

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación



**DISEÑO Y DESARROLLO DE UN SERVICIO PARA CLASIFICAR EL
NIVEL DE ESTRÉS FÍSICO BASADO EN LA TECNOLOGÍA DE
MACHINE LEARNING 'KNOWLEDGE GRAPHS'.**

TRABAJO FIN DE MÁSTER

Jaime de Frutos Cerezo

2020

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

**Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos**

TRABAJO FIN DE MÁSTER

**DISEÑO Y DESARROLLO DE UN SERVICIO PARA
CLASIFICAR EL NIVEL DE ESTRÉS FÍSICO BASADO EN
LA TECNOLOGÍA DE MACHINE LEARNING
'KNOWLEDGE GRAPHS'.**

Autor
Jaime de Frutos Cerezo

Tutor
Joaquín Luciano Salvachúa Rodríguez

Departamento de Ingeniería de Sistemas Telemáticos

2020

Resumen

Los sistemas basados en inteligencia artificial han sido un tema recurrente y muy abordado en los últimos años. Es una de las tecnologías con más futuro, pero de la cual, no se está consiguiendo los resultados con los que se soñaba en un principio.

Actualmente, el Machine Learning está en la cúspide de las tecnologías IA. A su vez, uno de los recursos más conocidos son las redes neuronales, las cuales, como muchas otras soluciones de predicciones, obtienen buenos resultados. Sin embargo, no se sabe cómo se han obtenido. El proceso para calcular estos resultados lo podemos definir como una caja negra, no se sabe lo que hace por dentro, pero al introducir valores obtenemos una salida, más o menos válida.

Con el siguiente paso de la IA, se pretende evitar estas cajas negras. Los “Knowledge Graphs” son grafos que, gracias a nodos y relaciones, podremos trazar el origen de los resultados, además de obtener conocimiento nuevo.

En este TFM se aborda esta nueva tecnología aplicándola a un caso de uso en específico: La predicción del nivel de estrés a partir de los resultados de varios análisis biométricos. Se han desarrollado todos los componentes de la arquitectura necesarios para el servicio, como un portal web, base de datos (en nuestro caso, orientada a grafos), canales de comunicación y componente de predicción con datos en “streaming”.

El propósito fundamental es la investigación y el estudio de los Knowledge Graphs, así como las distintas tecnologías envueltas en el desarrollo de un despliegue de Big Data.

Palabras Clave

Aprendizaje Automático, Inteligencia Artificial, Microservicios, Macrodatos, Grafos de Conocimiento, VueJS, Docker, virtualización, Kubernetes, Spark, Kafka, Zookeeper, Express, Neo4J, NoSQL, Python, Scala, JavaScript.

Abstract

Artificial Intelligence systems have been a recurrent and approached topic over the past years. It is one of the most promising technologies. However, the results that are been obtained are not the ones we dreamed of at the beginning.

Nowadays, Machine Learning is at the top of IA. One of its well-known technologies are Neural Networks, which, as many other prediction solutions, can obtain good results. However, we cannot come up with the process by which these results have been obtained. We can define this process as a “Black Box”, we do not know what is inside this box, but we obtain an output introducing some inputs.

With IA’s next step, we pretend to avoid these black boxes. Knowledge Graphs are graphs which, thanks to nodes and relations, can trace the origin of the results. What is more, we could obtain new knowledge from these graphs.

In this Final Master’s Degree Project, I pretend to approach this technology by developing a specific use case: Physical stress level prediction from data previously obtained through biometric analysis. I have developed every component needed for an optimum architecture like Web Page, data base (Graph data base in our case), communication channels and a component dedicated to the live prediction from streaming data.

The main purpose is the investigation and study of Knowledge Graphs, as well as many technologies involved in the development of a Big Data deployment.

Keywords

Machine Learning, Artificial Intelligence, Microservices, Big Data, Knowledge Graphs, VueJS, Docker, virtualization, Kubernetes, Spark, Kafka, Zookeeper, Express, Neo4J, NoSQL, Python, Scala, JavaScript.

Índice general

Resumen.....	i
Palabras Clave	i
Abstract.....	iii
Keywords.....	iii
Índice general.....	v
Índice de figuras	ix
Glosario	xi
1 Introducción	1
1.1 Motivaciones	1
1.2 Objetivos	1
2 Estado del arte	3
2.1 Knowledge Graphs	3
2.2 VueJS y Express	4
2.3 Kafka y Zookeeper	4
2.4 Spark-Streaming	5
2.5 Neo4j	6
2.6 Docker y Kubernetes	6
3 Análisis y Diseño	8
3.1 Requisitos del sistema	8
3.2 Arquitectura del sistema	9
3.3 Descripción del sistema.....	10
3.4 Casos de uso	10
3.5 Diagrama de secuencia de la información	12
4 Desarrollo e Implementación.....	15
4.1 Dataset WESAD	15

4.2	Implementación de solución basada en Knowledge Graph.....	17
4.3	Implementación del servicio de clasificación del nivel de estrés en local	19
4.3.1	Servidor Express y Front End con VueJS.....	19
4.3.2	Kafka y Zookeeper	21
4.3.3	Clasificación con Spark-Streaming	22
4.3.4	Neo4j	23
4.4	Dockerización de servicios	23
4.5	Despliegue con Kubernetes.....	25
4.6	Pruebas.....	28
4.7	Descripción del repositorio	28
5	Mantenimiento y Operación	36
6	Conclusiones y Líneas futuras	37
6.1	Conclusiones.....	37
6.2	Objetivos Cumplidos	37
6.3	Líneas Futuras.....	38
6.3.1	Knowledge Graphs	38
6.3.2	VueJS y Express	38
6.3.3	Kafka y Zookeeper	38
6.3.4	Spark.....	39
6.3.5	Neo4J	39
6.3.6	Kubernetes	39
7	Bibliografía	41
	Anexo I. Manual de Usuario.....	48
	Ejecución en local.....	48
	Lanzar servicio usando Docker-compose.....	49
	Lanzar el servicio usando Kubernetes.....	50
	Posibles problemas con Kubernetes	50
	Knowledge Graphs scripts.....	51
	Anexo II. Aplicabilidad del RGPD.....	51
	Anexo III. Aspectos Éticos, Económicos, Sociales y Ambientales.....	53
	Introducción	53

Descripción de impactos relevantes relacionados con el proyecto.....	53
Análisis detallado de alguno de los principales impactos.....	54
Conclusiones	54
Anexo IV. Presupuesto Económico	55

Índice de figuras

Figura 1: Ejemplo de Knowledge Graph.....	1
Figura 2: Eras de la computación	3
Figura 3. Logos de VueJS y Express	4
Figura 4: Arquitectura típica de Kafka y Zookeeper	5
Figura 5: Logo de Spark	5
Figura 6: Logo de Neo4J	6
Figura 7: Logos de Docker y Kubernetes	7
Figura 8: Arquitectura del servicio	9
Figura 9: Modelo de dominio de los datos en circulación en el sistema	10
Figura 10: Diagrama de casos de uso.....	11
Figura 11: Diagrama de Secuencia del flujo principal del servicio	12
Figura 12: Diagrama de Secuencia del flujo secundario del servicio	12
Figura 13: Estructura de datos de las interacciones 1, 2 y 3.....	13
Figura 14: Estructura de datos de las interacciones 4, 5 y 6.....	13
Figura 15: Diagrama de Secuencia de las acciones necesarias para recuperar todo el grafo.....	13
Figura 16: Árbol de ficheros del dataset utilizado	15
Figura 17: Árbol de ficheros de los directorios del dataset utilizado	16
Figura 18: Estructura de datos del dataset inicial.....	16
Figura 19: Esquema de la "Tercera Era de la Computación"	17
Figura 20: Esquema del grafo emulado	17
Figura 21: Esquema de la transformación de dimensionalidad.....	18
Figura 22: Score Function del algoritmo ComplEx	18
Figura 23: Vista de la ruta raíz con VueJS.....	20
Figura 24: Vista de la ruta "/predict" con VueJS	20
Figura 25: Vista de la ruta "/data/:id" con VueJS.....	21
Figura 26: Precisión del modelo RF con PySpark	22
Figura 27: Precisión y matriz de confusión del modelo RF con Scikit-Learn.....	22
Figura 28: Flujo de datos del componente Spark	23
Figura 29: Arquitectura del servicio montado con docker-compose.....	24
Figura 30: Captura del panel de control de minikube.....	26
Figura 31: Arquitectura de los Servicios de tipo NodePort.....	27
Figura 32: Árbol de ficheros y directorios del proyecto.....	28
Figura 33: Árbol del directorio Docker	29
Figura 34: Árbol del directorio Notebooks.....	29

Figura 35: Árbol del directorio Python.....	30
Figura 36: Árbol del directorio Scala.....	31
Figura 37: Árbol del directorio k8s.....	32
Figura 38: Árbol del directorio Web.....	33
Figura 39: Árbol del directorio frontend.....	33
Figura 40: Árbol del directorio src.....	34
Figura 41: Árbol de componentes VueJS usados en el Front End.....	35
Figura 42: Flujo de trabajo de los equipos de desarrollo y operación.....	36
Figura 43: Ejemplo de generalización y categorización de datos	38
Figura 44: Ejemplo de anonimización de un conjunto de datos.....	51
Figura 45: Principios de la Privacidad según el RGPD	53

Glosario

ML: Machine Learning o Aprendizaje Automático.

IA: Inteligencia artificial.

RGPD: Reglamento General de Protección de Datos.

API: Application Programming Interface o Interfaz de Programación de Interfaces.

TFM: Trabajo Fin de Máster.

WESAD: Wearable Stress and Affect Detection.

NaN: Not a Number.

MRR: Mean Reciprocal Rank.

UI: User Interface o Interfaz de Usuario.

KG: Knowledge Graph o grafo de conocimiento.

RF: Random Forest (Algoritmo de ML).

K8s: Kubernetes.

BBDD: Bases de Datos.

IDE: Integrated Development Environment o Entorno de Desarrollo Integrado.

UPM: Universidad Politécnica de Madrid.

AWS: Amazon Web Services.

1 Introducción

1.1 Motivaciones

En la actualidad, el “Machine Learning” [1] o Aprendizaje Automático (en adelante ML) aparece con más frecuencia en nuestro día a día, aunque no nos demos cuenta. La mayoría de la gente no técnica en el tema, lo primero que piensa cuando escuchan el término ML es Inteligencia Artificial (en adelante IA). Aunque todavía se está muy lejos del desarrollo de una tecnología digna de “Skynet”, nos encontramos en un punto bastante avanzado desde el surgimiento del término de IA en el año 1956, durante la Conferencia de Dartmouth [2].

Desde las constantes búsquedas en Google hasta el proceso más delicado dentro de una empresa, pasando por las redes sociales, casi todo proceso tecnológico incluye cierto nivel de ML hoy en día. Esto no sería posible si esta tecnología no aportase cierto nivel de resultados. Cada vez más, las empresas apuestan por la integración de sus tecnologías ya consolidadas con tecnologías ML o incluso el desarrollo de unas nuevas con el ML como base.

Debido a este aumento de interés por las empresas, y a mi propio interés por tecnologías de despliegue a gran escala, como las usadas en Microservicios [3], se ha elegido el que muchos creen que es el nuevo hito en la historia del ML, los “Knowledge Graphs” o Grafos de Conocimiento (en adelante KG), como base de mi Trabajo Fin de Máster.

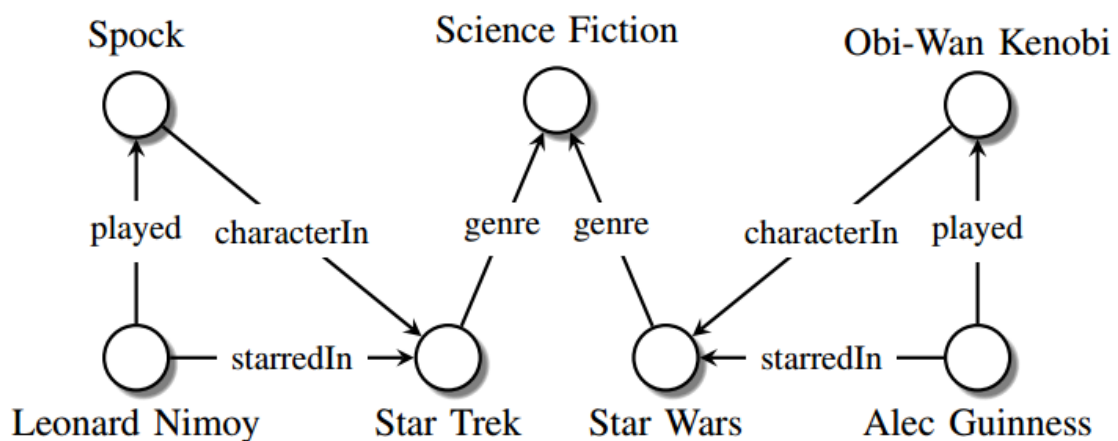


Figura 1: Ejemplo de Knowledge Graph

1.2 Objetivos

Con este TFM se pretende abordar los siguientes temas:

- El estudio de las tecnologías actuales orientadas a KG.
- El desarrollo de una arquitectura completa y escalable con tecnologías de Microservicios, como Kubernetes.

- Mejorar habilidades en el uso de lenguajes de programación funcionales como Scala.
- Aprender a usar una de las tecnologías emergentes para el desarrollo de “Front End” como VueJS.
- La aplicación de todas las tecnologías anteriores a un caso de uso real: **Un servicio de clasificación del nivel de estrés físico**. Este servicio sería desarrollado a partir del uso de un dataset con parámetros biométricos tomados a partir de unos sujetos de prueba. Para una funcionalidad completa, se desarrollarían todos los componentes de la arquitectura necesaria: Front End o parte de visualización, Back End o parte de lógica y persistencia gracias al uso de una base de datos.

Al final de la memoria, se recorrerá esta lista viendo los resultados obtenidos.

2 Estado del arte

El termino ML fue acuñado en 1959 por Arthur Samuel [4], aunque ya por los años 40, Isaac Asimov [5] daba pie al interés por la IA con su novela de ciencia ficción “Yo, Robot” y sus tres reglas de la robótica [6]. Desde entonces y a lo largo de todo el siglo XX, la IA ha tenido muchos altibajos: Desde su era “Dorada” de 1956-1974 [7] en la que las empresas invertían grandes cantidades de dinero para su desarrollo (con escasos resultados) hasta sus “inviernos” [8] como el de 1974-1980 en los que las empresas no veían avances sustanciales en esta tecnología.

En nuestros días, el ML se encuentra bien asentado dentro de múltiples empresas y procesos tecnológicos en los que cumple un papel fundamental. Los mejores sistemas de “Business Intelligence” [9] en empresa implementan soluciones ML y consiguen llegar a ahorrar millones a estas.

En este TFM, se implementarán y estudiarán múltiples tecnologías, usadas tanto en el ámbito del ML como en otros procesos paralelos.

2.1 Knowledge Graphs

Algunos expertos están empezando a divisar una “Tercera Era de la Computación” [10] con el surgimiento y estudio de los KG. Al contrario que las bases de datos tradicionales, donde se guardan datos y una vez extraídos, se procesan para obtener conocimiento, los KG son grafos donde se puede observar directamente dicho conocimiento.

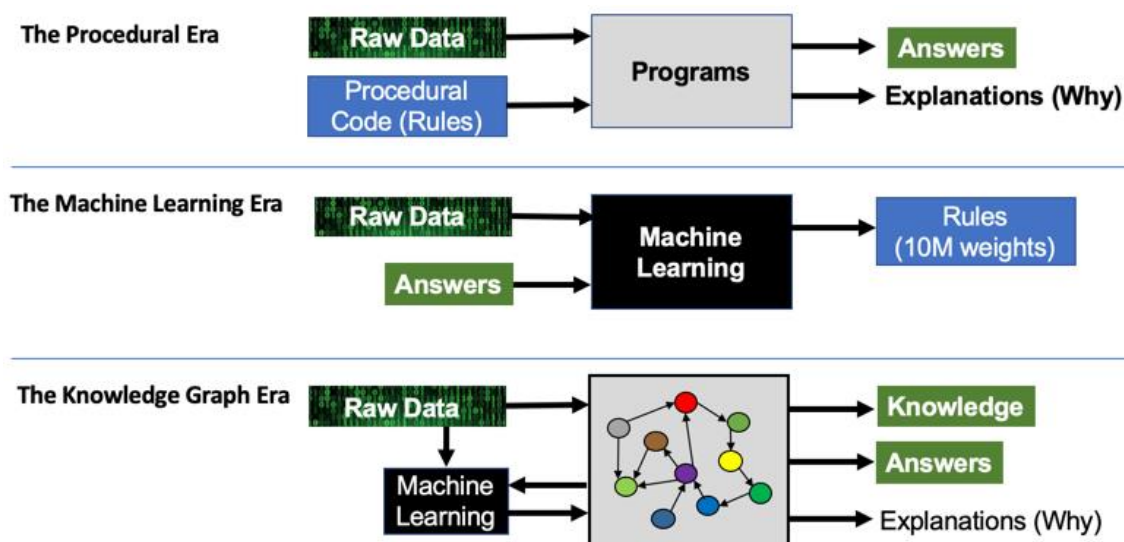


Figura 2: Eras de la computación

Hoy en día, los esfuerzos orientados a esta novedosa tecnología se centran en el “Graph Embedding” [11] o en la transformación de la dimensionalidad de estos grafos. Esto se traduce

en, la posibilidad de llevar estos grafos a un espacio tratable y manejable por los algoritmos “clásicos” de ML. En este TFM se usará Ampligraph [12], una librería de Python que implementa un gran número de algoritmos para tratar KG.

2.2 VueJS y Express

Para el desarrollo “Front End” de la arquitectura desarrollada en este TFM se han usado VueJS y Express. VueJS es un “UI Framework” de JavaScript [13], cuya popularidad se ha visto aumentada últimamente. En un principio, se pensó utilizar AngularJS [14], pero ante su “baja” popularidad estos 2 últimos años (sigue siendo el segundo más popular después de React, pero ha perdido interés frente a VueJS [15]), se decidió cambiar por una tecnología con una utilización en auge.



Figura 3. Logos de VueJS y Express

Para servir el código HTML, CSS y JavaScript generado por este Framework, se ha utilizado Node-Express [16], una librería o dependencia que te permite construir un servidor en Node [17]. A su vez, este será utilizado para facilitar una API para facilitar la comunicación con los siguientes niveles de la arquitectura. (Esta arquitectura queda descrita en el apartado de Análisis y Diseño).

2.3 Kafka y Zookeeper

Tanto Kafka [18] como Zookeeper [19] son proyectos de Apache [20]. Zookeeper es un software que permite la comunicación entre procesos distribuidos, tales como Kafka. Este último, es una plataforma de “Streaming” de datos que se utilizará para recoger toda la información enviada por el API anteriormente mencionado. Kafka actuará como un servicio de colas de mensajes del estilo “Publicación-Suscripción” [21]. Se necesita correr sobre un Zookeeper para que todas las instancias que se creen de Kafka estén comunicadas entre sí.

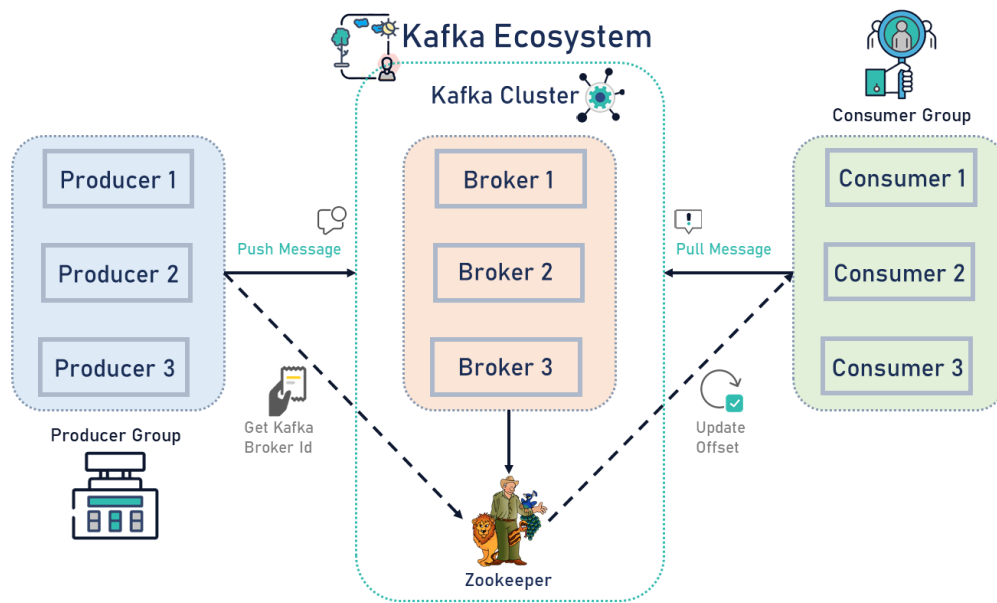


Figura 4: Arquitectura típica de Kafka y Zookeeper

2.4 Spark-Streaming

Apache Spark [22] es un motor de procesamiento de datos a gran escala, desarrollado para utilizar en Java, Scala o Python. Spark-Streaming [23] es la versión que te permite hacer el procesado “en vivo”.

En este TFM, se ha creado un proyecto de Spark usando Scala [24], un lenguaje similar a Java, que permite realizar una programación funcional [25], más adecuada para el proceso de datos en paralelo que la programación imperativa [26], en la que podemos reflejar lenguajes como Java, C o C++. Con este proyecto, se realiza una clasificación o predicción en vivo de los datos recibidos a través de Kafka. Tras la predicción, estos datos son almacenados en una base de datos junto con la predicción.



Figura 5: Logo de Spark

El modelo de ML usado en la predicción se ha calculado con PySpark [27], el motor desarrollado para Python. Se ha elegido usar un modelo Random Forest [28] ya que suele propiciar resultados óptimos.

Nota: Cabe señalar que el estudio, selección y optimización del modelo usado en este apartado no está concebido como parte del proyecto.

2.5 Neo4j

Neo4j [29] es una base de datos NoSQL [30] orientada a grafos bastante popular, sobre todo, porque es de código abierto para especificaciones o rendimientos medios-bajos. (Si se necesitasen mejores prestaciones, se tendría que acudir a alguna solución de Neo4j, u otra base de datos, de pago o con licencia). Se usará para almacenar los datos calculados en el nivel de Spark y así crear un KG. Neo4j, vista a través del Teorema de CAP [31] a la hora de distribuirla, se centra en potenciar la Consistencia y Disponibilidad frente a la Tolerancia a particiones. Esta característica es bien recibida al usar una arquitectura prevista para manejar gran cantidad de llamadas como la usada en este TFM.



Figura 6: Logo de Neo4J

2.6 Docker y Kubernetes

Las tecnologías basadas en contenedores, como son los microservicios, están muy extendidas entre las empresas hoy en día. Docker [32] es la elección por excelencia y sobre todo las soluciones basadas en esta, como Docker-compose [33] o Kubernetes [34]. En este proyecto, se propone como objetivo, llegar a desarrollar una solución de despliegue basada en Kubernetes, pasando antes por la “dockerización” de los servicios usados y posteriormente, el uso de Docker-compose para un primer despliegue más simple y menos potente que el que se propone en los objetivos.

Estos contenedores permitirán un rápido despliegue y escalado de los servicios, evitando a su vez posibles problemas de integración entre equipos, ya que permiten lanzar servicios en máquinas predefinidas y con la configuración que uno desee. Siempre se consigue el mismo estado final al usar soluciones Docker independientemente de en qué máquina se lancen o de la configuración que estas tengan.



Figura 7: Logos de Docker y Kubernetes

3 Análisis y Diseño

Para poder poner en práctica todas las tecnologías con las que se quería trabajar, se decidió desarrollar un caso práctico: Un servicio que clasifique, en tiempo real, el nivel de estrés del usuario. Con este servicio, se conseguirá poblar un grafo de conocimiento y así aplicar las distintas soluciones desarrolladas de KG.

Durante las siguientes secciones se recurrirá en varias ocasiones a la notación UML [35], la cual hace uso de varios elementos para describir el sistema a analizar. Por ejemplo, los diferentes actores que interactúan con el sistema, casos de uso y relaciones, etc.

3.1 Requisitos del sistema

En este apartado se describirán los diferentes requisitos de calidad o restricciones que deberá cumplir el sistema. Estos aspectos se contemplarán desde una posible fase de “development”. Se seguirá el estándar ECSS-E-ST-40C [36] para tratar los siguientes requisitos no funcionales:

- **Prestaciones:** Este requisito involucra aspectos como tiempos de carga, capacidades del sistema, frecuencias, velocidades... que no se han tenido en cuenta durante el desarrollo. Sin embargo, para pasar a producción sería necesario evaluar este requisito.
- **Interfaces:** El servicio no interacciona con ningún otro servicio externo, por lo que este requisito no aplica.
- **Operación:** Se pretende que los usuarios disfruten del sistema a través de una interfaz web. Como posible trabajo a futuro se podría desarrollar una solución dedicada para dispositivos móviles.
- **Recursos:** De forma similar al apartado de prestaciones, durante el desarrollo no se ha tenido en cuenta los recursos consumidos por el sistema. Queda como posible línea futura el optimizado de estos.
- **Verificación:** Se han realizado pruebas para comprobar el correcto funcionamiento de los distintos microservicios o partes que componen la arquitectura desarrolladas. Se describirán más adelante.
- **Pruebas de aceptación:** Estas pruebas involucran la puesta en operación del sistema. Se han realizado pruebas sencillas sobre el principal flujo de navegación al desplegar con Kubernetes y Docker. Se desarrollarán más en adelante.
- **Documentación:** Esta memoria actuará como documentación del desarrollo, incluyendo la bibliografía al final descrita.
- **Seguridad:** En un servicio ya en producción y fases siguientes, se debe garantizar mínimo las 3 dimensiones “canónicas” de la seguridad: Integridad, disponibilidad y confidencialidad. En la versión desarrollada solo se respeta la disponibilidad mediante el uso de Kubernetes. Queda para trabajos futuros el implementar soluciones seguras.

- **Portabilidad:** En una primera instancia, el sistema se ha desarrollado para que funcione en los navegadores más utilizados hoy en día (Chrome, Mozilla Firefox, Microsoft Edge y Safari). Como se ha dicho, se deja como línea futura el desarrollo de una solución móvil o aplicación, que se ejecutaría tanto en Android como IOs.
- **Fiabilidad:** En todo momento, se ha intentado evitar el surgimiento de errores. La gran mayoría de errores podrían aparecer en la interacción entre componentes, utilización de distintas versiones o sistemas base... Esto se consigue evitar con la dockerización del sistema, con la que se obtiene siempre entornos de ejecución idénticos (entre muchas otras ventajas analizadas más adelante) independientemente del sistema que corra o lance el servicio.
- **Mantenibilidad:** Este aspecto se ha ignorado desde el punto de vista de código y versionado de librerías. Por otra parte, se ha tenido en cuenta en el apartado de ciclo de vida de los microservicios, ya que Kubernetes ofrece soluciones para un cómodo escalado y administración de la arquitectura basada en contenedores.
- **Salvaguarda o “safety”:** El uso incorrecto del sistema no plantea daño personal alguno. Sin embargo, el no tener en cuenta la privacidad de los usuarios puede acarrear serios problemas, entre ellos, consecuencias legales (Incumplimiento del RGPD [37]).

3.2 Arquitectura del sistema

A continuación, se muestra la arquitectura seguida para el desarrollado tanto en local como con tecnologías de contenedores:

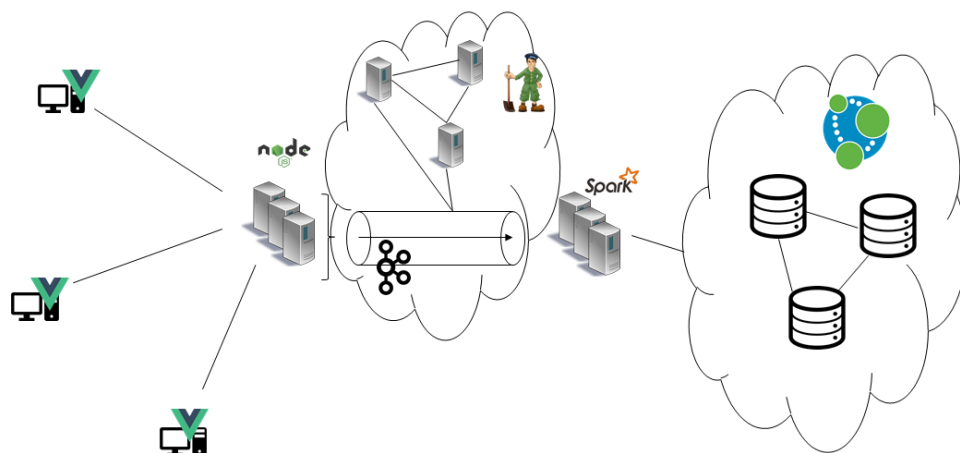


Figura 8: Arquitectura del servicio

Como se puede observar, se usan hasta 6 tecnologías distintas. Esto ha implicado que las mayores dificultades se han encontrado a la hora de integrar los distintos microservicios.

3.3 Descripción del sistema

En este apartado se describirá el esquema de datos que se comparte entre los distintos servicios de la arquitectura. Este, se ha visto influenciado en gran medida por el esquema del dataset utilizado para entrenar el modelo, el cual se verá en mayor profundidad en el apartado de desarrollo.

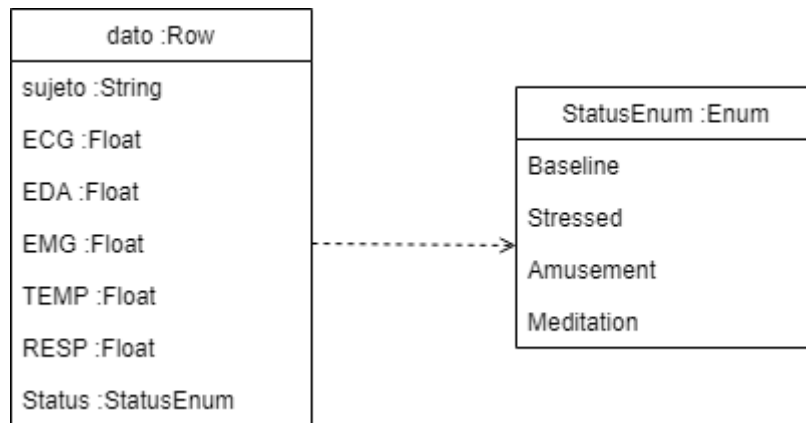


Figura 9: Modelo de dominio de los datos en circulación en el sistema

Se cuenta con estos 7 valores. Sin embargo, el modelo solo necesita de los 5 valores de tipo float (ECG, EDA, EMG, TEMP, RESP). A partir de estos, el modelo ofrece el valor de “status” o estado en el que se encuentra el sujeto. Estos estados pueden ser “Baseline” o normal, “Stressed” o estresado, “Amusement” o contento y “Meditation” o meditación. Estos vienen definidos en la descripción del dataset. Por último, “sujeto” sirve para identificar al conjunto de los datos atados a él. Para evitar problemas con el RGDP, este campo debería ser o estar anonimizado (en un anexo se analizará el proyecto desde una perspectiva de la privacidad o aplicabilidad del RGPD).

3.4 Casos de uso

Usando UML, se muestran los distintos casos de uso del servicio desarrollado:

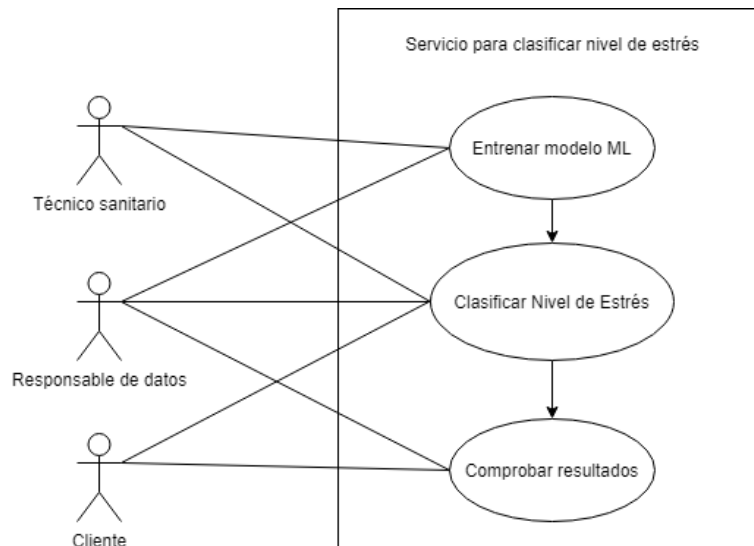


Figura 10: Diagrama de casos de uso

Pueden interactuar 3 tipos distintos de actores con el sistema:

- Técnico sanitario: Encargado de operar la maquinaria necesaria para realizar un análisis a los clientes. El sistema sería alimentado con estos datos.
- Responsable de datos: Encargado de administrar el sistema y desarrollar las soluciones ML pertinentes. También sería el encargado de administrar la gran cantidad de datos que maneja el sistema.
- Cliente: Beneficiario o usuario del servicio.

Se cuenta con 3 casos de uso muy básicos. Cada caso depende del anterior ya que ninguna acción se puede realizar sin antes haber hecho la acción o caso anterior.

Se parte de un primer caso de uso en el que sujetos de pruebas ceden sus análisis para, a partir de estos, calcular un modelo de ML para el siguiente caso de uso.

En el siguiente caso, un técnico sanitario, tomaría los datos necesarios usando las herramientas especializadas para ello. Estas son:

- Electrocardiograma [38] para el dato ECG.
- Sensor para medir actividad electro dérmica [39] para el dato EDA
- Sensor para tomar electromiografías [40] para el dato EMG.
- Sensor de temperatura [41] para el dato TEMP.
- Sensor piezoeléctrico para medir respiración [42] para el dato RESP.

En la versión desarrollada, este paso se simula y es el cliente quien directamente introduce los valores a través de la interfaz web. A continuación, estos datos recorren toda la arquitectura hasta llegar a la base de datos. Antes de llegar a esta, Spark hace una clasificación en vivo, con los datos enviados y el modelo calculado previamente, y clasifica

el estado del cliente. Una vez guardados los datos, el cliente puede pedir los resultados a través de la interfaz usando su identificador.

3.5 Diagrama de secuencia de la información

Para representar la interacción del cliente con el sistema y la transmisión de la información a lo largo de la arquitectura, se muestra el siguiente diagrama de secuencia:

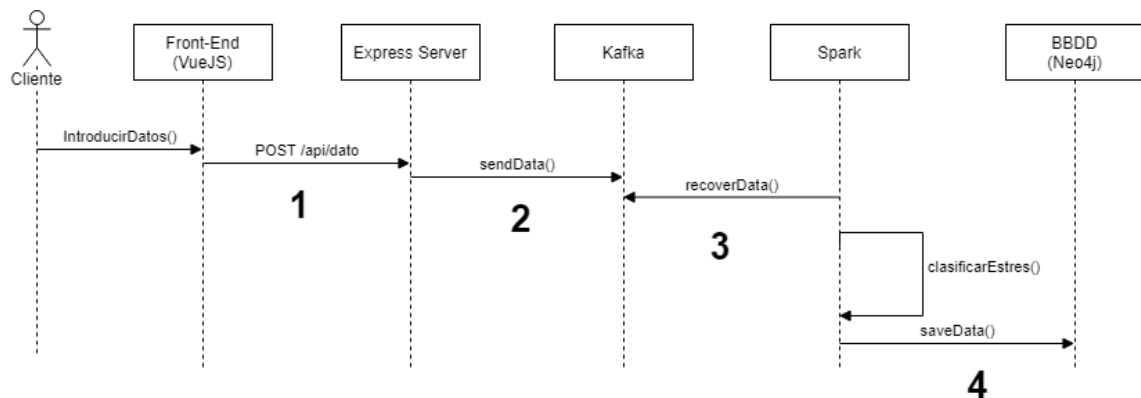


Figura 11: Diagrama de Secuencia del flujo principal del servicio

Este diagrama cubre el segundo caso de uso. En primer lugar, el cliente introduce sus datos en un formulario web junto con un identificador. Al enviar el formulario, los datos, en formato JSON, son recibidos por el servidor de Express (Se hace una petición POST a la API desarrollada). Este, los reenvía a Kafka que se encargará de hacerlos llegar a Spark. Hasta aquí, no ha habido transformación alguna en los datos. Spark introduce un nuevo campo “status” resultado de la clasificación del modelo de ML. Por último, estos datos se almacenan en Neo4j, creando nodos o entidades para los datos y relaciones entre estos. Tras múltiples secuencias, se dispondría de un KG.

Para representar el tercer caso de uso, se usa el siguiente diagrama de secuencia:

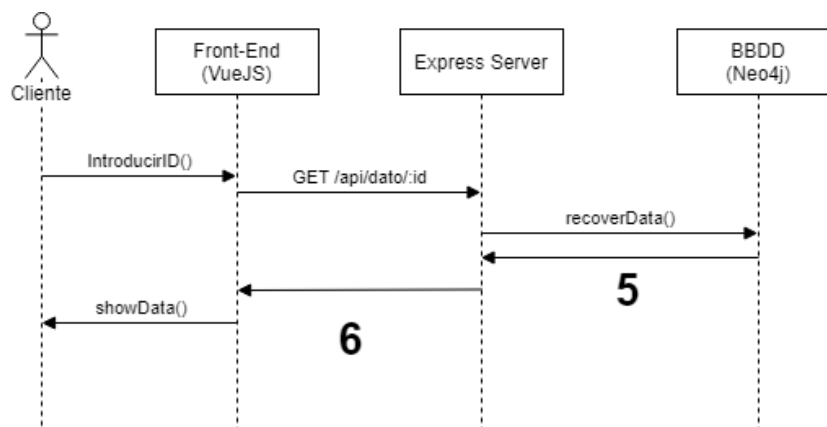


Figura 12: Diagrama de Secuencia del flujo secundario del servicio

En este, el cliente introduciría su identificador, elegido en el anterior caso de uso, para hacer una petición a través de la interfaz web. VueJS envía una petición GET a la API

desarrollada de Express y este reclama los datos relacionados con el identificador directamente a Neo4J.

En las distintas interacciones de los diagramas anteriores, los datos siguen una estructura fija. A continuación, se enseña dichas interacciones:

- Interacciones 1-2-3:

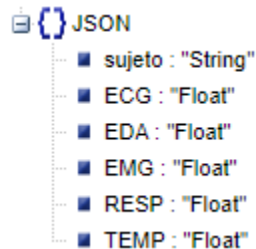


Figura 13: Estructura de datos de las interacciones 1, 2 y 3

- Interacción 4-5-6:

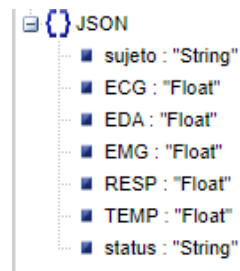


Figura 14: Estructura de datos de las interacciones 4, 5 y 6

Una vez se tiene un Knowledge Graph, se sigue el siguiente diagrama de secuencia para usar las tecnologías con las que tratarlos:

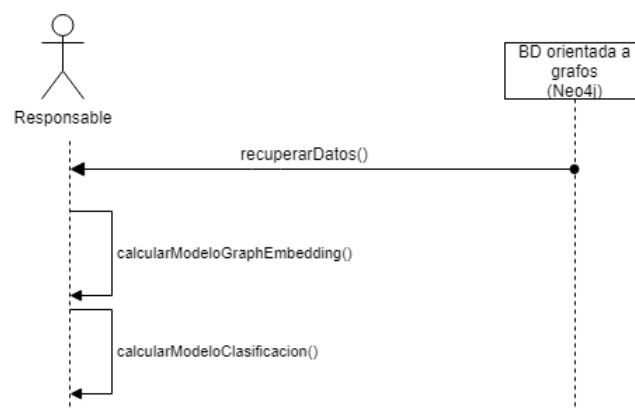


Figura 15: Diagrama de Secuencia de las acciones necesarias para recuperar todo el grafo

Primero, el responsable, recupera todos los nodos almacenados en Neo4j junto a sus relaciones. Aplica sobre estos datos varias transformaciones y calcula un modelo para cambiar la dimensionalidad del grafo. Tras esto, se obtiene un conjunto de valores

entendibles por los modelos clásicos de ML. Más adelante, se explicará con más detalle este proceso.

Nota: Este último diagrama no pertenece al servicio, por lo que no tiene un caso de uso asociado.

4 Desarrollo e Implementación

A lo largo de este apartado, se describirán los distintos pasos que se han seguido para desarrollar este TFM. Todo el código está disponible en un repositorio remoto [43]. El desarrollo se podría dividir en dos partes:

- La implementación de una solución aislada a partir de modelos de adaptación de Knowledge Graphs.
- La implementación del servicio descrito en el apartado de análisis y diseño.

Antes de comenzar con estas implementaciones, se dedicará un apartado al dataset que se ha usado a lo largo del TFM.

4.1 Dataset WESAD

Este dataset [44] [45], contiene los datos de un estudio realizado a 15 personas anónimas. En este estudio, se medían varios aspectos biométricos de estos sujetos y estos rellenaban un cuestionario para ver cómo se sentían (estresados, relajados...).

A continuación, se muestra la estructura del archivo descargado:

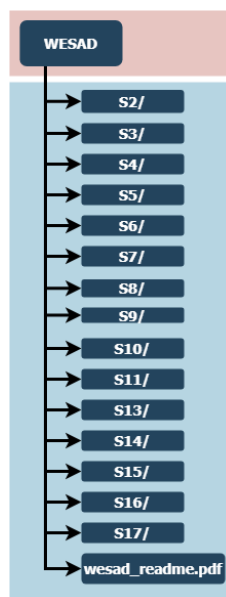


Figura 16: Árbol de ficheros del dataset utilizado

Para cada sujeto, obtenemos una carpeta con sus datos:

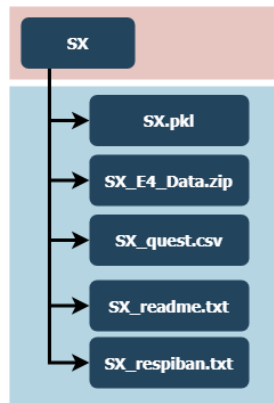


Figura 17: Árbol de ficheros de los directorios del dataset utilizado

Dentro de cada carpeta, se ha utilizado el archivo Sx.pkl, el cual contiene todos los datos en crudo. A continuación, se muestra la visualización de una entrada del archivo:

```

In [2]: df
Out[2]: {'signal': {'chest': {'ACC': array([[ 0.95539999, -0.222, ..., -0.55799997],
[ 0.32579997, -0.2218, ..., 0.55379999],
[ 0.96920992, -0.21960002, ..., 0.53920001],
...
[ 0.87179995, -0.12379998, ..., 0.30419999],
[ 0.87360003, -0.12339997, ..., 0.30260003],
[ 0.87020004, -0.12199998, ..., 0.30220002]]),
'ECG': array([[ 0.62142334],
[ 0.62832471],
[ 0.61852527],
...
[-0.00544739],
[ 0.00912732],
[ 0.0046741 ]]),
'EMG': array([[ -0.60444031],
[ 0.00434875],
[ 0.00517273],
...
[-0.61716614],
[-0.62897644],
[-0.02257482]]),
'EDA': array([[5.25054932],
[5.26733398],
[5.24381391],
...
[0.36048089],
[0.36582947],
[0.365448 ]]),
'Temp': array([[30.126758],
[30.129517],
[30.138214],
...
[31.459229],
[31.484283],
[31.456268]], dtype=float32),
'Resp': array([[ -1.14898602],
[-1.12457275],
[-1.15263857],
...
[-1.103321045],
[-1.08642578],
[-1.09716093]]),
'wrist': {'ACC': array([[ 62., -21., 107.],
[ 66., 13., 53.],
[ 41., 9., 15.],
...
[ 41., 25., 11.],
[ 39., 27., 22.],
[ 56., 26., 10.]]), 'BVP': array([[ -59.37],
[-53.42],
[-44.4 ],
...
[ 18.26],
[ 18.68],
[ 19.71]]), 'EDA': array([[1.138257],
[1.125448],
[1.011465],
...
[0.052688],
[0.073363],
[0.045113]]), 'TEMP': array([[35.41],
[35.41],
[35.41],
...
[34.23],
[34.23],
[34.23]]),
'Label': array([0, 0, 0, ..., 0, 0, 0], dtype=int32),
'subject': 'S2'}
  
```

Figura 18: Estructura de datos del dataset inicial

Al tener una gran cantidad de datos (≈ 63 Millones) y muchos campos a tener en cuenta, se ha decidido tener en cuenta solo los relativos a los sensores RespiBAN: ECG, EDA, EMG, RESP y TEMP. Estos ya han sido descritos en el apartado de diseño y análisis.

Para poder adaptar este dataset, se han desarrollado varios scripts para hacer un procesado previo, incluyendo una reducción del dataset, transformaciones... También se han desarrollado varios notebooks de Jupyter [46] para obtener una visualización cómoda de estadísticas del dataset.

4.2 Implementación de solución basada en Knowledge Graph

En esta implementación se ha intentado seguir el siguiente esquema:

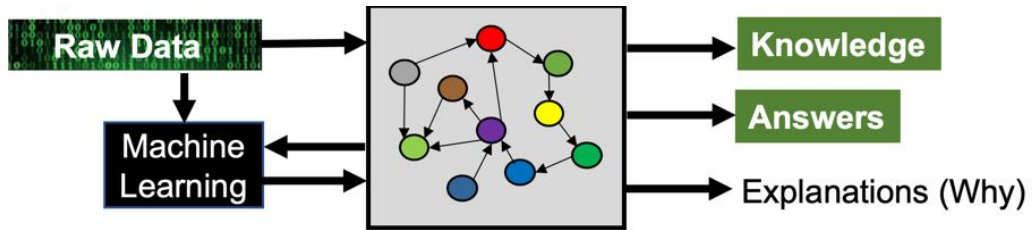


Figura 19: Esquema de la "Tercera Era de la Computación"

Como se puede observar, se trabaja alrededor de un grafo, el cual aporta conocimiento y respuestas a las preguntas que le hacemos (peticiones o “queries”) junto a una explicación de por qué nos responde esto (gracias a las relaciones entre nodos). La interacción clave y más compleja es la implicada en la comunicación del grafo y la caja que representa al ML. Este trabajo se centra en dicha interacción.

Para esta solución, se han desarrollado varios scripts en Python [47] usando la librería AmpliGraph siguiendo el siguiente tutorial [48].

El primer paso, una vez se tiene un dataset limpio, es transformar este en un grafo. Esto se consigue recorriendo el dataset y relacionando todos los campos entre sí. En el caso del dataset de WESAD, se ha decidido seguir el siguiente esquema:

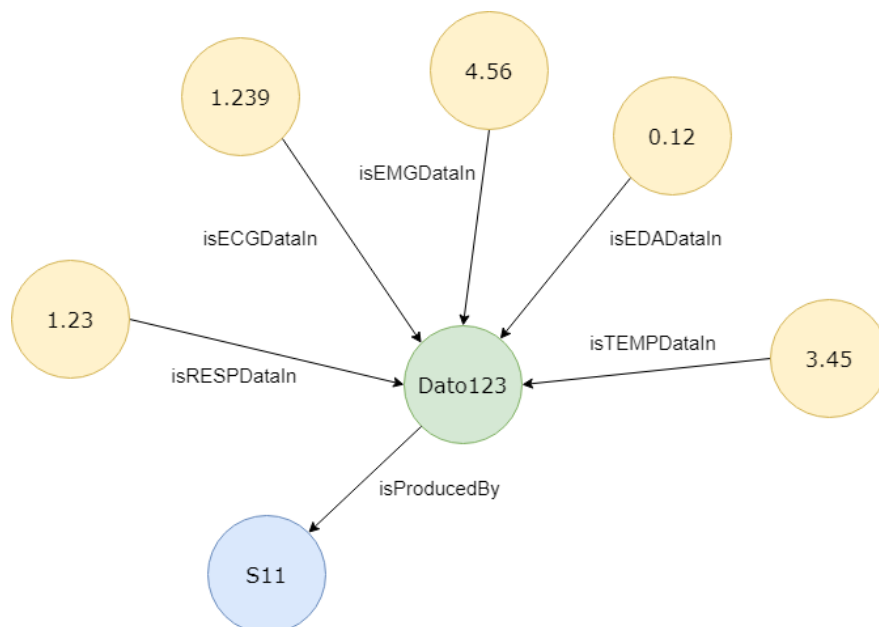


Figura 20: Esquema del grafo emulado

Una vez se tiene en un objeto guardado la representación de un grafo, se puede empezar a entrenar un “Knowledge Graph Embedding Model” o modelo para transformar la

dimensionalidad de un grafo [49]. Al transformar la dimensionalidad, se consigue “traducir” o transformar el grafo a una estructura de datos entendible por los algoritmos clásicos de ML.

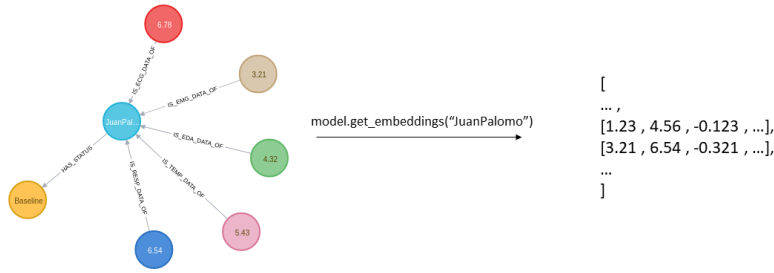


Figura 21: Esquema de la transformación de dimensionalidad

La API [50] de AmpliGraph ofrece varios algoritmos a usar en el modelo. En este caso, se ha decidido usar ComplEx, ya que, según el estado del arte [51], es el más eficiente. En las siguientes publicaciones [49], se describen en profundidad los algoritmos ofrecidos por la API.

Cabe señalar la gran diferencia entre usar “tensorflow” [52] “básico”, una librería dedicada al ML, y su versión optimizada para el cálculo con tarjetas gráficas, “tensorflow-gpu” [53]. Al principio del desarrollo, se usaba la primera opción, la cual llevaba una media de 12 horas de cálculo, dependiendo de la complejidad. Tras la implementación de la segunda opción, los mismos cálculos no tomaban más de 3 horas.

$$f_{\text{ComplEx}} = \text{Re}(\langle \mathbf{r}_p, \mathbf{e}_s, \overline{\mathbf{e}_o} \rangle)$$

Figura 22: Score Function del algoritmo ComplEx

Para evaluar este modelo, se ha usado una medición estadística llamada “Mean Reciprocal Rank” o “mrr_score” [54]. Esta es usada para evaluar procesos en los que se produce una lista de posibles valores ordenados por su probabilidad de exactitud, ante una petición. También se usa la función “hits_at_n_score” [55] que cede AmpliGraph, la cual calcula el porcentaje de peticiones que son respondidas correctamente según la posición del valor correcto. Por ejemplo, se tiene una petición que devuelve una lista de posibles resultados, que se ejecuta 4 veces. En la primera ejecución, se obtiene el valor correcto en la tercera posición o rango. En la segunda, se obtiene este valor a la primera. En la tercera, en la quinta posición. Y en la cuarta, en la tercera otra vez. Si se pide el “hits_at_n_score” del rango 3, se obtiene un 75% ya que 3 de las 4 peticiones, obtienen el resultado en la tercera posición o menor. Si se pide el “hits_at_n_score” del rango 1, se obtiene un 25%...

Tras el entrenamiento del modelo, ya se puede trabajar con algoritmos clásicos de ML. En los scripts desarrollados, se ha usado XGBClassifier de la librería xgboost [56]. Se ha elegido este algoritmo ya que no se ve interrumpido si se le inyecta valores de tipo “NaN”, un tipo inválido para la mayoría de los algoritmos de otras librerías, como Scikit-Learn [57]. Estos

valores aparecen si se pide la transformación de una entidad desconocida para el modelo, lo cual puede ser recurrente dependiendo del dataset usado, o al separar el dataset en dos, uno de entrenamiento y otro de prueba. Al usar un dataset más apropiado, con datos de tipo categórico con un espacio de datos acotado o finito, no surgiría este problema pues se “asegura” que el modelo conozca la entidad a transformar. Una solución al uso de datos numéricos con un amplio o infinito espacio de datos (Por ejemplo, números reales), sería la generalización o agregación y categorización de los datos. Esta técnica de agregación de datos es comúnmente usada en el área de anonimato de datos en privacidad [58] (Página 52 de 108). No se ha implementado esta solución por problemas de tiempo, pero como medida temporal se ha redondeado todos los datos para reducir un poco el espacio de posibles datos. Con estas soluciones cabe señalar que se perderá precisión en los cálculos de los algoritmos clásicos de ML.

En un principio, este desarrollo encontró muchos inconvenientes por problemas de entendimiento de los modelos para transformar la dimensionalidad. Una vez se vio y entendió que no se podían transformar entidades o conjuntos de datos que no participaban en el entrenamiento del modelo, el desarrollo fue trivial. Para la obtención de un modelo ideal, es necesario el entrenar con todo el espacio de posibles datos, lo cual con un espacio con datos no categóricos es imposible.

A continuación, se presentan multitud de publicaciones para continuar con el estudio. [59]

4.3 Implementación del servicio de clasificación del nivel de estrés en local

Esta sección se dividirá en varias subsecciones donde se explicará en detalle el desarrollo de los distintos componentes de la arquitectura.

4.3.1 Servidor Express y Front End con VueJS

Como primer punto de entrada a la arquitectura, se ha decidido usar un servidor Express debido a su simplicidad y potencial. A su vez, se ha usado VueJS para el desarrollo de la interfaz web. Se han construido 3 vistas distintas, una por cada ruta usada:

- “/”: Ruta base donde dar la bienvenida al servicio.

¡Bienvenidos a la Tercera Era del Machine Learning!



GitHub del proyecto: github.com/Jaimedfc/TFM

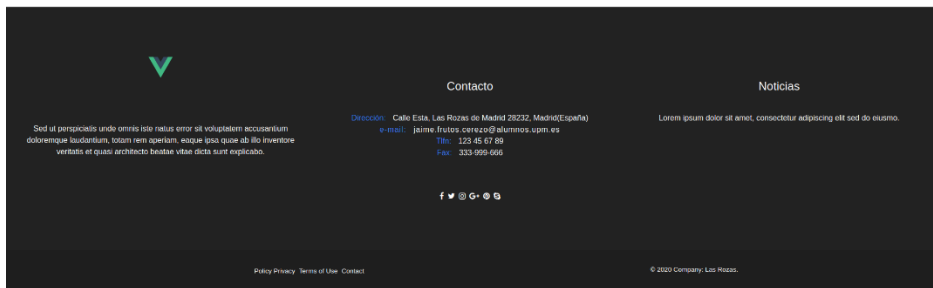



Figura 23: Vista de la ruta raíz con VueJS

- “/predict”: Ruta donde se encuentra el formulario a rellenar para empezar el flujo de datos de la arquitectura.


TFM de Jaime de Frutos Cerezo

Predecir nivel de Estrés

Sujeto a buscar:

Se recomienda esperar unos segundos tras rellenar el formulario inferior para buscar datos.

Por favor, rellene el siguiente formulario:

Sujeto:

Introducir un valor anonimizado, ej. "313"

Valor de ECG:

Valor de EMG:

Valor de EDA:

Valor de TEMP:

Valor de RESP:



Contacto

Dirección: Calle Esta, Las Rozas de Madrid 28232, Madrid(España)

e-mail: jaimede.frutos.cerezo@alumnos.upm.es

Tel: 123 45 67 89

Fax: 333-999-666

Noticias

Lorem ipsum dolor sit amet, consectetur adipiscing elit sed do eiusmod.

f

t

@

+

o

s

Policy

Privacy

Terms of Use

Contact

© 2020 Company, Ltd. Moscow.

Figura 24: Vista de la ruta "/predict" con VueJS

- “/data/:id”: Ruta donde comprobar la clasificación de los datos identificados por “:id”.



Figura 25: Vista de la ruta "/data/:id" con VueJS

Con este Front End, se hacen peticiones a una API muy básica construida en el servidor Express. La API es la siguiente:

- GET '/api/dato/:id': Petición que ejecuta una query a la base de datos y devuelve los datos relacionados con el nodo con el identificador pedido.
- POST '/api/dato': Petición con la que se envían los datos del formulario para que Express los envíe a su vez a Kafka, el siguiente componente de la arquitectura.

4.3.2 Kafka y Zookeeper

Estos componentes se han usado siguiendo los tutoriales de las páginas oficiales [60]. Para poner en funcionamiento este componente, debemos arrancar Zookeeper con:

- “\$ /kafka_2.11-2.4.0/bin/zookeeper-server-start.sh /kafka_2.11-2.4.0/config/zookeeper.properties”.

Con este comando de consola, se indica que queremos arrancar Zookeeper con la configuración predeterminada para un desarrollo en local.

Una vez tenemos Zookeeper corriendo, el encargado de controlar Kafka, ya podemos arrancar este con los siguientes comandos:

- “\$ /kafka_2.11-2.4.0/bin/kafka-server-start.sh /kafka_2.11-2.4.0/config/server.properties”.
- “\$ /kafka_2.11-2.4.0/bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic myTopic”.

Con el primero, corremos Kafka usando la configuración predeterminada para local y con el segundo comando indicamos que nuestro tópico por el cual filtrar los mensajes que le

interesan a Spark es “myTopic”. También se indica un factor de replicación de “1” y una partición. Estos parámetros no han sido sujeto de estudio.

4.3.3 Clasificación con Spark-Streaming

Antes de desarrollar este componente, se ha necesitado calcular un modelo de ML. Se ha escogido como algoritmo un Random Forest [28], un algoritmo basado en la generación de muchos algoritmos de árbol [61] quedándose con la composición de los que mejor resultado dan.

En un primer momento, se intentó implementar los resultados de los scripts de KGs, pero al comprobar su mal funcionamiento ante entidades o datos nuevos, se descartó la idea. Tras esto, se intentó usar un modelo RF desarrollado en Python con la librería Scikit-Learn. También se encontraron problemas debido a su mala integración con Spark-Scala.

Finalmente, se construyó un modelo RF usando la implementación en Python de Spark, PySpark, la cual no muestra problemas de implementación con su compañero de Scala. Cabe destacar la velocidad de cálculo, alrededor de 10 minutos que, comparado las implementaciones anteriores del orden de 45-60 minutos supone una gran mejoría. También hay que señalar que se consigue alrededor de un 80% de precisión cuando con la librería de Scikit-Learn se conseguía un 47%.

```
Total number of rows: 31471
Number of training set rows: 22066
Number of test set rows: 9405
Test Error = 0.331207
```

Figura 26: Precisión del modelo RF con PySpark

	precision	recall	f1-score	support
1	0.56	0.68	0.61	3246
2	0.42	0.31	0.36	1877
3	0.00	0.00	0.00	1004
4	0.36	0.46	0.40	2250
accuracy			0.46	8377
macro avg	0.33	0.36	0.34	8377
weighted avg	0.41	0.46	0.42	8377
[[2193 108 109 836]				
[603 580 0 694]				
[445 189 0 370]				
[701 503 0 1046]]				

Figura 27: Precisión y matriz de confusión del modelo RF con Scikit-Learn

Una vez calculado el modelo, se pudo proceder al desarrollo del código encargado de este componente usando Scala [24]. Primero, se codificó la integración con Kafka. Hay multitud de

ejemplos online por lo que no se encontraron problemas. A continuación, se codificó las transformaciones de los datos leídos:

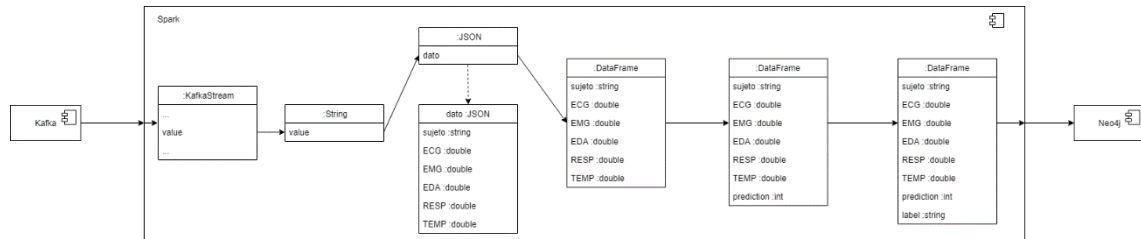


Figura 28: Flujo de datos del componente Spark

Por último, hecha la clasificación, solo restaba guardar los resultados en Neo4j. Se establece el identificador como nodo central y el resto de datos se guardan como nodos con relaciones hacia el nodo identificador, para una cómoda recuperación de estos.

En última estancia, hay que mencionar el entorno de desarrollo o IDE empleado en este microservicio, IntelliJ IDEA [62]. Es un IDE de pago que incluye facilidades para programar con Scala y compilar los proyectos con sbt. La UPM provee una licencia de estudiantes para su uso.

4.3.4 Neo4j

Para el desarrollo con Neo4j, simplemente se ha usado un contenedor Docker para evitar la instalación en local. Esto no afecta al desarrollo. Se ha ejecutado el contenedor con el siguiente comando [63]:

```
"docker run --name testneo4j -p7474:7474 -p7687:7687 -v $HOME/neo4j/data:/pruebas --env=NEO4J_AUTH=neo4j/test --rm neo4j:latest"
```

Con él se dice que se quiere correr Docker y exponer los puertos correspondientes para hacer accesible el contenedor. Los puertos expuestos son los siguientes:

- 7474: Puerto para acceder al contenedor con el protocolo http. También se proporciona una interfaz web para interactuar con la base de datos.
- 7687: Puerto para interactuar con el contenedor con el protocolo bolt [64].

4.4 Dockerización de servicios

Una vez se tiene todo el sistema construido y corriendo en local, se procedió al despliegue usando contenedores Docker. A continuación, se muestra el esquema de contenedores y las conexiones entre estos:

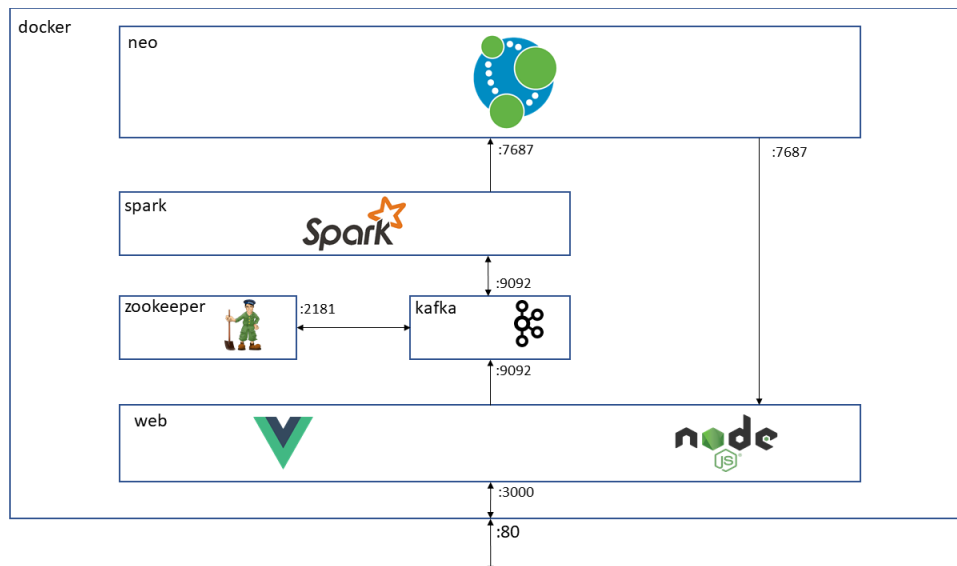


Figura 29: Arquitectura del servicio montado con docker-compose

Solo se han construido 2 contenedores Docker de los 5 microservicios construidos, ya que, se han usado contenedores ya construidos por la comunidad para el resto. La única dificultad que surge es la configuración de dichos contenedores. Los contenedores construidos son:

- Web [65]: Se parte de una imagen con Node instalada, se clona el repositorio con todo el código del proyecto, se instalan las dependencias necesarias con npm, se expone el puerto 3000 y se construye y corre el servidor.
- Spark [66]: Se parte de una imagen con un openjdk [67] (versión 8) instalada, se descargan e instalan spark (para correr el contenedor) y sbt para compilar el código a ejecutar. Se clona el repositorio, se copia el modelo calculado en local y se compila el código de Scala con sbt.

Las imágenes de la comunidad usadas son las siguientes:

- Zookeeper [68].
- Kafka [69].
- Neo4j [70].

Una vez se tienen todos los contenedores disponibles en DockerHub, se puede proceder al desarrollo con Docker-compose.

Las dificultades encontradas han sido relacionadas con las configuraciones de los contenedores ajenos (se contaba con una buena documentación por lo que no se encontraron muchos problemas) y la integración entre estos. Gracias a las funcionalidades NAT [71] que ofrece Docker-compose y al uso de variables de entorno [72], esta última complejidad fue solventada “cómodamente”.

4.5 Despliegue con Kubernetes

El siguiente paso y último, fue la construcción de los artefactos necesarios para el uso de Kubernetes.

Se comenzó usando la herramienta Kompose [73], la cual proporciona artefactos base si se le proporciona un archivo Docker-compose. Estos artefactos se dividen en:

- Deployments [74]: Ficheros YAML donde establecer la configuración de los contenedores a usar en cada pod [75]. Una vez hecha la configuración en el paso anterior, la programación de estos ficheros es trivial.
- Services [76]: Ficheros YAML donde describir la exposición de los distintos servicios o contenedores. Son necesarios para la integración de microservicios.
- Ingress [77]: Fichero YAML donde describir los puntos de entrada a los contenedores desde el exterior. Se puede entender como un ejemplo muy concreto de Service.

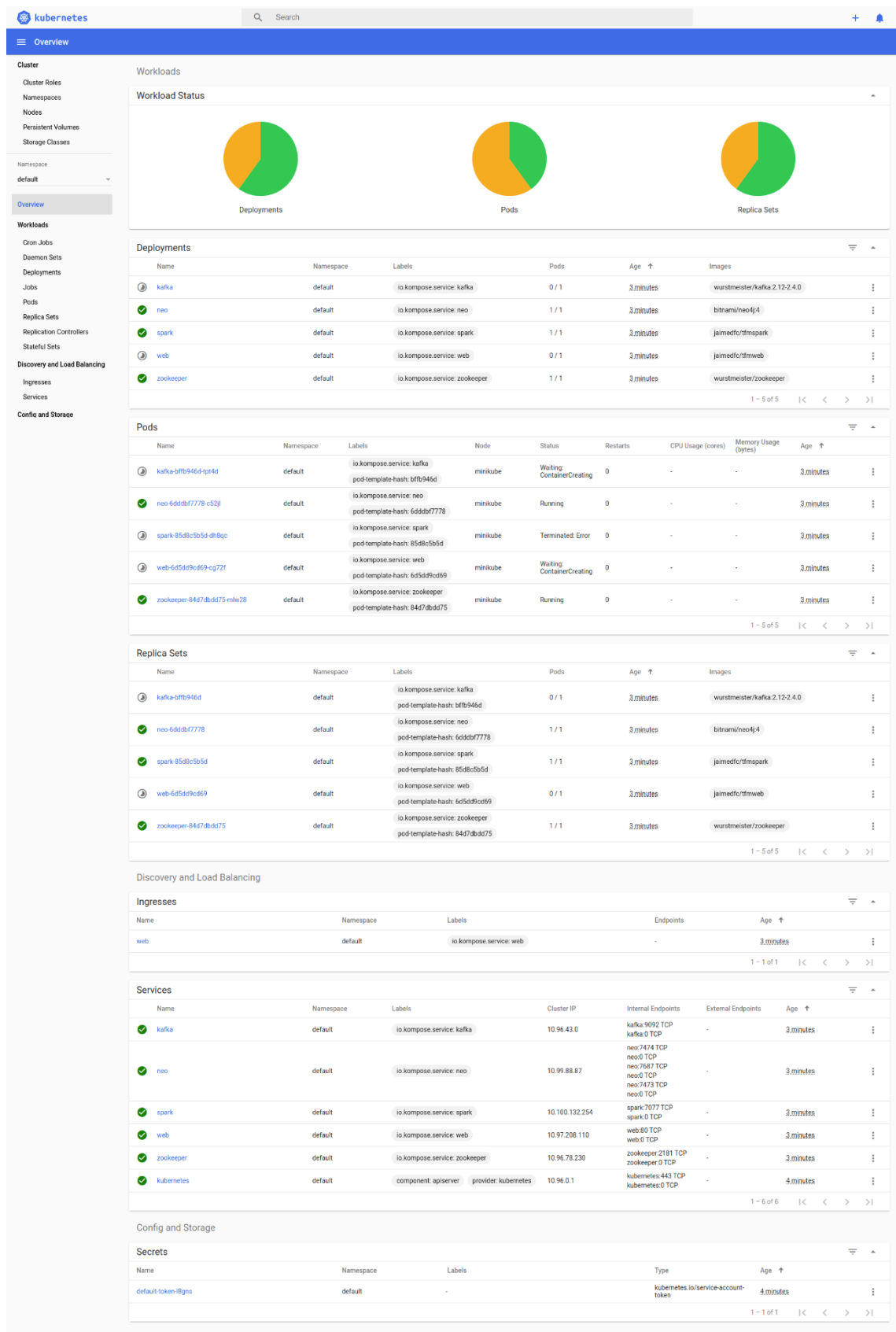


Figura 30: Captura del panel de control de minikube

Los únicos problemas, a parte del hecho de usar una tecnología prácticamente desconocida, se encontraron en la configuración de los Deployments y en la integración de estos:

- Se usó la misma configuración que en el paso anterior, sin embargo, al estar usando K8s aparecieron ciertos problemas con Kafka. En un principio se pensó que era un problema de integración entre web y Kafka, ya que el problema se detectaba en el primero. Al hacer una petición para introducir datos al sistema, saltaba un código de error 400 (“Bad Request” o que se está accediendo al recurso de forma errónea). Tras muchas pruebas y comprobar los logs de ambos contenedores, se vio que el problema era el Deployment de Kafka, el cual no se conseguía levantar correctamente. Tras “bucear” un poco en foros, se vio que K8s, al llamar un deployment Kafka, creaba una variable de entorno llamada “KAFKA_PORT” que sobrescribía una variable necesaria. Se solucionó volviendo a sobrescribir esa variable en la configuración del Deployment.
- En un primer momento, se usaron Services de tipo “LoadBalancer”, el cual era el más recomendado en foros, sin embargo, para un despliegue en local usando herramientas como Minikube [78] o kubectl [79], se deben usar servicios de tipo “NodePort”. Los Servicios LoadBalancer están orientados al despliegue en nubes, mientras NodePort simplemente mapea un puerto del nodo donde se despliega con la dirección del contenedor o Service interno.

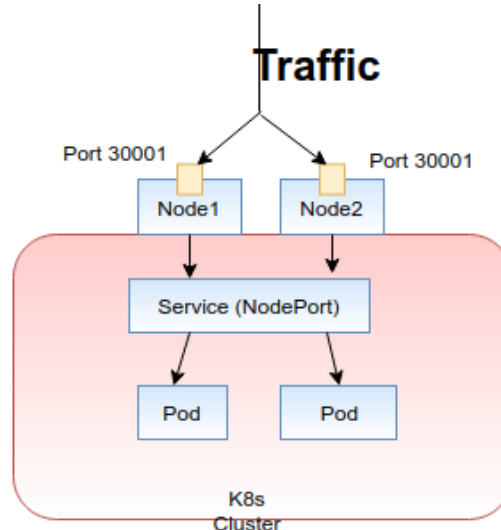


Figura 31: Arquitectura de los Servicios de tipo NodePort

Una vez se tienen todos los ficheros necesarios, solo queda arrancar el nodo donde se ejecutarán los microservicios o Pods y desplegar en este toda la arquitectura. Para realizar esto, se han usado las herramientas minikube, para desplegar y administrar el nodo, y kubectl, para desplegar y administrar los Pods. Un Pod [75] es la unidad mínima usada en K8s, en cada Pod se puede desplegar uno o varios contenedores. Si se decide desplegar

varios contenedores en un Pod, es porque se dedican a realizar el mismo trabajo o forman parte del mismo microservicio.

4.6 Pruebas

Al acabar cada hito del desarrollo, se han realizado pruebas sobre el flujo de datos en el servicio. Simplemente, se introducían datos a través del Front End y se comprobaba que circulaban correctamente por toda la arquitectura hasta llegar a la base de datos. Para realizar esto, en el desarrollo en local era una acción trivial: Observar la consola de cada tecnología en funcionamiento. Para la parte de contenedores Docker, se hizo un uso intensivo de la herramienta de mostrado de logs por pantalla de cada contenedor. El comando es el siguiente:

En Docker/Docker-compose: `$docker logs <ID del contenedor>`

En K8s: `$kubectl logs <ID del Deployment>`

4.7 Descripción del repositorio

Acabado el desarrollo, se procede a describir el repositorio [43] donde se puede encontrar todo el código escrito. Al clonar el proyecto, uno se encuentra con lo siguiente:

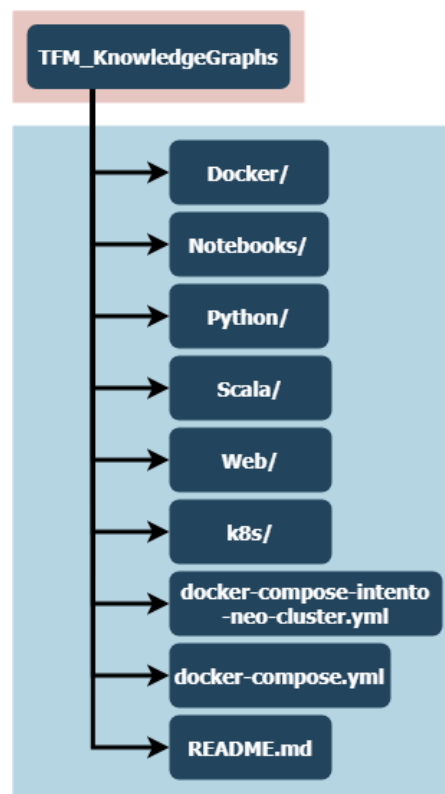


Figura 32: Árbol de ficheros y directorios del proyecto

- **Fichero README.md:** Fichero generado por GitHub al crear el repositorio. En él, se puede escribir información referente al proyecto para informar al que quiera utilizarlo en un futuro. Por ejemplo, se puede escribir un “Manual de uso” para la instalación y ejecución del proyecto, entre otras cosas.

- **Fichero docker-compose.yml:** Fichero usado por la herramienta Docker-compose para levantar toda la arquitectura siguiendo las configuraciones de contenedores descritas en el propio fichero.
- **Fichero docker-compose-intento-neo-cluster.yml:** Fichero similar al anterior en el que se pretendía desplegar un clúster de BBDD de Neo4j. Al encontrar numerosos problemas para configurar la integración de este clúster con el resto de la arquitectura, se decidió abandonar por falta de tiempo.
- **Directorio Docker:** Carpeta que contiene los Dockerfile desarrollados para definir los contenedores del Front End y Spark.

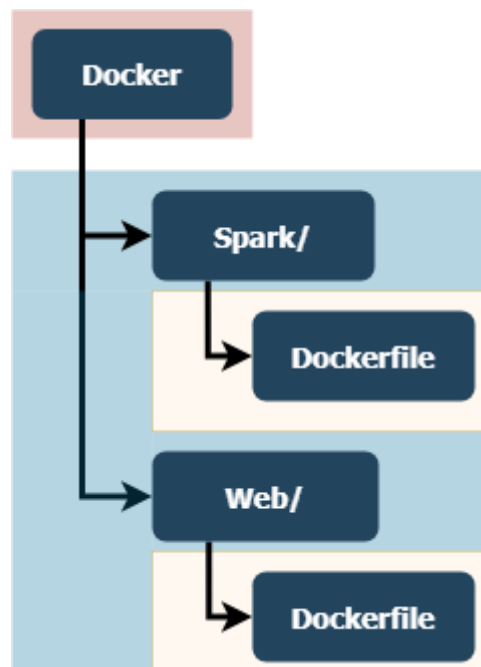


Figura 33: Árbol del directorio Docker

- **Directorio Notebooks:** Carpeta que contiene los notebooks de Jupyter desarrollados.

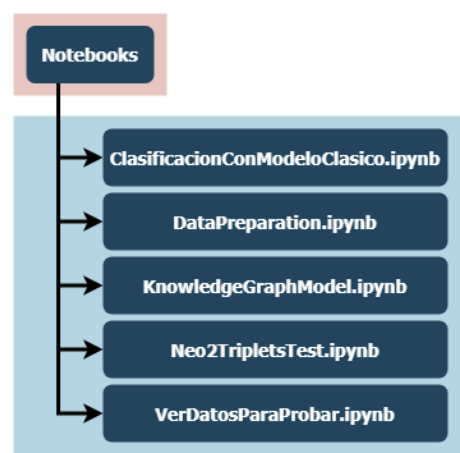


Figura 34: Árbol del directorio Notebooks

- **Fichero ClasificacionConModeloClasico.ipynb:** Notebook en el que se calcula un modelo RF con la librería SKLearn [80], usando el mismo dataset que en el modelo calculado para la parte de Spark, evaluando el resultado.
- **Fichero DataPreparation.ipynb:** Notebook en el que se hace una pequeña visualización de los datasets que contiene WESAD, se seleccionan los campos deseados y se guarda un dataset limpio con el que poder trabajar.
- **Fichero KnowledgeGraphModel.ipynb:** Notebook en el que se hicieron las primeras pruebas con la librería Ampligraph. Da errores de ejecución pues la librería tensorflow y los notebooks no funcionan bien en conjunto.
- **Neo2TripletsTest.ipynb:** Notebook en el cual se hicieron pruebas para el desarrollo del script Neo2Triplets.py.
- **Fichero VerDatosParaProbar.ipynb:** Notebook de pruebas en el que se visualiza el esquema de datos del dataset utilizado en los modelos.
- **Directorio Python:** Carpeta en la que guardar todos los scripts desarrollados en Python.

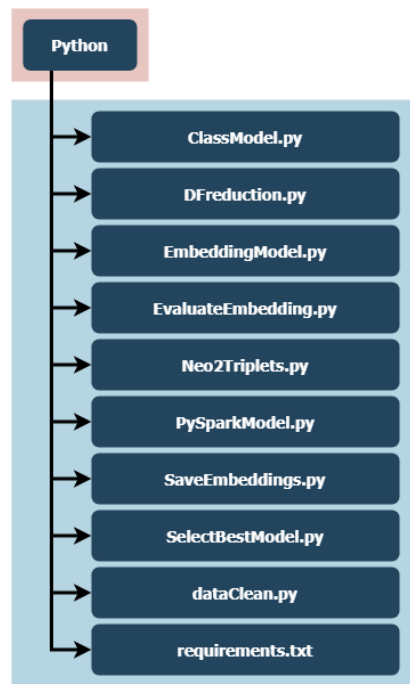


Figura 35: Árbol del directorio Python

- **Fichero ClassModel.py:** Script que crea un modelo de clasificación con la librería XGBClassifier tomando como datos la salida del modelo de transformación de dimensionalidad del grafo.
- **Fichero DFredution.py:** Script para reducir el tamaño del dataset procesado. Reduce el dataset 1000 veces. (Un dataset con 5000 filas se quedaría en un dataset con 5 filas).
- **Fichero EmbeddingModel.py:** Script principal en el que a partir del dataset reducido, se obtiene un modelo para transformar la dimensionalidad de un

grafo evaluando su rendimiento. Se puede adaptar para que tome como datos de entrada las entidades y relaciones descargadas con el script Neo2Triplets.py.

- **Fichero EvaluateEmbedding.py:** Script que carga el modelo generado por Ampligraph junto al dataset utilizado y evalúa su rendimiento.
 - **Fichero Neo2Triplets.py:** Script para descargar la información de Neo4J, nodos y relaciones, y guardarla en un dataset para luego aplicar los demás scripts.
 - **Fichero PySparkModel.py:** Script utilizado para calcular y guardar el modelo RF usado por el microservicio de Spark.
 - **Fichero SaveEmbeddings.py:** Script usado para guardar un dataset, generado tras aplicar el modelo de transformación de la dimensionalidad, en diferentes formatos (CSV, JSON).
 - **Fichero SelectBestModel.py:** Script usado para calcular los mejores hiperparámetros a pasar al modelo de transformación de la dimensionalidad.
 - **Fichero dataClean.py:** Script con la misma funcionalidad que DataPreparation.ipynb. Carga todos los datasets de WESAD, los junta y selecciona los campos a usar en el dataset definitivo.
 - **Fichero requirements.txt:** Fichero que contiene todas las dependencias usadas por los scripts instalables con pip [81], herramienta manejadora de bibliotecas de Python.
- **Directorio Scala:** Carpeta contenedora del proyecto de Scala, encargado de hacer la clasificación en el microservicio de Spark.

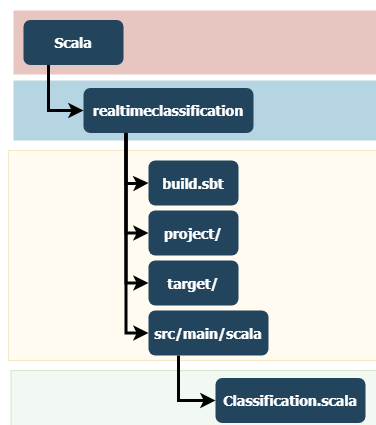


Figura 36: Árbol del directorio Scala

- **Directorio realtimeclassification:** Carpeta a compilar con la herramienta sbt.
- **Fichero build.sbt:** Fichero de configuración de sbt.
- **Directorio project:** Carpeta autogenerada por IntelliJ.
- **Directorio target:** Carpeta destino del fichero compilado del proyecto Scala.
- **Fichero src/main/scala/Classification.scala:** Fichero contenedor del código con la lógica del microservicio de Spark.

- **Directorio k8s:** Carpeta que contiene todos los ficheros necesarios para el despliegue con K8s.

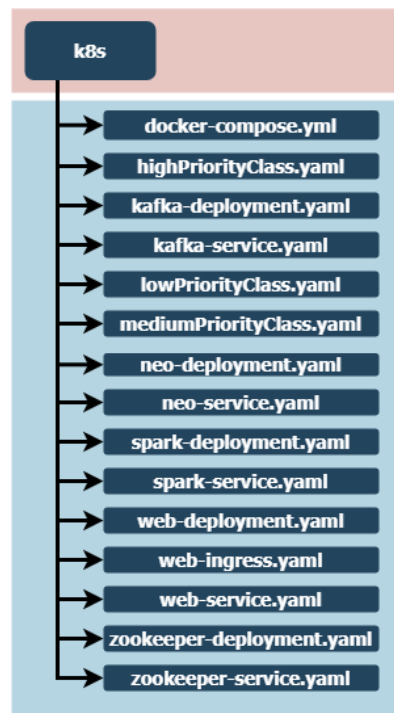


Figura 37: Árbol del directorio k8s

- **Fichero docker-compose.yml:** Fichero con la definición del despliegue usando Docker-compose adaptado para el uso de Kompose y generar los demás ficheros.
- **Ficheros *-deployment.yaml:** Ficheros en los que se configura cada Pod o microservicio.
- **Ficheros *-service.yaml:** Ficheros en los que se configura la exposición de cada Pod o microservicio al exterior para la integración con otros microservicios.
- **Fichero web-ingress.yaml:** Fichero con la configuración necesaria para exponer la arquitectura al exterior y poder acceder al servicio.
- **Ficheros *Class.yaml:** Ficheros con la definición de los tipos de prioridades de los microservicios. Estas prioridades indican el orden de arrancado de los Pods, cuanta más alta es la prioridad, antes se despliega.
- **Directorio Web:** Carpeta contenedora del código de Express y VueJS.

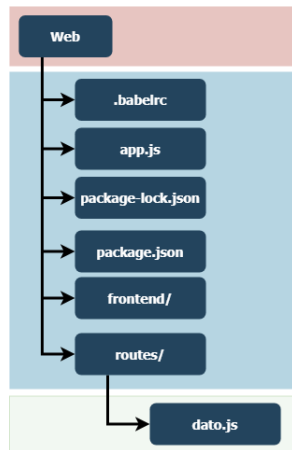


Figura 38: Árbol del directorio Web

- **Fichero app.js:** Fichero a ejecutar por Node que contiene la lógica del servidor Express.
- **Fichero .babelrc:** Fichero de configuración de babel [82]. Librería usada para la traducción y compilación de ficheros JavaScript.
- **Fichero package-lock.json:** Fichero autogenerado por npm.
- **Fichero package.json:** Fichero con el registro de librerías usadas en el microservicio de Express.
- **Directorio routes:** Carpeta contenedora del código con la lógica de la ruta “/api/dato”.
- **Directorio frontend:** Carpeta contenedora del código del Front End con VueJS.

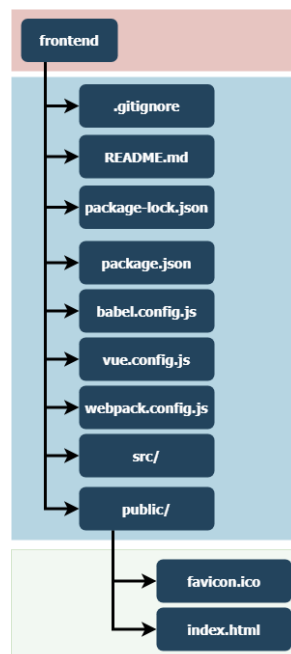


Figura 39: Árbol del directorio frontend

- **Fichero .gitignore:** Fichero usado por Git en el que establecer que ficheros debe ignorar la herramienta.

- **Fichero README.md:** Fichero autogenerado con una breve explicación para correr el proyecto de VueJS.
- **Fichero babel.config.js:** Fichero de configuración de babel.
- **Fichero package-lock.json:** Fichero autogenerado por npm.
- **Fichero package.json:** Fichero con el registro de librerías usadas en el proyecto de VueJS.
- **Fichero vue.config.js:** Fichero de configuración de VueJS. En él, se establece donde se debe “buildear” o compilar el proyecto (/Web/public).
- **Fichero webpack.config.js:** Fichero de configuración de webpack [83]. Librería usada para compilar el proyecto.
- **Directorio public:** Directorio con la imagen a usar en la pestaña del navegador y una página index.html a rellenar con VueJS.
- **Directorio src:** Carpeta contenedora del código de VueJS.

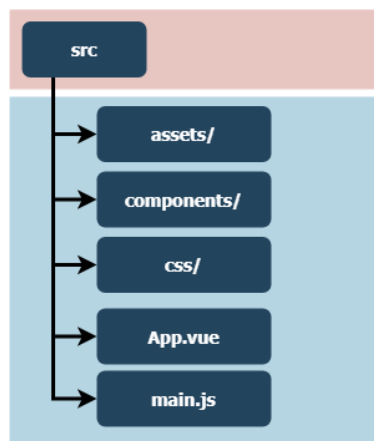


Figura 40: Árbol del directorio src

- **Fichero main.js:** Fichero principal a correr con Node. En él, se define las rutas a usar y los componentes a renderizar en cada una.
- **Fichero App.vue:** Componente principal de VueJS contenedor de los demás.
- **Directorio assets:** Directorio que contiene imágenes usadas en el portal web.
- **Directorio css:** Carpeta con el fichero css del portal web.
- **Directorio components:** Carpeta donde almacenar los componentes VueJS desarrollados. A continuación, se muestra un árbol con los componentes utilizados. Se han usado componentes ya desarrollados con Bootstrap [84] junto con otros propios.

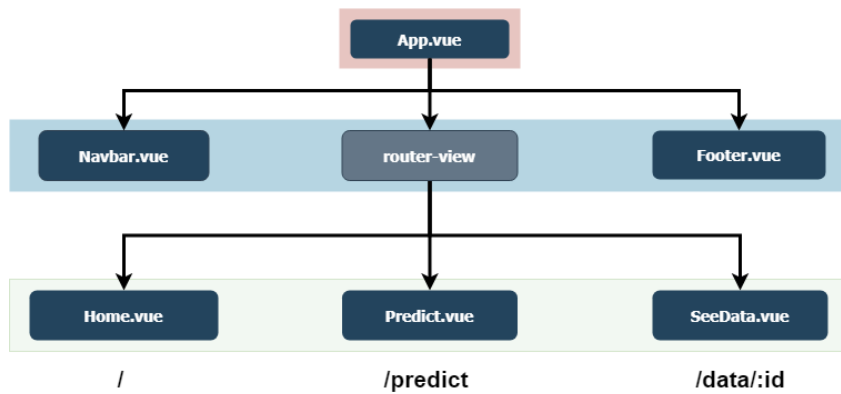


Figura 41: Árbol de componentes VueJS usados en el Front End

- **Home.vue:** Componente cargado al acceder a “/”. Contiene la portada del portal web.
- **Predict.vue:** Componente cargado al acceder a “/predict”. Contiene dos formularios: Uno para introducir datos en el servicio y otro para solicitar la información almacenada en la base de datos.
- **SeeData.vue:** Componente cargado al acceder a “/data/:id”. Al ser montado, hace una petición para conseguir los datos relacionados con la persona identificada por “:id”.
- **Navbar.vue:** Componente usado en todas las rutas que renderiza una barra de navegación haciendo uso de componentes externos de Bootstrap.
- **Footer.vue:** Componente usado en todas las rutas que renderiza un pie de página ejemplo reciclado de otros trabajos propios [85] y adaptado a VueJS.

Nota: Estos componentes se pueden visualizar en el apartado “Servidor Express y Front End con VueJS”.

5 Mantenimiento y Operación

Con el despliegue de la arquitectura o la finalización de la implementación no acaba un servicio software. El mantenimiento es una de las partes que más ingresos genera a las distintas empresas y es una de las partes donde más esfuerzo se invierte. Esta aplicación no va a ser distinta.

En lo referente a disponibilidad o escalabilidad del servicio, K8s cumple una función crítica. El uso de esta tecnología supone una gran ventaja frente a otros tipos de despliegues. Los contenedores permiten un escalado rápido y cómodo, sin apenas intervención de un operador. Con un comando y unos pocos minutos tienes un servicio completo desplegado y funcionando. Si el desarrollo hubiese acabado sin usar Docker, el escalado del servicio sería insostenible e inoperable debido a la cantidad de componentes y variables a tener en cuenta (Interconexiones, datos, configuraciones...).

Respecto al grafo construido a partir del servicio, conviene reiniciarlo o vaciarlo tras pasar un cierto número de instancias, ya que, si no se respetase este límite, los futuros modelos para transformar su dimensionalidad consumirían demasiado tiempo sin aportar una mejora significativa en la precisión. También cabe señalar que, a mayor cantidad de datos, mayor es el tiempo de recuperación y borrado de estos.

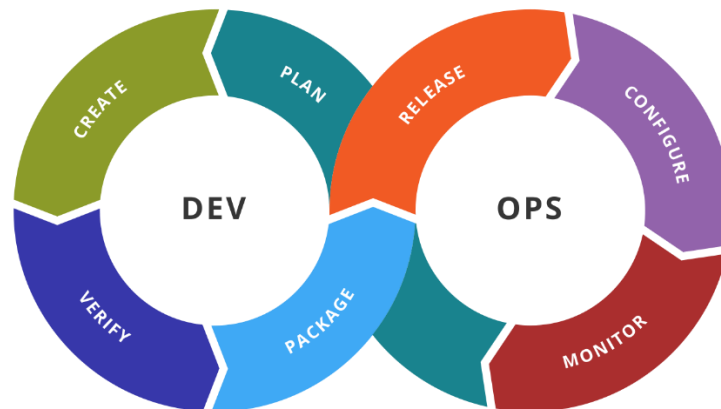


Figura 42: Flujo de trabajo de los equipos de desarrollo y operación

6 Conclusiones y Líneas futuras

Para finalizar esta memoria, se describirán en los siguientes apartados, las conclusiones obtenidas, objetivos cumplidos y líneas futuras o posibles trabajos pendientes para mejorar el proyecto.

6.1 Conclusiones

Para comenzar las conclusiones, se hará un breve comentario acerca del tiempo que ha llevado todo el proyecto. En primer lugar, las pruebas relacionadas con Knowledge Graphs llevaron más tiempo del planeado en un principio. La principal causa fue el desconocimiento sobre los modelos y el gran tiempo que conllevaba el cálculo de cada modelo (de 3 a 12 horas dependiendo de su complejidad). Por otra parte, la arquitectura desarrollada, a pesar de haber programado cosas muy sencillas o básicas, al tener que integrar todas las partes entre sí, ha llevado también bastante tiempo. Se entiende que, en un grupo de trabajo multidisciplinar, el mismo trabajo se hubiese hecho en mucho menos tiempo y se hubiese obtenido un resultado mucho más completo. En total, Knowledge Graphs se estudiaron en 2 meses y la arquitectura costó otros 2 meses. Al principio, se pretendía emplear 1 mes en Knowledge Graphs y 3 meses en el resto, llegando a obtener resultados más pulidos e incluso llegar a desplegar en alguna nube comercial actual como Amazon Web Services o Google Cloud usando Kubernetes.

En cuanto a la dificultad, tanto KGs como la arquitectura tienen una alta barrera de entrada. Se necesitan conocimientos en muchos campos para su desarrollo, desde ML, estadística, bases de datos orientadas a grafos, tecnologías orientadas a contenedores... Sin embargo, todas estas tecnologías son las más usadas en entornos empresariales o son el futuro de estos, por lo que no hay arrepentimiento alguno por elegir este TFM. Se ha disfrutado el aprendizaje de estas tecnologías.

Para acabar, al realizar este proyecto, se han comprobado de primera mano las muchas posibilidades que ofrecen estas tecnologías o las que pueden llegar a ofrecer. No sorprende que hoy en día, se ofrezcan multitud de empleos bien remunerados buscando capacidades para desarrollar estos proyectos, por ejemplo, “FullStack developers”, ingenieros de datos...

6.2 Objetivos Cumplidos

Se ha conseguido desarrollar un servicio completo haciendo uso de una arquitectura propia de entornos de Big Data. Durante este desarrollo, se han fortalecido e incrementado conocimientos en: La tecnología de despliegue basada en contenedores, Kubernetes, lenguajes funcionales como Scala y nuevos “frameworks” de JavaScript como VueJS.

Por último, se ha tenido un primer contacto con tecnologías o modelos relacionados con Knowledge Graphs. Aunque el dataset empleado no haya sido el óptimo al cual aplicar esta tecnología, se han aportado posibles soluciones o mejoras para adaptar los datos. Como se ha comentado anteriormente, el uso de datos no categóricos o numéricos dificulta el cálculo de

modelos, por ello, se propuso el muestreo y categorización o generalización de los datos, perdiendo con ello un poco de información.

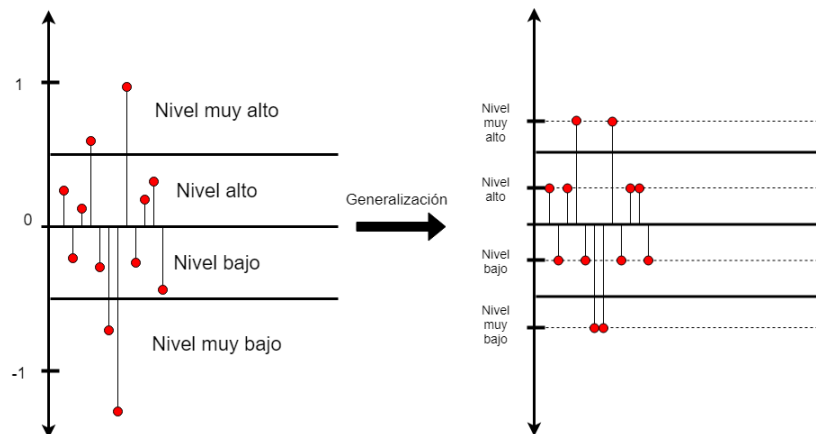


Figura 43: Ejemplo de generalización y categorización de datos

6.3 Líneas Futuras

A pesar de dar por finalizado el proyecto, se dejan muchas tareas a realizar para mejorarlo. A continuación, se citan algunas divididas por tecnologías.

6.3.1 Knowledge Graphs

- Implementar más algoritmos para calcular el “Embedding Model” y su evaluación, ya que solo se usa ComplEx.
- Entender e implementar más librerías [86] para usar esta tecnología.
- Implementar la solución descrita de generalización de los valores numéricos para hacer más eficiente el modelo.

6.3.2 VueJS y Express

- Implementar la seguridad al completo de este componente: Comunicaciones seguras, middlewares en la API desarrollada para evitar el uso por agentes externos al sistema, implementar algún tipo de autenticación, prevenir la inyección de código a través de los inputs...
- Mejorar la interfaz de usuario y adaptar el formulario de ingreso de datos a la futura solución con generalización de estos. (Cambiar inputs normales por desplegables).
- Implementar una solución para dispositivos móviles.

6.3.3 Kafka y Zookeeper

- Realizar una configuración orientada a clúster de máquinas Zookeeper y Kafka para un despliegue totalmente escalable.
- Cifrar las comunicaciones con Kafka e implementar algún tipo de autenticación para poder solicitar la información de Kafka.

6.3.4 Spark

- Configurar una pareja de máquinas Spark que actúen como maestro y esclavo para poder exprimir la principal ventaja de esta tecnología, la paralelización de cálculos o procesados.
- El modelo de ML usado actualmente no ha sido ajustado ni adaptado a los datos. Es un modelo con los hiperparámetros básicos. Habría que optimizar estos hiperparámetros para obtener una precisión mayor en la clasificación.

6.3.5 Neo4J

- Desarrollar con K8s y Docker un clúster de máquinas Neo4J para aumentar la disponibilidad de este componente.
- Implementar cifrado de conexiones y un sistema de autenticación.

6.3.6 Kubernetes

- Adaptar los ficheros desarrollados para un posible despliegue del servicio en una nube pública como AWS o Google Cloud.

7 Bibliografía

- [1] V. Maini, «Medium - Machine Learning for humans,» 19 Agosto 2017. [En línea]. Available: <https://medium.com/machine-learning-for-humans/why-machine-learning-matters-6164faf1df12>.
- [2] J. McCarthy, M. L. M. N. R. y C. S. , «A PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE,» 31 Agosto 1955. [En línea]. Available: <https://web.archive.org/web/20080930164306/http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html>.
- [3] «Microservices Architecture,» [En línea]. Available: <https://microservices.io/>.
- [4] «Arthur Samuel - Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Arthur_Samuel.
- [5] «Isaac Asimov - Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Isaac_Asimov.
- [6] «Three Laws of Robotics - Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Three_Laws_of_Robotics.
- [7] «Golden years of Artificial Intelligence - Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/History_of_artificial_intelligence#The_golden_years_1956%E2%80%931974.
- [8] «AI Winter - Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/AI_winter.
- [9] L. Adley, «How Machine Learning is Improving Business Intelligence,» [En línea]. Available: <https://insidebigdata.com/2019/02/16/how-machine-learning-is-improving-business-intelligence/>.
- [10] D. McCreary, «Knowledge Graphs: The Third Era of Computing - Medium,» [En línea]. Available: <https://medium.com/@dmccreary/knowledge-graphs-the-third-era-of-computing-a8106f343450>.
- [11] F. Tong, «Graph Embedding for Deep Learning,» [En línea]. Available: <https://towardsdatascience.com/overview-of-deep-learning-on-graph-embeddings->

4305c10ad4a4.

- [12] «Ampligraph - Python Library docs,» [En línea]. Available: <https://docs.ampligraph.org/en/1.3.1/index.html>.
- [13] «VueJS WebPage,» [En línea]. Available: <https://vuejs.org/>.
- [14] «AngularJS WebPage,» [En línea]. Available: <https://angularjs.org/>.
- [15] E. Elliott, «Top JavaScript Frameworks and Topics to Learn in 2020 and the New Decade,» [En línea]. Available: <https://medium.com/javascript-scene/top-javascript-frameworks-and-topics-to-learn-in-2020-and-the-new-decade-ced6e9d812f9>.
- [16] «ExpressJS WebPage,» [En línea]. Available: <https://expressjs.com/es/>.
- [17] «NodeJS WebPage,» [En línea]. Available: <https://nodejs.org/es/>.
- [18] «Apache Kafka WebPage,» [En línea]. Available: <https://kafka.apache.org/>.
- [19] «Apache Zookeeper WebPage,» [En línea]. Available: <https://zookeeper.apache.org/>.
- [20] «Apache Software Foundation WebPage,» [En línea]. Available: <https://www.apache.org/>.
- [21] V. Raut, «Kafka hands-on Guide to using publish-subscribe based messaging system (PART I) - Medium,» [En línea]. Available: <https://medium.com/@vrushaliraut1234/kafka-hands-on-guide-to-using-publish-subscribe-based-messaging-system-part-i-cda9eada3b06>.
- [22] «Apache Spark WebPage,» [En línea]. Available: <https://spark.apache.org/>.
- [23] «Apache Spark-Streaming WebPage,» [En línea]. Available: <https://spark.apache.org/streaming/>.
- [24] «The Scala Programming Language WebPage,» [En línea]. Available: <https://www.scala-lang.org/>.
- [25] P. Hudak, «Conception, evolution, and application of functional programming languages.,» [En línea]. Available: <http://www.dbnet.ece.ntua.gr/~adamo/languages/books/p359-hudak.pdf>.
- [26] «Imperative Programming - Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Imperative_programming.
- [27] «Spark Python API Docs,» [En línea]. Available:

<https://spark.apache.org/docs/latest/api/python/index.html>.

- [28] T. Yiu, «Understanding Random Forest,» [En línea]. Available: <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>.
- [29] «Neo4J WebPage,» [En línea]. Available: <https://neo4j.com/>.
- [30] «Bases de Datos NoSQL - AWS,» [En línea]. Available: <https://aws.amazon.com/es/nosql/>.
- [31] S. S. Nazrul, «CAP Theorem and Distributed Database Management Systems,» [En línea]. Available: <https://towardsdatascience.com/cap-theorem-and-distributed-database-management-systems-5c2be977950e>.
- [32] «Docker WebPage,» [En línea]. Available: <https://www.docker.com/>.
- [33] «Overview of Docker Compose,» [En línea]. Available: <https://docs.docker.com/compose/>.
- [34] «¿Qué es Kubernetes? - Kubernetes WebPage,» [En línea]. Available: <https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/>.
- [35] «UML WebSite,» [En línea]. Available: <https://www.uml.org/>.
- [36] «Estándar ECSS-E-ST-40C,» [En línea]. Available: <https://standards.globalspec.com/std/1266612/E-ST-40C>.
- [37] «Reglamento General de Protección de Datos,» [En línea]. Available: <https://www.boe.es/doue/2016/119/L00001-00088.pdf>.
- [38] «Ficha técnica Sensor ECG,» [En línea]. Available: http://www.biosignalsplux.com/datasheets/ECG_Sensor_Datasheet.pdf.
- [39] «Ficha Técnica Sensor EDA,» [En línea]. Available: http://www.biosignalsplux.com/datasheets/EDA_Sensor_Datasheet.pdf.
- [40] «Ficha Técnica Sensor EMG,» [En línea]. Available: http://www.biosignalsplux.com/datasheets/EMG_Sensor_Datasheet.pdf.
- [41] «Ficha Técnica Sensor de Temperatura,» [En línea]. Available: http://www.biosignalsplux.com/datasheets/TMP_Sensor_Datasheet.pdf.
- [42] «Ficha Técnica Sensor de Respiración,» [En línea]. Available: http://www.biosignalsplux.com/datasheets/PZT_Sensor_Datasheet.pdf.

- [43] «TFM_KnowledgeGraphs - GitHub Jaimefc,» [En línea]. Available: https://github.com/Jaimefc/TFM_KnowledgeGraphs.
- [44] «WESAD Dataset,» [En línea]. Available: <http://archive.ics.uci.edu/ml/datasets/WESAD+%28Wearable+Stress+and+Affect+Detection%29#>.
- [45] A. R. R. D. C. M. a. K. V. L. Philip Schmidt, «Introducing WESAD, a Multimodal Dataset for Wearable Stress and Affect Detection,» [En línea]. Available: <https://dl.acm.org/doi/10.1145/3242969.3242985>.
- [46] «Jupyter Project WebSite,» [En línea]. Available: <https://jupyter.org/>.
- [47] «Python Programming Language WebSite,» [En línea]. Available: <https://www.python.org/>.
- [48] «Clustering and Classification using Knowledge Graph Embeddings - Ampligraph Tutorial,» [En línea]. Available: <https://docs.ampligraph.org/en/1.3.1/tutorials/ClusteringAndClassificationWithEmbeddings.html>.
- [49] A. S. a. P. T. Chandrahas, «Towards Understanding the Geometry of Knowledge Graph Embeddings,» [En línea]. Available: <https://www.aclweb.org/anthology/P18-1012.pdf>.
- [50] «AmpliGraph API WebSite,» [En línea]. Available: <https://docs.ampligraph.org/en/1.3.1/api.html>.
- [51] «Embedding Models Performance - AmpliGraph,» [En línea]. Available: <https://docs.ampligraph.org/en/1.3.1/experiments.html>.
- [52] «Tensorflow WebSite,» [En línea]. Available: <https://www.tensorflow.org/>.
- [53] «Tensorflow GPU Guide,» [En línea]. Available: <https://www.tensorflow.org/guide/gpu>.
- [54] M. Taifi, «MRR vs MAP vs NDCG: Rank-Aware Evaluation Metrics And When To Use Them,» [En línea]. Available: <https://medium.com/swlh/rank-aware-recsys-evaluation-metrics-5191bba16832>.
- [55] «hits_at_n_score Function - AmpliGraph API,» [En línea]. Available: https://docs.ampligraph.org/en/1.3.1/generated/ampligraph.evaluation.hits_at_n_score.html#ampligraph.evaluation.hits_at_n_score.

- [56] «XGBClassifier - Python xgboost API,» [En línea]. Available: https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.XGBClassifier.
- [57] «Scikit-Learn WebSite,» [En línea]. Available: <https://scikit-learn.org/stable/>.
- [58] «Anonymisation: Code of practice,» [En línea]. Available: <https://ico.org.uk/media/1061/anonymisation-code.pdf>.
- [59] «Awesome Knowledge Graph - Github Papers,» [En línea]. Available: <https://github.com/shaoxiongji/awesome-knowledge-graph>.
- [60] «Apache Kafka QuickStart,» [En línea]. Available: <https://kafka.apache.org/quickstart>.
- [61] P. Gupta, «Decision Trees in Machine Learning,» [En línea]. Available: <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052>.
- [62] «IntelliJ IDEA - JetBrains WebSite,» [En línea]. Available: <https://www.jetbrains.com/es-es/idea/>.
- [63] «Run Neo4J with Docker,» [En línea]. Available: <https://neo4j.com/developer/docker-run-neo4j/>.
- [64] «Bolt Protocol WebSite,» [En línea]. Available: <https://boltprotocol.org/>.
- [65] «jaimedfc/tfmweb Docker Image,» [En línea]. Available: <https://hub.docker.com/r/jaimedfc/tfmweb>.
- [66] «jaimedfc/tfmspark Docker Image,» [En línea]. Available: <https://hub.docker.com/r/jaimedfc/tfmspark>.
- [67] «OpenJDK WebSite,» [En línea]. Available: <https://openjdk.java.net/>.
- [68] «Wurstmeister Zookeeper Docker Image,» [En línea]. Available: <https://github.com/wurstmeister/zookeeper-docker>.
- [69] «Wurstmeister Kafka Docker Image,» [En línea]. Available: <https://github.com/wurstmeister/kafka-docker>.
- [70] «Bitnami Neo4J Docker Image,» [En línea]. Available: <https://hub.docker.com/r/bitnami/neo4j/>.
- [71] «Networking in docker-compose,» [En línea]. Available: <https://docs.docker.com/compose/networking/>.

- [72] «Environment variables in docker-compose,» [En línea]. Available: <https://docs.docker.com/compose/environment-variables/>.
- [73] «Kompose - GitHub,» [En línea]. Available: <https://github.com/kubernetes/kompose>.
- [74] «Deployments - Kubernetes,» [En línea]. Available: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>.
- [75] «Pods - Kubernetes,» [En línea]. Available: <https://kubernetes.io/docs/concepts/workloads/pods/>.
- [76] «Services - Kubernetes,» [En línea]. Available: <https://kubernetes.io/docs/concepts/services-networking/service/>.
- [77] «Ingress Service - Kubernetes,» [En línea]. Available: <https://kubernetes.io/docs/concepts/services-networking/ingress/>.
- [78] «Minikube Docs,» [En línea]. Available: <https://minikube.sigs.k8s.io/docs/>.
- [79] «kubectl Overview,» [En línea]. Available: <https://kubernetes.io/docs/reference/kubectl/overview/>.
- [80] «Random Forest Classifier - SKLearn,» [En línea]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>.
- [81] «PIP WebSite,» [En línea]. Available: <https://pypi.org/project/pip/>.
- [82] «BabelJS WebSite,» [En línea]. Available: <https://babeljs.io/>.
- [83] «Webpack WebSite,» [En línea]. Available: <https://webpack.js.org/>.
- [84] «Bootstrap VueJS WebSite,» [En línea]. Available: <https://bootstrap-vue.org/>.
- [85] «Footer React Component - GitHub Jaimedfc,» [En línea]. Available: <https://github.com/Jaimedfc/TFG/blob/master/src/components/Footer.js>.
- [86] «Awesome Knowledge Graph Embedding Approaches,» [En línea]. Available: <https://gist.github.com/mommi84/07f7c044fa18aaaa7b5133230207d8d4>.
- [87] «Recital 26 - GDPR,» [En línea]. Available: <https://gdpr-info.eu/recitals/no-26/>.
- [88] «El perfil del Ingeniero de Telecomunicación - COIT,» [En línea]. Available: <https://www.coit.es/sites/default/files/informes/pdf/mapa-del-estudiante-de-ingenieria-de->

telecomunicacion_0.pdf#pdfjs.action=download.

- [89] «Precio - Kubernetes Engine Google Cloud,» [En línea]. Available:
<https://cloud.google.com/kubernetes-engine/pricing?hl=es>.

Anexo I. Manual de Usuario

A continuación, se ofrecen varias formas de desplegar el servicio desarrollado:

Ejecución en local.

Se necesita tener instaladas las siguientes herramientas:

- Git.
- NodeJS y npm.
- Zookeeper y Kafka (versión 2.11-2.4.0).
- sbt.
- Spark (versión 2.4.5-hadoop2.7).
- Docker.
- Python (versión 3 o superior).
- Dataset WESAD.
- PySpark

En primer lugar, se clona el repositorio con:

```
git clone https://github.com/Jaimedfc/TFM_KnowledgeGraphs && cd /TFM_KnowledgeGraphs
```

Es necesario crear una carpeta Data en la raíz del proyecto. Aquí se almacenarán los datos del proyecto.

```
mkdir Data
```

Se instalan las dependencias necesarias con pip y se ejecutan los siguientes scripts (adaptando las rutas del código) para preprocesar y limpiar el dataset WESAD y reducirlo:

```
cd Python
pip3 install -r requirements.txt
python3 dataClean.py
python3 DFreduction.py
```

En /Data, se encontrarán varios ficheros, el más interesante es reducedDataset001.csv. Es el fichero que usará PySpark para generar el modelo al ejecutar:

```
python3 PySparkModel.py
```

Una vez calculado el modelo, se puede comenzar a correr el servicio. Primero, se lanza Zookeeper con:

```
/kafka_2.11-2.4.0/bin/zookeeper-server-start.sh /kafka_2.11-2.4.0/config/zookeeper.properties
```

Se lanza Kafka y se crea el tópico “myTopic” (cada comando en una consola distinta):

```
/kafka_2.11-2.4.0/bin/kafka-server-start.sh /kafka_2.11-2.4.0/config/server.properties  
/kafka_2.11-2.4.0/bin/kafka-topics.sh --create --bootstrap-server  
localhost:9092 --replication-factor 1 --partitions 1 --topic myTopic
```

Se usa Docker para lanzar un contenedor con Neo4j, mapeando los puertos necesarios:

```
docker run -p7474:7474 -p7687:7687 -v $HOME/neo4j/data:/pruebas --  
env=NEO4J_AUTH=neo4j/test --rm neo4j:latest
```

Se compila el proyecto de Spark usando sbt, modificando el fichero Classification.scala:

- En la línea 22 hay que poner la ruta absoluta al proyecto.
- Comentar la línea 27 y descomentar la 28.

```
cd /TFM_KnowledgeGraphs/Scala/realtimeclassification && sbt package
```

Se lanza el archivo compilado usando spark-submit, indicando la dirección de neo4j y las credenciales de acceso:

```
./spark-submit --packages org.apache.spark:spark-sql-kafka-0-  
10_2.11:2.4.5,neo4j-contrib:neo4j-spark-connector:2.4.1-M1 --class  
Classification --conf spark.neo4j.bolt.url=bolt://localhost:7687 --  
conf spark.neo4j.bolt.user=neo4j --conf spark.neo4j.bolt.password=test  
/TFM_KnowledgeGraphs/Scala/realtimeclassification/target/scala-  
2.11/realtimeclassification_2.11-1.0.jar
```

Por último, se lanza el frontend. Para ello, se instala las dependencias necesarias, se compila la parte de VueJS y se lanza Express:

```
cd /TFM_KnowledgeGraphs/Web/frontend && npm install  
npm run build  
cd .. && npm install  
npm run devbabel
```

Ya se puede uno conectar a <http://localhost:3000> y probar el servicio.

Lanzar servicio usando Docker-compose

Se necesita instalar la herramienta docker-compose y ejecutar lo siguiente estando en /TFM_KnowledgeGraphs:

```
docker-compose up
```

Ya se puede uno conectar a <http://localhost:80> y probar el servicio.

Para parar el servicio y limpiar entorno:

```
Ctrl + C  
docker-compose down
```

Lanzar el servicio usando Kubernetes

Se necesitan las siguientes herramientas:

- minikube.
- kubectl.

Una vez instaladas, se comienza lanzando el nodo donde se desplegará todo el servicio con:

```
minikube start --vm-driver=virtualbox --force --disk-size=15g --memory='4g' --cpus=4
```

Nota: Se pueden modificar los recursos usados por el nodo para adaptarlo a las capacidades de cada máquina.

Lanzado el nodo, se puede usar un dashboard o panel de control que proporciona minikube para administrar el entorno:

```
minikube dashboard
```

Ahora, solo resta indicar los ficheros yaml que contienen todo el proyecto, modificando los ficheros deployment. Ejecutar:

```
minikube ip
```

Con ese comando se obtiene la dirección IP del nodo donde se ejecuta el servicio. A continuación, es necesario cambiar todas las direcciones IP de los ficheros *-deployment.yaml por esta nueva IP.

Ahora, desplegamos los microservicios con:

```
cd k8s
kubectl create -f kafka-service.yaml,neo-service.yaml,spark-
service.yaml,web-service.yaml,zookeeper-service.yaml,kafka-
deployment.yaml,neo-deployment.yaml,spark-deployment.yaml,web-
deployment.yaml,zookeeper-deployment.yaml,web-
ingress.yaml,highPriorityClass.yaml,mediumPriorityClass.yaml,lowPrior-
ityClass.yaml
```

Tras unos minutos, en el dashboard, en el apartado que controla los Ingress, aparecerá el enlace de acceso al proyecto.

Posibles problemas con Kubernetes

Es posible que, si el dashboard no funciona o tarda demasiado el obtener un enlace a través del Ingress, se necesite habilitar los “addons” pertinentes en minikube:

```
minikube addons enable dashboard
minikube addons enable ingress
```


Knowledge Graphs scripts

Una vez se ha usado el servicio, se pueden hacer cálculos con el grafo que se ha generado en Neo4J. Todo el código necesario se puede encontrar en /TFM_KnowledgeGraphs/Python.

Nota: No se recomienda lanzar los scripts sin antes revisarlos. Hay que adaptarlos a cada situación.

Anexo II. Aplicabilidad del RGPD

En este anexo, se estudiará si los datos usados en el proyecto están o no sujetos al RGPD. Este apartado se apoyará en el considerando número 26 del RGPD [87], en el cual se estipula que, cualquier dato anonimizado, aquel mediante el cual no se puede identificar a una persona, queda excluido de la aplicación del RGPD.

Según se ha explicado en el apartado de líneas futuras, los datos usados en el servicio final deberán estar categorizados. Para hacer un uso óptimo de los algoritmos usados en los modelos de KGs, se deben usar un conjunto finito de datos. Al usar datos de tipo numérico, cumplir esta condición se hace imposible, por ello, se propone generalizar estos datos. Se pasaría de un conjunto infinito de posibles valores, a uno definido por rangos. Si un dato cae dentro del rango, se categorizaría con el nombre del rango.

Por otra parte, los datos introducidos en el sistema deben ir acompañados de un identificador para poder recuperarlos y consultarlos individualmente. Este identificador, en un servicio ya en funcionamiento y desplegado, debe ser asignado por el sistema. Si se quiere evitar aplicar el RGPD, es necesario que este anonimizado.

Siguiendo la explicación anterior, un conjunto de datos ejemplo se vería así:

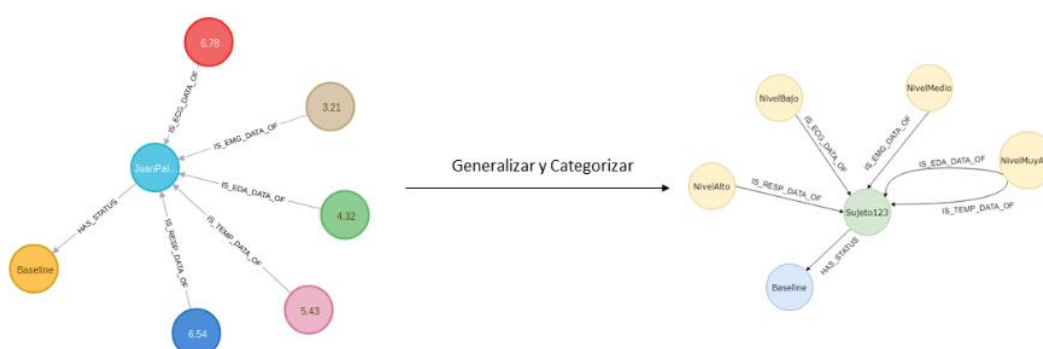


Figura 44: Ejemplo de anonimización de un conjunto de datos

Aplicando los tratamientos anteriores, se consigue el anonimato total de los datos usados. Así se consigue evitar tener que aplicar el RGPD e invertir gran cantidad de recursos y tiempo en adaptar el servicio al reglamento.

A pesar de no aplicarse el RGPD, se seguirán los principios descritos en este para el tratamiento de datos:

- **Limitación y minimización del tratamiento de los datos:** Para ambos tipos de interesados, los participantes en el entrenamiento del modelo del microservicio de Spark y los propios usuarios del servicio, solo se emplearían datos relativos a la biométrica anonimizados, omitiendo posibles datos que puedan usarse para identificar a los sujetos. (Nombre, edad, dirección...). En referente a los datos involucrados en el entrenamiento de los modelos, estos serían eliminados tras las pruebas que confirmen el funcionamiento de estos. En cuanto a los datos de los segundos interesados, se guardarían en una base de datos orientada a grafos para su consulta por estos y otro personal como responsables del sistema para observar estadísticas o cálculo de nuevos modelos a partir de estos nuevos datos.
- **Legitimación del tratamiento:** En un caso real, estos datos serían utilizados bajo el expreso consentimiento de los sujetos. Por otra parte, al orientarse este TFM a un ecosistema empresarial, no se contempla el usar sujetos menores de edad que puedan causar un problema al obtener su consentimiento.
- **Exactitud de los datos:** Se usarán datos extraídos de los sensores descritos en anteriores secciones y su precisión recae en las especificaciones técnicas de estos.
- **Transparencia:** Toda información referente al tratamiento realizado sobre los datos estaría disponible en todo momento para los interesados.
- **Lealtad con el interesado:** Se comunicaría el proceso que recorrerán los datos de los interesados antes de que estos expresen su consentimiento. No se contemplaría ningún tipo de cambio en el proceso tras obtener los datos. En caso de necesitar cambiar los procesos, al no poseer ninguna información de contacto de los interesados para informales de este hecho, se tendría que empezar desde cero, eliminando todos los datos, es decir, se obtendrían nuevos datos para entrenar, comunicando estos nuevos procesos a los nuevos interesados. También se tendría una actitud proactiva a la hora de informar en todo momento a los interesados.
- **Seguridad:** Al estar los datos anonimizados, una brecha de seguridad no sería perjudicial para los interesados. Aun así, se garantizaría el cumplimiento de las distintas dimensiones de la seguridad: Integridad, disponibilidad, confidencialidad, trazabilidad y accesibilidad.



Figura 45: Principios de la Privacidad según el RGPD

Anexo III. Aspectos Éticos, Económicos, Sociales y Ambientales

Introducción

El servicio desarrollado se posiciona en un contexto médico. Tiene como objetivo ayudar a empresas con el cuidado y bienestar diario de sus empleados. A su vez, se ayuda a consolidar una tecnología pionera en el ámbito de ML, los Knowledge Graphs, para una futura aplicación en otros campos dentro de la empresa, como en la Inteligencia de Negocio.

La implementación de este servicio o la puesta en uso de este genera ciertos impactos de índole éticos, económicos, sociales y ambientales, los cuales se comentan más adelante.

Descripción de impactos relevantes relacionados con el proyecto

A continuación, se describirán los principales impactos que genera el servicio divididos por los distintos aspectos en los que se sitúan:

- **Aspectos éticos:** Un mal uso del servicio puede incurrir fácilmente en un incumplimiento del derecho de la privacidad estipulado en el RGPD. Por otra parte, con este servicio se consigue acercar a la gente las tecnologías de ML y así cambiar los prejuicios que tienen muchos sobre la IA.
- **Aspectos económicos:** Este servicio o proyecto no tiene ningún impacto en este campo.
- **Aspectos sociales:** Con un correcto uso del servicio por parte de las empresas, la vida laboral de muchos empleados puede verse muy beneficiada y mejorada. Con un seguimiento del estado de los empleados, se pueden tomar medidas rápidamente ante situaciones de estrés críticas y perjudiciales, tanto para el empleado como para la empresa. Por otra parte, si el propósito es la toma de decisiones sobre empleados

dependiendo de su estado psíquico, a parte de ser un incumplimiento del RGPD, puede llevar a un impacto perjudicial para la sociedad.

- **Aspectos ambientales:** El servicio sería operado con tecnología de computación en la nube. Estas tecnologías hacen un uso masivo de recursos eléctricos y componentes electrónicos que, aunque cada vez son más eficientes y limpios, siguen siendo un punto crítico en el impacto ambiental que tienen algunas empresas como Amazon o Google.

Análisis detallado de alguno de los principales impactos

Los principales impactos son los relacionados con la privacidad de los clientes del servicio y los relacionados con la mejora de su vida laboral. Acerca de la privacidad de los usuarios, se ha realizado un análisis de aplicabilidad del RGPD (Anexo II). En este se explica cómo se debe hacer un correcto uso del servicio sin afectar a la privacidad de los usuarios del servicio.

Con relación al otro impacto principal, cabe comentar que mejorar la vida laboral de los empleados de una empresa puede llegar a suponer un impacto altamente positivo para el futuro de las empresas. Gracias a un buen estado de los empleados, las empresas pueden ver incrementada su eficacia.

Conclusiones

Para finalizar con este anexo, cabe comentar que, en general, si se hace un correcto uso del servicio, sin dañar la privacidad de los usuarios, puede llegar a tener impactos muy positivos, en especial, en el ámbito laboral. Sin embargo, en zonas no reguladas por el RGPD, donde las limitaciones al uso de datos personales son más laxas, se pueden llegar a generar impactos muy negativos a los usuarios del servicio.

Anexo IV. Presupuesto Económico

Para el desarrollo del presupuesto económico, se han tenido en cuenta las siguientes suposiciones:

- El sueldo medio de un ingeniero de telecomunicaciones es de 52.711€ al año según el COIT [88], repartidos en 14 pagas al año trabajando 20 días al mes (4 semanas), 8 horas diarias resulta en un sueldo de 23,53 €/hora.
- La duración del desarrollo ha sido de 360 horas, equivalente a 12 créditos, valiendo 30 horas cada crédito.
- Al usar software de código abierto, este no repercute en las cuentas.
- El despliegue se realiza usando el servicio de Kubernetes de Google Cloud, el cual supone un coste de 0,10 USD (0,089 €) cada hora por clúster [89]. Un año de este servicio equivale a 779,64 €.

Tras estas consideraciones, se obtiene el siguiente presupuesto:

COSTE DE MANO DE OBRA (coste directo)		Horas	Precio/hora	Total	
		360	23,53 €	8.470,8 €	
COSTE DE RECURSOS MATERIALES (coste directo)		Precio de compra	Uso en meses	Amortización (en años)	Total
Ordenador personal (Software incluido)		1.000,00 €	6	5	100,00 €
Servicio K8s de Google Cloud por 1 año		779,64 €			779,64 €
COSTE TOTAL DE RECURSOS MATERIALES					879,64 €
GASTOS GENERALES (costes indirectos)		15%	sobre CD		1.402,57 €
BENEFICIO INDUSTRIAL		6%	sobre CD+CI		645,18 €
SUBTOTAL PRESUPUESTO					11.398,19 €
IVA APLICABLE				21%	2.393,62 €
TOTAL PRESUPUESTO					13.791,81 €

Se obtiene un presupuesto de 13.791,81 €. Sin embargo, este valor supone una cota inferior, ya que, el presupuesto de un desarrollo profesional involucraría a más gente, contratación de servicios en la nube a medida y un coste de operación y mantenimiento, entre otras cosas.