos basados en datos propuestos en los campos del *machine learning* y *deep learning,* que a pesar de tener un enfoque probabilístico, en realidad "aprenden" de las coincidencias en las observaciones. En las siguientes subsecciones se detallan algunos aspectos relevantes para la definición de lo que en este trabajo se predice (productividad) y como se predice (análisis de regresión).

3.3.1 Medición de la productividad de jugadores en deportes de conjunto

Para los responsables de la toma de decisiones en los deportes, así como para los miembros de los medios de comunicación y los fanáticos, es crucial saber cuales fueron los mejores jugadores en un encuentro. Esto se ha tratado de responder desde que se fundaron las ligas organizadas de deportes de conjunto en el siglo XIX [6]. Como respuesta, los equipos deportivos comenzaron a rastrear las estadísticas de los jugadores. Habitualmente, se realiza un seguimiento de una multitud de estadísticas para separar el rendimiento del individuo, de los resultados que se observan para el equipo. Se conoce entonces quién ganó o perdió el encuentro y con estadísticas de rendimiento, los equipos esperan determinar qué jugadores fueron responsables del éxito (o fracaso) del equipo. No obstante, para que las estadísticas de los jugadores tengan valor, estos números deben estar conectados a los resultados[6].

Entre las métricas de rendimiento de un equipo pueden tenerse métricas de efectividad asociadas directamente al número de anotaciones recibidas o generadas. Por su parte, la productividad abarca una concepción más general, en la que se componen diferentes estadísticas de acciones que favorecen a obtener la victoria o sufrir una derrota. Existen diferentes propuestas para la medición de la productividad de los jugadores, como es el caso de la calificación del mariscal de campo de algun equipo de la National Football League- NFL- de los Estados Unidos que tiene en

cuenta el número de jugadas terminadas, pases de touchdown lanzados, intercepciones lanzadas e intentos de pase [6]. Otro enfoque presente en la NFL es la medición del rendimiento en un deporte de invasión compleja y proveniente del béisbol y del baloncesto en donde se emplea un análisis de regresión para vincular los resultados; específicamente, los resultados de la temporada final con las estadísticas rastreadas por jugador, pero también a nivel de equipo[6]. Esto introduce otro tipo de estrategias que tiene como objetivo calcular la productividad a nivel de equipo, una de ellas es cálculo del factor total utilizando el índice de Malmquist que ha sido aplicado anteriormente al fútbol, específicamente, al cálculo de la productividad de los equipos de fútbol españoles de primera división [7].

3.3.2 Definición de un problema de regresión

Sea el conjunto de entrenamiento que comprende N observaciones de x, escritas $\widehat{x} \equiv (x_1, x_2, ..., x_N)^T$, junto con las observaciones correspondientes de los valores de la variable objetivo t, denotado $\widehat{t} \equiv (t_1, t_2, ..., t_N)^T$. El objetivo es explotar la información que contiene el conjunto de entrenamiento $\widehat{x}$ para hacer predicciones del valor de la variable objetivo $\widehat{t}$ usando algún nuevo valor x_i de la variable de entrada. Esto implica tratar implícitamente de descubrir la función que define el comportamiento de los datos, lo que lo hace un problema enormemente difícil ya que se tiene que generalizar desde un conjunto de datos finito[8]. Una representación geométrica de dicho ajuste puede observarse en la **Fig. 2.(a)**. Además, los datos observados generalmente están corrompidos por fuentes ruido, por lo que para una x_k dada existe incertidumbre sobre el valor apropiado para t_k. En particular, y como caso ilustrativo, es posible ajustar los datos usando una regresión polinomial de la forma

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j, \qquad (1)$$

donde M es el orden del polinomio, y x^j denota x elevado a la potencia de j. Los coeficientes polinómicos $w_0, ..., w_M$ se denotan colectivamente por el vector **w**. Aunque la función polinómica $y(x, \mathbf{w})$ es una función no lineal de x, es una función lineal de los coeficientes **w**. Los valores de los coeficientes se determinarán ajustando el polinomio a los datos de entrenamiento. Esto puede hacerse minimizando una función de error que mida el desajuste entre la función $y(x, \mathbf{w})$, para cualquier valor dado de **w**, y los puntos de datos del conjunto de entrenamiento. Una elección simple de la función de error, que se usa ampliamente, viene dada por la suma de los cuadrados de los errores entre las predicciones $y(x_n, \mathbf{w})$ para cada punto de datos x_n y los valores objetivo correspondientes t_n, de modo que se minimiza

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2, \qquad (2)$$

donde se incluye el factor de 1/2 para su posterior conveniencia. La función de error *E(**w**)* es una cantidad no negativa que sería cero si, y solo si, la función $y(x, \mathbf{w})$ pasará exactamente a través de cada punto de datos de entrenamiento. La **Fig. 2.(b)** describe la interpretación geométrica del procesos de minimización del error cuadrático medio, que es como comúnmente se le conoce a la ecuación (2).

3.4 Arquitecturas de microservicios

Antes de empezar con la etapa del desarrollo del proyecto desde el punto de vista arquitectónico es necesario abordar los conceptos descritos en las siguientes subsecciones.

3.4.1 WSGI and ASGI

WSGI[7] (*Web Service Gateway Interface)* es una especificación que describe la comunicación entre un servidor web y aplicaciones desarrolladas en *Python*. Su principal objetivo es estructurar las peticiones que llegan al servidor web y entregarla a una aplicación externa que responda a las solicitudes. Por su parte un servidor ASGI[8] (*Asynchronous Server Gateway Interfac*e), que es considerado el sucesor de WSGI, cumple la misma función de comunicación entre servidores web, *frameworks* y aplicaciones web, pero agregando características que permiten ejecutar funciones asíncronas.

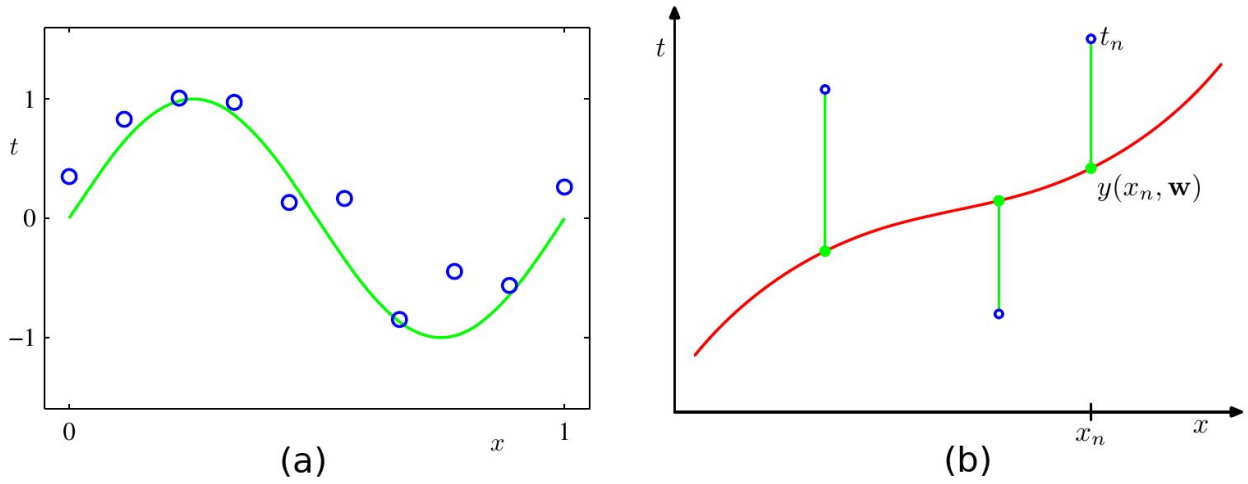

Figura 2. Diagrama de construcción de la visualización de datos: (a) Gráfico de un conjunto de datos de entrenamiento de $N = 10$ puntos, que se muestran como círculos azules, cada uno de los cuales comprende una observación del conjunto de entrada $\widehat{x}$ junto con la variable objetivo correspondiente $\widehat{t}$. La curva verde representa el ajuste polinomial a calcular. (b) La función de error de la ecuación (2) corresponde a semisuma de los cuadrados de los desplazamientos (mostrados por las barras verticales verdes) de cada punto de datos desde la función $y(x, \mathbf{w})$. Fuente: [8].

[7] WSGI: https://wsgi.readthedocs.io
[8] ASGI: https://asgi.readthedocs.io/en/latest/

3.3.2 Microservicios

La idea principal de este enfoque es desarrollar aplicaciones mediante la composición de pequeños servicios, cada uno, realizando una mínima acción o responsabilidad, permitiendo así la interacción entre ellos mediante algún protocolo de comunicación como HTTP (*HyperText Transfer Protocol*) o GRCP (*Remote Procedure Calls*) [9]. Esta composición de responsabilidades convierte un conjunto de microservicios en una funcionalidad comercial de la aplicación. No existe actualmente una forma concreta de escribir aplicaciones utilizando este diseño, como lo explica Martin Flower en su blog personal: "*existen ciertas características comunes en torno a la organización, en torno a la capacidad empresarial, la implementación automatizada, la inteligencia en los puntos finales y el control descentralizado de idiomas y datos"*[9]. Algunas características que deben cumplir estos sistemas se refieren a unidades de software reemplazables e independientes orientadas a las necesidades comerciales requeridas por el producto. La vida útil de un microservicio debería basarse en un flujo simple de operaciones: recibir un llamado, efectuar una lógica y responder a la solicitud, donde finalmente un sistema externo se encarga de la respuesta y procede a realizar otros llamados. Este esquema permite crear componentes más fáciles de probar y escalar. Normalmente los productos se desglosan en cantidades enormes de requisitos donde la característica "atómica" de los microservicios beneficia positivamente las necesidades de cada componente dentro del proceso de desarrollo [10]. Es decir, ahora se tienen sistemas completamente políglotas y descentralizados tanto en la persistencia de datos como en los lenguajes de programación utilizados por cada componente.

Para lograr construir efectivamente un microservicio es tendencia usar tecnologías como *Docker* que utilizan contenedores *Linux* que aíslan la implementación del código respecto a la infraestructura y fases de configuración del proyecto. Sin embargo, controlar cada componente resulta ser una tarea complicada en términos de comunicación y despliegue; es por ello que en ambientes de producción es común utilizar sistemas de más alto nivel como *Kubernetes* que

[9] Microservices: https://martinfowler.com/articles/microservices.html

orquestan los contenedores para que corran en armonía preservando y extendiendo las ventajas de estos como lo es un factor de réplica y resiliencia a fallos.

4. Metodología

Para el cumplimeinto de los objetivos especificos del proyecto, los pasos metodológicos planteados se describen en las siguientes subsecciones en orden cronológico de ejecución.

4.1 Adquisición de datos

Después de una extensa búsqueda de conjuntos de datos relacionados con formaciones tácticas de equipos de fútbol, el mejor candidato de conjunto de datos de carácter público que se encontró fue la base de datos del fútbol Europeo (KES: *Kaggle European Soccer Database*)[10]. Esta conjunto de datos contiene información de aproximadamente 26k partidos, 11k jugadores y 300 equipos de fútbol profesional europeo, pertenecientes al torneo principal de 11 países, desde la temporada 2008 hasta la temporada 2016. La base de datos recopila información de tres fuentes externas y las unifica en una base de datos relacional SQL. La base de datos inicial está conformada por las siguientes tablas:

- Países: 11 países Europeos y sus respectivos IDs
- Ligas: 11 ligas de fútbol profesional, su respectivo país y su ID
- Encuentros: 26.0k partidos con 115 columnas que caracterizan cada uno de ellos, con información relevante acerca de la formación y los jugadores del equipo local y visitante, así como el número de goles, tiros de esquina, tarjetas amarillas y rojas, faltas etc.

[10] *Kaggle - European Soccer Database*: https://www.kaggle.com/hugomathien/soccer

- Jugadores: 11.1k jugadores con información esencial de su contextura física y fecha de nacimiento.
- Atributos de jugadores: 42 atributos estimados para cada jugador en diferentes puntos temporales dentro de las temporadas 2008 a 2016, lo que conlleva a un total de 184k registros.
- Equipos: 299 equipos con sus respectivos identificadores, información de su nombre completo y su nombre abreviado.
- Atributos de equipos: 25 atributos estimados para cada equipo en diferentes puntos temporales dentro de las temporadas 2008 a 2016, lo que conlleva a un total de 1458 registros.

4.2 Análisis exploratorio de los datos

Como punto de partida se fijó la tabla "Encuentros" como conjunto de datos principal, dado que el principal interés fue predecir la formación táctica con la cual un equipo de fútbol pueda obtener una mayor probabilidad de victoria contra un rival dado. n este conjunto de datos es precisamente donde se encuentran las coordenadas de las formaciones tácticas para el equipo local y visitante en cada encuentro registrado. De esta manera, las demás tablas se fijaron como información suplementaria a integrar dentro del conjunto de datos de encuentros.

Al analizar los atributos por encuentro para los 300 equipos que conforman el *dataset,* se obtuvieron las áreas de partidos ganados, perdidos y empatados para los equipos que jugaron de local, así como el área general de partidos jugados, que se visualizan en la **Fig. 1**. Se observó que cerca de la mitad de equipos están por debajo del promedio de encuentros, i.e., que mientras en ligas como la Española o la Inglesa cada equipo tiene una alta cantidad de encuentros en cada temporada, en ligas menos populares como la Suiza o Belga la cantidad de partidos que juegan estos equipos es relativamente bastante inferior.

Se procedió entonces a cuantificar la cantidad de encuentros registrados por liga como primer criterio de selección de datos. Los resultados de este análisis se presentan en la **Fig. 2**. Como resultado de dicho conteo de partidos, se evidenció que las seis ligas más representativas del KES y representadas por el color salmón en la **Fig.1** son: *France League 1, England Premier League, Spain LIGA BBVA, Italy Serie A, Germany 1. Bundesliga* y *Netherlands Eredivisie.* Por consiguiente, se decidió inicialmente enfocar el desarrollo de la plataforma predictiva en estas seis ligas, que coinciden con el conocimiento popular como las ligas de mayor interés para los aficionados.

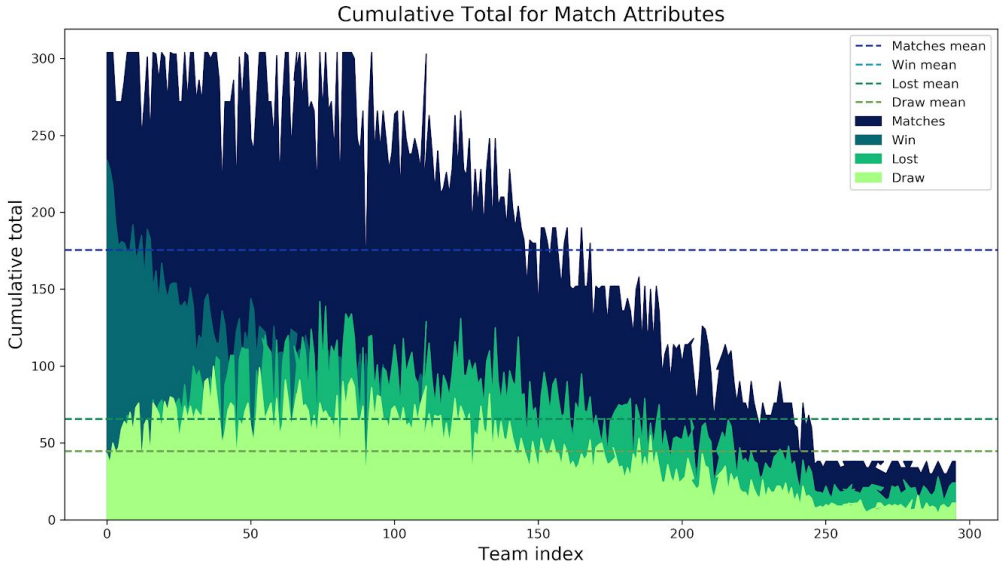

Figura 3. Atributos por partido para cada equipo registrado en el conjunto de datos de encuentros.

Una vez definidas las principales ligas por el número de encuentros disputados, se procedió a analizar los conjuntos de datos suplementarios. El principal conjunto fue el de las habilidades por jugador. La **Fig. 3** muestra las representaciones polares o comúnmente conocidas como "de radar", para tres jugadores bastante representativos dentro del conjunto de datos: Sergio Ramos, Andrés Iniesta y Cristiano Ronaldo. Por jugador, el *dataset* proporciona la estimación cuantitativa de 38 habilidades a lo largo de diferentes puntos temporales entre las temporadas

2008 a 2016, extraídas directamente de la base de datos del videojuego FIFA. Al mismo tiempo, dado el alto número de datos faltantes en este *dataset* para la *Netherlands Eredivisie,* se decidió retirarla del conjunto de datos de encuentros. De esta manera se obtuvo conjuntos de datos de encuentros y de habilidades de jugadores mucho "más densos", evitando así estrategias de imputación simples que pudieran sesgar los resultados.

La información de habilidades por jugador permitió enriquecer los datos correspondientes a las formaciones tácticas, ya que como se observa en la **Fig. 4** el sistema de referencia y las coordenadas de los jugadores, tienen una representación bastante simple que no describe en detalle las formaciones tácticas y los aspectos técnicos del equipo.

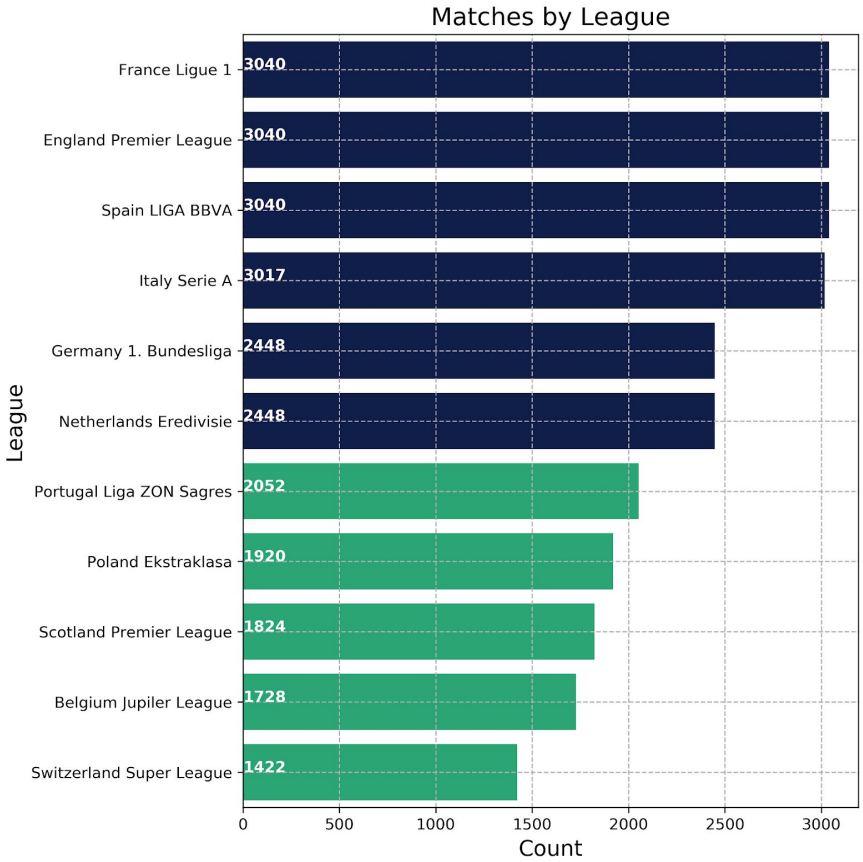

Figura 4. Conteo de partidos por liga.

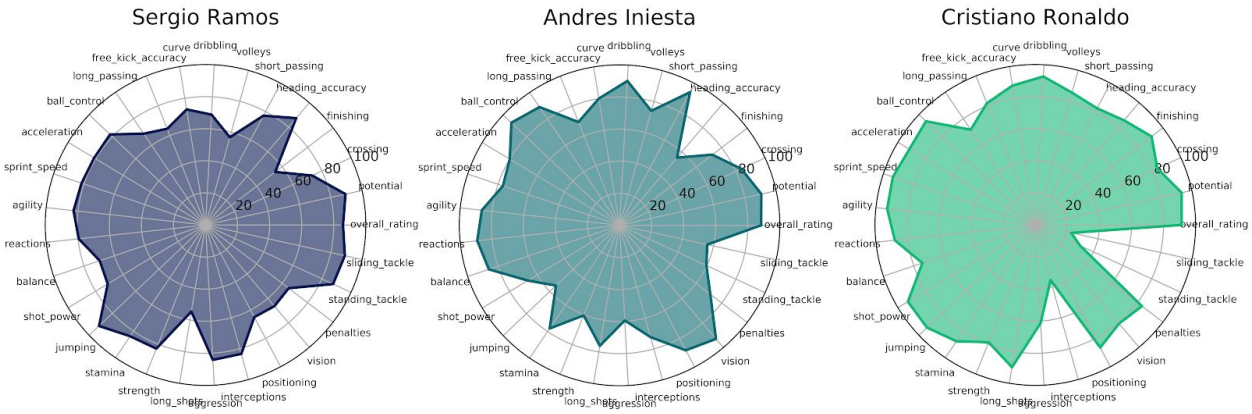

Figura 5. Representación en forma de radar de los atributos de tres jugadores populares de diferentes épocas: Sergio Ramos, Andrés Iniesta y Cristiano Ronaldo.

Al igual que los jugadores, los equipos también cuentan con una caracterización general de rendimiento en diferentes puntos temporales entre la ventana de tiempo del KES. La **Fig. 4** muestra la comparación entre dos equipos de la liga Inglesa.

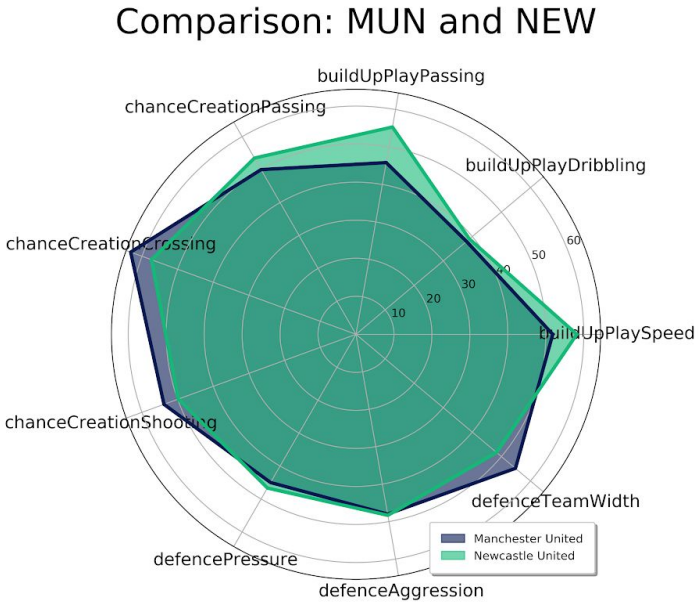

Figura 6. Comparación entre los equipos *Manchester United* (MUN) y *Newcastle* (NEW).

A la hora de analizar los datos de un determinado encuentro, la inclusión de los atributos de cada jugador permite complementar la información del grafo acíclico no dirigido[11] que se forma a partir de las coordenadas de los jugadores. De esta manera se definió la información con la que los estimadores realizan la predicción de las productividad de una determinada formación táctica.

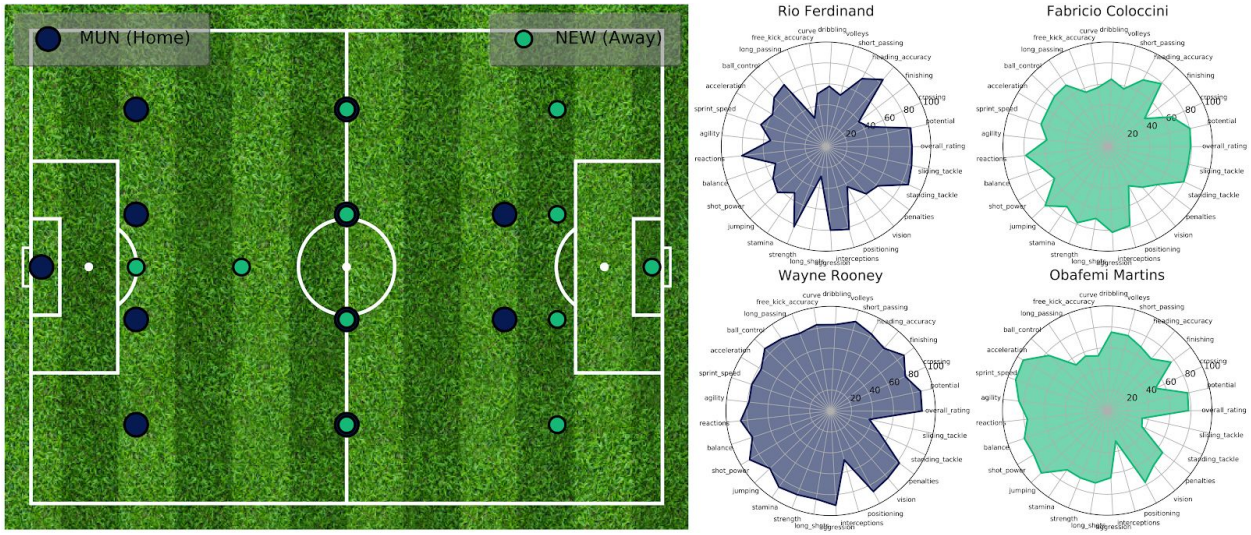

Figura 7. Gráfico de las formaciones tácticas para el encuentro entre el *Manchester United* (MUN) y el *Newcastle United* (NEW), por la *England Premier League* en la temporada 2008, junto a la información complementaria de los atributos por jugador.

4.3 Esquema unificado de datos y base de datos no relacional con MongoDB

Debido a la firme intención de enriquecer los datos de las formaciones tácticas para cada partido a través de la inclusión de las habilidades de cada jugador (principal conclusión del análisis estadístico de la base de datos del fútbol Europeo KES), se propuso entonces un esquema de datos unificado representado en la **Fig. A.1** del anexo **A**.

[11] Wikipedia - Graph: https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)

Para el almacenamiento de los datos se usó una base de datos no relacional, de manera que los datos sean mutables con el tiempo y permitiendo así la incorporación de actualizaciones de habilidades de jugadores y de equipos. Particularmente las entidades del conjunto de datos comprende una jerarquía y composicionalidad robusta, en donde un jugador posee un conjunto de habilidades, pero a su vez éste pertenece a un equipo con información propia de rendimiento y conformado por un total de once jugadores. El conjunto de datos principal registra adicionalmente la información por encuentro, teniendo así toda la información anterior para dos equipos. Esta complejidad es precisamente la que se ve resuelta con la implementación de bases de datos no relacionales, como es el caso de *MongoDB* y *Apache Cassandra*.

Para este proyecto se utilizó *MongoDB* por su excelente integración con el *framework FastAPI*. Esta comunicación entre *Python* y *MongoDB* se realizó a través de *mongoengine,* un DOM (*Document-Object Mapper*) que cumple la misma función de un ORM (*Object-Relational Mapper*) pero para bases de datos de documentos, como también se le conoce a las no relacionales.

4.4 Modelamiento de la productividad como un problema de regresión

Dentro del ciclo de vida de la ciencia de datos[12], el modelamiento corresponde a la etapa en la que se entrenan los estimadores de los diferentes algoritmos de *machine learning* a utilizar. Para esto, es necesario haber definido las características (*features*) y la variable objetivo (*target*), como se expuso en la sección 3.3.2. Por consiguiente, en las siguientes subsecciones se detalla el procedimiento conocido como ingeniería de características o *feature engineering*[13] que transforma los datos para hacerlos más "representativos" ante el problema de predicción, la definición de la productividad acorde a lo presentado en la subsección 3.3.1 como variable objetivo o *target* y detalles generales de la implementación de los estimadores.

[12] Microsoft Azure - What is the Team Data Science Process?: https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/overview
[13] Wikipedia - Feature engineering: https://en.wikipedia.org/wiki/Feature_engineering

4.4.1 Ingeniería de características y definición de la productividad

En términos generales[14], los pasos metodológicos para la transformación de características del *dataset* y la definición de la productividad como variable objetivo, se listan a continuación:

1. Incorporación al conjunto de datos de encuentros la información pivoteada de los atributos de los jugadores y los atributos de los equipos.
2. División del campo de juego en cuatro subáreas: a) a defender, b) mediocampo a defender, c) mediocampo a atacar y d) a atacar. Esto define los roles de los jugadores en la formación técnica y pasan a ser nuevas características dentro del *dataset*, como se muestra en la **Fig. 8**.
3. Cálculo de los centros geométricos para cada subárea del campo de juego.
4. Definición de los espacios de juego efectivos ofensivos y defensivos (OEP-S : *Offensive Effective Play-Space* y DEP-S: *Defensive Effective Play-Space*) para cada equipo en cada subárea.
5. Cálculo del área geométrica y del centroide de los polígonos irregulares que conforman los DEP-Ss y los OEP-Ss.
6. Codificación de los planteamientos de ataque, defensa y equilibrado, a partir de la posición relativa entre el centro geométrico de las subáreas del campo y los centroides de cada espacio de juego.
7. Codificación del balance de fuerzas entre los dos equipos a través de la intersección de las áreas geométricas de los DEP-Ss y los OEP-Ss y la "ponderación de habilidades individuales" planteada a partir de la información de los jugadores que conforman cada espacio de juego.
8. Cómputo de la productividad en ataque y defensa a través de las estadísticas del partido como goles marcados, tiros de esquina a favor, remates, remates de media distancia, tiros libres y goles recibidos, tiros de esquina en contra, remates recibidos, tiros libres

[14] Por temas de confidencialidad y un actual proceso de publicación en curso, la empresa guane Enterprises S.A.S permitió unicamente una descripción general de estos procedimiento.

recibidos y demás, respectivamente. Ambas métricas de productividad se ponderan por la posesión del balón y por la condición de local o visitante.

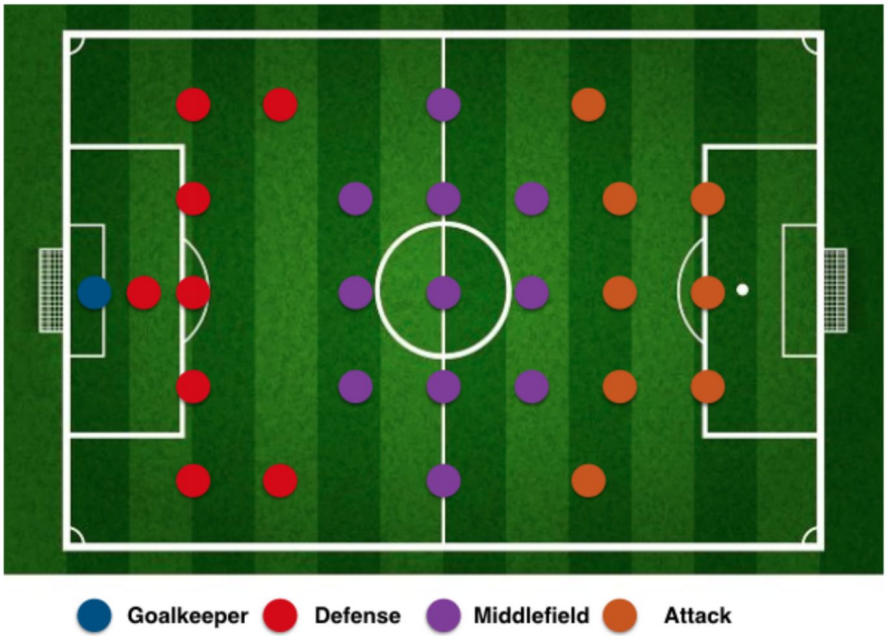

Figura 8. Descripción de los roles defensa, mediocampista y atacante presentes en el KES. Fuente: [11]

4.4.2 Entrenamiento de regresores de *machine learning*

Con miras a implementar una plataforma web capaz de entrenar varios algoritmos predictivos a "petición" del usuario, se optó por reducir el espacio de posibles algoritmos a los enmarcados entre el conjunto de técnicas clásicas de *machine learning*, excluyendo así todo el conjunto de redes neuronales artificiales (ANN: *Artificial Neural Networks*) pertenecientes al *deep learning*. Entre las principales razones de exclusión de las ANNs están: *i)* su bajo rendimiento predictivo en datos tabulares y en conjuntos de pocas muestras, *ii)* altas demandas computacionales para el

entrenamiento, pues requieren en ocasiones de GPU (*Graphics Processing Unit*) y *iii)* tiempos de entrenamiento grandes que no favorecen las experiencia de usuario en una plataforma predictiva.

Se utilizaron dos enfoques para el entrenamiento y generación de los modelos: global y "por rival". Para el global, el *dataset* de entrenamiento está conformado por el 80% de los registros del conjunto de datos de encuentros que fue filtrado y seleccionado anteriormente para contener únicamente encuentros de las cinco ligas más representativas de Europa. Al ser global, el modelo intenta generalizar el comportamiento para cualquier confrontación de equipos, a partir de las coincidencias en las observaciones de este conjunto de datos global. Por su parte, el enfoque "por rival" define un *dataset* mucho más pequeño pero particularmente centrado en el equipo local y en su rival (equipo visitante). Dicho *dataset* se compone de la información de los últimos cinco partidos jugados en la liga contra cualquier rival, tanto para el equipo de referencia como para su respectivo rival y todos los registros de encuentros previos en los que el equipo de referencia se enfrentó a su rival en la misma condición de localidad (local o visitante) a predecir.

Para el modelamiento se implementaron los siguientes algoritmos: *Support Vector Regressor* (SVR) *with Linear Kernel, Decision Tree Regressor* (DTR)*, Random Forest* (RF) *y Gradient Boosting Regressor* (XGBR). Los tres primeros fueron implementados en el *framework* H2O 3[15] y el último en la biblioteca *XGBoost*[16] (*eXtreme Gradient Boosting*). La elección de estos *frameworks* se basó principalmente en los criterios de robustez (distribuidos y paralelizados), rendimiento y rapidez. En cuanto a la búsqueda de hiperparametros se utilizó la estrategia *Grid Search*, bajo el criterio de *cross-validation*, siendo este de los más simples pero con tiempos de ejecución bastante bajos (característica deseada para la plataforma predictiva). Finalmente, los modelos fueron exportados en formato MOJO para las implementaciones hechas en H2O 3 y en formato pickle para el *XGBoost*.

[15] H2O 3: https://www.h2o.ai/products/h2o/#features
[16] XGBoost: https://xgboost.readthedocs.io/en/latest/index.html

4.5 Creación de microservicios bajo la arquitectura API RESTu

La idea principal de una arquitectura basada en *API REST* (*Representational State Transfer*) es que por medio de un conjunto de reglas y especificaciones las aplicaciones puedan comunicarse y transferir información entre sí, usando el protocolo HTTP (*Hypertext Transfer Protocol*). HTTP permite crear operaciones sencillas como *POST*, *GET*, *PUT* o *DELETE* que transforman los datos de la manera que más convenga dentro del flujo de la aplicación. Este concepto central, al juntarse con la idea minimalista de microservicios, permite tener aplicaciones más pequeñas, fácilmente mantenibles en el tiempo, desacoplando la arquitectura del proyecto y facilitando de igual forma que se cumpla el principio de tener servicios que cumplan funciones muy específicas dentro de la aplicación, comunicados entre sí por medio de HTTP.

4.5.1 *Pydantic* como validador de esquemas

Pydantic[17] es una biblioteca de *Python* que tiene por objetivo ayudar en el diseño de estructuras de datos complejas que puedan ser validadas según sea el tipo de atributo. Funciona mediante el marco OOP (*Object Oriented Programming*) declarando la estructura como una clase y asignando los tipos de datos utilizando *Typing* a sus atributos. Los usos de *Pydantic* pueden extenderse desde usarlo solo para validar las estructuras de datos complejos o utilizar instancias de las clases para determinar el flujo de la aplicación. Los tipos de datos en *Python* son interpretados en tiempo de ejecución debido a la naturaleza débilmente tipada del lenguaje. Esto hace que sea complicado determinar en muchos casos los retornos o los parámetros de las funciones incurriendo así en la inyección de posibles errores en el código. Allí es donde *Pydantic* ofrece una experiencia de desarrollo mucho más fluida al tener la posibilidad de verificar los tipos de datos justamente en la construcción de las instancias de las clases, ayudando a establecer un esquema de datos sólido para todas la estructuras utilizadas en el proyecto.

[17] : *Pydantic*: https://pydantic-docs.helpmanual.io

4.5.2 *FastAPI*

Es un marco web moderno construido para crear API's usando *Python* 3.6+, dado que es un *framework* rápido y optimizado dado que internamente *Uvicorn* para cargar y servir la aplicación, además tiene otras características que están dentro de las necesidades del proyecto pues es intuitivo, fácil de codificar, con una curva de aprendizaje baja y sobre todo robusto dado que es necesario esa estabilidad a la hora de tener códigos listos para producción. Asimismo, otra de las características que llama la atención del uso de este *framework*, es la implementación interna de *OpenAPI* que facilita el acceso automático a documentación intuitiva para que tanto desarrollador como usuario pueda acceder a los servicios creados de manera interactiva.

4.6 Empaquetando microservicios en contenedores de *Docker*

Luego de realizar la definición e implementación de cada microservicio se estableció cada artefacto como una imagen de *Docker* definida como un servicio web creado en *FastAPI*. Los manifiestos se describieron utilizando la sintaxis imperativa para la configuración del contenedor llamada *dockerfile*. Cada una de las definiciones contenían todo lo necesario para ejecutar las bibliotecas y establecer los complementos necesario para cada uno de los microservicios.

En el fragmento de código mostrado en la **Fig. 9,** la **línea 2** indica la imagen base del nuevo contenedor, lejanamente podemos pensar en la imagen como un "sistema operativo base*"*. Se eligió *NodeJs* en su versión 11 corriendo sobre *Alpine Linux* (distribucion de linux famosa por ser ligera y segura), la fase de desarrollo del documento termina en la **línea 5** con la instalación de las dependencias utilizadas por el proyecto con el comando `npm install`. Luego, sigue la etapa de construcción con la **línea 10**, se distingue de la fase anterior debido a que en este paso el proyecto se compila o construye y todas las dependencias deberían estar listas desde la fase de desarrollo. Al ser un contenedor que proporciona las vistas de la plataforma se requiere de un

servidor de archivos estáticos para completar la fase de producción, para esto, la pila de tecnologías principales de *guane Enterprises* recomienda el uso de *Nginx*. La **línea 13** ocupa una nueva imagen basada en una versión liviana de *Nginx.* Además, se copian los archivos generados de la construcción de la etapa anterior. Finalmente, la **línea 22** ejecuta el proceso de *Nginx* que escucha sobre el puerto 80 y desde allí sirve los archivos. Cabe resaltar que todos los contenedores, en general, tienen diferentes configuraciones de estos manifiestos condicionadas por las necesidades de los microservicios, es decir, si alguno requiere *Python*, *Golang* u otro lenguaje se deben tener las consideraciones respectivas para estos entornos de desarrollo. Por ejemplo, si se usa una base de *Python* es bastante probable que el manejador de dependencias sea *Pip* o en el caso de *Java Maven*.

```
1 # Fase de desarrollo
2 FROM node:11.1-alpine as develop-stage
3 WORKDIR /app
4 COPY package*.json ./
5 RUN npm install
6 COPY . .
7
8 # Fase de construcción/compilación
9 FROM develop-stage as build-stage
10 RUN npm run build
11
12 # Fase de producción
13 FROM nginx:1.15.7-alpine as production-stage
14 COPY --from=build-stage /app/dist /var/www/html
15 RUN mkdir -p /var/log/nginx
16 RUN mkdir -p /var/www/html
17 RUN chown nginx:nginx /var/www/html
18 # Fase de configuración específica
19 COPY nginx_config/nginx.conf /etc/nginx/nginx.conf
20 COPY nginx_config/default.conf /etc/nginx/conf.d/default.conf
21 EXPOSE 80
22 CMD ["nginx", "-g", "daemon off;"]
```

Figura 9. *dockerfile* del microservicio *WebApp*.

La **Fig. 9** muestra un ejemplo de manifiesto para el microservicio que construye y despliega la aplicación web desarrollada en *VueJs*. En particular se utilizó el modo *multi-stage* de *Docker que*

habilita al desarrollador la opción de utilizar pasos intermedios para la construcción del contenedor con la finalidad de crear imágenes limpias de un tamaño mínimo.

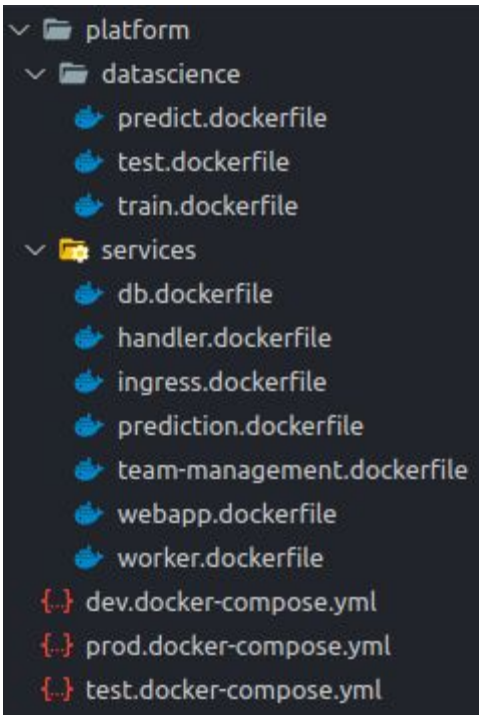

Figura 10. Manifiesto de los contenedores.

En la **Fig. 10** se listan las descripciones de los contenedores en los archivos con extensión *.dockerfile* . Específicamente se hace una distinción entre los contenedores orientados a las tareas de ciencia de datos y los otros microservicios que sirven las operaciones de la plataforma. De esa forma, las acciones y comunicación entre ambas fases del proyecto se comunican mediante microservicios intermedios que funcionan como *middleware* y operadores del flujo de control de la plataforma.

Si bien `docker-compose` permite gestionar más limpiamente la construcción, ejecución y comunicación entre contenedores resulta ser una opción que se queda corta en entornos de producción. Por un lado, no tiene una gestión de réplicas para cada contenedor, es decir, si un contenedor presenta errores no hay una política de resiliencia y tolerancia a fallas que permita

restaurar los microservicios de la plataforma. *Docker-compose* no es una solución escalable debido a sus falencias en la gestión de servicios, monitoreo y comunicación entre contenedores, es por esto que se buscó alternativas más robustas como *Kubernetes*.

4.7 Orquestación de contenedores con *Kubernetes*

Con el equipo de desarrollo e infraestructura se evaluó la viabilidad de implementar un cluster de *Kubernetes* dando como conclusión que por su factor de réplicas, monitoreo, montaje y la posibilidad de hacer despliegues canarios era precisamente la tecnología que el proyecto requería.

Primero se definieron los Pods que serán desplegados. Un pod es un objeto mínimo dentro de *Kubernetes* que alberga, normalmente, una relación uno a uno con los contenedores; sin embargo, pueden encontrarse casos donde un Pod esté compuesto de varios servicios. Podemos subir la abstracción de un Pod mediante *deployments* donde se puede configurar un factor de réplica que *Kubernetes* fielmente va a preservar. En otras palabras, si definimos un factor de réplica 3 para el *frontend* entonces el controlador del orquestador intentara tener vivas 3 instancias de los contenedores sobre los nodos del cluster.

También con *Kubernetes* se usaron procesos asíncronos programados que ejecutaban flujos de trabajo para las actividades computacionalmente más costosas y las que tomaban más tiempo procedimentalmente como el entrenamiento de múltiples modelos y el procesamiento de datos. Por ejemplo, un *CronJob* era activado cada cierto tiempo verificando información nueva que pudiera servir de ingesta para los modelos desarrollados por el equipo de ciencia de datos. También, varios de estos procedimientos programados desarrollaban tareas de *ETL* (*extract-transfer-load*) de API's de datos públicos o practicando algún tipo de *web scraping* en sitios de información considerada relevante para el proyecto.

Frente a todos los microservicios se utilizaron dos abstracciones adicionales de *Kubernetes*: *Load balancer* y *Ingress server*. Normalmente la gestión de red(*networking*) en sistemas donde intervienen tantos pequeños servicios se hace complicada a medida que crecen las réplicas y se agregan más funcionalidades que impliquen robustecer el diseño de la arquitectura. Estas dificultades se mitigaron balanceando el tráfico con servicios que los proveedores *Cloud* tienen muy bien establecidos dentro de su catálogo de productos. Es así como *Kubernetes* administra balanceadores de carga llamando directamente las API del proveedor *Cloud* donde se esté corriendo el Cluster. En otras palabras, el tráfico de red que ingresa al cluster se reparte en una entidad de *software* y *hardware* que tiene la capacidad de repartir y garantizar que nuestros recursos van a ser utilizados de una manera eficiente. Además de esto, el proyecto necesitaba una manera de distinguir los servicios usados por los equipos de ciencia de datos y desarrollo debido a los múltiples artefactos que intervienen en la fase de producción y esto fue realizado agregando un servidor de ingreso(i*ngress*) justamente luego del balanceador de carga que mediante reglas basadas en subdominios se enviaba dicho tráfico ya gestionado por el proveedor a los respectivos servicios que se le fueran asignados.

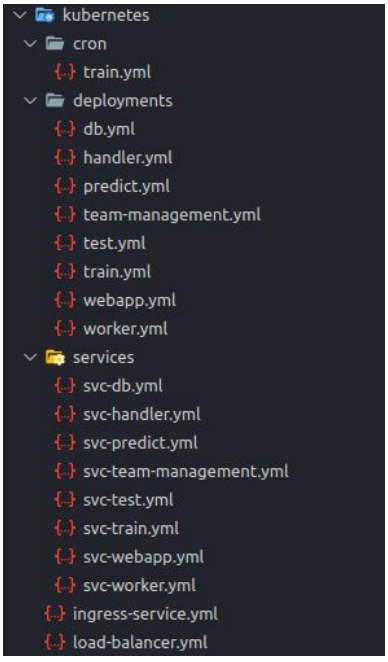

Figura 11. Manifiesto de *Kubernetes*.

En la fase final del proyecto se exploraron elementos adicionales que extienden mucho más las posibilidades de *Kubernetes* y queda propuesto para la siguiente fase del proyecto hacer una exploración de herramientas como *helm*, *rancher* e *Istio*. Por otro lado, *KubeFlow* es un proyecto de código abierto que ya ofrece operadores que facilitan los entornos de ciencia de datos simplificando los procesos los entrenamientos y despliegue de modelos utilizando los frameworks más modernos para desarrollar algoritmos de aprendizaje de máquina y profundo.
En la **Fig. 11** puede observarse los distintos manifiestos utilizados para el proyecto.

Kubernetes ofreció al proyecto una versatilidad asombrosa en términos del despliegue resolviendo todas las falencias que tenía el proceso con *docker-compose* agregando una complejidad necesaria a la cual se le pueden agregar fases del desarrollo de software más profundas para garantizar calidad y mantenibilidad del producto como lo es las pruebas de integración, el despliegue continuo y la integración continua.

4.8 Integración continua y despliegue continuo

Para cada microservicio se utilizó la semántica de versionamiento *semver 2.0*[18] con entregas programadas cada dos semanas;. sin embargo, también se subieron a producción correcciones de *bugs* entre *releases* que corrigieron incidencias inyectadas en el código. Es por ello que se encontró la necesidad de utilizar sistemas para desplegar e integrar el proyecto continuamente eligiendo a *jenkins* para este propósito. El *workflow* implementado fue:

1. La rama *master* del repositorio de los microservicios fue registrada en un *trigger* de eventos de *jenkins.*

[18] *Semver*: https://semver.org/

2. *Jenkins* monitorea la rama *master* y al detectar un cambio o un commit nuevo procede a ejecutar las tareas asociadas a recetas de procedimientos previamente descritas para cada microservicio.
3. *Jenkins* clona el proyecto en un espacio de almacenamiento específico y ejecuta el comando `docker build` para construir las imágenes de los microservicios.
4. Sobre las imágenes se efectúan las pruebas unitarias, pruebas de integración y se ofusca el API del microservicio para detectar fallos cuando el tráfico crece. Si el reporte de pruebas indica buenos resultados *Jenkins* pasa al siguiente paso, si no, detiene el proceso e informa donde fallo.
5. Al pasar las pruebas las imágenes se envían a un *registry* privado que la almacena y a su vez genera una señal a *Kubernetes* para indicarle que debe ingresar en el cluster un nuevo *deployment* con la nueva versión del microservicio.
6. Finalmente, *Kubernetes* realiza un despliegue canario del microservicio y lo distribuye en el Cluster.

5. Resultados y Análisis

En este apartado, serán presentados los resultados obtenidos al final del proyecto, en las siguientes subsecciones se mostrarán desde el enfoque arquitectónico de *software*, interfaces gráficas de la *WebAPP* y ciencia de datos respectivamente:

5.1 Diseño de arquitectura de *software*

El resultado más contundente de la práctica profesional fue lograr llevar a un entorno de producción la plataforma cognitiva, en la cual, los equipos de ciencia de datos y desarrollo encontraron una gran utilidad en términos de automatización de procesos y estabilidad. Los objetivos profesionales se culminaron con buenas expectativas desarrollando todas las fases del

proceso de *software* incluyendo marcos modernos como las tareas de operaciones y el montaje de una aplicación compleja de alta demanda tanto computacional como a nivel de tráfico de red.

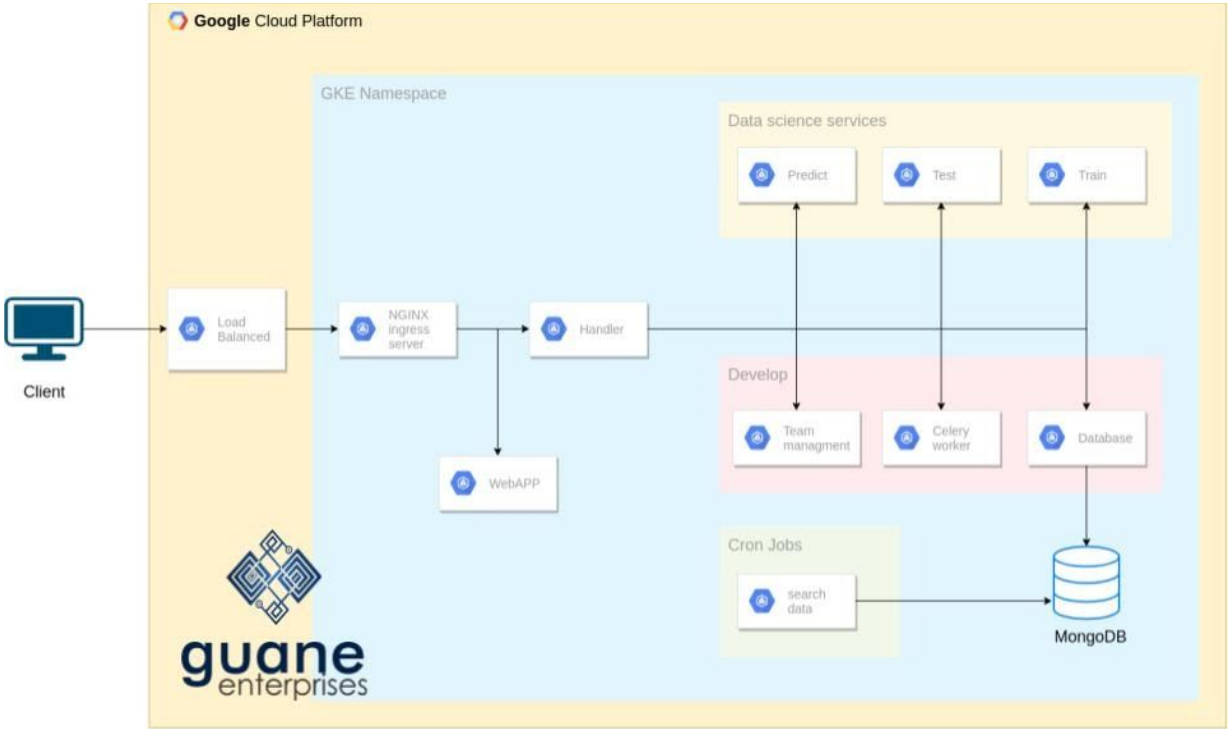

Figura 12. Arquitectura *Cloud* de la solución.

En la **Fig. 12** muestra la arquitectura final de la solución estableciendo como interactúan cada artefacto en el despliegue sobre *Google Cloud*. Es importante resaltar que la comunicación entre los clientes y los componentes internos del sistema ocurren mediante mensajes utilizando el protocolo HTTP. Sin embargo, toda esta complejidad queda a cargo de los microservicios, el servidor ingress y el balanceador de carga y solo se permite la interacción directa entre la *WebApp* y el API de la plataforma a través de sus respectivos subdominios.

5.2 Interfaz gráfica de la plataforma cognitiva

Otro de los resultados fue la integración de los microservicios a una *WebApp*, en la cual se visualiza la información de una manera más intuitiva para el usuario final. Entre las

funcionalidades que se pueden explorar en las interfaz gráfica está la carga de datos y la posibilidad de escoger el enfrentamiento entre equipos de una misma liga en determinada temporada. Al procesar los datos se muestra información relacionada con las habilidades de los equipos y de los jugadores en forma de gráficos de radar que en son polígonos que representan ciertas habilidades siendo el círculo el valor perfecto en todas ellas. También en la sección principal del enfrentamiento aparece el mapa de la cancha con la alineación inicial del partido. Además, en la barra lateral se extienden características como ver el histórico de consultas realizadas a la plataforma, estadísticas de uso y finalmente una sección de administración que en versiones futuras será implementada para el manejo de usuarios.

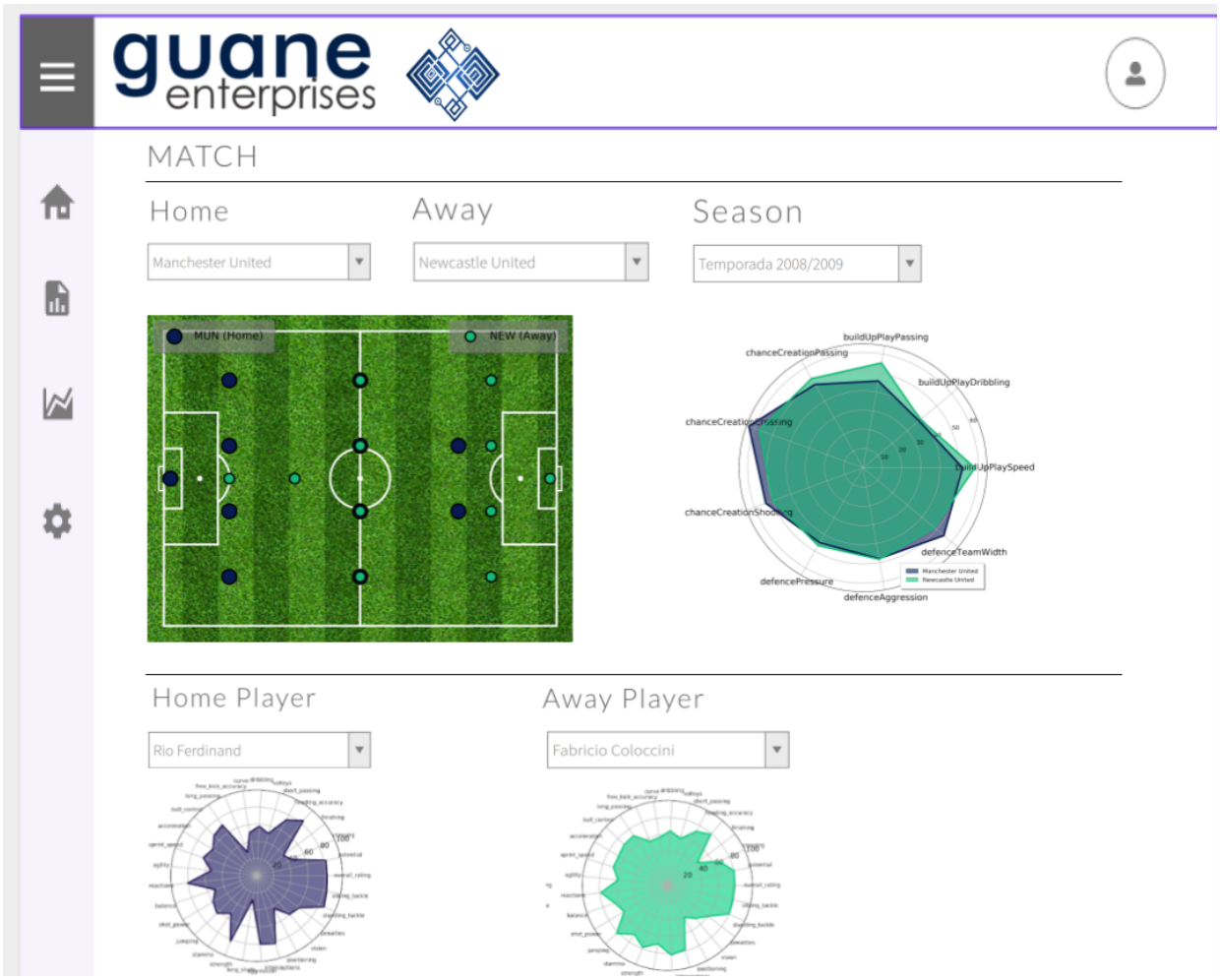

Figura 13. Interfaz gráfica del *WebAPP*.

5.2 Evaluación de modelos

Para el enfoque global de modelación, se entrenaron los siguientes regresores *Support Vector Regressor* (SVR) *with Linear Kernel, Decision Tree Regressor* (DTR)*, Random Forest* (RF) *y Gradient Boosting Regressor* (XGBR). Los resultados obtenidos para las métricas de rendimiento *Root-Mean-Square Error* (RMSE), *Mean Absolute Error* (MAE), *Pearson Correlation Coefficient* (PCC) y *Mean Absolute Percentage Error* (MAPE), fueron consignados en la **Tabla 1**.

Algoritmo	**RMSE**	**MAE**	**PCC**	**MAPE**
SVR	2.844	2.147	0.670	32.735 %
DTR	2.259	1.707	0.792	15.089 %
RF	1.113	0.945	0.974	8.599 %
XGBR	**0.562**	**0.398**	**0.989**	**3.640 %**

Tabla 1. Métricas de rendimiento para los regresores entrenados en el enfoque global.

El *XGBoost* fue el mejor regresor para la predicción de la productividad de la formación táctica inicial. Por su parte, para el enfoque "por rival"[19] el mejor de los regresores registro MAPEs entre 2.8 % y 7.5%.

6. Conclusiones

Los sistemas que usan tecnologías como *Docker* y *Kubernetes* orientadas a arquitecturas de microservicios, se adaptan muy bien a las necesidades de los productos que incluyan un componente basado en inteligencia artificial como el *machine learning*. Este tipo de arquitectura

[19] Por temas de confidencialidad y un actual proceso de publicación en curso, la empresa *guane Enterprises S.A.S* permitió unicamente una descripción general de estos resultados.

permite construir ambientes en los cuales se pueden realizar procesos de entrenamiento de algoritmos, despliegues de predictores, validaciones y pruebas, de una manera desacoplada y atómica, mejorando el rendimiento de estos procedimientos y a su vez la experiencia de usuario.

Esta atomicidad en los servicios, es uno de los múltiples estándares de código que resultan ser relevantes y casi obligatorios para el desarrollo de *software* de calidad. En proyectos donde intervienen diferentes desarrolladores, estos lineamientos basados en buenas prácticas de desarrollo facilita y mejora la construcción de los productos de manera colectiva. Particularmente, la estructura de los microservicios aquí diseñados, pasó a ser un *boilerplate* para otros proyectos de la empresa en la que realizo la practica empresarial. La adopción estándares de desarrollo como las arquitecturas basadas en microservicios y patrones estructurales bien definidos, permitieron una uniformización en el desarrollo y una mejor comunicación entre equipos.

Son precisamente los microservicios un diseño arquitectónico puramente resiliente que tiene la capacidad de resistir un volumen alto de peticiones siempre y cuando existan elementos externos que faciliten su comunicación a nivel de infraestructura. Por ejemplo, los factores de réplicas y servicios como balanceadores de carga son necesarios para que este tipo de aplicaciones logre soportar el tráfico, objetivo que si bien los microservicios no logran por sí solos combinados con estos componentes se convierten en una solución eficaz.

Referencias

[1] Lutz, M. (2001). *Programming python*. " O'Reilly Media, Inc.".

[2] Martin, R. C. (2009). *Clean code: a handbook of agile software craftsmanship*. Pearson Education.

[3] Anaya, M. (2018). Clean Code in Python: Refactor your legacy code base. Packt Publishing Ltd.

[4] Haslwanter, T. (2016). *An Introduction to Statistics with Python*. Springer International Publishing.

[5] Bobriakov, I. (2019, February 28). How to Choose the Right Chart Type [Infographic]. Retrieved April 23, 2020, from https://www.datasciencecentral.com/profiles/blogs/how-to-choose-the-right-chart-type-infographic

[6] Quinn, K. G. (Ed.). (2011). *The economics of the National Football League: The state of the art* (Vol. 2). Springer Science & Business Media.

[7] Espitia-Escuer, M., & Garcia-Cebrian, L. I. (2008). Measuring the productivity of Spanish first division soccer teams. *European Sport Management Quarterly*, *8*(3), 229-246.

[8] Bishop, C. M. (2006). Pattern recognition and machine learning. springer.

[9] Rodger, R. J. (2018). *The tao of microservices*. Manning Publications Company.

[10] Newman, S. (2019). Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O'Reilly Media.

[11] Pelechrinis, K., & Winston, W. (2018). Positional value in soccer: Expected league points added above replacement. *arXiv preprint arXiv:1807.07536*.

Anexos

A. Esquema de base de datos.

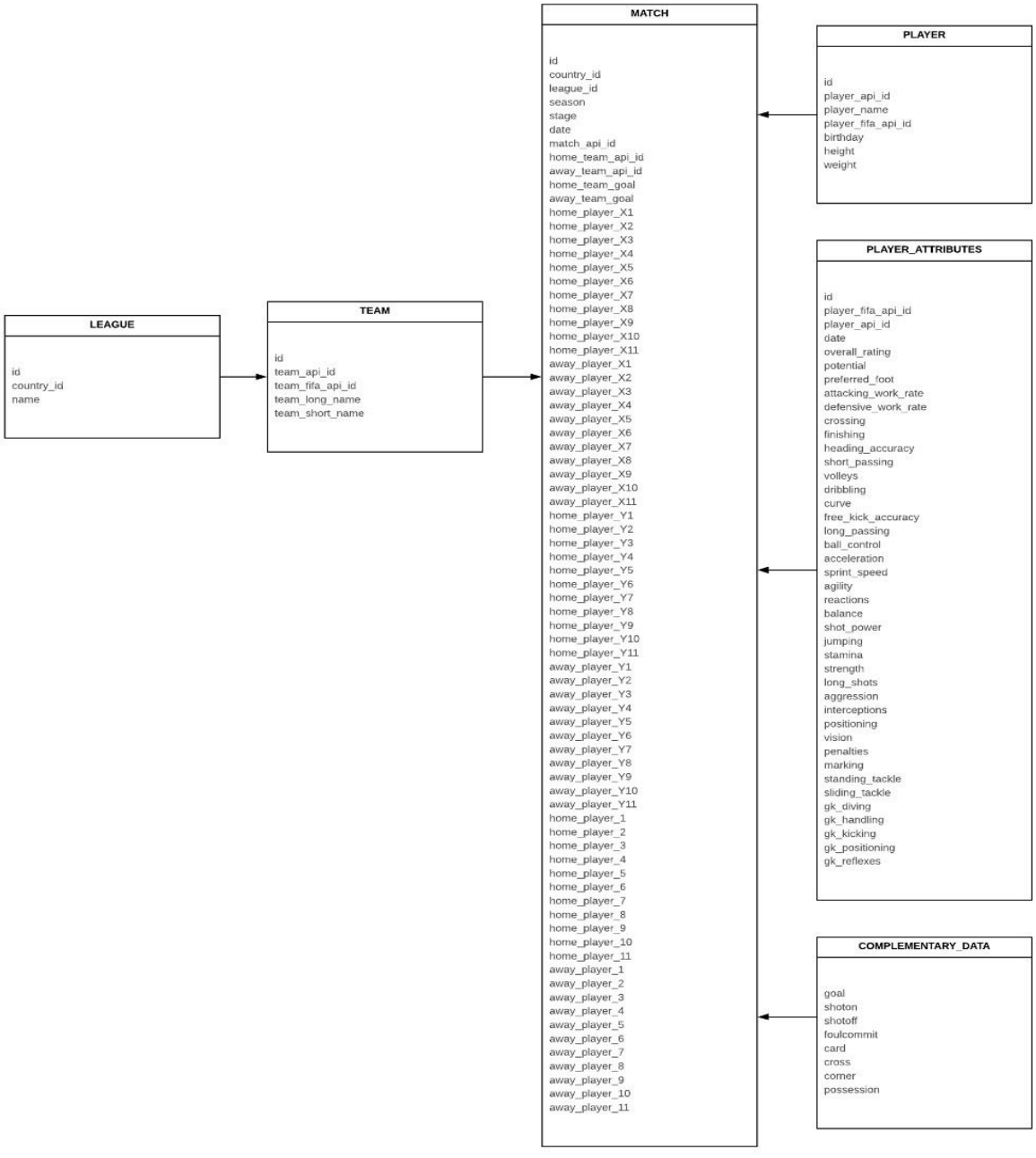

Figura A.1. Esquema de base de datos no relacional en MongoDB.