Deusto

Universidad de Deusto

ESTUDIO DE USO DE NUEVOS ALGORITMOS DE EDGE COMPUTING PARA LA GENERACIÓN EFICIENTE DE GEMELOS DIGITALES EN ENTORNOS PRODUCTIVOS INDUSTRIALES

Tesis doctoral presentada por

NEREA GÓMEZ LARRAKOETXEA

Bilbao, 9 de octubre de 2023

ESTUDIO DE USO DE NUEVOS ALGORITMOS DE EDGE COMPUTING PARA LA GENERACIÓN EFICIENTE DE GEMELOS DIGITALES EN ENTORNOS PRODUCTIVOS INDUSTRIALES

Tesis doctoral presentada por
NEREA GÓMEZ LARRAKOETXEA

Dentro del programa de
INGENIERÍA PARA LA SOCIEDAD DE LA INFORMACIÓN Y DESARROLLO SOSTENIBLE

Facultad de Ingeniería

Dirigida por el Dr. Borja Sanz Urquijo
y por el Dr. Jon Garcia Barruetabeña

La doctoranda | El director | El director

Bilbao, 9 de octubre de 2023

Agradecimientos

Es momento de expresar mi profundo agradecimiento a todas las personas que han sido fundamentales en el logro de esta importante etapa en mi vida. Quiero comenzar agradeciendo a mis directores de tesis, Borja y Jon, cuya guía y apoyo han sido indispensables para la realización de este trabajo. Su sabiduría y consejos tanto en el ámbito profesional como personal han hecho que este camino sea gratificante y enriquecedor, no podía haber elegido mejor. También quiero agradecer a Iker, quien, aunque no haya sido director, ha desempeñado un papel fundamental en mi crecimiento como investigadora.

Mi reconocimiento se extiende a la empresa colaboradora, cuya participación ha permitido llevar a cabo esta tesis en un sector que me apasiona. Agradezco su colaboración y la oportunidad de trabajar en un entorno tan inspirador.

La Universidad de Deusto ha sido mi segunda casa durante estos últimos años, y quiero expresar mi agradecimiento a todo el personal de la institución por el trato cercano y cariñoso que he recibido en todo momento. Quiero hacer una mención especial a mis compañeros de Deustotech, quienes han contribuido a que mi experiencia en la universidad sea aún más enriquecedora. A los bedeles, por el servicio inmejorable que ofrecen día a día. Además, quiero agradecer especialmente a Agustín Zubillaga por darme la oportunidad de formar parte de Deustotech Computing tras una primera entrevista en la cafetería. Fue un paso fundamental en mi trayectoria. Gracias, Agus.

Ha sido un orgullo trabajar con los diferentes equipos con los que he colaborado, y quiero expresar mi gratitud a todos ellos.

Por último, pero no menos importante, quiero agradecer a mi familia y amigos por su apoyo constante a lo largo de estos años. Han sido un pilar fundamental en mi camino hacia este logro. A todos ellos, mi más sincero agradecimiento.

Resumen

Esta tesis doctoral se ha desarrollado en el contexto de la aparición del Internet de las cosas (IoT) y la digitalización de las empresas, situación que ha llevado a un aumento significativo en la cantidad de datos generados y recopilados. Debido a los inconvenientes, como latencia o sobrecarga de red, producidos al procesar los datos en la nube surge el concepto *Edge Computing* como una solución para procesar datos lo más cerca posible de la fuente de generación. La tesis se ha desarrollado en colaboración con una planta de producción de automóviles y el objetivo de la investigación ha sido mejorar el desarrollo de gemelos digitales con el fin de aumentar la calidad y eficiencia de la planta. Para ello, esta tesis doctoral se enfoca en investigar si la combinación de *Edge Computing* con técnicas de inteligencia artificial puede mejorar el desarrollo de gemelos digitales en los procesos de una planta de producción. Es por eso que los objetivos definidos para conseguirlo se basan en analizar el comportamiento de los recientes dispositivos que han salido al mercado de *Edge Computing* al procesar diferentes algoritmos de inteligencia artificial, validar la eficacia de modelos *Edge* en comparación con modelos más completos, y mejorar la optimización del flujo de datos en este tipo de dispositivos a través de la compresión de datos. Estos objetivos se han definido con el objetivo de reducir los tiempos de cálculo y predicción de los procesos en la industria, permitiendo respuestas más rápidas y eficientes.

Nerea Gómez Larrakoetxea

Índice general

Agradecimientos **i**

Preface **iii**

Índice general **v**

1 Introducción **1**
1.1 Contextualización . . . 1
1.2 La Digitalización en la producción. ¿Qué efectos tiene? . . . 1
1.3 Situación industrial actual . . . 4
1.3.1 Contexto tesis doctoral . . . 5
1.3.2 *Edge Computing* . . . 7
1.3.3 *Cloud Computing* vs *Edge Computing* . . . 9
1.3.4 Ventajas de Edge Computing . . . 10

2 Estado del arte **13**
2.1 IoT y digitalización . . . 13
2.2 Edge Computing . . . 16
2.3 Optimización del flujo de datos . . . 18
2.3.1 Teorema de Bayes . . . 20
2.3.2 Clasificador de Naive Bayes . . . 20
2.3.3 *Autoencoder* . . . 23
2.4 Algoritmia . . . 24
2.4.1 *Machine Learning* o aprendizaje automático . . . 24
2.4.2 Algoritmos de árbol de decisión . . . 25
2.4.3 Algoritmos de agrupación . . . 43
2.4.4 Algoritmos de Redes Neuronales Artificiales . . . 49
2.5 Estudio crítico del estado del arte . . . 53
2.6 Hipótesis y Objetivos . . . 56

3 Análisis hardware **59**
3.1 Experimentación . . . 59
3.1.1 Pasos de la experimentación . . . 60
3.1.2 Random Forest . . . 63
3.1.3 Logistic Regression . . . 67
3.1.4 K Nearest Neighbor . . . 72
3.1.5 Neural Network . . . 76
3.1.6 Gaussian Naive Bayes . . . 80

3.2 Resultados obtenidos 84
3.2.1 Resultados obtenidos por algoritmo 85
3.2.2 Resultados obtenidos por tamaño de archivo 88
3.3 Conclusiones 90

4 Modelos Edge **95**
4.1 Experimentación 95
4.1.1 Conjunto de datos 96
4.1.2 Procesamiento de los datos 100
4.2 Experimentación conjunto de datos "Bosch" 103
4.3 Experimentación conjunto de datos de pintado 104
4.3.1 Resultados modelo Esmalte 108
4.3.2 Resultados modelo Apresto 110
4.3.3 Resultados modelo Apresto y Esmalte unificados 110
4.3.4 Comparativa resultados de precisión entre los modelos de Esmalte, Apresto y ambos modelos unificados 112
4.4 Conclusiones 112

5 Optimización del flujo de datos **115**
5.1 Experimentación 115
5.1.1 Reducción de instancias 118
5.1.2 Reducción de dimensionalidad 119
5.1.3 Pasos de la experimentación 130
5.1.4 Arquitectura 131
5.1.5 Validación 132
5.1.6 Resultados - Reducción de instancias 133
5.1.7 Resultados - Reducción de dimensionalidad 137
5.2 Conclusiones 138

6 Conclusiones **141**

Bibliografía **151**

Índice de figuras

1.1 Ejemplos de digitalización en empresas de fabricación 2
1.2 Diagrama del flujo de trabajo del proceso de Pintura 6
1.3 Diagrama del flujo de trabajo del proceso de Pintura con subprocesos. Fuente: Automotive Paints and Coatings [**streitberger2008automotive**] 7

2.1 Diagrama de flujo del algoritmo *Random Forest* 26
2.2 Diagrama de flujo de entrenamiento del algoritmo *Random Forest* 27
2.3 Estructura de una Red Neuronal 50

3.1 Gráfica tiempos de entrenamiento y test para Random Forest en Jetson Nano . 64
3.2 Gráfica tiempos de entrenamiento y test para Random Forest en GoogleCoral 65
3.3 Gráfica tiempos de entrenamiento de Jetson Nano y Google Coral con Random Forest 67
3.4 Gráfica tiempos de test de Jetson Nano y Google Coral con Random forest . . 68
3.5 Gráfica tiempos de entrenamiento y test para Logistic Regression en Jetson Nano 69
3.6 Gráfica tiempos de entrenamiento y test para Logistic Regression en Google Coral 70
3.7 Gráfica tiempos de entrenamiento de Jetson Nano y Google Coral con Logistic Regression 71
3.8 Gráfica tiempos de test de Jetson Nano y Google Coral con Logistic Regression 71
3.9 Gráfica tiempos de entrenamiento y test de Jetson Nano con K Nearest Neighbor 73
3.10 Gráfica tiempos de entrenamiento y test de Google Coral con K Nearest Neighbor 74
3.11 Gráfica tiempos de entrenamiento de Jetson Nano y Google Coral con K Nearest Neighbor 75
3.12 Gráfica tiempos de test de Jetson Nano y Google Coral con K Nearest Neighbor 76
3.13 Gráfica tiempos de entrenamiento y test de Jetson Nano con Neural Network 77
3.14 Gráfica tiempos de entrenamiento y test de Google Coral con Neural Network 78
3.15 Gráfica tiempos de entrenamiento de Jetson Nano y Google Coral con Neural Network 79
3.16 Gráfica tiempos de test de Jetson Nano y Google Coral con Neural Network . 80
3.17 Gráfica tiempos de train y test de Jetson Nano con Gaussian Naive Bayes . . . 81
3.18 Gráfica tiempos de train y test de Google Coral con Gaussian Naive Bayes . . 82
3.19 Gráfica tiempos de train de Jetson Nano y Google Coral con Gaussian Naive Bayes 83
3.20 Gráfica tiempos de test de Jetson Nano y Google Coral con Gaussian Naive Bayes 84
3.21 Gráfico circular Hardware óptimo de entrenamiento y test 85
3.22 Gráfico circular Hardware óptimo en Random Forest 85
3.23 Gráfico circular Hardware óptimo en Logistic Regression 86

3.24 Gráfico circular Hardware óptimo en K Nearest Neighbor 86
3.25 Gráfico circular Hardware óptimo en Neural Network 87
3.26 Gráfico circular Hardware óptimo en Gaussian Naive Bayes 87
3.27 Gráfico de barras para hardware más óptimo para el entrenamiento en función del tamaño de archivo 89
3.28 Gráfico de barras que muestra el hardware más rápido para el test en función del tamaño de archivo 90

4.1 Conjunto de datos "Bosch" dividido en líneas de producción 97
4.2 Conjunto de datos de Pintado de vehículos dividido en los subprocesos de esmalte y apresto 100
4.3 Estructura del conjunto de datos original 101
4.4 Estructura del conjunto de datos pivotado 101
4.5 Estructura del conjunto de datos transformado 102

5.1 Reducción del conjunto de datos con el objetivo de obtener resultados similares 116
5.2 Diagrama de flujo de compresión bayesiana 119
5.3 Estructura del autoencoder 120
5.4 Configuración del Autoencoder para la reducción de 24 a 12 121
5.5 Configuración del Autoencoder para la reducción de 24 a 6 122
5.6 Configuración del Autoencoder para la reducción de 24 a 2 123
5.7 Código del Autoencoder para la reducción de 24 columnas a 2 125
5.8 Código del Autoencoder para la reducción de 24 columnas a 6 126
5.9 Código del Autoencoder para la reducción de 24 columnas a 2 128
5.10 Diagrama de compresión bayesiana: paso a paso 130
5.11 Arquitectura de nodos Edge 132

Índice de cuadros

3.1 Tiempos de rendimiento para Random Forest 63
3.2 Tiempos de rendimiento para Logistic Regression 67
3.3 Tiempos de rendimiento para K Nearest Neighbor 72
3.4 Tiempos de rendimiento para Neural Network 76
3.5 Tiempos de rendimiento para Gaussian Naive Bayes 80
3.6 Modelos de comportamiento para entrenamiento Jetson Nano 92
3.7 Modelos de comportamiento para test Jetson Nano 92
3.8 Modelos de comportamiento para entrenamiento Google Coral 93
3.9 Modelos de comportamiento para test Google Coral 93

4.1 Comparativa de eficacia de los modelos creados con los subconjuntos de datos y el modelo creado con el conjunto de datos original. 103
4.2 Comparativa de eficacia de los modelos creados con los subconjuntos de datos y el modelo creado con el conjunto de datos original. 103
4.3 Resultados de precisión del modelo de Esmalte con diferentes configuraciones 109
4.4 Resultados de precisión del modelo de Esmalte con diferentes configuraciones 109
4.5 Resultados de precisión del modelo de Apresto con diferentes configuraciones 110
4.6 Resultados de precisión del modelo de Apresto y Esmalte unificados con diferentes configuraciones 111
4.7 Resultados de precisión del modelo de Apresto y Esmalte unificados con diferentes configuraciones 111
4.8 Comparativa de resultados de precisión entre los modelos de Esmalte, Apresto y ambos modelos unificados 112

5.1 Resultados con 25 % de compresión 136
5.2 Resultados con 50 % de compresión 136
5.3 Resultados con 75 % de compresión 137
5.4 Resultados con 50 % y 75 % de compresión 138
5.5 Resultados con 90 % de compresión 138

Introducción 1

1.1 Contextualización . 1
1.2 La Digitalización en la producción. ¿Qué efectos tiene? 1
1.3 Situación industrial actual 4
1.3.1 Contexto tesis doctoral 5
1.3.2 *Edge Computing* . . 7
1.3.3 *Cloud Computing* vs *Edge Computing* . . 9
1.3.4 Ventajas de Edge Computing 10

1.1. Contextualización

La Cuarta Revolución Industrial, también conocida como *Industria 4.0*, está transformando el funcionamiento de las empresas, buscando que la empresa se convierta en una organización inteligente [1]. Los principales objetivos de la *Industria 4.0* son lograr un mayor nivel de eficiencia operativa y de productividad, así como un mayor nivel de automatización [2]. Otra novedad de la *Industria 4.0* es que gracias al "Internet de las Cosas" (IoT) existe la posibilidad de disponer de toda la información necesaria en tiempo real, independientemente del proceso o de su ubicación en la fábrica, generando así respuestas más rápidas a las demandas del mercado [3]. A pesar de que el término IoT empezó a utilizarse hace más de dos decadas, poco se sabía entonces de la importancia que tendría unos años después. En la actualidad, la palabra IoT se utiliza como un término genérico para referirse a la conexión entre Internet y el mundo físico a través del despliegue generalizado de dispositivos distribuidos espacialmente con capacidades de identificación, detección o actuación integradas [4]. El IoT es una de las tecnologías habilitadoras que han permitido llevar a cabo la digitalización y automatización de los procesos.

La Industria 4.0 o Cuarta Revolución Industrial busca transformar la forma de trabajo en las empresas con la creación de organizaciones inteligentes para mejorar la eficiencia y productividad. El IoT es una tecnología clave que permite tener acceso a información en tiempo real y responder rápidamente a las demandas del mercado. La Industria 4.0 se basa en la digitalización y automatización de los procesos.

1.2. La Digitalización en la producción. ¿Qué efectos tiene?

La digitalización en la producción industrial es una de las tareas prioritarias a las que se enfrentan las empresas y la sociedad hoy en día [5]. De manera general, la digitalización puede verse como un aumento de la generación, el análisis y el uso de datos para aumentar la eficiencia interna de la empresa [6]. Los efectos de la digitalización para las empresas se pueden clasificar en desarrollo de productos y fabricación más eficientes, productos y servicios más sofisticados y cadenas de valor más integradas (ver Figura 1.1).

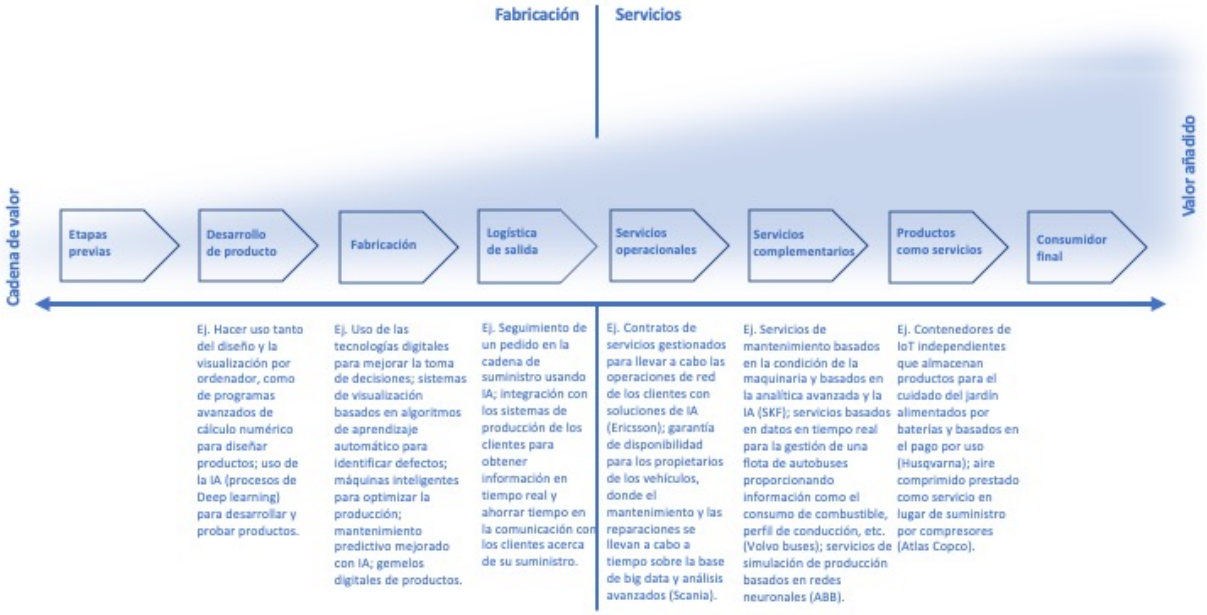

Figura 1.1: Ejemplos de digitalización en relación con las diferentes actividades de la cadena de valor de las empresas de fabricación [6].

La digitalización en la producción industrial es una prioridad para las empresas y la sociedad. Se trata de aumentar la generación, análisis y uso de datos para mejorar la eficiencia interna de la empresa. Los efectos de la digitalización incluyen una fabricación y desarrollo de productos más eficientes, productos y servicios más sofisticados y cadenas de valor más integradas, lo que añade valor a las diferentes actividades en la cadena de valor.

Como se puede observar en la Figura 1.1, la digitalización acarrea una serie de beneficios (añade valor) en las diferentes actividades de la cadena de valor [6]:

- **Desarrollo de producto.** La digitalización tiene el potencial de hacer más eficiente el desarrollo de productos de las empresas de fabricación. La digitalización del desarrollo de productos reduce la necesidad de utilizar prototipos físicos. Las empresas utilizan programas informáticos con interfaces tanto internas como externas, lo que requiere la realización de pruebas mediante modelos en entornos asistidos por ordenador para comprobar el rendimiento y la funcionalidad del producto. Estos procedimientos de prueba pueden realizarse rápidamente al tiempo que permiten verificar distintos resultados. Las pruebas son una parte importante de las actividades de desarrollo de productos de las empresas. Volvo Cars [7], al igual que muchas otras empresas de automoción, simula y realiza diferentes pruebas de motores de forma digital antes de tomar la decisión de llevar un motor específico a la producción. En el pasado, había que construir un motor y probarlo a diferentes temperaturas y altitudes y en distintas condiciones ambientales. En función de los resultados de estas pruebas,

se adaptaban los motores, lo que suponía altos costes y largos plazos antes de la producción a gran escala. Esto demuestra la importancia de la digitalización en un sector como es el de la automoción, ya que muchas empresas de fabricación, la digitalización ha puesto fin a estas formas de trabajo.

- **Fabricación.** La mayor parte de las empresas de fabricación están llevando a cabo una amplia labor para hacer más competitiva su producción mediante la digitalización, pero sobre todo, las empresas del sector automóvil (este sector supone el 11.7 % del PIB de España y emplea a 75.000 personas de forma directa y 250.000 de forma indirecta [8]). Pueden aumentar el rendimiento y la calidad, minimizar el número de averías y paradas de la cadena de producción haciendo que el proceso de fabricación sea más inteligente mediante el uso de tecnologías digitales. Sin embargo, la amplitud y la profundidad de estos esfuerzos difieren sustancialmente entre las empresas. En algunos casos, estas actividades se limitan a vincular diferentes fuentes de datos para mejorar la toma de decisiones y lograr la integración entre las máquinas distribuidas por la fábrica. Las empresas determinan las razones de las perturbaciones de la producción que dan lugar a una reducción de la eficacia operativa, detectando las anomalías y registrando las averías para identificar las causas fundamentales. Las empresas líderes también realizan muchas otras actividades que permite la digitalización. Por ejemplo, los sistemas de visualización por ordenador que utilizan algoritmos de aprendizaje automático para identificar defectos en el proceso de fabricación reducen la necesidad de sacar productos o materiales de la línea de producción y comprobarlos manualmente [6]. Además, aunque los productos finales utilizados por los clientes no están conectados a la nube, las empresas de fabricación suelen emplear "gemelos digitales" o modelos de un producto construido que reflejan el proceso de fabricación completo y pueden permitir mejoras basadas en el rendimiento del producto en un entorno real. El mantenimiento predictivo mejorado por la IA está tomando el relevo del mantenimiento preventivo. El uso de acelerómetros, sensores digitales y algoritmos de software avanzados permite informar de las condiciones de las máquinas

La digitalización está aumentando la eficiencia y mejorando la calidad en la industria de fabricación. Se utilizan tecnologías digitales para mejorar el desarrollo de productos, la producción y la toma de decisiones en la fábrica. Esto incluye la utilización de programas informáticos para verificar el rendimiento y la funcionalidad de los productos antes de la producción, la integración de diferentes fuentes de datos, y sistemas de visualización que utilizan aprendizaje automático para detectar defectos en el proceso de fabricación.

en tiempo real. Esto se traduce en un ahorro de tiempo y recursos, menos paradas costosas de la producción y menores costes de mantenimiento e inspección.

Además, los "gemelos digitales" permiten mejoras basadas en el rendimiento del producto en el mundo real. El mantenimiento predictivo basado en la inteligencia artificial también está mejorando el mantenimiento y reduciendo los costes.

- **Productos y servicios avanzados.** Las tecnologías digitales están integradas no sólo en la cadena de valor, sino también en los propios productos. Cada vez más empresas manufactureras integran tecnologías en productos ya establecidos para hacerlos más "inteligentes". Los datos generados por los productos en uso pueden aumentar el rendimiento o dar lugar a nuevas funcionalidades. Por ello, la integración de tecnologías digitales permite a la empresa y a sus clientes recoger datos valiosos sobre el producto y las aplicaciones que dependen de él. Estos datos permiten que los productos sean supervisados, optimizados, controlados y, a veces [6], funcionen de forma autónoma.
- **Cadenas de valor integradas.** La digitalización permite una mayor integración de las cadenas de valor, lo que aumenta la eficiencia de las distintas funciones de la empresa, reduce los plazos de entrega y permite un mejor control de las operaciones. Las empresas de fabricación comparten datos no solo dentro de la empresa, sino también a través de los límites de la misma con proveedores y clientes. Este cambio permite a las empresas y a sus proveedores tener un mejor control de los materiales y componentes y reducir los inventarios. La digitalización también facilita la trazabilidad a lo largo de las actividades de la cadena de valor de la empresa, por ejemplo, mediante el seguimiento de la posición en la cadena de suministro de un pedido específico de un cliente, utilizando el aprendizaje automático, o vinculando los productos finales a los datos del proceso de fabricación y los materiales de entrada utilizados [6].

1.3. Situación industrial actual

Debido a la aparición del IoT, y como consecuencia, la digitalización de las empresas, se ha producido un enorme aumento del volumen de datos que estas manejan gracias a la facilidad que ofrece a la hora de recoger el dato y almacenarlo. De hecho, solo entre 2016 y 2018, se produjo un aumento del 569 % [9]. Actualmente, la mayoría de los datos recogidos

a través de los dispositivos IoT se procesan en la nube [10], como solución para descentralizar la computación. Sin embargo, los recursos para hacer frente a esta computación en la nube se encuentran en grandes centros de datos alejados de la mayoría de los usuarios finales. En consecuencia, existe una importante latencia de comunicación entre ambos [10]. Además, el gran volumen de datos que se intercambia entre los usuarios y la nube añade carga a la red, provocando un grave cuello de botella en el ancho de banda de la red y la latencia de las comunicaciones.

En consecuencia, el auge del IoT hace que la computación en la nube ya no sea una solución óptima a la hora de satisfacer las diversas necesidades de la sociedad inteligente actual para el procesamiento de datos [11], además de no ser la opción más adecuada [12], ya que cada vez hay más dispositivos que recogen y generan datos en el borde de la red, y no es solucionable desde el punto de vista físico, es decir, no se soluciona aumentando el ancho de banda. Por consiguiente se necesitan aplicaciones que se desplieguen en el borde de la misma para procesar la información lo más cerca del proceso posible. Para ello, se ha propuesto el concepto de *'Edge Computing'* [13].

La digitalización de las empresas ha llevado a un aumento del 569 % en el volumen de datos entre 2016 y 2018. Esto ha resultado en latencia de comunicación y cuellos de botella en el ancho de banda debido a la gran cantidad de datos intercambiados entre los usuarios y la nube. Por lo tanto, se necesitan aplicaciones en el borde de la red para procesar la información de manera más eficiente y cercana al proceso y evitar los problemas causados al ser procesados en la nube. Es por eso que se ha propuesto el concepto "*Edge Computing*".

1.3.1. Contexto tesis doctoral

Es en este contexto de *Edge Computing* es donde se desarrolla esta tesis doctoral que parte de la necesidad de generar conocimiento sobre el proceso a partir de los datos de proceso de un *OEM* [1]. Así, esta tesis se ha realizado en colaboración con una empresa de automoción cuya planta de producción tiene como objetivo la fabricación de 610 vehículos diarios. Esta es una de las mayores plantas de producción de vehículos a nivel mundial y produce vehículos para todos los países del mundo, por lo que existen muchas configuraciones posibles para cada vehículo (p.e. volante a la derecha/izquierda). La digitalización de esta planta de producción ha hecho que se recoja una cantidad elevada de variables por segundo durante el proceso de fabricación de los vehículos, y esto conlleva a la necesidad de ser capaces de procesar de manera eficiente esos datos para poder tomar decisiones durante el propio proceso de fabricación.

1: OEM: Original Equipment Manufacturer

Para gestionar los datos, se puede hacer uso tanto de la tecnología *Cloud Computing* como de la tecnología *Edge Computing*. Debido a que, como se ha mencionado anteriormente, con la tecnología *Cloud Computing* se genera cierta latencia entre la recogida del dato y la respuesta obtenida una vez procesado el dato [10]; y teniendo en cuenta que las plantas de producción demandan cada vez respuestas más rápidas [3], esta tesis se ha centrado en investigar la capacidad de gestionar los datos de este tipo de plantas en el borde de la red (*Edge Computing*). El objetivo industrial de esta investigación es generar gemelos digitales de pequeñas partes del proceso que permitan dotar de inteligencia al proceso para alcanzar los estándares exigidos en cuanto a calidad y eficiencia.

Esta tesis doctoral se desarrolla en el contexto de *Edge Computing* y trata de generar conocimiento sobre el proceso de producción de vehículos en una empresa de automoción que tiene como objetivo fabricar 610 vehículos diarios. La planta de producción es una de las más grandes del mundo y recopila una gran cantidad de datos durante el proceso de fabricación. Para gestionar estos datos, se está investigando la capacidad de hacerlo en el borde de la red (*Edge Computing*) en lugar de en la nube, debido a la latencia generada en la nube y a la necesidad de tener respuestas más rápidas.

Esta tesis doctoral se ha desarrollado en la nave de pintura de vehículos, ya que, la pintura es la fase más delicada de la fabricación de un vehículo. El proceso de pintar los vehículos es un cuello de botella en la fabricación de muchas plantas de automoción debido a la complejidad del proceso de pintura de carrocerías, las ajustadas labores de gestión de la producción y los rigurosos requisitos de calidad. Si al final del proceso de pintura una carrocería no pasa el control de calidad, las carrocerías son retrabajadas y aumentan significativamente los costes de fabricación.

A continuación, en la figura 1.2 se puede ver de manera general cuál es flujo de trabajo que tiene cada carrocería desde que entra en el proceso de ser pintada hasta que acaba siendo pintada.

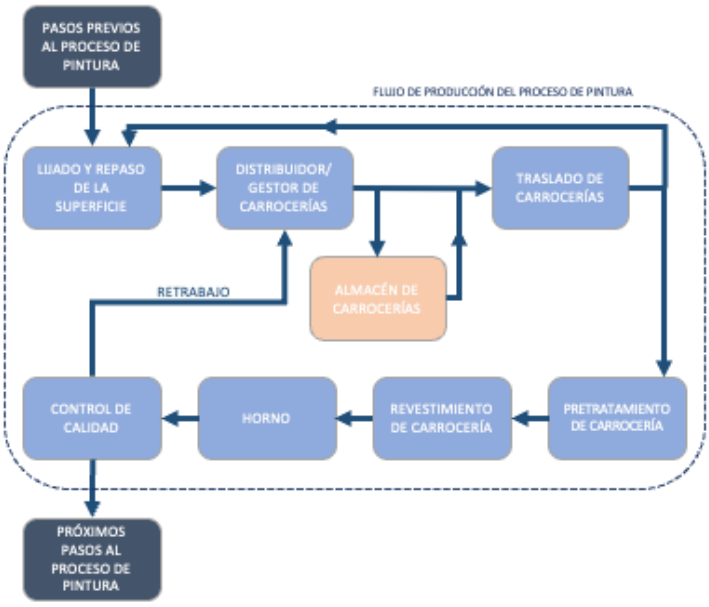

Figura 1.2: Diagrama del flujo de trabajo del proceso de Pintura en el contexto de desarrollo de esta tesis.

La carrocería sigue la línea de trabajo mostrada y va avanzando por cada uno de los procesos. Por otro lado, en la Fig. 1.3 se muestra el diagrama completo con los subprocesos correspondientes. Como se puede apreciar, el proceso de pintura no es un proceso sencillo, tiene cierta complejidad y una carrocería puede llegar a retrabajarse hasta 3 veces.

La tesis se enfoca en la fase de pintura de los vehículos, ya que es la más delicada y es un cuello de botella en la producción debido a la complejidad del proceso y los requisitos de calidad. El objetivo es generar gemelos digitales que permitan dotar de inteligencia al proceso para mejorar la calidad y eficiencia.

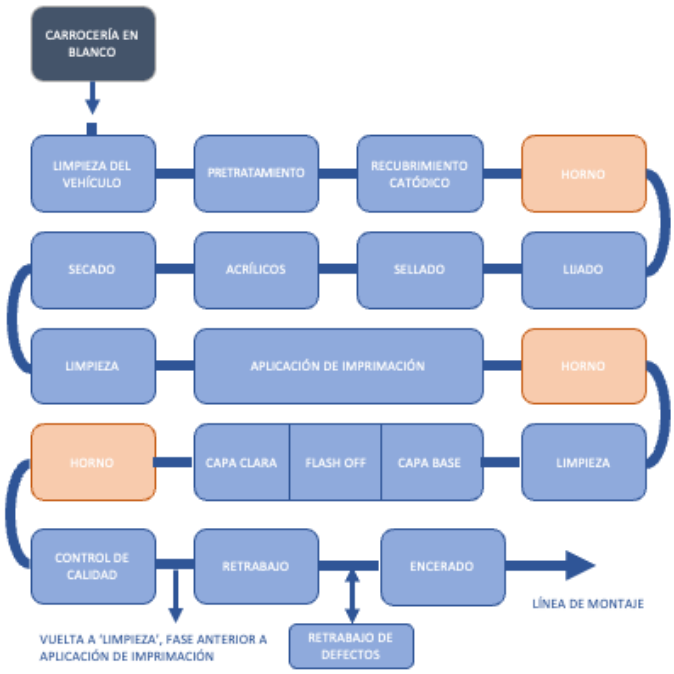

Figura 1.3: Diagrama del flujo de trabajo del proceso de Pintura en el contexto de desarrollo de esta tesis.

Al estar todos los subprocesos digitalizados, por cada carrocería que es pintada existe una gran cantidad de variables que son recogidas. Al ser el proceso de pintado el más complejo del proceso de fabricación, se ha decidido que la investigación se centre en este proceso de manera que luego pueda ser escalado y aplicado a otros procesos de fabricación.

1.3.2. *Edge Computing*

Edge Computing es un nuevo paradigma de computación para realizar cálculos en el borde de la red [11]. Los investigadores tienen diferentes definiciones de *Edge Computing*. Shi et al. [14] [15] [16] introdujeron el concepto de la siguiente manera: "El *Edge Computing* es un nuevo modo de computación en el borde de la red. Los datos del enlace descendente de la computación de borde representan el servicio en la nube, los datos del enlace ascendente representan el Internet de Todo (*Internet of Everything*), y el borde de la computación de borde se refiere a la computación y los recursos de red entre

En resumen, el *Edge Computing* es un nuevo enfoque de computación que consiste en proporcionar servicios y realizar cálculos en el borde de la red y generación de datos. La computación de borde coloca los recursos informáticos en el borde de la red, cerca del usuario y de la fuente de los datos, lo que reduce la latencia y mejora la eficiencia en términos de ancho de banda. Este enfoque se ha convertido en un área de investigación popular y se espera que satisfaga las necesidades críticas de la industria de la tecnología de la información en cuanto a conexión, tiempo real, optimización de datos, inteligencia, seguridad y privacidad.

la fuente de datos y la ruta del centro de *Cloud Computing*". Satyanarayanan [17], profesor de la Universidad Carnegie Mellon de Estados Unidos, describe el *Edge Computing* como: "El *Edge Computing* es un nuevo modelo informático que despliega recursos informáticos y de almacenamiento en el borde de la red, más cerca de los dispositivos móviles o sensores" [18]. Zha et al. [19] propusieron sobre la base de las dos definiciones anteriores que "*Edge Computing* es un nuevo modelo de computación cuyo objetivo es unificar los recursos que están cerca del usuario en la distancia geográfica o la distancia de la red para proporcionar computación, almacenamiento y red para el servicio de aplicaciones". La alianza industrial acerca del *Edge Computing* de China, define el *Edge Computing* como: "una plataforma abierta que integra capacidades básicas como la red, la computación, almacenamiento, aplicaciones, y proporciona servicios inteligentes de borde cercanos para satisfacer los requisitos clave de agilidad de la industria en materia de conexión, negocio en tiempo real, optimización de datos, aplicación, inteligencia, seguridad y privacidad, todo ello cerca del borde de la red o el origen de los datos" [20]. En otras palabras, el *Edge Computing* consiste en proporcionar servicios y realizar cálculos en el borde de la red y la generación de datos. La computación de borde consiste en migrar la red de la nube, la computación, las capacidades de almacenamiento y los recursos de la red, y proporcionar servicios inteligentes en el borde para satisfacer las necesidades críticas de la industria de la tecnología de la información en la vinculación ágil, el negocio en tiempo real, la optimización de datos, la inteligencia de aplicaciones, la seguridad y la privacidad, y cumple con los requisitos de baja latencia y gran ancho de banda en la red. La computación de borde se ha convertido en un punto caliente de investigación en la actualidad [21] [22] [23] [24].

En el paradigma del *Edge computing*, a diferencia del *Cloud Computing*, los recursos informáticos se sitúan en el borde de la red [10], estando más cerca del usuario y más cerca de la fuente de los datos [11]. Este enfoque reduce la latencia de la comunicación entre el dispositivo y los centros de procesamiento de datos en la nube existentes en el paradigma 'Cloud Computing'. Además, los datos pueden ser procesados y analizados cerca de los dispositivos finales, lo que reduce la demanda de ancho de banda en los enlaces de red.

1.3.3. *Cloud Computing* vs *Edge Computing*

Hasta ahora, el *Cloud Computing* tradicional se encargaba de transferir todos los datos al centro de computación en la nube a través de la red, y de esta manera resuelvía los problemas de computación y almacenamiento de forma centralizada [11]. En la literatura [25], se describe la historia del desarrollo del *Cloud Computing*. Durante la conferencia *Search Engine Strategies* (Miami) en agosto de 2006, el director general de *Google* propuso por primera vez el concepto *Cloud Computing*.

Con el desarrollo de motores de búsqueda representados por Google, la computación en nube empieza a mostrar una fuerte vitalidad. En la actualidad, el *Cloud Computing* se ha desarrollado gradualmente. Es una plataforma de servicios de red muy potente que incluye computación distribuida, equilibrio de carga, computación paralela, almacenamiento en red, virtualización y otras tecnologías. Sin embargo, hoy en día, con la popularización y el desarrollo del Internet de las Cosas en la vida de las personas, el número de dispositivos conectados está aumentando gradualmente, y se genera una gran cantidad de datos. El ancho de banda de la red del *Cloud Computing* no ha podido satisfacer las necesidades de rendimiento de los sistemas *real-time* [26]. Por lo tanto, el modelo *Cloud Computing* presenta graves carencias en cuanto a la carga, el tiempo real [27] [28] [29], el ancho de banda, consumo de energía y seguridad y privacidad de los datos [30].

La irrupción del *Edge Computing* busca solventar parte de los problemas *Cloud Computing*. Ambos deben convivir juntos y complementarse y desarrollarse de manera coordinada contribuyendo a la transformación digital de la industria. En el contexto del Internet de las cosas, si toda la cantidad de datos generados por los dispositivos conectados se transmiten a la nube, el *Cloud Computing* provocará una gran carga. En este momento, el *Edge Computing* es necesario para compartir la presión de la nube y encargarse de las tareas dentro de su alcance del borde. Cuando hay un problema en *Edge Computing*, los datos en la nube no se pierden. En algunos servicios de Internet, algunos datos deben ser devueltos a la nube para su procesamiento después de ser procesados mediante *Edge Computing*, como por ejemplo para hacer un análisis profundo de datos y compartirlos, por lo que requiere la

El *Cloud Computing* es una plataforma de servicios de red que se ha desarrollado gradualmente y que incluye tecnologías como la computación distribuida, equilibrio de carga, computación paralela, almacenamiento en red y virtualización. Sin embargo, con la popularización del Internet de las Cosas (IoT), el aumento de dispositivos conectados y la cantidad de datos generados ha puesto en cuestión el rendimiento en tiempo real y la seguridad y privacidad de los datos. El *Edge Computing* busca solucionar parte de estos problemas al compartir la carga con la nube, procesando datos localmente y en tiempo real, pero requiere la cooperación con el *Cloud Computing* para realizar un análisis profundo y compartir los datos. Ambos, *Cloud* y *Edge Computing*, deben trabajar de manera coordinada en muchos sectores y ámbitos para transformar la industria digital.

cooperación del *Cloud Computing* y *Edge Computing* [11]. En la actualidad, el desarrollo coordinado de *Edge Computing* y *Cloud Computing* se ha aplicado en muchos sectores y ámbitos como la fabricación inteligente [30], la energía [31], seguridad [32] y protección de privacidad [33]. Por ejemplo, en la fabricación inteligente en la producción industrial, el rol de la nube es controlar la totalidad. Por el contrario, en los nodos de *Edge Computing*, es necesario tener la función de detección en tiempo real para poder resolver los problemas que surjan a tiempo [11]. El *Edge Computing* es una extensión del *Cloud Computing* que tiene sus propias características y comparte algunas con el *Cloud Computing*. Las principales características del *Cloud Computing* es que puede capturar el conjunto entero, puede procesar una gran cantidad de datos, realizar un análisis profundo y desempeña un papel importante en el procesamiento de datos de manera asíncrona, ayudando a la toma de decisiones empresariales [11]. El *Edge Computing* se centra en local, y puede desempeñar un mejor papel en el análisis inteligente a pequeña escala satisfaciendo las necesidades en tiempo real.

1.3.4. Ventajas de Edge Computing

El *Edge Computing* es un enfoque de procesamiento de datos que se basa en procesar los datos en dispositivos locales en lugar de enviarlos a la nube. Esto permite una respuesta más rápida y procesamiento en tiempo real. Además, reduce el costo de transmisión, el ancho de banda y el consumo de energía.

El *Edge Computing* muestra unas ventajas evidentes gracias a que almacena y procesa los datos en los dispositivos sin subirlos a la nube [11]: Procesamiento y análisis de datos en tiempo real: el rápido crecimiento de la captura de datos y la consecuente presión del ancho de banda de la red son desventajas del *Cloud Computing* [34]. En comparación con el *Cloud Computing*, el *Edge Computing* tiene como ventaja principal la velocidad de respuesta. El *Edge Computing* está más cerca de la fuente de datos. El almacenamiento de datos y las tareas de computación pueden ser llevadas a cabo en el nodo de *Edge Computing*, lo que reduce el proceso de transmisión de datos. Destaca por la proximidad a los usuarios y proporciona mejores servicios inteligentes, mejorando el rendimiento de la transmisión, garantizando el procesamiento en tiempo real y reduciendo el *delay* [11]. En definitiva, el *Edge Computing* proporciona una variedad de servicios de respuesta rápida, siendo una carácterística muy importante en ciertos sectores como son la fabricación y las líneas de producción.

El *Cloud Computing* requiere que todos los datos sean subidos

a la nube para su posterior procesamiento unificado, ya que es un método de procesamiento centralizado. Durante ese proceso, hay riesgo de pérdida de datos y no se puede garantizar la seguridad y privacidad de los mismos. Sin embargo, otra de las ventajas es que como en el *Edge Computing* cada nodo es responsable solo de sus tareas, el procesamiento de los datos se hace en local sin subir a la nube, por lo que se evitan los riesgos de transmisión de datos en red garantizando la seguridad de los datos. Además, si un nodo es atacado, solo afecta a los datos locales de ese nodo y no a todos [11]. Otra de las claras ventajas del *Edge Computing* es el bajo consumo de energía y el bajo coste de ancho de banda, puesto que al no tener que subir datos a la nube, la carga del ancho de banda se reduce. El *Edge Computing* es de pequeña escala, y en términos de producción, las empresas pueden reducir el coste del procesamiento de los datos en local. Por lo tanto, el *Edge Computing* reduce la cantidad de datos transmitidos en la red, reduce el coste de transmisión y la presión del ancho de banda de la red, reduce el consumo de energía de los equipos locales y mejora la eficiencia de computación [11].

Por otro lado, el *Cloud Computing* es un enfoque de procesamiento centralizado que requiere la subida de datos a la nube, lo que puede tener riesgos de pérdida de datos y seguridad.

A continuación, se ha realizado un exhausto análisis del estado del arte acerca del IoT en los procesos de digitalización donde se describe como a lo largo de los años se ha ido utilizando esta tecnología, y también se han considerado las diferentes investigaciones que se han hecho con diferentes enfoques sobre *Edge Computing*. Además, también se ha recogido en el estado del arte las diferentes estrategias desarrolladas a lo largo de los años sobre la optimización en el flujo de datos en modelos de inteligencia artificial, ya que se quiere ahondar en este ámbito con el objetivo de optimizar los modelos de inteligencia artificial en dispositivos *Edge Computing*.

Estado del arte 2

2.1. IoT y digitalización

Fue en 1999 cuando se introdujo por primera vez el término que hoy en día se utiliza frecuentemente en el paradigma de la Industria 4.0: "Internet de las Cosas" (IoT). Fue en una conferencia magistral ante una multinacional americana cuando Kevin Ashton con el objetivo de llamar la atención de los ejecutivos sobre la gestión automatizada de la cadena de suministro mencionó la palabra [35].

Una década más tarde, en 2010, el término IoT comenzó también a asociarse con las infraestructuras de identificación por radiofrecuencia (RFID) en red en el MIT(Massachussetts Institute of Technology) [36].

La tendencia hacia un desarrollo de productos más eficaces basado en la digitalización lleva más de una década entre las empresas de fabricación. Un cambio notable que se ha producido desde 2017 entre los principales fabricantes es la adopción de la inteligencia artificial (IA) en su proceso de desarrollo de productos. Por ejemplo, la IA está permitiendo realizar pruebas de software más eficaces, más rápidas y más baratas. La IA también se está utilizando para diseñar productos. Por ejemplo, la empresa de telecomunicaciones Ericsson utiliza la IA para poner a punto los complejos filtros utilizados en su red 5G, que antes requerían que una persona recibiera una formación especializada de seis meses [6].

2.1 IoT y digitalización 13
2.2 Edge Computing . 16
2.3 Optimización del flujo de datos . . . 18
2.3.1 Teorema de Bayes . 20
2.3.2 Clasificador de Naive Bayes 20
2.3.3 *Autoencoder* 23
2.4 Algoritmia 24
2.4.1 *Machine Learning* o aprendizaje automático 24
2.4.2 Algoritmos de árbol de decisión 25
2.4.3 Algoritmos de agrupación 43
2.4.4 Algoritmos de Redes Neuronales Artificiales 49
2.5 Estudio crítico del estado del arte . . . 53
2.6 Hipótesis y Objetivos 56

Uno de los avances más recientes es la construcción de fábricas completamente digitales. Por ejemplo, el fabricante de vehículos comerciales Scania ha invertido 220 millones de dólares en una nueva fábrica de cabinas de camiones pesados para su mercado europeo [6]. Las únicas actividades manuales que quedan están relacionadas con el suministro de materiales para la producción y la vigilancia y el mantenimiento de la producción. La capacidad de producción de Scania se ha duplicado y el número de averías y paradas de producción se ha reducido.

El término "Internet de las cosas" (IoT) fue introducido por primera vez en 1999 por Kevin Ashton. Desde entonces, la tendencia en la fabricación ha sido hacia un desarrollo más eficiente basado en la digitalización y la adopción de la inteligencia artificial (IA). Empresas como Scania y SKF han invertido en fábricas digitales para mejorar la competitividad y la calidad de sus productos.

Otra empresa, SKF, el mayor fabricante de rodamientos del mundo, también tiene fábricas digitales. Basándose en la experiencia de una fábrica de pruebas que construyó en Suecia, SKF está construyendo otra nueva fábrica en Alemania. Al igual que Scania, SKF ha mejorado su planificación de la producción porque el proceso de fabricación puede modificarse sin causar interrupciones ni errores. Los estudios muestran que las empresas están invirtiendo en fábricas digitales porque creen que las harán más competitivas y producirán bienes de mejor calidad con menores costes de producción. Otra consecuencia importante de las fábricas digitales que aún no se ha materializado es que, al necesitarse menos personas para la producción, la actividad puede posicionarse estratégicamente con una menor consideración de los costes salariales. Podría ser beneficioso trasladar las instalaciones de producción a algún lugar con una mano de obra digitalmente competente, especialmente para empresas como SKF con instalaciones de producción en 103 lugares del mundo. Aunque la primera fábrica digital de SKF era una fábrica de pruebas,

la empresa redujo el número de empleados que trabajaban en y con la fábrica en un 77 %; y todavía se espera una reducción de personal aún mayor en relación con las futuras fábricas digitales [6].

ABB, líder mundial en tecnologías de energía y automatización, como sus robots están conectados a la nube gracias al "Internet de las Cosas", le permite realizar un mantenimiento predictivo a través de inteligencia artificial y crear simulaciones de producción [6].

Las empresas de automoción están conectando sus vehículos a la nube para habilitar nuevos servicios complementarios. Por ejemplo, Scania tiene 500.000 vehículos conectados a la nube en la carretera. Permiten optimizar las rutas, realizar un mantenimiento predictivo y dotar de personal más eficiente a los camiones [6].

La automatización de las fábricas está llevando a una disminución de la necesidad de mano de obra y una mayor eficiencia en la producción. Empresas como ABB utilizan IoT para realizar mantenimiento predictivo y crear simulaciones de producción. Las empresas de automoción también están conectando sus vehículos a la nube para habilitar nuevos servicios.

Muchas empresas recurren también a la digitalización y a los productos conectados para permitir la conexión directa con los usuarios finales. Husqvarna, uno de los principales fabricantes de productos eléctricos para exteriores, comercializaba y vendía antes sus productos principalmente a través de distribuidores. Sin embargo, la digitalización ha desencadenado una nueva estrategia y ha permitido el desarrollo de servicios innovadores basados en la venta directa a los usuarios finales [6].

Siemens en Amberg, también es un claro ejemplo de la digitalización de la producción [37]. Utilizan un software de simulación para crear un gemelo digital de cada modelo de producción para previsualizar cómo sería en el mundo real incluso antes de fabricarlo. La existencia de un gemelo digital para la fase de diseño de la producción reduce el

tiempo entre esa fase y la de comercialización de la producción en un 30 %. Esto permite hacer un uso más eficiente de la energía, reduciendo los costes en un 40 %.

Otra entidad de reconocido prestigio que se beneficia del uso de estas nuevas tecnologías en la Industria 4.0 y que es capaz de aprovechar mejor sus recursos y responder de forma más rápida y flexible a las necesidades de cada cliente es Adidas [38]. El principal resultado ha sido un aumento de la producción equivalente a 30 millones de pares de zapatillas. Hay que señalar que no sólo las organizaciones se benefician de ello, sino también el cliente, por supuesto: Un par de zapatillas puede diseñarse y probarse con simulación 3D y realidad aumentada, y fabricarse inmediatamente después con robots autónomos. El consumidor puede comprobar su apariencia antes de que se fabrique el par [39].

2.2. Edge Computing

A lo largo de los años, se han propuesto diferentes enfoques para lograr funcionalidades del *Edge Computing*, como el *Edge Accelerated web Browsing* (EAB) propuesto en 2015 y el algoritmo *Successive Convex Approximation* (SCA) propuesto en 2015.

A lo largo de los años se han investigado diferentes enfoques que pretendían alcanzar algunas de las funcionalidades del 'Edge Computing' actual.

En 2015, Takahashi *et al.* [40] propusieron el prototipo *Edge Accelerated web Browsing* (EAB), cuyo objetivo era lograr la ejecución de aplicaciones web con una técnica de descarga diferente. Esta técnica se basaba en el procesamiento del contenido web en el servidor EAB mientras los flujos de audio y vídeo se decodifican en función de las capacidades del hardware del cliente.

En 2015, Alfo, Sardellitti *et al.* [41] propusieron el algoritmo Successive Convex Approximation (SCA) con el objetivo de optimizar la descarga computacional a través de múltiples puntos de acceso de radio densamente desplegados. Utilizaron un sistema de comunicación multicelular MIMO [42] para atender las solicitudes de los usuarios de realizar sus tareas computacionales en el servidor central.

Debido al creciente uso de aplicaciones basadas en Deep Learning [43], se han llevado a cabo varias investigaciones para utilizar técnicas de Deep Learning en placas embebidas. En 2020, Ullah *et al.* [44] realizaron un experimento de comparación y concluyó con que la placa Jetson Nano [45] desarrollada por Nvidia tiene un mejor rendimiento que Raspberry Pi.

Además, debido al creciente uso de aplicaciones de *Deep Learning*, se han llevado a cabo varias investigaciones en torno a la utilización de técnicas de *Deep Learning* en placas embebidas. Las investigaciones han comparado diferentes dispositivos acelerados por GPU, como Nvidia Jetson Nano, Nvidia Jetson TX2 y Nvidia Jetson Xavier, concluyendo que la Jetson TX2 es la más eficiente en términos de eficiencia energética y velocidad de ejecución.

En 2019, Peng *et al.* [46] realizaron un estudio de investigación sobre los sistemas integrados de GPU de Nvidia: Jetson Xavier [47], Jetson TX2 [48] y Jetson Nano [45]. El resultado fue que Nvidia Jetson TX2 tiene la mejor eficiencia energética en comparación con los demás.

En 2020, Jo *et al.* [49] evaluaron los dispositivos 'Edge' acelerados por GPU y, para ello, realizaron un conjunto de pruebas de aprendizaje profundo en el dispositivo para medir su rendimiento. Llevaron a cabo la experimentación comparando Jetson TX2 [48] y Jetson Nano [45] a través de pruebas con Redes Convolucionales. Como conclusión, el hardware Jetson Nano muestra un rendimiento 2 veces menor.

En 2020, Basulto-Lantsova *et al.* [50], también realizaron una investigación basada en la comparativa

entre Nvidia Jetson TX2 y Nvidia Jetson Nano. Se llevó a cabo con seis imágenes de diferentes tamaños y dos variantes en cuanto al tamaño de la imagen de la plantilla. Se observó que la Jetson TX2 superaba a la Jetson Nano en velocidad de ejecución.

2.3. Optimización del flujo de datos

En el campo de la optimización del flujo de datos en modelos de inteligencia artificial, se han desarrollado varias estrategias para minimizar el número de instancias y disminuir la complejidad de la red. Uno de los primeros métodos utilizados fue la poda, y más tarde se introdujo el "*daño cerebral óptimo*".

En cuanto a lo ya desarrollado acerca de la optimización del flujo de datos en modelos de inteligencia artificial, a lo largo de los años se han desarrollado varias estrategias tanto para minimizar el número de instancias en los modelos como para disminuir la complejidad de la estructura de la red. Uno de los primeros métodos utilizados en ambos escenarios fue la poda. Ya en 1997, Wilson y R. Martínez presentaron tres algoritmos utilizados para reducir el número de instancias del conjunto de datos de entrenamiento [51] y por otro lado, LeCun, Denker y Solla [52] introdujeron el "*daño cerebral óptimo*", una nueva técnica dirigida a reducir el tamaño de la red de aprendizaje eliminando sistemáticamente los pesos. Además, también se han desarrollado algoritmos con técnicas bayesianas que utilizan la poda de pesos con un resultado satisfactorio [53].

En cualquier caso, al realizar la compresión bayesiana no se ha tenido en cuenta el formato de las matrices, por lo que la poda de pesos es ineficiente para la compresión. Para solucionar este problema, se utilizó el formato de columnas dispersas comprimidas (CSC) [54]. En este caso, utilizaron la compresión bayesiana para reducir las estructuras de la red, pero no los datos de entrada a la red.

En 2017, también se quiso afrontar el problema desde un enfoque bayesiano [55]. Introdujeron el uso de prioridades jerárquicas para podar nodos en lugar de pesos individuales y también determinaban la precisión óptima del punto fijo para codificar los pesos. Investigaciones anteriores en 2016 [56] y en 2015 [57] se habían centrado en el punto fijo óptimo y en la cuantificación de hash, respectivamente. En cuanto al punto fijo óptimo, implementaron un método con el objetivo de determinar el ancho de banda del punto fijo a través de las capas de la *DCN*. [1] Esta investigación concluyó que, la optimización del ancho de bits en las *DCN* presenta una reducción superior al 20 % del tamaño del modelo sin haber perdido precisión con el conjunto de datos *CIFAR-10* [58]. [2] En lo que a la cuantificación de *hash* respecta, Wenlin Chen *et al.* [57] utilizaron una transformada del coseno discreta (DCT) [3] para modificar los pesos de los filtros de la frecuencia. Para ello, emplearon una función *hash* de bajo coste para agrupar al azar los parámetros de frecuencia.

1: DCN: *Deep Convolutional Network*.

2: CIFAR-10: Conjunto de datos que consta de 60.000 imágenes a color de 32x32 divididas en 10 clases, 6.000 imágenes por clase. Hay 50.000 imágenes de entrenamiento y 10.000 para test.

3: DCT: *Discrete cosine transform*.

Por otro lado, en cuanto a la reducción de dimensiones a través de *autoencoders*, en 2016 Dilokthanakul et al. [59] tras estudiar los *autoencoders* variacionales para el clustering no supervisado a través de DGM [4], observaron un problema de sobrerregularización que ocurre en los autocodificadores variacionales normales. Para solucionarlo, propusieron la utilización de la información mínima con el objetivo de mejorar la agrupación no supervisada.

4: DGM: *Deep Generative Modeling*. Son redes neuronales con muchas capas ocultas entrenadas para aproximar distribuciones de probabilidad complicadas y de alta dimensión utilizando un gran número de muestras.

2.3.1. Teorema de Bayes

La teoría de Thomas Bayes [60] consiste en proporcionar un método para calcular la probabilidad o las distribuciones de probabilidad de un suceso, conociendo información de antemano sobre ese suceso. Supongamos que hay n estados posibles, etiquetados como $S_1, S_2, , S_n$. A priori, la probabilidad de que S_i sea el estado real es $P(S_i)$. Sea A un suceso que tiene una probabilidad $P(A|S_i)$ de ocurrir dado que S_i es el estado verdadero, entonces, la probabilidad absoluta de que ocurra el evento A a priori es:

También se han desarrollado algoritmos con técnicas bayesianas, pero la compresión bayesiana no tuvo en cuenta el formato de las matrices y la poda de pesos fue ineficiente. En 2017, se introdujo el uso de prioridades jerárquicas y la optimización del ancho de bits en las DCN, lo que resultó en una reducción del 20 % del tamaño del modelo sin pérdida de precisión. Además, se han estudiado los autoencoders variacionales y se ha propuesto la utilización de información mínima con el objetivo de solucionar el problema de sobrerregularización.

$$P(A) = P(A|S_i)P(S_i) + P(A|S_2)P(S_2) + +P(A|S_n)P(S_n)$$

Dado que el evento A ha ocurrido, la probabilidad a posteriori de que S_i sea el estado verdadero es:

$$P(S_i|A) = \frac{P(A|S_i)P(S_i)}{P(A)}$$

2.3.2. Clasificador de Naive Bayes

La clasificación consiste en asignar un objeto (instancia, dato) a una clase (categoría) [61]. Un clasificador bayesiano es un clasificador probabilístico basado en el teorema de Bayes [62]. Utilizando estos clasificadores, se puede predecir una instancia así como la probabilidad de que la instancia pertenezca a una clase/categoría. Por ejemplo, una imagen puede clasificarse como un paisaje, retrato, entorno urbano, etc. Desde un punto de vista matemático, el proceso de clasificación consiste en

asignar una clase, c, de un conjunto de clases C, a una instancia dada, representada por un vector de características o atributos. Estos clasificadores parten del supuesto de que el peso de un atributo en una clase determinada es independiente de todos los demás atributos. Esta hipótesis se denomina "*Independencia de clase condicional*" y está destinada a simplificar el cálculo [61].

Clasificar lo que se percibe a través de los sentidos es algo natural en el ser humano; básicamente nos permite abstraer la información, trasladándola a una representación más adecuada para la toma de decisiones. La clasificación ofrece muchos campos de aplicación, como por ejemplo:

- Control de calidad en la industria: clasificar una pieza o un producto como correcto o defectuoso.
- Sistemas de seguridad: identifican, por ejemplo, si una persona tiene o no acceso a una zona determinada.
- Vehículos inteligentes: detección de peatones en la carretera, clasificación de objetos detectados mediante cámaras u otros sensores.
- Lectores de correo electrónico: filtran los mensajes que son spam.
- Análisis de imágenes médicas: detección de tumores en las radiografías.
- Sistemas biométricos: asignan una imagen de una huella dactilar a la persona correspondiente.

En consecuencia, es importante diseñar clasificadores, ya sea en hardware o en software, que puedan ayudar a resolver estos problemas [61]. El clasificador bayesiano funciona de la siguiente manera [61]:

Theorem 2.3.1 ■ *Imagina que Z es un conjunto de muestras de entrenamiento, y que cada muestra tiene una etiqueta que pertenece a una clase. Existen K clases,* $C_1, C_2, , C_k$ *Cada muestra está representada por un vector n-dimensional,* $X = X_1, X_2, , X_n$ *que representa los valores medios de los n atributos* $A_1, A_2, , A_n$*, respectivamente.*

- *El clasificador predice que la muestra X pertenece a la clase con mayor probabilidad a posteriori, bajo la condición de X. Es decir, se predice que X pertenece a la clase* C_i *si y solo si:*

  $$P(C_i|X) > P(C_j|X) \text{ donde } 1 \leq j \leq m, j \neq i.$$

  De esta manera, se podría encontrar la clase que maximiza $P(C_i|X)$*. La clase* C_i *para la que se maximiza* $P(C_i|X)$ *se denomina "Hipótesis máxima a posteriori". Por el teorema de Bayes:*

  $$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

- $P(X)$ *es el mismo para todas las clases, por lo que solo* $P(X|C_i)P(C_i)$ *necesita ser maximizado. Si a priori las probabilidades de las clases,* $P(C_i)$*, se desconocen, entonces se supone que las clases son igualmente probables:* $P(C_1) = P(C_2) == P(C_k)$ *y, por lo tanto,* $P(X|C_i)$ *sería maximizada. En caso contrario, se maximizaría* $P(X|C_i)P(C_i)$*. Las probabilidades de las clases a priori pueden ser estimadas mediante:* $P(C_i) = freq(C_i, T)/|T|$*.*

- *Con el fin de predecir la etiqueta de la clase X, se evalúa* $P(X|C_i)P(C_i)$ *para cada clase* C_i*. El*

clasificador predice que la etiqueta de clase de X es C_i si y solo si es la clase que maximiza $P(X|C_i)P(C_i)$.

2.3.3. *Autoencoder*

Un *autoencoder* es una red neuronal que consta de un codificador y un decodificador. El *autoencoder* crea un proceso de discretización que se basa en muestras cuyo objetivo es submuestrear el objeto de entrada reduciendo el tamaño [63]. Por un lado, el objetivo del codificador es transformar los datos de alta dimensionalidad en datos de baja dimensionalidad generando un vector latente, mientras que el objetivo del decodificador es obtener el vector latente C_{AE} o datos de baja dimensionalidad y transformarlos en datos de alta dimensionalidad [64]. La red entera aprende la función de identidad $X_{out} = x$ optimizando los pesos y el sesgo de cada unidad de la misma. La diferencia entre x y X_{out} se define como la función de pérdida, *'loss function'*. Las funciones de pérdida más utilizadas son la del error absoluto medio (MAE) [5] y la del error cuadrático medio (MSE)[6] [64]. La función que representa la pérdida de los *autoencoders* es la siguiente:

$$min(f_{loss} : (W^T(W_x + b) + b'), x))$$

El método utilizado para entrenar los pesos y reducir la diferencia entre x y X_{out} es *Back propagation* [7] .

Las matrices de pesos y de sesgos del codificador están representadas por W^T y b, respectivamente. Las funciones del codificador y del decodificador son las siguientes [64]:

5: MAE: *Mean Absolut Error*. Es una métrica de evaluación de modelos que se utiliza con los modelos de regresión. El error medio absoluto de un modelo con respecto a un conjunto de pruebas es la media de los valores absolutos de los errores de predicción individuales sobre todas las instancias del conjunto de pruebas. Cada error de predicción es la diferencia entre el valor real y el valor predicho para la instancia [65].

6: MSE: *Mean Squared Error*. El error cuadrático medio es una métrica de evaluación de modelos que se utiliza a menudo con los modelos de regresión. El error cuadrático medio de un modelo con respecto a un conjunto de pruebas es la media de los errores de predicción al cuadrado sobre todas las instancias del conjunto de pruebas. El error de predicción es la diferencia entre el valor real y el valor predicho para una instancia [65].

7: *Back propagation*: Es un algoritmo de aprendizaje supervisado. Utiliza un proceso iterativo que ajusta los parámetros de peso de la red en función del gradiente de una medida de error. El procedimiento se implementa calculando un valor de error para cada unidad de salida, y retropropagando los valores de error a través de la red.

$$E(x) : W(x) + b$$
$$D(x) : W^T + b'$$

2.4. Algoritmia

Con el objetivo de validar las experimentaciones realizadas en capítulos posteriores, se han utilizado diferentes tipos de algoritmos de inteligencia artificial pertenecientes a la familia de algoritmos de aprendizaje automático o *Machine Learning*. Esto se ha hecho con el fin de validar cada experimento en las diferentes familias de algoritmos existentes. A continuación, se va a describir el estado del arte tanto de los algoritmos como de la familia a la que pertencen.

2.4.1. *Machine Learning* o aprendizaje automático

El *Machine Learning* o aprendizaje automático es una rama de la inteligencia artificial que se utiliza para crear modelos basados en datos para hacer predicciones o tomar decisiones sin ser explícitamente programados. Un modelo de clasificación es un ejemplo de aprendizaje supervisado, cuya finalidad es predecir etiquetas de clase categóricas para nuevos datos.

El aprendizaje automático (*Machine Learning, ML*) es el análisis de algoritmos computacionales que se utilizan como una rama de la inteligencia artificial que evoluciona automáticamente a través de la experiencia [66]. Para hacer predicciones o tomar decisiones sin estar explícitamente programados para ello, los algoritmos de aprendizaje automático crean un modelo basado en datos de muestra, definidos como "datos de entrenamiento" [67]. La clasificación es uno de los modelos de aprendizaje automático supervisado cuyo objetivo es predecir las etiquetas de clase categóricas de nuevas (valores discretos, no ordenados, pertenencia a un grupo) basándose en observaciones previas [68].

En las siguientes secciones se describen las familias y los algoritmos de *Machine Learning* que se han utilizado.

2.4.2. Algoritmos de árbol de decisión

Los árboles de decisión utilizan diversos algoritmos para decidir dividir un nodo en dos o más subnodos. La forma de hacer subnodos aumenta la homogeneidad de los subnodos consecuentes [69]. El Árbol de Decisión divide los nodos en todas las combinaciones variables disponibles y luego selecciona la división que da lugar a los subnodos más homogéneos [70].

Los árboles de decisión son herramientas de aprendizaje automático que dividen nodos en subnodos con el objetivo de aumentar la homogeneidad de los subnodos. Los dos algoritmos de árboles de decisión utilizados en las experimentaciones de tesis doctoral son *Random Forest* y *J48*.

A lo largo de las experimentaciones realizadas en esta tesis doctoral, se han utilizado dos algoritmos diferentes que pertenecen a la familia de árboles de decisión: *Random Forest* y *J48*.

Random Forest

El algoritmo *Random Forest*, propuesto en 2001 por L. Breiman [71], se considera como uno de los algoritmos más representativos del *Machine Learning* y se caracteriza por su sencillez y eficacia [72].

Random Forest es un clasificador formado por un conjunto de clasificadores estructurados en árbol con vectores aleatorios independientes distribuidos de forma idéntica donde cada árbol emite un voto unitario en la entrada X para la clase más popular [73]. Se genera un vector aleatorio independiente de los anteriores vectores aleatorios de la misma distribución y se genera un árbol utilizando

el conjunto de datos de entrenamiento [74] y se combinan los resultados obtenidos con el objetivo de crear un modelo más sólido el generado por cada árbol por separado [75].

Cada árbol se genera siguiendo un proceso de dos etapas [76]:

- En primer lugar, con el conjunto de datos se crea un número considerable de árboles de decisión donde cada uno de ellos tiene un subconjunto de variables aleatorias m (predictores) donde $m < M$ (M = total de variables predictores).
- En segundo y último lugar, cada árbol crece hasta su máxima extensión

El algoritmo *Random Forest* es un clasificador que combina varios árboles de decisión con vectores aleatorios independientes y distribuidos de forma idéntica para crear un modelo más sólido. Cada árbol se crea a partir de un subconjunto de variables aleatorias y crece hasta su máxima extensión. Es un algoritmo sencillo y eficaz.

En la Fig. 2.1 se puede apreciar el diagrama de flujo que sigue el algoritmo *Random Forest*.

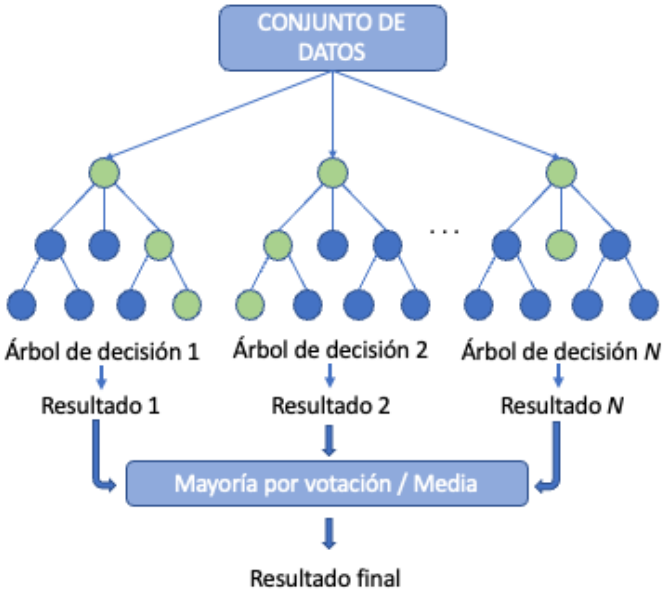

Figura 2.1: Diagrama de flujo del algoritmo *Random Forest*

El funcionamiento del algoritmo *Random Forest*, tal y como muestra la Fig. 2.1 es el siguiente. Utilizando árboles de decisión como clasificadores básicos,

implementa el método método bootstrap para obtener múltiples subconjuntos de muestras [77], crea un arbol de decisión utilizando cada uno de los subconjuntos y combina varios árboles de decisión en un Random Forest. Cuando se tiene la muestra clasificada, el resultado final de la clasificación se decide mediante una votación sobre el Árbol de decisión [78]. Generalmente, los investigadores aumentan la precisión del clasificador reduciendo la asociación entre clasificadores [79]. Una vez generado el *Random Forest*, se toman las muestras del conjunto de datos de *test* y se utilizan las reglas de cada árbol de decisión generado aleatoriamente para pronosticar el resultado y almacenar el resultado esperado (objetivo), y a través de una votación se elige el pronóstico final, considerándose este como el pronóstico conseguido por el algoritmo *Random Forest* [80].

El algoritmo Random Forest utiliza árboles de decisión como clasificadores básicos y implementa el método *Bootstrap* para generar múltiples subconjuntos de muestras. Cada subconjunto se usa para crear un árbol de decisión, y luego se combinan varios árboles de decisión en un *Random Forest*. La clasificación final se decide mediante una votación de los árboles de decisión. La predicción se realiza utilizando las reglas de cada árbol de decisión generado aleatoriamente para predecir el resultado, y el resultado final es la clase con el máximo número de votos.

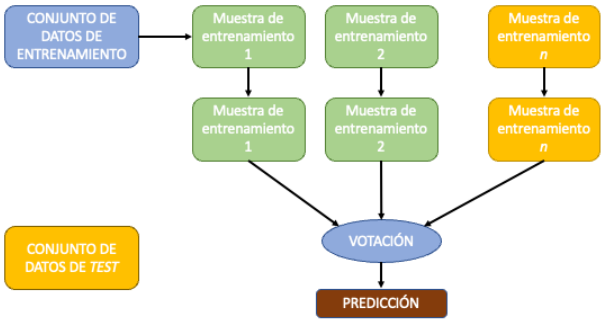

Figura 2.2: Diagrama de flujo de entrenamiento del algoritmo *Random Forest*

Como se puede observar en la Fig 2.2, el diagrama de flujo del algoritmo básico de *Random Forest* es el siguiente:

El algoritmo *Random Forest* se compone de muchos árboles de decisión [81], y a continuación se

muestran los pasos para el desarrollo del algoritmo [82]:

1. Selecciona "*K*" muestras de manera aleatoria del conjunto de datos completo donde se encuentran las "*m*" muestras, donde *k*<*m*.
2. Calcular el nodo "*d*" entre las muestras "*K*" utilizando el mejor punto de división.
3. Utilizar la mejor división para dividir la red en nodos hijos.
4. Repita los pasos de 1 a 3 hasta alcanzar el número de nodos "*n*".
5. Desarrollar un árbol que contenga "*n*" número de árboles repitiendo los pasos de 1 a 4 durante "*n*" número de veces.

Una vez generado el clasificador de *Random Forest*, el siguiente paso es realizar la predicción. La predicción se hace de la siguiente manera. Se cogen las muestras del conjunto de datos de *test* y utiliza las reglas de cada árbol de decisión generado aleatoriamente para predecir el resultado y almacenar el resultado objetivo. La decisión se toma en base a la siguiente fórmula [76]:

$$H(x) = \arg_{max} \sum_{i=1}^{k} I(h_i(x) = Y) \qquad (2.1)$$

Donde:
x = Conjunto de datos de *test*
h_i = Árbol de decisión simple
Y = Variable de salida (etiqueta de clasificación)
I = Función de indicación
H = Modelo de *Random Forest*

Con todo esto, como se ha mencionado anteriormente [80], se recoge el resultado de clasificación de cada árbol de prueba para la muestra de prueba y el resultado final de la clasificación es la clase con el máximo número de votos.

Ventajas del algoritmo Random Forest [8] A continuación se muestran cuáles son las principales ventajas del algoritmo *Random Forest* [83]:

- La posibilidad de ser utilizados para realizar clasificación o predicción. Cuando realiza clasificación, el resultado es la clase con mayor número de votaciones generadas por cada árbol, es decir, cada nueva muestra se pasa por cada uno de los árboles y se le asigna la clase más votada. Sin embargo, para realizar predicción, el resultado es el promedio de la salida de todos los árboles.
- Rendimiento similar al de técnicas más complejas siendo un modelo simple de entrenar.
- Técnica de lo más certera cuando se trata de bases de datos grandes, y con un desempeño muy eficiente.
- Proveé información acerca de qué variables son importantes a la hora de realizar la clasificación. Como consecuencia de esto, esta técnica se utiliza para reducir dimensionalidad [76] [67].
- Aún con grandes pérdidas de datos, es capaz de mantener la precisión.
- Permite el análisis de la relación entre variables [84].

8: El algoritmo *Random Forest* es un modelo de clasificación o predicción con una serie de ventajas, tales como: rendimiento similar a técnicas más complejas y eficiente con bases de datos grandes, capacidad de identificar variables importantes, mantener precisión con pérdidas de datos y permitir el análisis de relación entre variables.

Desventajas del algoritmo Random Forest

Por otro lado, este algoritmo también tiene ciertas desventajas que hay que tener en cuenta [83] [67]:

Sin embargo, también presenta desventajas, como el riesgo de sobre-entrenamiento (*overfitting*), dificultad en la interpretabilidad de los resultados, limitaciones en la naturaleza continua de las predicciones, posible sesgo en predictores categóricos con varios niveles y poco control sobre el modelo.

- Uno de los problemas principales de este algoritmo es que hay que tener cuidado con el *over-fitting* o sobre-entrenamiento.
- La interpretabilidad de los resultados graficados es de alta dificultad.
- Las predicciones al no ser de naturaleza continua, no pueden ir más allá que el rango de valores del conjunto de datos.
- Cuando existen predictores en formato categórico con varios niveles, el resultado podría estar sesgado hacia predictores con más niveles.
- No existe mucho control sobre el modelo.

J48

En cuanto al algoritmo J48, fue propuesto por Quinalan [85] en 1993. Es una implementación de Java del algoritmo C4.5 [86] en la herramienta de minería de datos de código abierto de *Weka* [9]. El algoritmo C4.5 se basa en el algoritmo $ID3^2$, que intenta encontrar árboles de decisión pequeños o simples. Estas son algunas de las principales premisas que rigen este algoritmo [85]:

9: Weka: Waikato Environment for Knowledge Analysis es un software libre con licencia que contiene herramientas de visualización y algoritmos para el análisis de datos y modelado predictivo.

- En caso de que todos las muestras sean de la misma clase, el árbol es una hoja, y por lo tanto, la hoja se devuelve etiquetada con esta clase.
- En cada nodo del árbol, J48 elige un atributo de los datos que mejor divida el conjunto de muestras en subconjuntos. Su criterio es la ganancia de información más alta y normalizada como resultado de elegir un atributo para dividir los datos. El atributo que tenga la

mayor ganancia de información normalizada es el elegido para tomar la decisión.
- En función del criterio de selección actual se encuentra el mejor atributo para ramificar.

J48 es un algoritmo de minería de datos basado en el algoritmo *C4.5*. Fue implementado en *Java* en la herramienta de minería de datos *Weka*. La idea detrás de J48 es encontrar árboles de decisión simples y pequeños. En cada nodo del árbol, J48 elige un atributo de los datos que divide mejor el conjunto de muestras en subconjuntos utilizando la ganancia de información más alta y normalizada como criterio. El atributo con la mayor ganancia de información normalizada se utiliza para tomar la decisión en cada nodo.

En cada nodo del árbol, J48 elige un atributo de los datos que que mejor divide el conjunto de muestras en subconjuntos [85]. Su criterio criterio es la ganancia de información más alta y normalizada que resulta de elegir un atributo para dividir los datos. El atributo atributo que tenga la mayor ganancia de información normalizada se para tomar la decisión. Para cada atributo, se calcula la ganancia calculada y la ganancia más alta se utiliza en el nodo de decisión.

Cálculo de ganancia

Para el proceso de calcular la ganancia, como se ha mencionado en las premisas anteriores, se usa la *Entropía*, una medida del desorden de los datos. La entropía es calculada de la siguiente manera [86] :

$$Entropy(\vec{y}) = -\sum_{j=1}^{n} \frac{|y_j|}{|\vec{y}|} log \frac{|y_j|}{|\vec{y}|} \qquad (2.2)$$

Iterando sobre todos los valores posibles de $|\vec{y}|$ La Entropía es:

$$Entropy(j|\vec{y}) = \frac{|y_j|}{|\vec{y}|} log \frac{|y_j|}{|\vec{y}|} \qquad (2.3)$$

Finalmente, se define de la siguiente manera:

$$Gain(\vec{y}, j) = Entropy(|\vec{y}| - Entropy(j|\vec{y})) \tag{2.4}$$

El objetivo es maximizar la Ganancia, dividiendo por la entropía global debida a la división del argumento $\vec{y}$ por el valor j.

Poda

Este es un paso importante para el resultado debido a los valores anómalos. Todos los conjuntos de datos contienen un pequeño subconjunto de instancias que no están bien definidas y difieren de los otros en su vecindad. Tras la creación completa del árbol, que debe clasificar todas las instancias del conjunto de entrenamiento, se realiza la poda. Esto se hace para reducir los errores de clasificación causados por la particularización del conjunto de entrenamiento: se hace para que el árbol sea más general.

Logistic Regression

El algoritmo de *Logistic Regression* proporciona un método para modelizar una variable de respuesta binaria, que toma los valores 1 y 0 [87]. Se considera el algoritmo más idoneo para situaciones en las que se debe predecir la aparición de un resultado binario a partir de una o más variables independientes (predictoras) [88] [89] [90].

El algoritmo *Logistic Regression* fue inventado en el siglo XIX por Pierre François Verhulst, matemático francés, para describir el crecimiento de las poblaciones humanas y el curso de las reacciones químicas autocatalíticas [91]. Verhulst publicó sus

sugerencias, editadas por Quetelet entre 1838 y 1847 [92]. El modelo logístico se ajustaba muy bien a la evolución real de la población de Francia, Bélgica, Essex (Reino Unido) y Rusia hasta principios de la década de 1830.

El algoritmo *Logistic Regression* es una técnica de análisis de datos que permite predecir un resultado binario (1 o 0) a partir de una o más variables independientes. Fue inventado en el siglo XIX para describir el crecimiento de poblaciones humanas y reacciones químicas. Se utiliza para tomar decisiones basadas en si un suceso ha ocurrido o no, y es especialmente adecuado para modelar situaciones en la salud, donde se determina el estado de un paciente como enfermo o sano.

La función logística fue descubierta de nuevo en 1920 por Pearl y Reed en un estudio sobre el crecimiento demográfico de EE.UU [93]. *Logistic Regression* se utiliza cuando el método de investigación se centra en si un suceso ha ocurrido o no, más que en cuándo ha ocurrido (no se utiliza información temporal). Resulta especialmente adecuada para modelos que implican la toma de decisiones (sí o no), por lo que se utiliza ampliamente en estudios de ciencias de la salud para determinar el estado del paciente (enfermo o sano). Existen formas más complejas que pueden tratar situaciones en las que la variable pronosticada tiene más de dos categorías; se habla entonces de regresión logística policotómica o multinomial [94].

Como en todos los modelos, se hacen ciertas suposiciones para ajustar el modelo a los datos. LR no supone una relación lineal entre las variables dependientes e independientes, sino entre el logit [10]del resultado y los valores predictores [96]. La variable dependiente debe ser categórica; las variables independientes no tienen por qué ser de intervalo; ni distribuidas normalmente, ni relacionadas linealmente, ni de igual varianza dentro de cada grupo. Y, por último, las categorías (grupos) deben ser mutuamente excluyentes y exhaustivas. Una instancia sólo puede estar en un grupo y cada caso debe pertenecer a uno de los grupos. El algoritmo *Logistic Regression* tiene la capacidad de adaptarse tanto a variables independientes cate-

10: Logit: función que tiene como objetivo calcular el logaritmo de la razón de momios [95].

góricas como continuas. Aunque la potencia del análisis aumenta si las variables independientes se distribuyen normalmente y tienen una relación lineal con la variable dependiente [97]. Investigaciones de estos supuestos muestran que esta técnica puede emplearse de forma algo más flexible que las técnicas de regresión tradicionales, lo que la hace adecuada para muchas situaciones clínicamente relevantes. Para determinados casos, el algoritmo *Logistic Regression* calcula la probabilidad de que una instancia con con sus respectivas variables independientes pertenezca a la categoría modelada. Se necesitan conjuntos de muestras grandes para este algoritmo porque los coeficientes de máxima verosimilitud son estimaciones de muestras grandes [98].

La variable dependiente debe ser categórica y las variables independientes pueden ser categóricas o continuas. El algoritmo calcula la probabilidad de que una instancia pertenezca a una categoría. Se requieren conjuntos de muestras grandes para obtener resultados precisos y estudios con tamaños de muestra pequeños o medianos pueden sobrestimar los resultados.

Los estudios realizados con tamaños de muestra pequeños o medianos que emplean el algoritmo *Logistic Regression* sobrestiman el resultado [89] [99].

Además de sus múltiples usos para desarrollar modelos que predigan acontecimientos en las ciencias físicas [100], económicas [101] [102] y ciencias políticas [103], el algoritmo *Logistic Regression* se aplica cada vez más en la investigación médica [104] [105] [106] y en la automoción [107] [108] [109]. Ejemplos de casos de uso de este algoritmo en medicina incluyen un estudio de los factores que predicen si se producirá una mejora o no tras una intervención [110] [111] o la presencia o ausencia de una enfermedad en relación a diversos factores [112]; y en automoción, se ha utilizado este algoritmo para minimizar roturas en el proceso de estampación de automóviles [107], para determinar la importancia relativa de las cargas máximas modeladas de

la columna vertebral, las cargas de las manos, la cinemática del tronco y las cargas acumuladas de la columna vertebral como predictores del dolor lumbar declarado en la industria de la automoción [108]; y también se ha realizado una investigación sobre la identificación y validación de un modelo de regresión logística para predecir lesiones graves asociadas a colisiones de vehículos de motor [109].

La regresión logística multivariable es una técnica estadística sofisticada y se ha expresado preocupación por su uso e interpretación [113] [114] [115] [116]. Las inquietudes se han centrado en los supuestos asociados al uso apropiado, la interpretación correcta y la información completa de las regresiones logísticas multivariables. La calidad del análisis de la regresión logística depende en gran medida de que los investigadores comprendan los supuestos inherentes al método y sigan los principios desarrollados para garantizar su correcta aplicación [117].

El modelo - Logistic Regression

La regresión logística es un algoritmo que asigna un peso a cada variable predictor para estimar su contribución a la variable dependiente, que es una variable binaria con valores 0 y 1.

El algoritmo *Logistic Regression* asigna a cada predictor un coeficiente que mide su contribución independiente a la variación de la variable dependiente. La variable dependiente Y toma el valor 1 si la respuesta es "Sí" y toma el valor 0 si la respuesta es "No". La forma del modelo para las probabilidades predichas se expresa como logaritmo natural (ln) [117]:

$$\ln\left(\frac{P(Y)}{1-P(Y)}\right) = \beta_0+\beta_1X_1+\beta_2X_2++\beta_kX_k \quad (2.5)$$

Y,

$$\frac{P(Y)}{1-P(Y)} = e^{\beta_0+\beta_1X_1+\beta_2X_2++\beta_kX_k} \quad (2.6)$$

$$P(Y) = e^{\beta_0+\beta_1X_1+\beta_2X_2++\beta_kX_k} - P(Y)e^{\beta_0+\beta_1X_1+\beta_2X_2++\beta_kX_k} \quad (2.7)$$

donde, $\ln\left(\frac{P(Y)}{1-P(Y)}\right)$ es el logaritmo (probabilidades) de los resultados, Y es el resultado dicotómico; $X_1, X_2, , X_k$ son las variables predictoras, $\beta_0, \beta_1, \beta_2, , \beta_k$ son los coeficientes de regresión del modelo y β_0 el intervalo.

El modelo utiliza la máxima probabilidad para estimar los parámetros y se expresa como un logaritmo natural. La regresión logística busca predecir correctamente la categoría de la variable dependiente para cada caso, y utiliza la curva logística para representar la correlación entre las variables. La curva logística permite una asociación acotada entre 0 y 1, donde los valores predichos se acercan a 0 a niveles bajos de la variable independiente y se acercan a 1 a niveles altos de la misma.

En la ecuación 2.7, el modelo de regresión logística relaciona directamente la probabilidad de Y con las variables predictoras. Esa ecuación tiene como objetivo que la regresión logística estime los *k+1* parámetros β desconocidos. Para ello se recurre a la estimación de máxima probabilidad, que consiste en hallar el conjunto de parámetros para los que la probabilidad de los datos observados es mayor. Los coeficientes de regresión indican el grado de asociación entre cada variable independiente y el resultado. Cada coeficiente representa la cantidad de cambio que esperaríamos en la variable de respuesta si se produjera un cambio de una unidad en la variable de predicción. El objetivo de la regresión logística es predecir correctamente la categoría del

resultado para cada caso utilizando el mejor modelo. Para lograr este objetivo se crea un modelo que incluye todas las variables predictoras que son útiles para predecir la correspondiente variable de respuesta. La regresión logística calcula la probabilidad de éxito sobre la probabilidad de fracaso. Los resultados del análisis se presentan en forma de cociente de probabilidades [117].

La variable dependiente binaria tiene los valores 0 y 1 y el valor predicho (probabilidad) debe estar acotado para caer dentro del mismo rango [118]. Para definir una asociación acotada entre 0 y 1, el algoritmo *Logistic Regression* utiliza la curva logística para representar la correlación entre la variable independiente y la dependiente [117]. A niveles muy bajos de la variable independiente, la probabilidad se aproxima a 0, pero nunca llega a 0. Del mismo modo, si la variable independiente aumenta, los valores predichos aumentan hacia arriba en la curva y se aproximan a 1, pero nunca llegan a ser iguales a 1.

Transformar una probabilidad en valores probabilísticos y logarítmicos

La transformación logística garantiza que los valores estimados no queden fuera del intervalo de 0 y 1. Esto se consigue en dos pasos: en primer lugar, la probabilidad se replantea como cociente, que se define como la relación entre la probabilidad de que se produzca el suceso y la probabilidad de que no se produzca. Por ejemplo, si un atleta tiene una probabilidad de 0.8 de ganar una carrera, las probabilidades de que gane son 0.8/(1 - 0.8) = 4:1. Para restringir los valores predichos entre 0 y 1, el valor de las probabilidades puede convertirse de nuevo en una probabilidad; de esta manera:

La transformación logística en la regresión logística asegura que los valores estimados estén dentro del rango de 0 a 1, convirtiendo la probabilidad en un cociente y luego en una probabilidad nuevamente. El cálculo del valor logit también limita los valores predichos para que no sean inferiores a 0. Valores logit negativos corresponden a probabilidades menores a 1, valores logit positivos corresponden a probabilidades mayores a 1 y un valor logit de 0 corresponde a una probabilidad de 1/2.

$$Probabilidad(evento) = \frac{cocientes(evento)}{1 + cocientes(evento)} \tag{2.8}$$

Por lo tanto, se puede demostrar que la probabilidad correspondiente es $4/(1 + 4) = 0{,}8$. Además, para que los valores de las probabilidades no bajen de 0, que es el límite inferior (no hay límite superior), hay que calcular el valor logit, que se calcula tomando el logaritmo de las probabilidades. Las probabilidades inferiores a 1 tienen un valor logit negativo, las probabilidades superiores a 1,0 tienen valores logit positivos y las probabilidades de 1,0 (correspondientes a una probabilidad de 0,5) tienen un valor logit de 0 [119].

La variable independiente "*Xi*" tiene un efecto en la variable dependiente, y cuando *Xi* aumenta en una unidad, las probabilidades de la variable dependiente también aumentan en un factor llamado "cociente de probabilidades". Este factor varía entre 0 y infinito positivo y muestra la cantidad relativa en que las probabilidades de la variable dependiente aumentan o disminuyen con el aumento de la variable independiente.

Interpretación de la probabilidad de éxito

Cuando una variable independiente *Xi* aumenta en una unidad ($X_i + 1$) permaneciendo constantes todos los demás factores, las probabilidades de la variable dependiente aumentan en un factor $exp(\beta_i)$ que se denomina cociente de probabilidades y oscila entre cero e infinito positivo [120]. Esto indica la cantidad relativa en que las probabilidades de la variable dependiente aumentan (OR > 1) o disminuyen (OR < 1) cuando el valor de la variable independiente correspondiente aumenta en una (1) unidad [117].

Selección de variables dependientes

En la mayoría de los casos, el resultado se puede clasificar fácilmente como que ha ocurrido o como que no ha ocurrido. Por ejemplo, ha habido un fallo de motor, o ha habido un fallo de producción o no,

se distinguen con relativa facilidad y se codifican como ocurridos o no ocurridos.

El resultado de un estudio puede ser fácilmente clasificado como ocurrido o no ocurrido, o puede ser dicotómico a pesar de derivarse de datos continuos, que se han recodificado como categóricos en un punto de corte. En algunos casos, los resultados continuos se pueden traducir fácilmente en un evento dicotómico, pero en otros la elección de la variable de resultado puede ser complicada. Muchas variables multicategoría o continuas pueden reducirse a dicotómicas.

Una vez realizada esta categorización, pueden estudiarse los predictores de ese resultado [121]. En otros casos, el resultado puede tratarse como dicotómico, pero, de realidad, se deriva de la censura de datos continuos; es decir, se ha producido un criterio de corte y los datos se han recodificado de continuos a categóricos en el punto de corte. En estos casos, la situación a la hora de elegir la variable de resultado puede ser más complicada [102]. En algunos casos, los resultados continuos se traducen con relativa facilidad en un evento dicotómico. Estos casos suelen referirse a medidas para las que se han desarrollado puntos de corte bien establecidos para la presencia de un evento [122]. Cabe destacar que muchas variables multicategoría o incluso continuas pueden reducirse a dicotómicas [94].

Selección de variables predictoras

Otro aspecto a tener en cuenta es la selección de las variables que se analizarán como posibles predictores del resultado. Esto sólo puede lograrse mediante un estudio minucioso de la literatura en relación con el resultado, con el fin de garantizar que se incluye toda la gama de predictores potenciales [106]. Sin embargo, la selección de variables predictoras presenta una serie de inconvenientes que pueden hacer que el modelo de regresión logística presentado parezca tener una varianza mayor o menor de la que puede tener en realidad [123]. Los resultados de cualquier modelo *Logistic Regression* dependerán de las variables seleccionadas como predictores potenciales; en pocas palabras, si una

variable no se selecciona para el análisis, no puede figurar en el modelo final. Sin embargo, la decisión de incluir o no ciertos factores en el conjunto de datos inicial puede influir en los resultados [124].

La selección de variables predictoras para un modelo de regresión logística es importante y debe hacerse cuidadosamente. La inclusión de variables no relacionadas puede inflar la aparente validez predictiva del modelo y la omisión de variables relacionadas puede tener importantes repercusiones en los resultados. El tamaño de la muestra y la presencia de datos faltantes son limitaciones que pueden causar sesgos en la selección de las variables. Agresti sugiere que se necesitan al menos 10 variables por cada instancia estudiada.

Además, si se tienen en cuenta los efectos de interacción entre las variables, la omisión de algunas de ellas podría tener importantes repercusiones en los resultados. Desgraciadamente, la solución no consiste simplemente en incluir tantas variables como sea posible, ya que la inclusión de variables que no están relacionadas con el resultado en cuestión (la adición de variables no relacionadas) tiende a inflar la aparente validez predictiva del modelo final [125]. No hay una forma óptima de saber si el conjunto de predictores elegidos es adecuado, pero una serie de reglas empíricas pueden demostrar que la elección es razonable. Por ejemplo, si la especificación (el grado en que los predictores identifican correctamente a los individuos que no muestran el resultado concreto, los verdaderos negativos) y la sensibilidad (el grado en que los predictores identifican correctamente a los individuos que muestran el resultado, los verdaderos positivos) del modelo son superiores al 80 %, es probable que los predictores elegidos sean válidos [126].

Es muy posible que existan limitaciones en un estudio concreto que provoquen un sesgo en la selección de los datos utilizados para el análisis. Una posible restricción es el tamaño de la muestra, que limita el número de variables que pueden estudiarse [117]. Existe cierto debate sobre el número de variables necesarias, pero Agresti [127] sugiere que se necesita un mínimo de 10 variables por cada instancia estudiada; una sugerencia basada en

pruebas estadísticas que confirmen la fiabilidad de las regresiones logísticas realizadas con diferentes números de variables por instancia [128].

Otra fuente de sesgo en la selección de las variables que se estudian es la de los datos que faltan, ya que puede llevar a excluir ciertas variables del análisis si faltan grandes cantidades de datos [117].

Evaluación del modelo

La validez del modelo de regresión logística puede evaluarse de varias formas [117]. En primer lugar, hay que evaluar el modelo completo (relación entre todas las variables independientes y la variable dependiente). En segundo lugar, hay que evaluar la importancia de cada una de las variables independientes. En tercer lugar, hay que evaluar la exactitud predictiva o la capacidad discriminatoria del modelo y, por último, se tiene que validar el modelo. Si el modelo completo funciona bien, la siguiente pregunta es qué importancia tiene cada una de las variables independientes. El coeficiente regresión logística de la i-ésima variable independiente muestra el cambio en las probabilidades logarítmicas previstas de obtener un resultado para un cambio unitario en la i-ésima variable independiente, en igualdad de condiciones [128]. Es decir, si la i-ésima variable independiente, con coeficiente de regresión b, se cambia en 1 unidad mientras que todos los demás predictores se mantienen constantes, se espera que las probabilidades logarítmicas del resultado cambien b unidades. Existen un par de técnicas diferentes diseñadas para evaluar la importancia de una variable independiente en la regresión logística: test del ratio de probabilidad [87] y el estadístico de Wald [87].

La validez del modelo de regresión logística se puede evaluar en cuatro áreas: el modelo completo, la importancia de cada variable independiente, la exactitud predictiva y la validación del modelo. Para evaluar la importancia de las variables independientes, se pueden usar dos técnicas: el test de ratio de probabilidad y el estadístico de Wald. El test de ratio de probabilidad también puede usarse para evaluar la contribución de cada predictor individual al modelo.

El test de ratio de probabilidad utilizado para eva-

luar el ajuste global del modelo también puede utilizarse para evaluar la contribución de los predictores individuales a un modelo determinado. Cuando se usa para un parámetro concreto, compara la probabilidad de obtener los datos cuando el parámetro es cero, L_0, con la probabilidad L_1 de obtener los datos evaluados en la estimación de máxima probabilidad del parámetro.

El test estadístico es calculado de la siguiente manera [117]:

$$G = -2ln\frac{L_0}{L_1} = -2ln(L_0 - L_1) \qquad (2.9)$$

Validación del modelo

La validez del modelo debe evaluarse con un conjunto de datos diferente al utilizado para su creación para evitar una sobrevaloración del resultado. La validación interna se realiza con una submuestra de los datos de creación y la validación externa se hace con un nuevo conjunto de datos independiente. Los métodos más utilizados para la validación interna son el data-splitting, el data-splitting repetido, la técnica jackknife y el bootstrapping. Un buen ajuste del modelo a un segundo conjunto de datos proporciona una garantía de capacidad de generalización, pero una falta de ajuste puede ser debido a un contexto diferente o a una verdadera falta de ajuste del modelo.

Cuando un modelo se valida utilizando los datos con los que se ha creado, es probable que el resultado esté sobrevalorado. Por lo tanto, la validez del modelo debe evaluarse realizando pruebas con un conjunto de datos diferente [129]. Si el modelo se desarrolla con una submuestra de observaciones y se valida con la muestra restante [130], se denomina validación interna. Los métodos más utilizados para obtener una buena validación interna son el data-splitting, el data-splitting repetido, la técnica jackknife y el bootstrapping [131].

Si la validez se comprueba con un nuevo conjunto de datos independientes de la misma población o de una población similar, se denomina validación externa. La obtención de un nuevo conjunto de datos permite comprobar el modelo en un contexto diferente. Si el primer modelo se ajusta al segundo conjunto de datos, existe cierta garantía de capacidad de generalización del modelo. Sin embargo, si

el modelo no se ajusta a los segundos datos, la falta de ajuste puede deberse a los diferentes contextos de los dos conjuntos de datos o a una verdadera falta de ajuste del primer modelo [132].

2.4.3. Algoritmos de agrupación

Para la familia de algoritmos de agrupación, solo se ha utilizado el conocido algoritmo *K Nearest Neighbor* [133].

K Nearest Neighbor

El algoritmo de clasificación KNN (*K Nearest Neighbor*) es uno de los métodos de clasificación más simples y comúnmente utilizados. Fue introducido por primera vez en 1951 por Fix y Hodges como un enfoque no paramétrico para la clasificación de patrones. Desde entonces, ha habido una amplia línea de investigación que incluye nuevos enfoques de rechazo, ajustes en relación a la tasa de error de Bayes, enfoques de ponderación de la distancia y métodos de "soft computing" y "fuzzy".

La clasificación mediante el algoritmo *K-nearest-neighbor (KNN)* es uno de los métodos de clasificación más importantes y sencillos y es un algoritmo que se suele usar como primera opción cuando hay poco o ninún conocimiento previo sobre la distribución de los datos [133]. La clasificación *k* del vecino más próximo se desarrolló a partir de la necesidad de realizar análisis discriminativos cuando se desconocen o son difíciles de determinar las densidades de probabilidad. En un informe no publicado de la Escuela de Medicina de Aviación de las Fuerzas Aéreas de EE.UU. de 1951, Fix y Hodges introdujeron un método no paramétrico para la clasificación de patrones que desde entonces se conoce como la regla del vecino más próximo k [134]. Más tarde, en 1967, se desarrollaron algunas de las propiedades formales de la regla del*K Nearest Neighbor*; por ejemplo, se demostró que para *k=1* y n$\rightarrow \infty$ el error de clasificación del *K Nearest Neighbor* está limitado por encima del doble de la tasa de error de Bayes [135]. Una vez establecidas estas características de la clasificación *K Nearest*

Neighbor, se inició una amplia línea de investigación que incluyó nuevos enfoques de rechazo [133], ajustes con respecto a la tasa de error de Bayes [136], enfoques de ponderación de la distancia [137] [138], métodos de *soft computing* [139] y métodos *fuzzy* [140] [141].

El algoritmo se basa en una función que mide la similitud entre dos instancias y clasifica una nueva instancia en función de la mayoría de los votos de los vecinos más cercanos. Hay varias formas de medir la distancia entre dos instancias: Euclideana, Manhattan y Minkowski.

K Nearest Neighbor es un clasificador no paramétrico convencional que proporciona un buen rendimiento para valores óptimos de *K* [142]. Se ha utilizado frecuentemente para resolver problemas de clasificación [143]. El algoritmo se basa en una función que mide la diferencia o similitud entre dos instancias [144] [145]. Existen varios grupos de clasificación y, a continuación, se clasifica una nueva instancia en función de los diferentes votos obtenidos de los vecinos. Existen varias técnicas para medir la distancia entre dos instancias: Euclideana, Manhattan y Minkowski [145].

Distancia Euclidiana

A pesar de que existen varias técnicas, la técnica más utilizada es la distancia Euclidiana, ya que proporciona facilidad, eficiencia y productividad [146]. La distancia Euclidiana estándar *d(x,y)* entre dos instancias x e y que se utiliza a menudo como función de distancia, se define de la siguiente manera:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(a_i(x) - a_i(y))^2)} \qquad (2.10)$$

Dada una instancia x, *K Nearest Neighbor* asigna la clase más común de x a los K vecinos más cercanos, como se muestra en la Ecuación *2.11*. *KNN* es

un ejemplo de *lazy learning* [147]. El *lazy learning* simplemente almacena los datos de entrenamiento en el momento del entrenamiento y retrasa su aprendizaje hasta el momento de la clasificación [144].

$$c(x) = argmax \sum_{i=1}^{k} \delta(c, c(y_i)) \tag{2.11}$$

donde $y_1, y_2, , y_k$ son los k vecinos más próximos de x, k es el número de vecinos, y $\delta(c, c(y_i)) = 1$ si $c = c(y_i)$ y $\delta(c, c(y_i)) = 0$ en caso contrario.

La técnica de KNN es un método de clasificación basado en la asignación de la clase más común a la instancia más cercana a partir de sus vecinos K. La distancia Euclidiana es la técnica más utilizada en KNN, aunque puede ser afectada por la maldición de la dimensionalidad si hay muchos atributos irrelevantes en el problema.

Aunque KNN ha sido ampliamente utilizado durante décadas debido a su simplicidad, eficacia y robustez, al basarse en una función de distancia que mide la diferencia o similitud entre dos instancias su sesgo inductivo corresponde a la suposición de que la clasificación de una instancia será más similar a la clasificación de otras instancias cercanas en el espacio euclidiano [144]. Por lo tanto, la forma de definir la función de distancia es crucial. La distancia entre instancias se calcula basándose en todos los atributos de la instancia. Sin embargo, cuando el problema que se investiga incluye atributos irrelevantes, esta distancia euclídea estándar se verá dominada por el gran número de atributos irrelevantes y se volverá inexacta. Esta dificultad, que surge cuando hay muchos atributos irrelevantes, se conoce a veces como la *maldición de la dimensionalidad* [148]. KNN es especialmente sensible a este problema. Un enfoque estricto [149] para hacer frente a la *maldición de la dimensionalidad* consiste en eliminar completamente los atributos

menos relevantes del espacio de atributos al calcular la distancia entre dos instancias. Este enfoque se conoce como selección de características. Para abordar este problema, se han presentado numerosos algoritmos. Por ejemplo, Kohavi et al. [150] propusieron un enfoque denominado *wrapper* para buscar un subconjunto óptimo de características. Además, los métodos de *greedy search* [151] y *genetic search* [152] suelen ser a menudo alternativas utilizadas.

Para hacer frente a la maldición de la dimensionalidad se han presentado varios enfoques, como la selección de características, la ponderación de atributos y la medida de no similitud. Algunos autores han propuesto soluciones específicas, como WAKNN y la medida de no similitud de Huang.

Otro enfoque eficaz para hacer frente al problema de la maldición de la dimensionalidad consiste en ponderar cada atributo de forma diferente al calcular la distancia entre dos instancias [144]. Este enfoque se conoce como ponderación de atributos. Así, la función de distancia debería definirse como:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} w_i^2 (a_i(x) - a_i(y))^2} \qquad (2.12)$$

donde $w_i (i = 1, , n)$ es la ponderación del atributo A_i.

Han [153] presentó un método para la ponderación de variables. Propone ponderar la importancia de las palabras discriminantes utilizando la información mutua entre cada palabra y la variable de clase a la hora de construit clasificadores de texto. En su artículo, llamó a su algoritmo mejorado *weight adjusted k Nearest Neighbor*, *WAKNN*. Friedman [154] da la definición completa de información mutua entre cada par de variables.

De esta manera, cuando todos los atributos son nominales, la función de distancia ponderada por atributos puede definirse como:

$$d(x, y) = \sum_{i=1}^{n} I_P(A_i; C)\delta(a_i(x), a_i(y)) \quad (2.13)$$

donde $I_P(A_i; C)$ es la información mutua entre la variable de atributo Ai y la variable de clase , y $\delta(a_i(x), a_i(y)) = 0$ si $a_i(x) = a_i(y)$; y $\delta(a_i(x), a_i(y)) = 1$ de lo contrario.

Además, Huang [155] también presentó una función de distancia basada en la frecuencia, que denominó medida de no similitud, para abordar el problema de la agrupación en dominios categóricos. Ahora, esa función se describe de la siguiente manera:

$$d(x, y) = \sum_{i=1}^{n} \frac{F(a_i(x)) + F(a_i(y))}{F(a_i(x))F(a_i(y))}\delta(a_i(x), a_x(y)) \quad (2.14)$$

donde $F(a_i(x))$ y $F(a_i(y))$ son, respectivamente, un recuento del número de instancias de entrenamiento que tienen valor de atributo $a_i(x)$ y $a_i(y)$, y $\delta(a_i(x), a_i(y)) = 0$ si $a_i(x) = a_i(y)$; y $\delta(a_i(x), a_i(y)) = 1$ de lo contrario. Esta función de distancia es similar a la distancia *chi-cuadrado* [156]. La métrica de diferencia de valor (VDM) presentada por Stanfill y Waltz [157] es otra función de distancia apropiada para atributos nominales. Una versión simplificada del VDM (sin los esquemas de ponderación) puede definirse como:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}\sum_{c=1}^{c}(P(c|a_i(x)) - P(c|a_i(y))} \quad (2.15)$$

donde C es el número de salidas (clases); $P(c|a_i(x))$ es la probabilidad condicional de que la clase de salida sea c dado que el atributo A_i de x tiene el valor $\alpha_i(x)$; $P(c|a_i(y))$ es la probabilidad condicional de que la clase de salida sea c dado que el atributo A_i de y tiene el valor $a_i(y)$.

Distancia Manhattan

La función de distancia Manhattan mide la distancia recorrida siguiendo un camino de cuadrícula desde un punto a otro, sumando las diferencias de los componentes correspondientes. También se conoce como distancia City Block y representa la distancia en una ciudad con calles en forma de cuadrícula, analizando las diferencias entre las coordenadas de dos objetos.

La función de distancia Manhattan calcula la distancia que habría que recorrer para llegar de un punto al otro si se sigue un camino similar a una cuadrícula. La distancia Manhattan entre dos elementos es la suma de las diferencias de los componentes correspondientes [158]. Es también conocida como distancia *City Block*. Representa distancia entre puntos de una cuadrícula de calles de una ciudad. Se analiza completamente las diferencias entre coordenadas de un par de objetos.

La fórmula para calcular la distancia entre un punto $X_1 = (X_{i1}, X_{i2})$ y otro punto $X_2 = (X_{j1}, X_{j2})$ es:

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i1} - x_{j2}| + + |x_{ip} - x_{jp}| \quad (2.16)$$

donde X_i y X_j j son los valores de la i-ésima variable y de la j en los puntos X respectivamente.

Distancia Minkowski

La distancia de Minkowski puede ser una generalización tanto de la distancia Euclidiana como de la distancia Manhattan.

La distancia de Minkowski puede ser una generalización tanto de la distancia euclidiana [145] como

de la distancia Manhattan [158]. Esta se define como [158]:

$$d(i,j) = (|x_{i1} - x_{j1}|q + |x_{i1} - x_{j2}|q + + |x_{ip} - x_{jp}|q)^{\frac{1}{q}} \tag{2.17}$$

2.4.4. Algoritmos de Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (RNA) han dominado el aprendizaje automático y han logrado avances en una amplia gama de aplicaciones industriales como la visión por computadora y el procesamiento del lenguaje natural. Están basadas en la teoría del aprendizaje supervisado, que se originó en la teoría del entrenamiento de perceptrones y se inspiró en el cerebro.

El aprendizaje con Redes Neuronales Artificiales (RNA), o aprendizaje profundo, ha irrumpido como un marco dominante en el aprendizaje automático en la actualidad [159], lo que ha dado lugar a grandes avances en una amplia gama de aplicaciones industriales tales como la visión por computación [160] o procesamiento natural del lenguaje [161]. La teoría del aprendizaje supervisado tiene sus raíces en la teoría del entrenamiento de perceptrones, que, a su vez, se inspiró en elfuncionamiento del cerebro [162]; la arquitectura jerárquica [163] y el principio convolucional [164] estaban estrechamente vinculados a nuestros conocimientos sobre el sistema visual de los primates [165] [166]. En la actualidad, existe un continuo intercambio de ideas desde la neurociencia hasta el campo de la inteligencia artificial [167]. A continuación se muestra una imagen sobre la estructura que tiene una red neuronal:

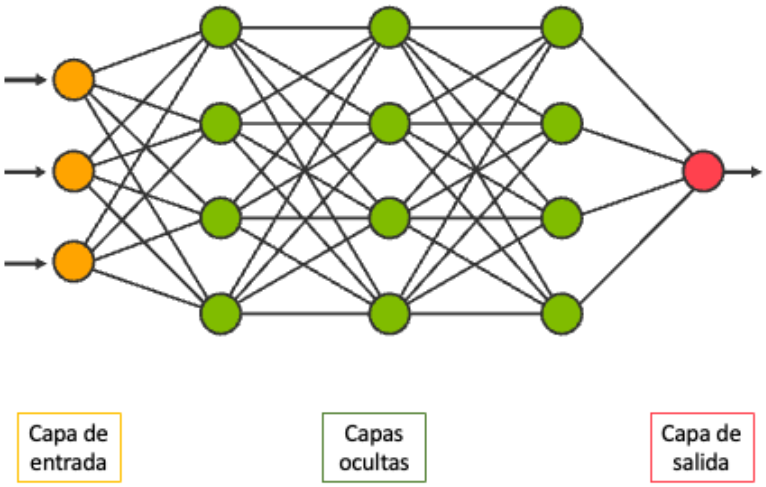

Figura 2.3: Estructura de una Red Neuronal

La arquitectura MLP es la más fundamental, consiste en múltiples capas de neuronas y sólo reciben entradas de la capa anterior y proyectan a la siguiente. Un estudio típico de RNA consta de aprendizaje de problemas, arquitectura de la red y algoritmo de entrenamiento.

Un estudio típico que utiliza redes neuronales consta de tres elementos básicos: el aprendizaje del problema, la arquitectura de la red y el algoritmo de entrenamiento. Los pesos de las conexiones entre unidades o neuronas en una red neuronal están limitados por la arquitectura de la red, pero sus valores específicos se asignan aleatoriamente en la inicialización [168]. Estas ponderaciones constituyen un gran número de parámetros, denominados colectivamente θ, que también incluyen otros parámetros del modelo, que deben entrenarse mediante un algoritmo. . El algoritmo de entrenamiento especifica cómo cambian los pesos de conexión para resolver mejor un problema de aprendizaje, para ajustarse a un conjunto de datos o realizar una tarea.

En el aprendizaje supervisado, un sistema aprende a ajustarse a un conjunto de datos que contiene una serie de instancias. Cada instancia se empareja con una salida objetivo [169]. Los símbolos en negrita representan vectores. El objetivo es aprender los parámetros 0 de una función de red neuronal que predice las salidas objetivo dadas las instancias de

entrada:

$$y^{(i)} = F(x^{(i)}, 0) \approx y^{(i)}_{target} \tag{2.18}$$

Más concretamente, el sistema se entrena para optimizar el valor de una función objetivo o, lo que es habitual, minimizar el valor de una función de pérdida [168]:

$$L = \frac{1}{N} \sum_i L(y^{(i)}, y^{(i)}_{target}) \tag{2.19}$$

Los pesos de las conexiones se limitan por la arquitectura de la red y se asignan aleatoriamente, pero se entrenan mediante un algoritmo para optimizar una función objetivo y predecir la salida objetivo.

donde, $L(y^{(i)}, y^{(i)}_{target})$ cuantifica la diferencia entre la salida objetivo $y^{(i)}_{target}$ y la salida real $y^{(i)}$. Las Redes Neuronales son increíblemente versátiles e incluyen una amplia gama de arquitecturas [170]. De todas las arquitecturas, la más fundamental es la arquitectura MLP. Esta arquitectura consiste en múltiples capas de neuronas, donde las neuronas de la capa (l) sólo reciben entradas de la capa (l -1) y sólo se proyectan a la capa (l + 1).

$$r^{(1)} = x, \tag{2.20}$$

$$r^{(l)} = f(W^l r^{l-1} + b^l), 1 < l < N, \tag{2.21}$$

$$y = W^{(N)} r^{N-1} + b^N \tag{2.22}$$

donde x es una entrada externa, r^l denota la actividad neuronal de las neuronas de la capa l, y W^l es la matriz de conexión de la capa *l-1* a la capa l. *f()* es la función de activación (normalmente no lineal) de las neuronas del modelo. La salida de la red se obtiene a través de la combinación de $W^{(N)}$, parámetros b^l y b^N que son los sesgos de las neuronas y de las unidades de salida, respectivamente. Si la red se entrena para clasificar, la salida suele normalizarse de forma que [168]:

$$\sum_j y_j = 1 \tag{2.23}$$

Las redes neuronales MLP (perceptrones multicapa) en teoría pueden calcular funciones arbitrarias, pero en la práctica están limitadas por el tamaño de la red. Se usan a menudo en combinación con otras arquitecturas más modernas o como parte de ellas.

donde, y_j representa la probabilidad estimada de la clase j. Cuando hay suficientes neuronas por capa, las redes neuronales con arquitectura MLP pueden, en teoría, calcular funciones arbitrarias [171]. Sin embargo, en la práctica, el tamaño de la red es limitado, y es posible que no se encuentren buenas soluciones mediante el entrenamiento aunque existan. La arquitectura MLP en redes neuronales se utiliza a menudo en combinación con arquitecturas de redes neuronales más modernas, o como parte de ellas.

En cuanto al entrenamiento, el método de aprendizaje característico en este tipo de redes neuronales es el descenso de gradiente estocástico (SGD) [172] [173]. Los parámetros entrenables, designados con-

juntamente como θ, se actualizan en la dirección opuesta del gradiente de la pérdida, $\delta L/\delta\theta$. De manera intuitiva, el parámetro j-ésimo de θ debe reducirse mediante el entrenamiento si la función de coste L aumenta [168]. Para cada paso del entrenamiento, como suele ser demasiado costoso evaluar la pérdida utilizando todo el conjunto de entrenamiento, la pérdida se calcula utilizando un pequeño número M de ejemplos de entrenamiento seleccionados aleatoriamente (un minilote), indexados por:

El método de entrenamiento típico en este tipo de redes es el descenso de gradiente estocástico (SGD), donde los parámetros se actualizan en la dirección opuesta del gradiente de la pérdida y se actualizan para reducir la pérdida si ésta aumenta. La pérdida se calcula utilizando un pequeño número (minilote) de ejemplos de entrenamiento seleccionados aleatoriamente.

$$\mathbb{B} = \{K_1, , k_M\}; L_{batch} = \frac{1}{M}\sum_{k\in\mathbb{B}} L(y^{(k)}, y^{(k)}_{target}), \tag{2.24}$$

Para simplificar, se asume un tamaño de minilote "*batch*" de 1 y se omite el lote en las siguientes ecuaciones.

A continuación, se va a proceder con el análisis crítico del estado del arte en el cual se va a examinar el estado y las limitaciones existentes en las investigaciones mencionadas en el estado del arte para de esta manera poder fijar los objetivos de investigación a posteriori.

2.5. Estudio crítico del estado del arte

Tras el estudio realizado sobre el estado del arte actual, cabe destacar que debido a la transformación digital masiva, se han realizado diferentes investigaciones acerca de las tecnologías que contribuyen en la Industria 4.0. Entre ellas, se han realizado

algunas en las cuales se han utilizado dispositivos hardware 'Edge computing' e incluso se han realizado comparativas entre estos dispositivos, pero siempre utilizando como datos de entrada imágenes y haciendo uso de la arquitectura *DNN* [11]. Además, se han realizado mediciones de rendimiento basadas en aspectos técnicos como uso de memoria, GPU, CPU, etc., pero se desconoce cuáles pueden llegar a ser los límites que estos dispositivos tienen a nivel de capacidad computacional cuando entrenan y validan conjuntos de datos. El límite para procesar un determinado algoritmo de inteligencia artificial, no tiene por qué ser el mismo que para procesar otro algoritmo de otra familia.

11: *Deep Neural Network:* Red Neuronal con más de dos capas de complejidad

El estudio del estado del arte de la Industria 4.0 ha revelado que hay una necesidad de evaluar el rendimiento de los dispositivos de hardware 'Edge computing' con algoritmos de inteligencia artificial diferentes a las *DNN*s y de diferentes tamaños de datos, tanto para entrenar como para validar modelos.

El límite también es variable dependiendo de si se va a utilizar el dispositivo para entrenar un conjunto de datos o únicamente se va a utilizar para validarlo. Es por eso, que tampoco existe información acerca de cómo se comportan estos dispositivos cuando procesan algoritmos de inteligencia artificial diferentes a las *DNN*s. De esta manera, existe la necesidad de evaluar los dispositivos "Edge" con el objetivo de conocer para qué situaciones específicas son útiles y para cuáles no (teniendo en cuenta tipo de algoritmo, tamaño de conjunto de datos, entrenamiento o validación). Esta evaluación requiere de realizar un estudio acerca del comportamiento que tiene cada familia de algoritmos de inteligencia artificial en este tipo de hardware cuando se enfrenta a diferentes tamaños de datos tanto a la hora de entrenar modelos como a la hora de validarlos.

Por otro lado, al tener estos dispositivos cierta limitación en cuanto a capacidad de computación en

comparación con los equipos de alta capacidad que se pueden utilizar en una arquitectura '*cloud*', es necesario centrar la investigación en estudiar la eficacia y precisión de modelos '*edge computing*' frente al tradicional '*cloud computing*'. De esta manera, se podría estudiar que a pesar de ser dispositivos limitados, se puede dividir un proceso modelizado con una arquitectura *cloud* en pequeños subprocesos que se modelicen en dispositivos '*edge*' y que mantienen la precisión del modelo tradicional, además de aportar las ventajas correspondientes al uso de la tecnología '*edge*'. Por último, tras conocer cuáles son los limites de los dispositivos '*edge*' con los conjuntos de datos de gran tamaño y tras haber validado la eficacia y precisión de estos modelos frente a modelos '*cloud*', es necesario estudiar técnicas que permitan modelizar procesos cuyo volumen de datos sea superior a los límites del dispositivo, ya que esto puede suponer en cierta medida un gran problema a la hora de hacer la transición de un modelo existente en '*cloud*' a '*edge*'. Para ello, es necesario estudiar técnicas de compresión de datos, ya que, el tamaño del conjunto de datos determina el rendimiento del algoritmo en este tipo de dispositivos. Hasta ahora, solo se han realizado estudios de cómo reducir la complejidad de redes neuronales. Ninguno de ellos se ha centrado en comprimir los datos de entrada para reducir la complejidad de ejecución del algoritmo. Por lo tanto, sería interesante poder comprimir los datos recogidos y que a la hora de entrenar modelos estos tuvieran la misma precisión que los modelos creados con los datos originales. De esta manera, en el caso de que el volumen de datos de la modelización de un subproceso industrial fuese superior al capaz de ser procesado por un

También es importante comparar la eficacia y precisión de modelos *Edge Computing* frente a los tradicionales *Cloud computing*, y estudiar técnicas de compresión de datos para reducir la complejidad de ejecución de algoritmos y poder utilizar la tecnología 'edge' en procesos con un volumen de datos superior a los límites del dispositivo.

dispositivo *'edge'*, en lugar de tener que utilizar una arquitectura *'cloud'*, se podría utilizar la arquitectura *'edge'* y así poder aprovechar las ventajas que esta tecnología ofrece.

2.6. Hipótesis y Objetivos

La pregunta investigación que se plantea es si la combinación de *Edge Computing* y técnicas de inteligencia artificial puede mejorar el desarrollo de gemelos digitales de partes del proceso de pintura en la industria de la automoción.

Teniendo identificada la problemática industrial actual y habiendo realizado el estudio crítico del estado del arte, se ha planteado la siguiente pregunta de investigación: *¿Puede la combinación de Edge Computing con técnicas de inteligencia artificial modelizar con suficiente precisión y mejorar el desarrollo de gemelos digitales de partes del proceso de pintura de una planta de producción del sector de la automoción?*

Los objetivos que se plantean para responder a la hipótesis son:

1. Analizar el comportamiento y las restricciones que los dipositivos *Edge Computing* tienen a la hora de procesar diferentes familias de algoritmos de inteligencia artificial.
2. Validación de la eficacia y precisión de pequeños modelos *Edge* representando subprocesos de un proceso industrial con respecto al modelo *cloud* que representa el proceso completo.
3. Comprimir conjuntos de datos de manera significativa manteniendo la precisión de los modelos generados a partir del conjunto de datos original.

Estos objetivos se han definido debido a la necesidad que existe de reducir los tiempos de cálculo y predicción del resultado del proceso para hacer frente a los retos tecnológicos de alto nivel que

requieren respuestas rápidas en un mundo cada vez más impulsado por los datos como es el sector de la automoción. Es por eso, que el primero de los objetivos descritos consiste en conocer en profundidad esta nueva tecnología para identificar los límites y el comportamiento de cada algoritmo en estos dispositivos tanto a la hora de entrenar como a la hora de validar modelos. Con este estudio se quiere facilitar la identificación del hardware a utilizar dependiendo del escenario en el que nos encontremos. Ese escenario contempla 3 variables que se van a analizar en el estudio:

- **Volumen de datos:** Tamaño del conjunto de datos que se quiere procesar.
- **Algoritmo a emplear:** Redes neuronales, árboles de decisión, regresión lineal, algoritmos de agrupación y algoritmos de tipo *Bayesiano*.
- **Tipo de procesamiento:** Entrenamiento o validación.

Para responder a la pregunta de investigación, se han establecido tres objetivos: analizar el comportamiento de los dispositivos Edge Computing con diferentes algoritmos de inteligencia artificial, validar la eficacia de pequeños modelos *Edge* que representen subprocesos de un proceso industrial y comprimir conjuntos de datos de manera significativa manteniendo la precisión y eficacia de los modelos.

Por otro lado, el segundo de los objetivos quiere demostrar que a pesar de ser dispositivos limitados, pueden facilitar la tarea de dividir un proceso modelizado con una arquitectura *cloud* en pequeños subprocesos que se modelicen en dispositivos *edge* y que no solo mantienen la eficacia del modelo tradicional, sino que además aportan las ventajas correspondientes al uso de esta tecnología.

Finalmente, unas vez completados los dos primeros objetivos, como una de las restricciones del *Edge Computing* depende directamente del volumen de datos que se quiere procesar, el último objetivo busca realizar una compresión de un conjunto de datos, siendo el conjunto resultante capaz de generar un modelo que mantenga la precisión y

eficacia del modelo creado con los datos originales. Es decir, el objetivo es entrenar modelos con un volumen de datos considerablemente menor pero sin que la precisión del modelo se vea afectada. De esta manera, en caso de que el volumen de datos de la modelización de un proceso industrial fuese superior al capaz de ser procesado por un dispositivo *edge*, en lugar de tener que utilizar una arquitectura *cloud*, se podría utilizar la arquitectura *edge*.

Estos objetivos tienen como fin reducir los tiempos de cálculo y predicción del proceso y permitir el uso de la arquitectura *Edge Computing* en lugar de la arquitectura *Cloud* en caso de ser necesario.

Análisis hardware 3

3.1. Experimentación

3.1 Experimentación . 59
3.1.1 Pasos de la experimentación 60
3.1.2 Random Forest . . 63
3.1.3 Logistic Regression 67
3.1.4 K Nearest Neighbor 72
3.1.5 Neural Network . . 76
3.1.6 Gaussian Naive Bayes 80
3.2 Resultados obtenidos 84
3.2.1 Resultados obtenidos por algoritmo . 85
3.2.2 Resultados obtenidos por tamaño de archivo 88
3.3 Conclusiones . . . 90

Esta experimentación se ha centrado no solo en conocer el rendimiento y los límites de los nuevos hardware de IoT que han salido al mercado y que permiten llevar a cabo el tan esperado 'Edge Computing', sino en medir la capacidad de los hardware para entrenar y validar modelos de forma paralela en el contexto de una cadena de producción con el objetivo de definir una arquitectura 'Edge'. Es decir, conociendo tanto el volumen de datos que genera cada estación de producción y la familia de algoritmos que se emplean en dicha estación, gracias a las pruebas de rendimiento realizadas, se puede estimar el tiempo necesario para realizar tanto el entrenamiento como la validación de los modelos y de esta manera, definir la estrategia 'Edge' para cada línea/estación de producción.

Además, conociendo los límites y capacidades máximas de procesamiento de los hardware, se podrá saber el nº de dispositivos hardware que hay que instalar en cada línea/estación. Se ha probado el comportamiento de diferentes familias de algoritmos (árboles de decisión, clasificación, redes neuronales, regresión, redes bayesianas) con el objetivo de crear un modelo de comportamiento para cada par algoritmo-hardware, y de esta manera, poder predecir cómo se va a comportar cada uno de ellos en los diferentes hardware 'Edge computing'. Los dispositivos hardware 'Edge Computing' utilizados en esta investigación son los siiguientes:

Esta experimentación se ha enfocado en medir el rendimiento y los límites de los nuevos hardware de IoT para llevar a cabo Edge Computing y determinar su capacidad para entrenar y validar modelos de forma paralela en una cadena de producción. Se han probado diferentes algoritmos para crear un modelo de comportamiento para cada par algoritmo-hardware y así predecir su comportamiento en diferentes hardware *Edge Computing*. Los dispositivos utilizados en la investigación son Nvidia Jetson Nano y Google Coral, ambos con una potencia mayor a un Mini PC pero con un consumo energético similar.

Nvidia Jetson Nano [45] y Google Coral [174]. Ambos hardware, a pesar de ser mucho más potentes que un Mini PC, no tiene un consumo energético mayor. El hardware Jetson Nano [45] tiene una CPU ARM Cortex-A57 MPCore de 4 núcleos, una GPU Nvidia Maxwell con 128 núcleos CUDA y una RAM de 4 GB. Por otro lado, el hardware Google Coral [174]], se basa en un pequeño chip Edge TPU (acelerador de inteligencia artificial). Además, contiene un microcontrolador ARM Cortex-M4F, una CPU ARM Cortex-A53 de cuatro núcleos y una GPU Vivante GC7000 Lite.

3.1.1. Pasos de la experimentación

A continuación, se realizará una breve descripción sobre las características del conjunto de datos utilizado en la experimentación, se explicará cómo se han procesado los datos, las pruebas de rendimiento realizadas y, finalmente, se indicará paso a paso la generación del modelo de comportamiento de cada par algoritmo-hardware.

Se utilizó un conjunto de datos público de la empresa Bosch para la experimentación, compuesto por datos adquiridos durante el proceso de fabricación de piezas.

Conjunto de datos

Para poder llevar a cabo la experimentación, se ha utilizado un conjunto de datos voluminoso (14,3 GB) de carácter público de la empresa Bosch [175]. El conjunto de datos lo componen piezas fabricadas con datos adquiridos durante el proceso de fabricación. Cada pieza del conjunto de datos tiene un identificador único y el objetivo es predecir qué piezas no pasarán el control de calidad (Representado como "Respuesta" = 1). El conjunto de datos

contiene datos numéricos, datos categóricos, fechas, y la etiqueta que indica si la pieza ha pasado el control de calidad o no. Tras un análisis más detallado del conjunto de datos, se ha descubierto que hay 2140 columnas categóricas, de las cuales 150 están vacías y no contienen información, por lo que se han eliminado. Además, de las columnas categóricas, 1490 de ellas son de valor único y 500 de valor múltiple. El resto de las columnas, se han convertido en columnas numéricas mediante la técnica "One hot Encoding" [176]. Por otro lado, el nombre de las columnas numéricas contiene implícitamente información acerca de la estación, línea de producción y el nº de test al que corresponde la columna. Por ejemplo, una columna llamada 'L2_S32_F3526' en un componente, indica que se encuentra en la línea de producción 2, en la estación 32, y el valor de la columna corresponde al nº de test 3526.

Se han eliminado 150 de las 2140 columnas categóricas debido a que no contenían información. Además, 1490 de las columnas categóricas eran de valor único y 500 de valor múltiple. Las columnas restantes se convirtieron en columnas numéricas mediante la técnica "One hot Encoding". También se han identificado las estaciones, líneas de producción y el número de test en el nombre de las columnas numéricas, y se han registrado 1157 columnas de tipo fecha en el conjunto de datos.

Por último, hay 1157 columnas de tipo fecha en el conjunto de datos. Estas columnas están formadas por la línea de producción, identificador de estación y fecha. Por ejemplo, 'L2_S21_D3254' indica que la pieza ha ido a través de la linea de producción 2, estación 21 y el valor corresponde al identificador de fecha 3254.

Procesamiento de datos

Para llevar a cabo la experimentación, el primer paso ha sido dividir el conjunto de datos procesado en diferentes tamaños: 5 MB, 100 MB, 250 MB, 350 MB, 500 MB, 700 MB y 900 MB. Con esta división se pretende estudiar el impacto del volumen de datos

en la capacidad de entrenamiento y predicción en los diferentes hardware.

Se ha analizado el rendimiento de cada uno de los algoritmos (Random Forest, Logistic Regression, K Nearest Neighbors, Neural Network, Gaussian Naive Bayes) con los diferentes tamaños del conjunto de datos en cada hardware 'Edge Computing'. Con esto, lo que se pretende es generar modelos de comportamiento para cada par algoritmo-hardware lo más representativo posible. Para cada par algoritmo-tamaño de datos se ha realizado 3 veces la experimentación y, por lo tanto, se han tomado 3 mediciones. Para el resultado final se ha cogido la media de las 3 mediciones.

En la experimentación, se ha dividido el conjunto de datos procesado en diferentes tamaños y se ha analizado el rendimiento de cada uno de los algoritmos (*Random Forest*, *Logistic Regression*, *K Nearest Neighbors*, *Neural Network*, *Gaussian Naive Bayes*) en cada hardware *Edge Computing*. Se han realizado tres mediciones para cada par algoritmo-tamaño de datos y se ha tomado la media de las mediciones para obtener el resultado final. El objetivo era generar modelos de comportamiento para cada par algoritmo-hardware lo más representativo posible.

Pruebas de rendimiento

Para llevar a cabo la experimentación, el primer paso ha sido dividir el conjunto de datos procesado en diferentes tamaños: 5 MB, 100 MB, 250 MB, 350 MB, 500 MB, 700 MB y 900 MB. Con esta división se pretende estudiar el impacto del volumen de datos en la capacidad de entrenamiento y predicción en los diferentes hardware. Se ha analizado el rendimiento de cada uno de los algoritmos (Random Forest, Logistic Regression, K Nearest Neighbors, Neural Network, Gaussian Naive Bayes) con los diferentes tamaños del conjunto de datos en cada hardware 'Edge Computing'. Con esto, lo que se pretende es generar modelos de comportamiento para cada par algoritmo-hardware lo más representativo posible. Para cada par algoritmo-tamaño de datos se ha realizado 3 veces la experimentación y, por lo tanto, se han tomado 3 mediciones. Para

el resultado final se ha cogido la media de las 3 mediciones.

En esta sección se describe cómo se han llevado a cabo las pruebas de rendimiento, dividiendo el conjunto de datos en diferentes tamaños y analizando el rendimiento de cinco algoritmos (*Random Forest, Logistic Regression, K Nearest Neighbors, Neural Network* y *Gaussian Naive Bayes*) en diferentes hardware de *Edge Computing*. Se ha tomado la media de las tres mediciones por cada par algoritmo-tamaño de datos para obtener los resultados finales. A partir de los resultados se han generado modelos de comportamiento para cada par algoritmo-hardware. Cada algoritmo tiene su correspondiente análisis, tabla de resultados y gráficos.

Experimentación

Una vez obtenidas las mediciones, se han generado modelos, que veremos a continuación, a través de los resultados mostrados en las tablas. Para cada algoritmo se ha generado un apartado con su correspondiente análisis, tabla de resultados y gráficos.

3.1.2. Random Forest

A continuación, la Tabla 3.1 muestra los resultados de los tiempos de procesamiento tanto para entrenar como para validar diferentes conjuntos de datos en los dispositivos hardware Jetson Nano y Google Coral.

Tabla 3.1: Tiempos de rendimiento para Random Forest

RANDOM FOREST				
Hardware	**Nvidia Jetson Nano**		**Google Coral**	
File Size	*Train Time (s)*	*Test Time (s)*	*Train Time (s)*	*Test Time (s)*
5 MB	13,74	0,63	21,52	1,22
100 MB	277	3,99	541,41	6,46
250 MB	1069,8	15,63	2889,35	28,46
350 MB	1449,8	20,43	2184,07	22,94
500 MB	2343,14	35,91	4661,28	48,35
700 MB	3628,62	56,31		
900 MB	4777,25	79,12		

Ahora, en las siguientes secciones se analizará cada dispositivo de manera independiente para acabar realizando una comparativa entre ambos dispositivos. Este patrón se repetirá para cada uno de los algoritmos.

Random Forest - Jetson Nano

La Figura 3.1 muestra el comportamiento del entrenamiento y la validación del algoritmo *Random Forest* en el hardware Jetson Nano. El tiempo de procesamiento aumenta exponencialmente cuando el archivo procesado supera los 100 MB.

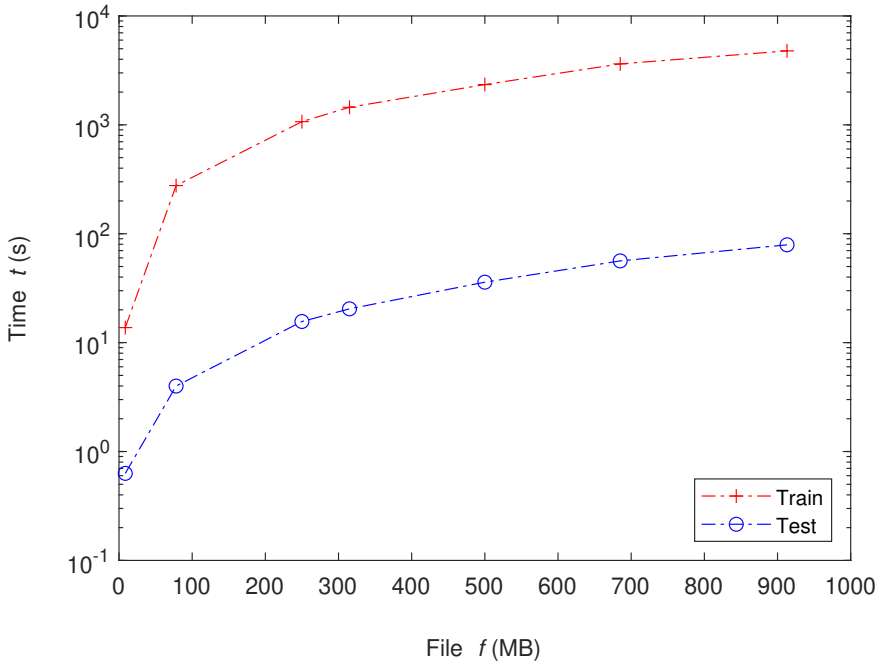

Figura 3.1: Gráfica tiempos de entrenamiento y test para Random Forest en Jetson Nano

La Figura 3.1 muestra como en este hardware tanto el entrenamiento como la validación se comportan de manera similar indiferentemente del volumen de datos que se procese. Cuando el archivo a procesar supera los 100 MB, el tiempo de procesamiento se incrementa exponencialmente tanto para el entrenamiento como para el test. Para realizar el entrenamiento de un archivo relativamente pequeño de 5 MB, el tiempo de procesamiento es algo mayor a 13 segundos, siendo el modelo que define el comportamiento en el dispositivo Jetson Nano el siguiente:

$$t(size) = a_i * size^{b_i} + c_i \tag{3.1}$$

$$a_i = 1976, b_i = 1145, c_i = 0$$

Donde *t(size)* será el tiempo estimado para procesar el archivo de tamaño *size* (MB). Por otro lado, el

test en este tipo de algoritmo y archivo (5MB) se procesa en poco más de medio segundo, y el modelo de comportamiento es:

$$t(size) = a_i * size^{b_i} + c_i \quad (3.2)$$

$$a_i = 0012, b_i = 1288, c_i = 0505$$

Random Forest - Google Coral

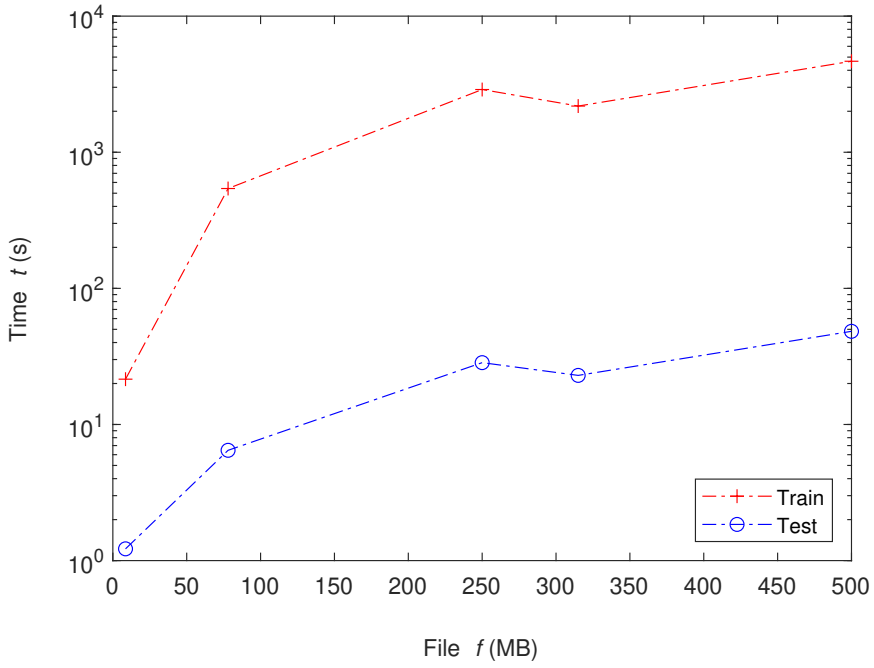

Figura 3.2: Gráfica tiempos de entrenamiento y test para Random Forest en GoogleCoral

En cuanto al comportamiento del hardware Google Coral frente al algoritmo Random Forest, en la Figura 3.2 puede observarse que es diferente al del hardware Jetson Nano. Sin embargo, es cierto que al igual que pasa en Jetson Nano, el comportamiento del entrenamiento y del test son similares aunque con la diferencia de tiempo. Cabe destacar el pico que existe entre los archivos de 200MB y 300 MB tanto para entrenamiento como para validación.

El comportamiento del hardware *Google Coral* con el algoritmo Random Forest reflejado en la Figura 3.2, muestra una diferencia en comparación con el hardware Jetson Nano. A pesar de esto, se observa una similitud en el comportamiento del entrenamiento y la validación en ambos casos, aunque con sus respectivas diferencias de tiempo. Es destacable el pico en los archivos de 200MB y 300MB para ambas fases.

Las funciones que mejor definen el comportamiento para el entrenamiento y el test son las siguientes:

$$t(size) = a_i * size^{b_i} + c_i \tag{3.3}$$
$$a_i = 953, b_i = 0995, c_i = -7566$$

$$t(size) = a_i * size^{b_i} + c_i \tag{3.4}$$
$$a_i = 0058, b_i = 1076, c_i = 0727$$

Random Forest - Jetson Nano vs Google Coral

En las Figuras 3.3 y 3.4 se puede observar la comparativa de entrenamiento y de test entre los dos hardware. Cabe destacar que el hardware Google Coral solo tiene graficado el tiempo hasta archivos de 500 MB, debido a que es el tamaño máximo de archivo que se puede procesar tanto a la hora de entrenar como a la hora de realizar test.

Las figuras 3.3 y 3.4 muestran una comparación del entrenamiento y test entre los dos hardware. En la comparativa del entrenamiento, se observa que el hardware *Google Coral* siempre tarda más tiempo en procesar archivos que el hardware *Jetson Nano*, con una diferencia mayor cuando el archivo supera los 100 MB. En el test, ocurre lo mismo en ambos hardware, y en el caso de *Jetson Nano*, el tiempo de procesamiento aumenta exponencialmente cuando el archivo es mayor a 100 MB. También se destaca un pico entre los archivos de 200MB y 300MB en el hardware *Google Coral* para el entrenamiento y la validación.

En la Figura 3.3, donde se ve la comparativa del entrenamiento, puede apreciarse que el hardware Google Coral siempre tarda más tiempo en procesar los archivos, independientemente del volumen de estos. Así mismo, cuanto más pequeño el archivo, más pequeña es la diferencia de tiempo entre ambos hardware, pero en cuanto el archivo supera los 100 MB de tamaño, la diferencia de tiempos empieza a ser considerablemente mayor. En el test (Figura 3.4) sucede exactamente lo mismo. En el hardware Jetson Nano, el tiempo de procesamiento se incrementa exponencialmente tanto en el entrenamiento como en el test cuando el archivo que

se está procesando es mayor a 100 MB. En el hardware Google Coral cabe destacar el pico que surge entre los archivos de 200MB y 300 MB tanto para entrenamiento como para validación.

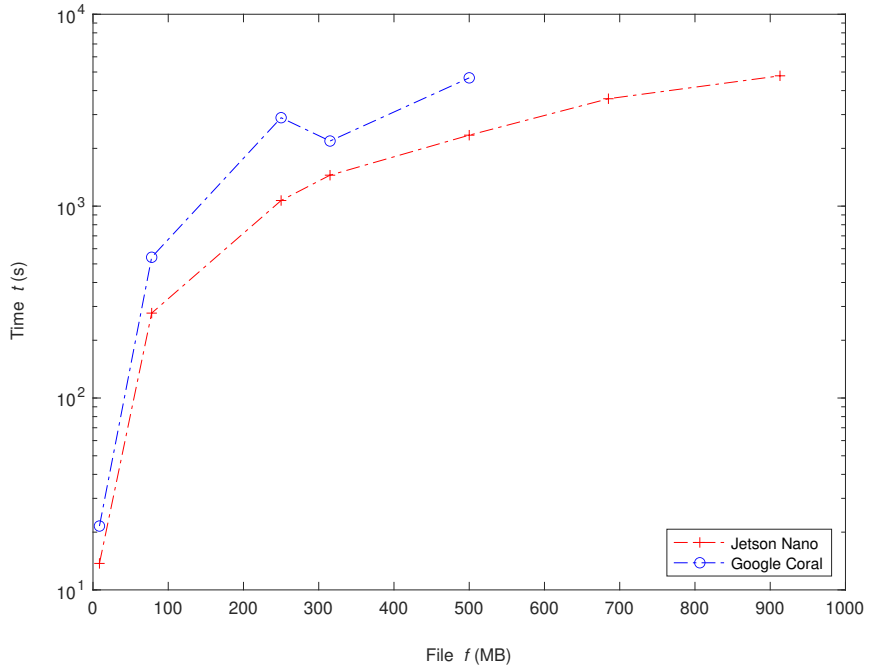

Figura 3.3: Gráfica tiempos de entrenamiento de Jetson Nano y Google Coral con Random Forest

3.1.3. Logistic Regression

A continuación, los resultados del algoritmo *Logistic Regression* en la Tabla 3.2

Tabla 3.2: Tiempos de rendimiento para Logistic Regression

LOGISTIC REGRESSION				
Hardware	**Nvidia Jetson Nano**		**Google Coral**	
File Size	*Train Time (s)*	*Test Time (s)*	*Train Time (s)*	*Test Time (s)*
5 MB	0,98	9,48	1,33	12,68
100 MB	10,95	94,45	12,32	108,36
250 MB	29,88	283,37	44,39	417,24
350 MB	45,28	447,87	33,88	316,12
500 MB	58,22	614,64	65,07	616,77
700 MB	133,33	878,93		
900 MB	176,1	1173,27		

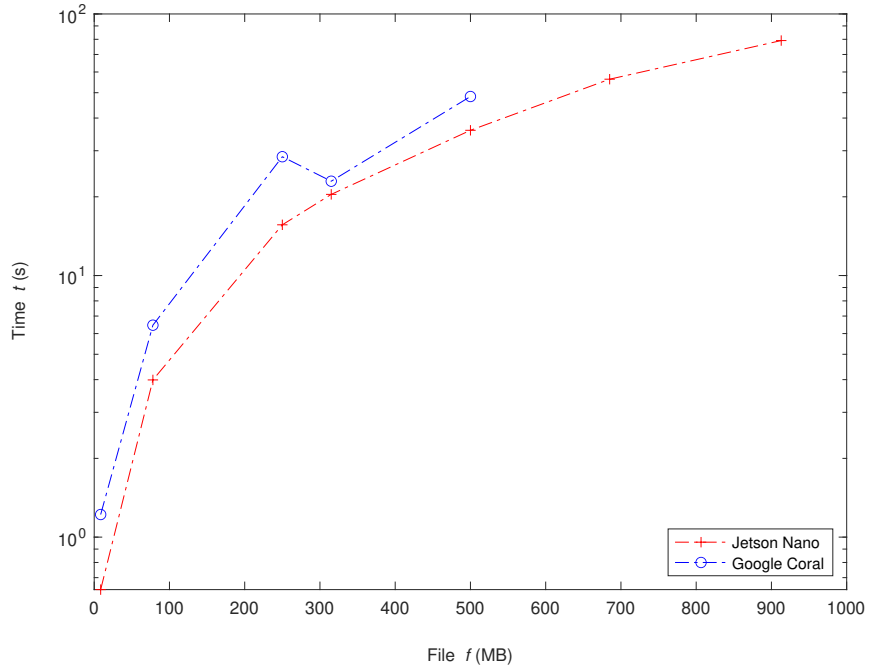

Figura 3.4: Gráfica tiempos de test de Jetson Nano y Google Coral con Random forest

Logistic Regression - Jetson Nano

El comportamiento del hardware Jetson Nano en el caso del algoritmo Logistic Regression (Fig. 3.5) es completamente diferente al que se ha podido ver en la Fig. 3.1 con el algoritmo Random Forest.

Como se ha mencionado, al contrario de lo que ocurre con el algoritmo Random Forest, el tiempo de procesamiento del test es mayor que el tiempo de entrenamiento, y los modelos que representan el comportamiento del entrenamiento y del test en el hardware Jetson Nano respectivamente son los siguientes:

En el algoritmo *Logistic Regression* pasa exactamente lo contrario que con el algoritmo *Random Forest*, es decir, el tiempo de procesamiento del test es mayor que el del entrenamiento.

$$t(size) = a_i * size^{b_i} + c_i \tag{3.5}$$

$$a_i = 0007, b_i = 1486, c_i = 3473$$

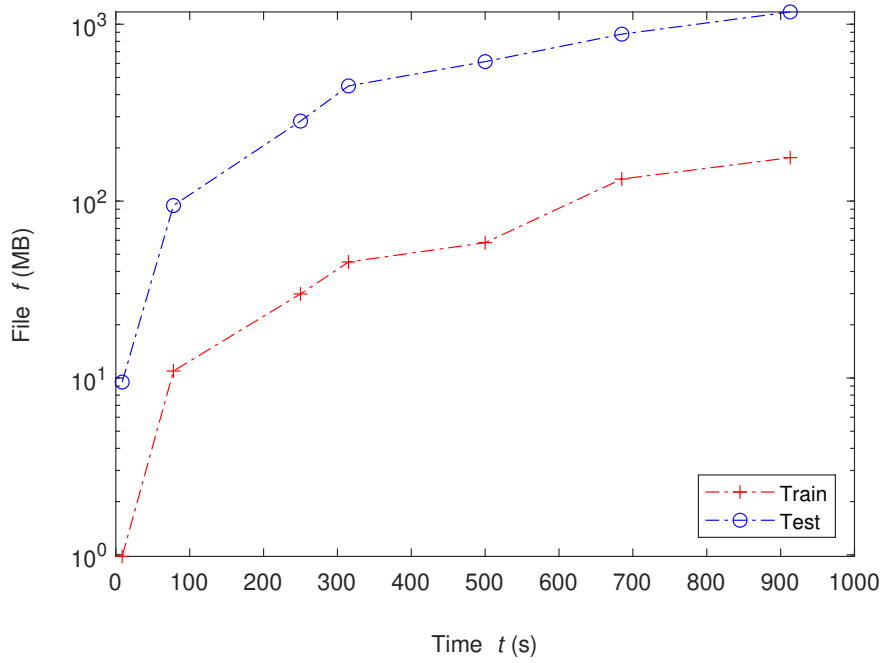

Figura 3.5: Gráfica tiempos de entrenamiento y test para Logistic Regression en Jetson Nano

$$t(size) = a_i * size^{b_i} + c_i \tag{3.6}$$

$$a_i = 1201, b_i = 101, c_i = -2158$$

Logistic Regression - Google Coral

A continuación, el comportamiento en Google coral:

En la Fig. 3.6 puede apreciarse que sucede lo mismo que en el hardware Jetson Nano (Fig. 3.5): el tiempo de procesamiento de test es mayor que el de entrenamiento. Además, tanto el entrenamiento como el test se comportan de manera idéntica con diferentes tamaños de archivos, y por tanto, los modelos que mejor representan el comportamiento del entrenamiento y de test respectivamente son los siguientes:

La Figura 3.6 muestra que ocurre lo mismo que en el hardware *Jetson Nano* (Figura 3.5): el tiempo de procesamiento del test es mayor que el del entrenamiento. El comportamiento del entrenamiento y el test es similar para diferentes tamaños de archivos.

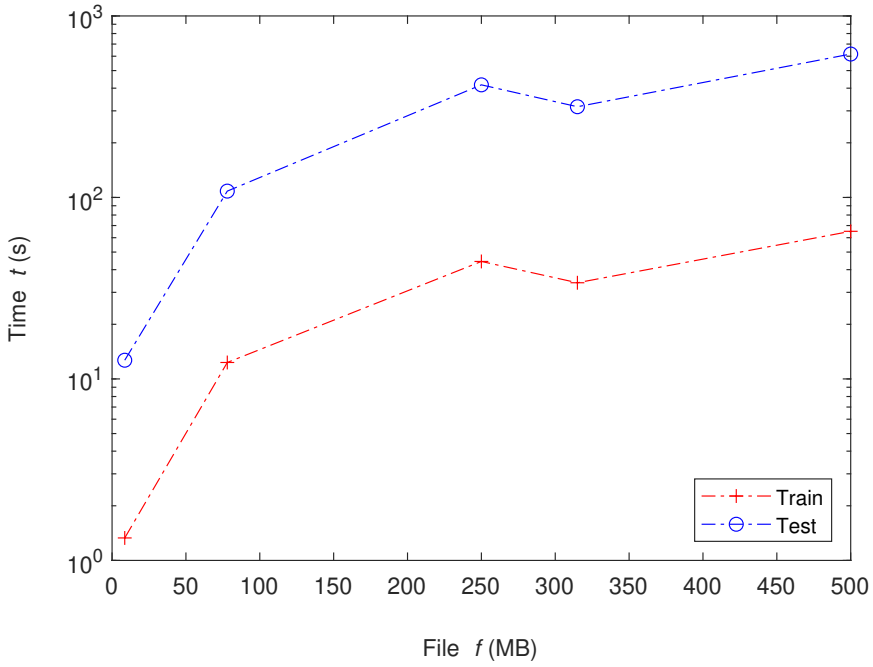

Figura 3.6: Gráfica tiempos de entrenamiento y test para Logistic Regression en Google Coral

$$t(size) = a_i * size^{b_i} + c_i \tag{3.7}$$
$$a_i = 0438, b_i = 0803, c_i = -1162$$

$$t(size) = a_i * size^{b_i} + c_i \tag{3.8}$$
$$a_i = 3431, b_i = 0833, c_i = -9134$$

Logistic Regression - Jetson Nano vs Google Coral

Las Figuras 3.7 y 3.8 muestran que ambos hardware tienen un comportamiento similar en el entrenamiento y en el test, aunque el hardware *Jetson Nano* es más rápido y tarda menos tiempo en procesar los archivos. Sin embargo, para archivos de alrededor de 350 MB, el hardware Google Coral es más rápido.

Tanto en la Fig. 3.7 como en la 3.8 se puede observar que ambos hardware se comportan de manera similar en el entrenamiento y en el test, aunque el hardware Jetson Nano es más rápido ya que requiere de menos tiempo para procesar los archivos. Sin embargo, cabe destacar un caso donde Google Coral obtiene un mejor tiempo de procesamiento,

y es cuando el archivo ronda los 350 MB.

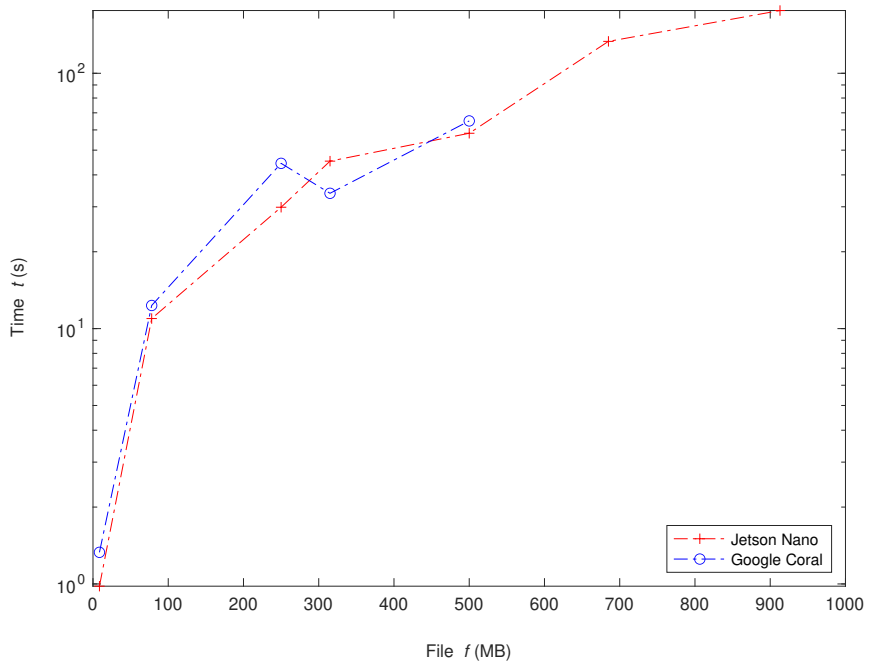

Figura 3.7: Gráfica tiempos de entrenamiento de Jetson Nano y Google Coral con Logistic Regression

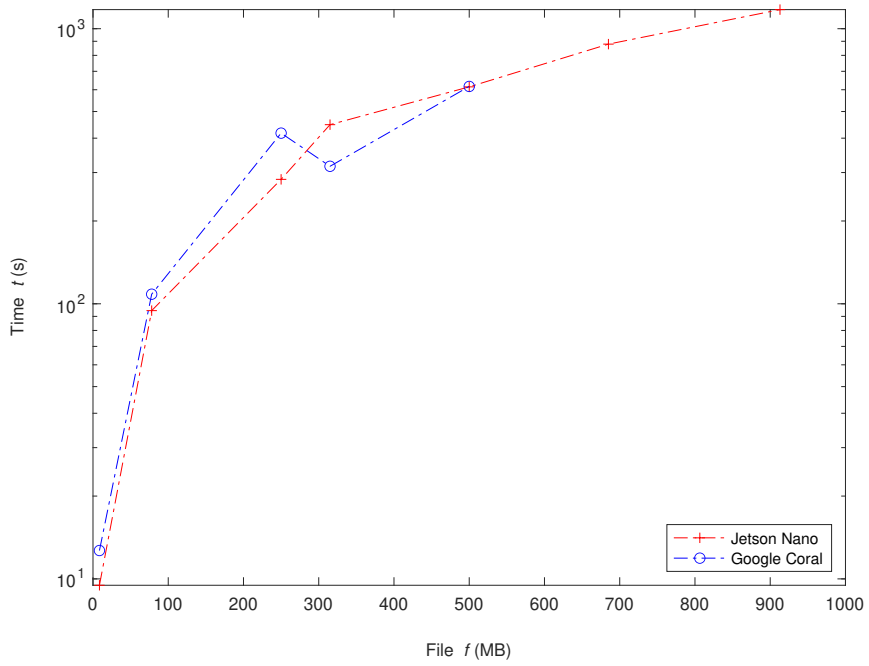

Figura 3.8: Gráfica tiempos de test de Jetson Nano y Google Coral con Logistic Regression

3.1.4. K Nearest Neighbor

Los resultados del algoritmo K Nearest Neighbor se muestran en la Tabla 3.3.

Tabla 3.3: Tiempos de rendimiento para K Nearest Neighbor

K NEAREST NEIGHBOR

Hardware	**Nvidia Jetson Nano**		**Google Coral**	
File Size	*Train Time (s)*	*Test Time (s)*	*Train Time (s)*	*Test Time (s)*
5 MB	0,03	15,08	0,03	4,25
100 MB	0,64	1625,06	0,13	383,53
250 MB	0,26	14334,43	0,47	
350 MB	0,32	23404,81	1,12	
500 MB	0,59	61069,39	2,2	
700 MB	6,01	177787,25		
900 MB	29			

K Nearest Neighbor - Jetson Nano

Al igual que en el algoritmo *Logistic Regression*, el tiempo de procesamiento del test es mayor que el del entrenamiento. Esto se debe a la complejidad del algoritmo *K Nearest Neighbor*.

En la Fig. 3.9 se aprecia que al igual que pasa con el algoritmo Logistic Regression, procesar el test requiere de mayor tiempo en comparación con el entrenamiento. Esto sucede debido a la complejidad del algoritmo K Nearest Neighbor a la hora de realizar el test, ya que lo que hace es calcular la distancia entre los vectores almacenados (creados con el entenamiento) y el nuevo vector, y se seleccionan los k vectores más cercanos [133].

Debido a la complejidad del algoritmo *K Nearest Neighbor* para realizar el test, no solo aumenta exponencialmente el tiempo de procesamiento a medida que aumenta el tamaño del archivo, sino que el hardware no puede procesar el test para archivos superiores a 700 MB. Sin embargo, en este hardware es posible entrenar archivos superiores a 700 MB.

Debido a la complejidad recientemente mencionada a la hora de realizar el test, no solo se incrementa exponencialmente el tiempo de procesamiento a medida que va aumentando el tamaño del archivo, sino que no puede procesar el test para archivos superiores a 700 MB. Esto significa que en este hardware sí que se puede entrenar archivos superiores a 700 MB, pero no procesar test. A continuación, el modelo que representa el comportamiento del entrenamiento y del test respectivamente:

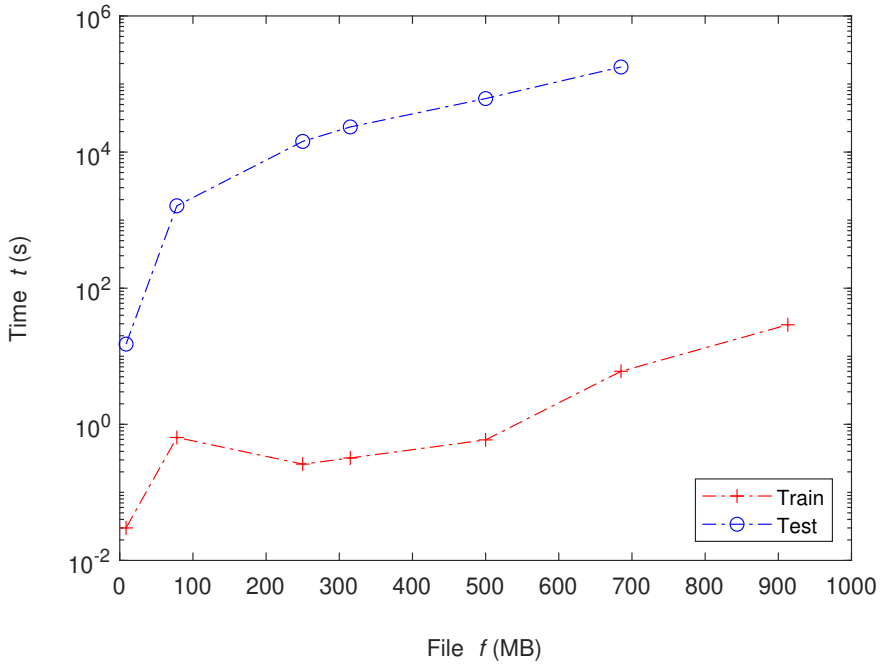

Figura 3.9: Gráfica tiempos de entrenamiento y test de Jetson Nano con K Nearest Neighbor

$$t(size) = a_i * size^{b_i} + c_i \tag{3.9}$$
$$a_i = 994e^{-16}, b_i = 5562, c_i = 0$$

$$t(size) = a_i * size^{b_i} + c_i \tag{3.10}$$
$$a_i = 00004, b_i = 3043, c_i = 0$$

K Nearest Neighbor - Google Coral

Como se puede ver en la Fig. 3.10, en este hardware sucede lo mismo que en el anterior, que el tiempo para procesar el test es superior al del entrenamiento. Además, en este dispositivo solo se puede procesar test de archivos iguales o inferiores a 100 MB. Sin embargo, el limite del entrenamiento llega hasta los 500 MB. Los modelos de comportamiento

En el hardware *Google Coral* también se requiere más tiempo para procesar el test que para el entrenamiento, al igual que en el hardware Jetson Nano. Además, en este caso el tamaño máximo de archivo para el test es de 100 MB, mientras que para el entrenamiento llega hasta los 500 MB.

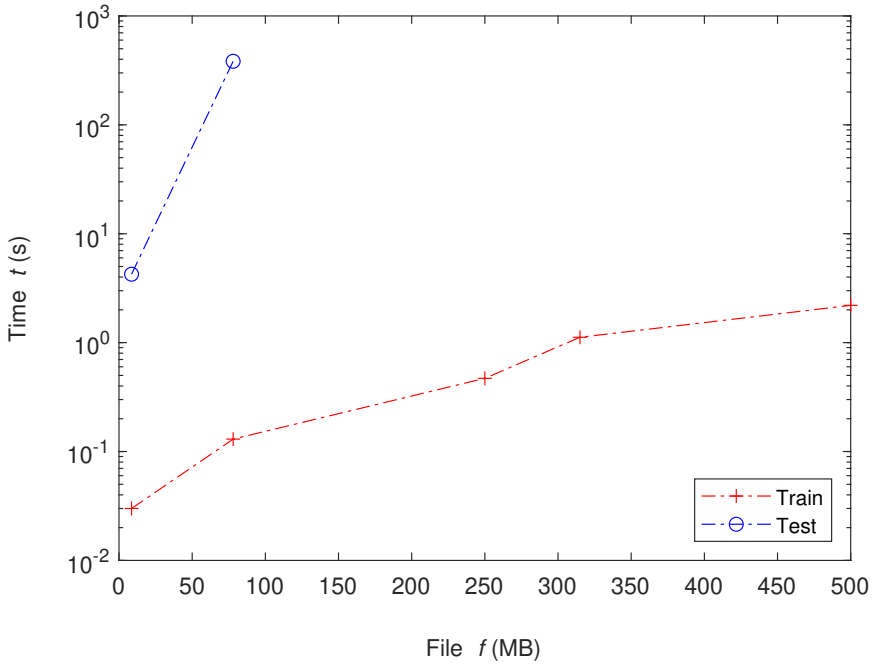

Figura 3.10: Gráfica tiempos de entrenamiento y test de Google Coral con K Nearest Neighbor

para el entrenamiento y test son los siguientes:

$$t(size) = a_i * size^{b_i} + c_i \tag{3.11}$$

$$a_i = 236e^{-5}, b_i = 1841, c_i = 0026$$

$$t(size) = a_i * size^{b_i} + c_i \tag{3.12}$$

$$a_i = 0052, b_i = 204, c_i = 0$$

K Nearest Neighbor - Jetson Nano vs Google Coral

El hardware *Jetson Nano* es más rápido que el *Google Coral* a partir de archivos de 200 MB, mientras que para archivos más pequeños, el hardware *Google Coral* es el que tiene un mejor rendimiento. En general, ambos dispositivos ofrecen un rendimiento similar en términos de tiempo y entrenamiento para el algoritmo *K Nearest Neighbor*.

En lo que a la comparativa de entrenamiento que se observa en la Fig. 3.11 se refiere, el hardware Jetson Nano ofrece un mejor rendimiento en cuanto a tiempo a la hora de procesar achivos superiores a 200 MB. Sin embargo, cuando el archivo es menor a 200 MB y se quiere realizar un entrenamiento, es el hardware Google Coral el que lo haría de manera más rápida.

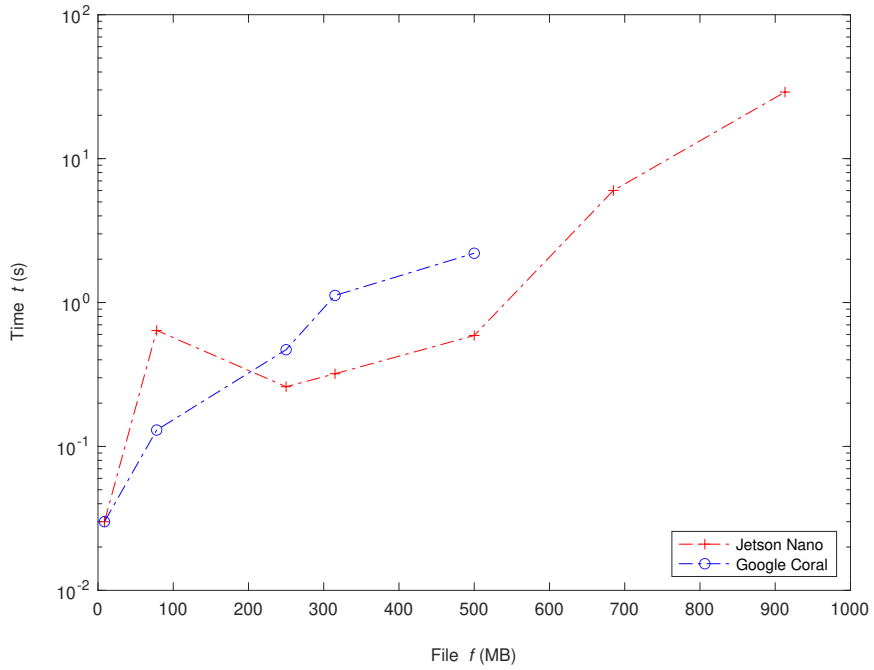

Figura 3.11: Gráfica tiempos de entrenamiento de Jetson Nano y Google Coral con K Nearest Neighbor

Por otro lado, en lo referente a la comparativa del test, en la Fig. 3.12, Google Coral es el dipositivo que procesa los archivos más rápidos siempre y cuando sean igual o inferiores a 100 MB. Para archivos superiores solo se podría realizar el test en el dispositivo Jetson Nano.

En cuanto al test con el algoritmo *K Nearest Neighbor*, el dispositivo *Google Coral* es más rápido para archivos iguales o inferiores a 100 MB, mientras que para archivos superiores solo es posible realizar el test en el hardware *Jetson Nano*.

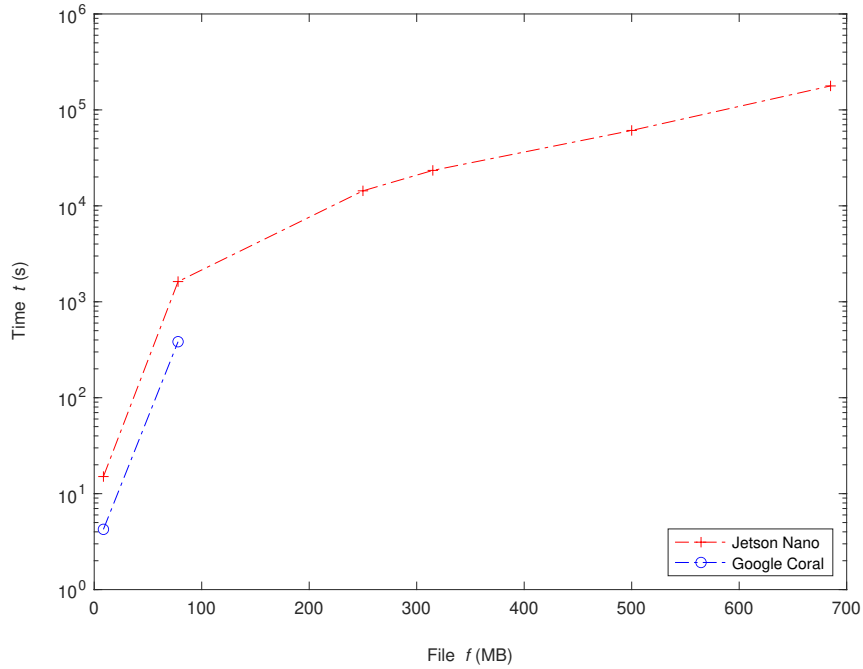

Figura 3.12: Gráfica tiempos de test de Jetson Nano y Google Coral con K Nearest Neighbor

3.1.5. Neural Network

En la siguiente tabla (3.4) se muestran los resultados del algoritmo *Neural Network*:

Tabla 3.4: Tiempos de rendimiento para Neural Network

NEURAL NETWORK				
Hardware	**Nvidia Jetson Nano**		**Google Coral**	
File Size	*Train Time (s)*	*Test Time (s)*	*Train Time (s)*	*Test Time (s)*
5 MB	1,25	0,03	0,62	0,04
100 MB	11,96	0,15	5	0,11
250 MB	36,61	0,60	18,46	0,32
350 MB	46,54	0,83	14,47	0,26
500 MB	83,56	1,00	28,96	0,50
700 MB	122,77	1,66		
900 MB	174,27	1,79		

La diferencia de tiempo entre el proceso de entrenamiento y el de test es abismal en este par algoritmo-hardware. Se necesitan 174 segundos para entrenar un archivo de 900 MB, y 2 segundos para realizar el test del mismo.

Neural Network - Jetson Nano

En este caso, entre el tiempo que necesita para entrenar y para realizar el test hay una diferencia abismal en este hardware. Como se puede ver en la Fig 3.13, para entrenar un archivo de 900

MB se necesitan 174 segundos aproximadamente, mientras que para ejecutar el test del mismo tamaño solo 2.

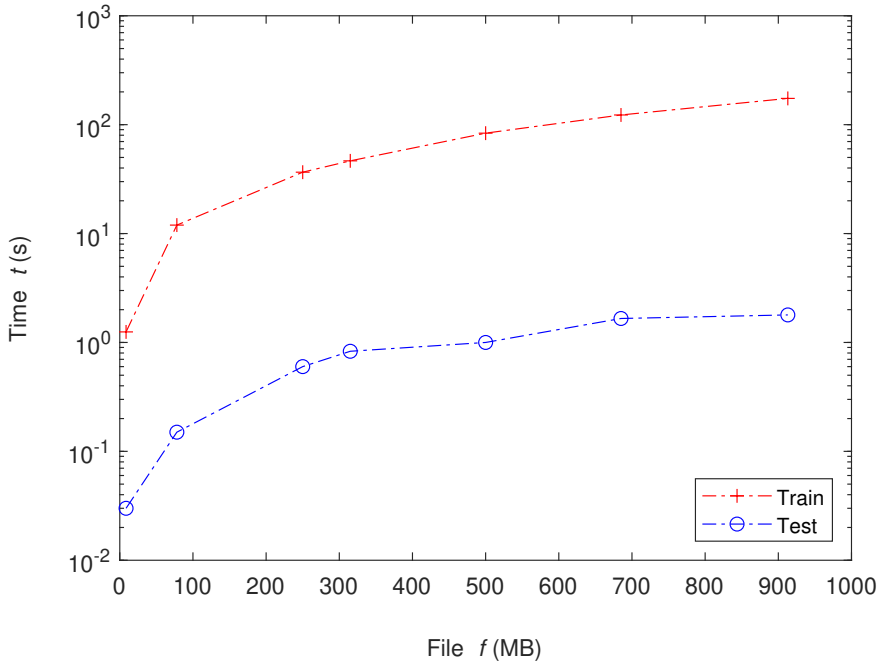

Figura 3.13: Gráfica tiempos de entrenamiento y test de Jetson Nano con Neural Network

A continuación, se muestran los modelos de comportamiento de entrenamiento y test respectivamente:

$$t(size) = a_i * size^{b_i} + c_i \tag{3.13}$$

$$a_i = 0035, b_i = 1245, c_i = 1982$$

$$t(size) = a_i * size^{b_i} + c_i \tag{3.14}$$

$$a_i = 0005, b_i = 0858, c_i = 0$$

El entrenamiento tarda más que el test en procesarse y a medida que aumenta el tamaño del archivo a procesar, tanto el entrenamiento como el test aumentan sus tiempos de procesamiento de manera similar.

Neural Network - Google Coral

En el hardware *Google Coral*, al igual que en el

Jetson Nano, tal y como se puede apreciar en la Fig. 3.14 tarda en realizar el entrenamiento más que el test, sin embargo, en este caso la diferencia no es tan abismal En cuanto a tiempos, tanto el en-

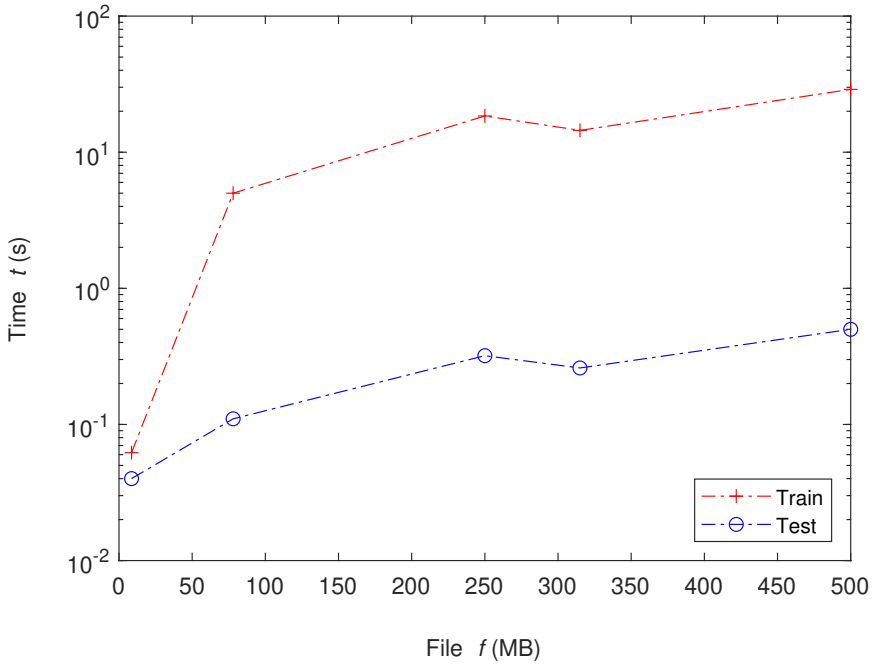

Figura 3.14: Gráfica tiempos de entrenamiento y test de Google Coral con Neural Network

trenamiento como el test se comportan de manera similar a medida que va aumentando el tamaño de archivo a procesar, y los modelos de entrenamiento y test son los siguientes:

$$t(size) = a_i * size^{b_i} + c_i \tag{3.15}$$

$$a_i = 0093, b_i = 0918, c_i = 0$$

$$t(size) = a_i * size^{b_i} + c_i \tag{3.16}$$

$$a_i = 0001, b_i = 0964, c_i = 0,034$$

Neural Network - Jetson Nano vs Google Coral

A pesar de que *Google Coral* solo puede procesar archivos de hasta 500 MB, ejecuta de media el doble de rápido el entrenamiento y el test(Fig. 3.15 y 3.16).

A pesar de que el hardware *Google Coral* solo puede procesar archivos de hasta 500 MB en este algoritmo, en promedio ejecuta el doble de rápido tanto el proceso de entrenamiento como el de test en comparación con el).

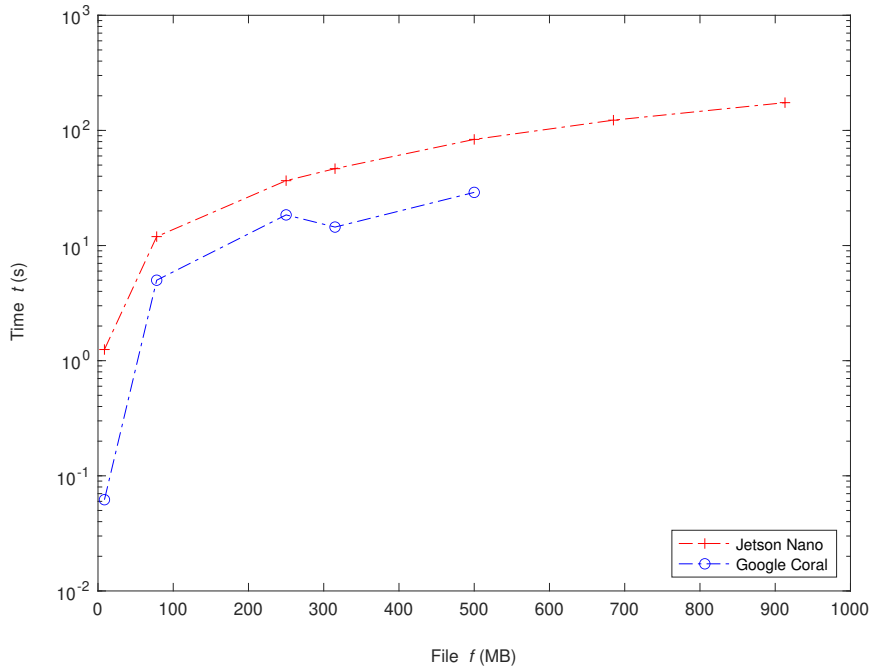

Figura 3.15: Gráfica tiempos de entrenamiento de Jetson Nano y Google Coral con Neural Network

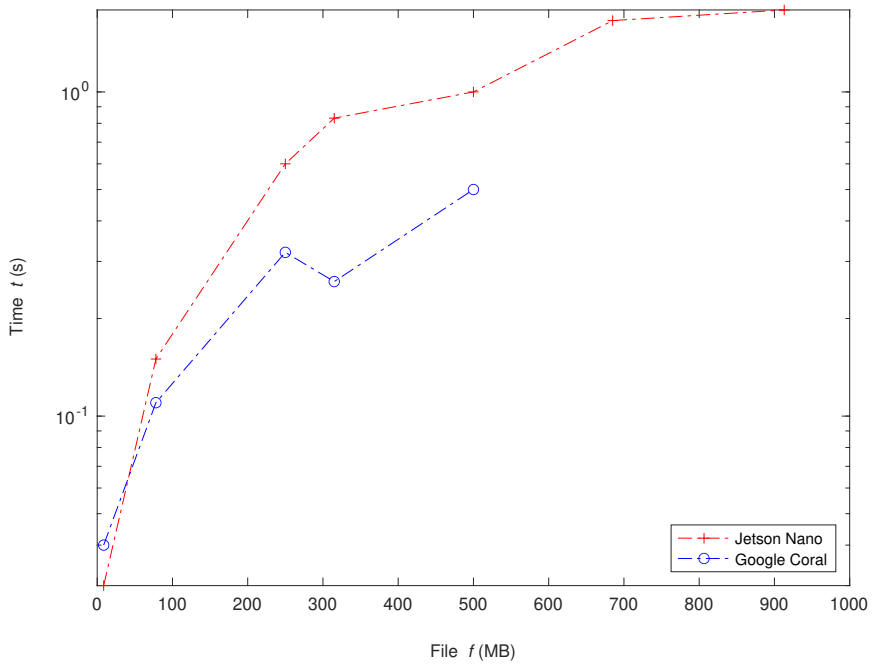

Figura 3.16: Gráfica tiempos de test de Jetson Nano y Google Coral con Neural Network

3.1.6. Gaussian Naive Bayes

A continuación, en la Tabla 3.5, se muestran los resultados obtenidos con el algoritmo Gaussian Naive Bayes.

Tabla 3.5: Tiempos de rendimiento para Gaussian Naive Bayes

GAUSSIAN NAIVE BAYES

Hardware	Nvidia Jetson Nano		Google Coral	
File Size	*Train Time (s)*	*Test Time (s)*	*Train Time (s)*	*Test Time (s)*
5 MB	0,06	0,04	0,12	0,05
100 MB	0,54	0,35	0,91	0,32
250 MB	1.94	0,98	3,3	1,2
350 MB	2,03	1,16	2,51	0,8
500 MB	28,03	1,98	4,95	2,01
700 MB	74,2	4,70		
900 MB	120,17	20,37		

Gaussian Naive Bayes - Jetson Nano

En este caso, como se puede ver en la Fig. 3.17 el tiempo de entrenamiento y test es muy similar

(siendo el tiempo de test algo menor) cuando el archivo es inferior a 350 MB. Cuando el archivo supera los 350 MB, la diferencia de tiempos entre el entrenamiento y el test empieza a ser considerablemente mayor, llegandose incluso a ser 15 veces mayor para archivos superiores a los 700 MB.

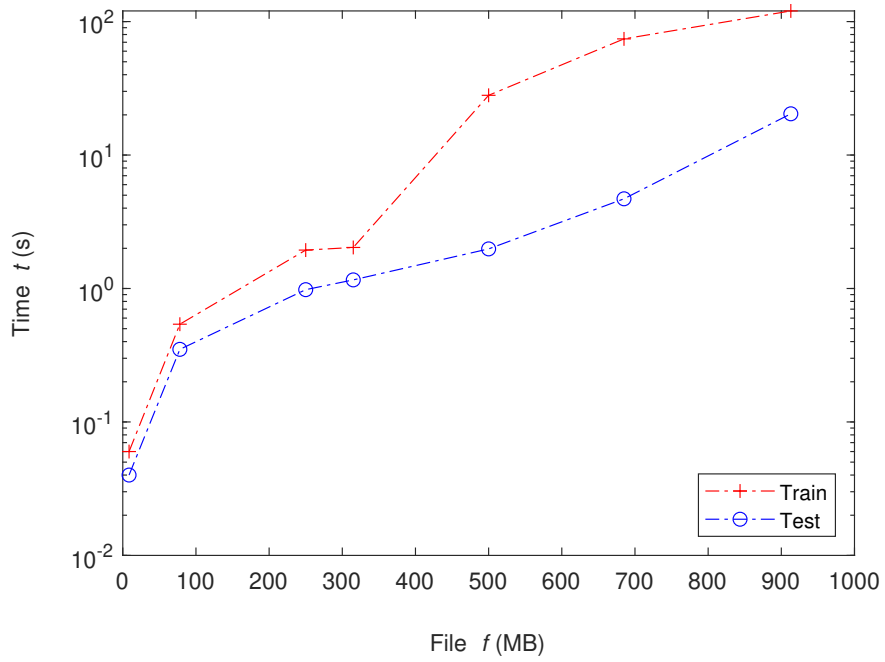

Figura 3.17: Gráfica tiempos de train y test de Jetson Nano con Gaussian Naive Bayes

Los modelos de comportamiento de entrenamiento y test para este par algoritmo-dispositivo son los siguientes:

$$t(size) = a_i * size^{b_i} + c_i \tag{3.17}$$

$$a_i = 5163e^{-5}, b_i = 2159, c_i = -3416$$

$$t(size) = a_i * size^{b_i} + c_i \tag{3.18}$$

$$a_i = 1345e^{-14}, b_i = 5122, c_i = 06274$$

El tiempo de entrenamiento y de test es muy similar cuando el archivo de datos es inferior a 350 MB, siendo el tiempo de test algo menor. Sin embargo, cuando el tamaño del archivo supera los 350 MB, la diferencia de tiempo entre el entrenamiento y el test comienza a ser considerablemente mayor (x15).

Gaussian Naive Bayes - Google Coral

A diferencia del hardware *Jetson Nano*, se mantiene constante la misma diferencia de tiempo entre el entrenamiento y el test a medida que aumenta el tamaño del archivo. Además, en este caso, el tiempo de test es menor.

En el hardware Google Coral sin embargo, se mantiene la misma diferencia de tiempo a medida que va aumentando el tamaño del archivo. En este caso también el test tarda menos en procesarse.

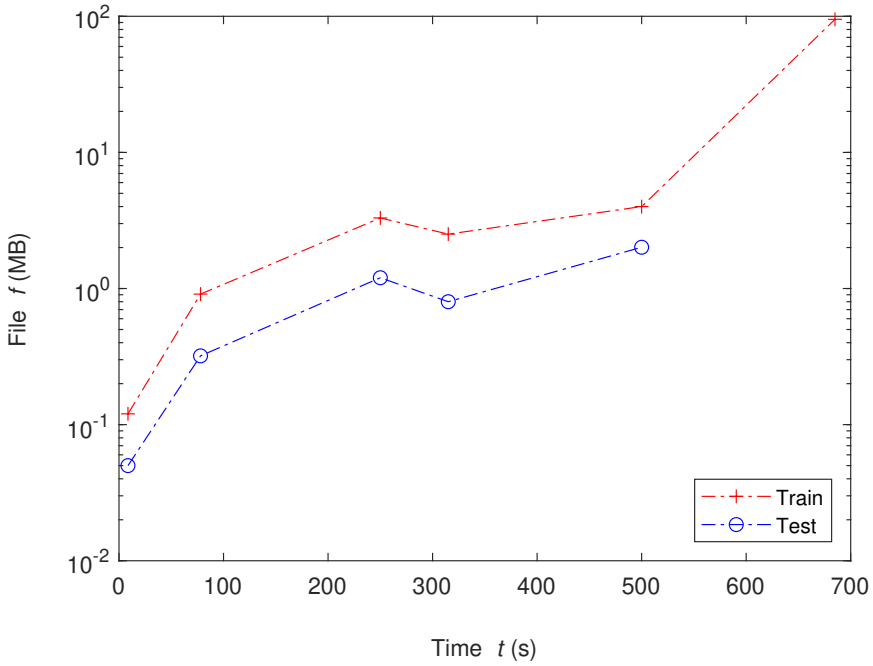

Figura 3.18: Gráfica tiempos de train y test de Google Coral con Gaussian Naive Bayes

Los modelos correspondientes a definir el comportamiento del entrenamiento y test de este algoritmo en Google coral son los siguientes:

$$t(size) = a_i * size^{b_i} + c_i \tag{3.19}$$

$$a_i = 4001e^{-27}, b_i = 1001, c_i = 0$$

$$t(size) = a_i * size^{b_i} + c_i \tag{3.20}$$

$$a_i = 0002, b_i = 1057, c_i = 0$$

Gaussian Naive Bayes - Jetson Nano vs Google Coral

En la comparativa de ambos hardware, se puede apreciar en la Fig. 3.19, a la hora de entrenar archivos menores de 300 MB, el hardware Google Coral es más rápido. Sin embargo, con archivos superiores, sería el hardware Jetson Nano el más rápido.

Para el algoritmo *Gaussian Naive Bayes*, para entrenar archivos menores de 300 MB el hardware más rápido es *Google Coral*, y para mayores el *Jetson Nano*. Para el test, ambos hardware obtienen tiempos similares, aunque el hardware *Google Coral* solo puede procesar el test archivos inferiores a 500 MB.

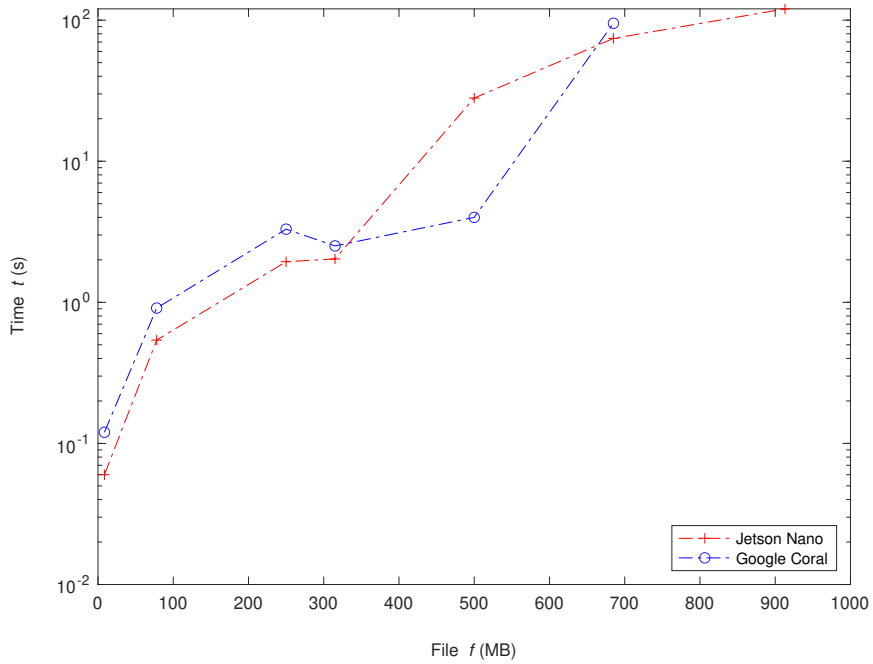

Figura 3.19: Gráfica tiempos de train de Jetson Nano y Google Coral con Gaussian Naive Bayes

En cuanto al test, como se puede ver en la Fig. 3.20 los dos hardware tienen tiempos similares, aunque si es cierto que Google Coral solo puede procesar test de archivos igual o inferiores a 500 MB.

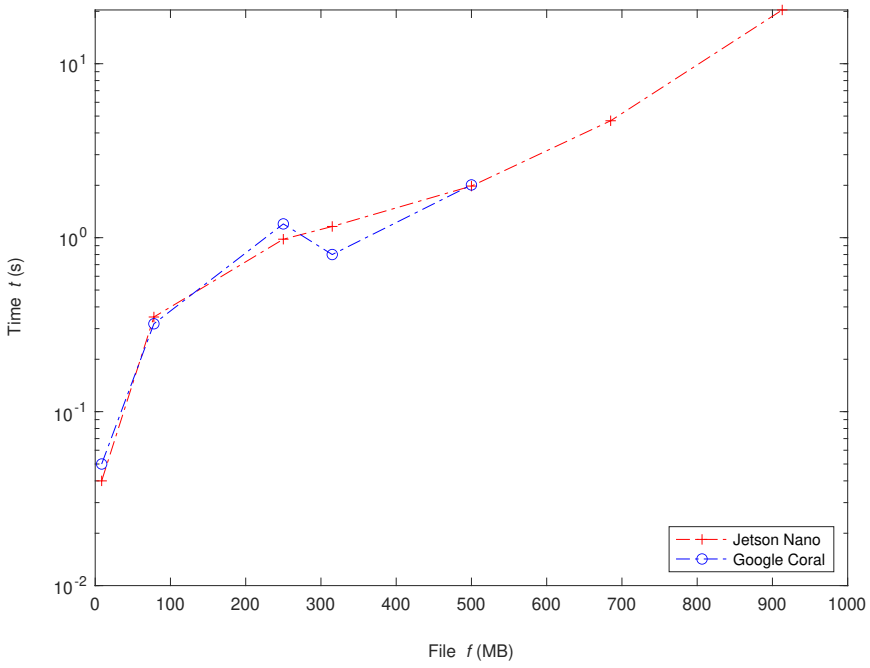

Figura 3.20: Gráfica tiempos de test de Jetson Nano y Google Coral con Gaussian Naive Bayes

3.2. Resultados obtenidos

En esta sección se va a realizar un análisis de los resultados que se han obtenido en el anterior apartado. Es decir, se va a traducir toda la experimentación a información visual que aporte valor a la investigación realizada. Como se ha podido ver en la sección anterior, la experimentación se ha dividido en resultados obtenidos por cada par algoritmo-hardware. Además, para cada par mencionado se ha dividido la experimentación en entrenamiento y test, debido a que el hardware se comporta de distinta manera para cada familia de algoritmos.

Como resumen general de los resultados obtenidos, en la Fig. 3.21 se muestra el resultado de las veces que cada hardware es más rápido habiendo cogido como datos cada entrenamiento y test que se ha hecho en la sección anterior.

Como resumen general de los resultados obtenidos, en la Fig. 3.21 se muestra el resultado de las veces que cada hardware es más rápido habiendo cogido como datos cada entrenamiento y test que se ha hecho en la sección anterior.

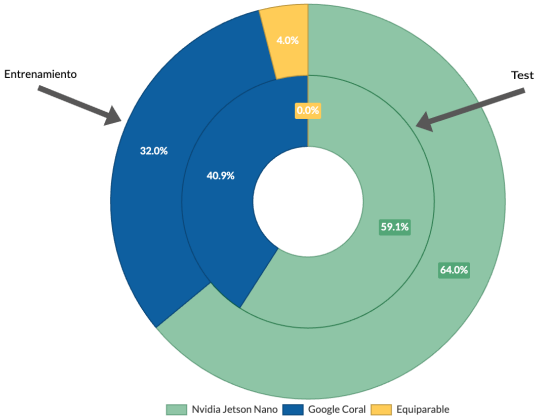

Figura 3.21: Gráfico circular Hardware óptimo de entrenamiento y test

Cabe destacar que sólo se han cogido los resultados que se podían comparar, es decir, en ciertos algoritmos el hardware Google Coral solo es capaz de procesar hasta 100 MB, por lo que solo se han tenido en cuenta para el gráfico los resultados de ambos hardware hasta 100 MB.

3.2.1. Resultados obtenidos por algoritmo

Con el algoritmo *Random Forest* el 100 % de las veces ha sido más rápido el hardware *Jetson Nano*, tanto para el entrenamiento como para el test.

Con el fin de conocer cómo afecta cada algoritmo en cada uno de los hardware, se han generado gráficos circulares para cada algoritmo con los reusultados obtenidos tanto en entrenamiento como test. A continuación se muestran uno a uno.

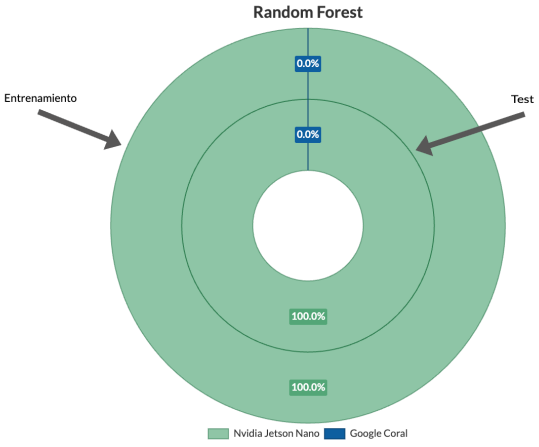

Figura 3.22: Gráfico circular Hardware óptimo en Random Forest

Como se puede ver en la fig. 3.22, en el caso del algoritmo Random Forest, en esta experimentación tanto para entrenamiento como para test el 100 % de las veces ha sido más rápido el hardware Nvidia Jetson Nano.

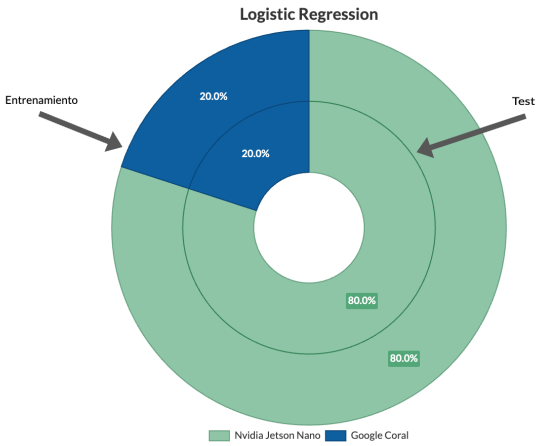

Figura 3.23: Gráfico circular Hardware óptimo en Logistic Regression

En el caso del algoritmo *Logistic Regression*, a la hora de entrenar y de realizar el test, el 80 % de las veces ha sido más rápido el hardware *Jetson Nano*.

En la Fig. 3.23, se aprecia que en este caso, tanto a la hora de entrenar como de realizar el test, en un 80 % de los casos el hardware Nvidia Jetson Nano es más rápido.

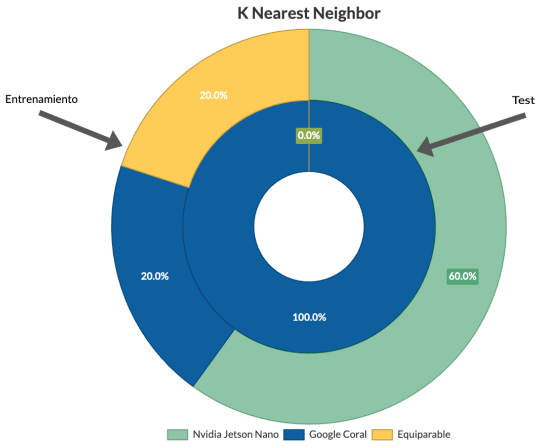

Figura 3.24: Gráfico circular Hardware óptimo en K Nearest Neighbor

En cuanto al algoritmo *K Nearest Neighbor*, tal y como muestra la Fig. 3.24 hay una diferencia notable entre el hardware óptimo en función de lo que se quiera procesar en él: entrenamiento o test.

Para el test, el 100 % de los archivod procesados en la experimentación han finalizado el proceso más rápido en el hardware Google Coral. Sin embargo, a la hora de entrenar, un 60 % de las veces ha sido el hardware Nvidia Jetson Nano el más veloz, mientras que el 20 % de las veces lo ha sido Google Coral y el otro 20 % han tenido el mismo tiempo de procesamiento ambos.

La experimentación realizada con el algoritmo *K Nearest Neighbor* ha concluido con que a la hora de realizar el test, el hardware *Google Coral* ha sido más rápido el 100 % de las veces. Sin embargo, en cuanto al entrenamiento, el hardware *Jetson Nano* ha sido más rápido el 60 % de las veces.

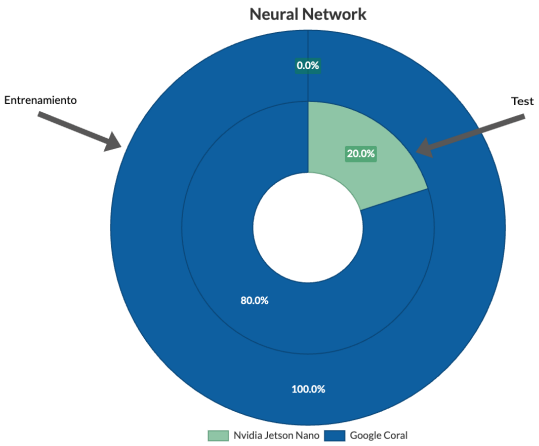

Figura 3.25: Gráfico circular Hardware óptimo en Neural Network

En lo referente al procesamiento de redes neuronales en los dispositivos hardware, como se puede observar en la Fig. 3.25, a la hora de realizar el test el 80 % de las veces ha sido más rápido el hardware Google Coral, mientras que para el entrenamiento lo ha sido el 100 % de las veces.

El hardware que mejor funciona a la hora de procesar redes neuronales es el *Google Coral*. Los experimentos han concluido con que este ha sido el hardware más rápido el 100 % de las veces a la hora de entrenar, y el 80 % de las veces para realizar el test.

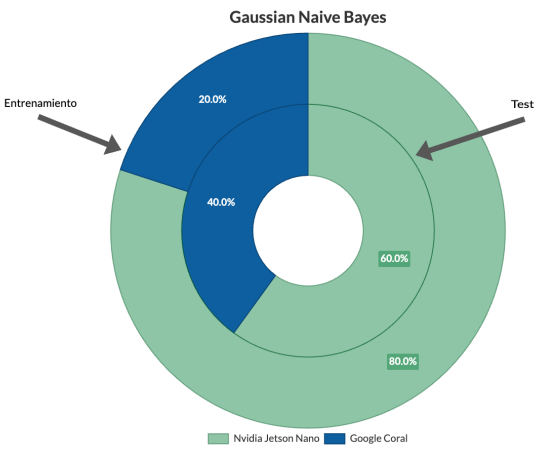

Figura 3.26: Gráfico circular Hardware óptimo en Gaussian Naive Bayes

Por último, el algoritmo *Gaussian Naive Bayes* el 60 % de las veces ha sido más rápido en el hardware *Nvidia Jetson Nano* a la hora de realizar el test y el 80 % de las veces a la hora de entrenar (Fig. 3.26).

A la hora de entrenar con el algoritmo *Gaussian Naive Bayes*, el hardware *Jetson Nano* ha sido el más rápido el 80 % de las veces, y el 60 % de las veces a la hora de realizar el test.

3.2.2. Resultados obtenidos por tamaño de archivo

Se han realizado estadísticas para analizar el impacto del tamaño del archivo en el procesamiento de datos en los hardware de *Edge Computing*. El hardware *Jetson Nano* es más rápido en la mayoría de los casos, con una tasa del 80 % de eficiencia para archivos de 250 MB y una tasa del 100 % para archivos de 700 MB y 900 MB. Es importante destacar que el hardware *Google Coral* no puede procesar archivos superiores a 500 MB por restricciones computacionales, por lo que el hardware *Jetson Nano* es el más rápido para entrenar archivos de ese tamaño.

Por otro lado, se ha realizado también estadísticas en función del tamaño de archivo que se procesa, para ver el impacto que tiene el tamaño del archivo a la hora de ser procesados en cualquiera de los dos hardware de *Edge Computing*. Para este caso concreto, se han cogido todos los datos, es decir, los datos que no se pueden procesar en uno de los hardware debido a restricciones computacionales pero en el otro si, se ha contabilizado como el más óptimo para este último.

A continuación, la Fig. 3.27 muestra 7 barras que representan cada una un tamaño de archivo (que puede consultarse en la leyenda inferior). Para cada tamaño, muestra las veces en las que cada hardware ha sido más rápido a la hora de procesar el entrenamiento de todos los algoritmos utilizados en la experimentación de la sección anterior.

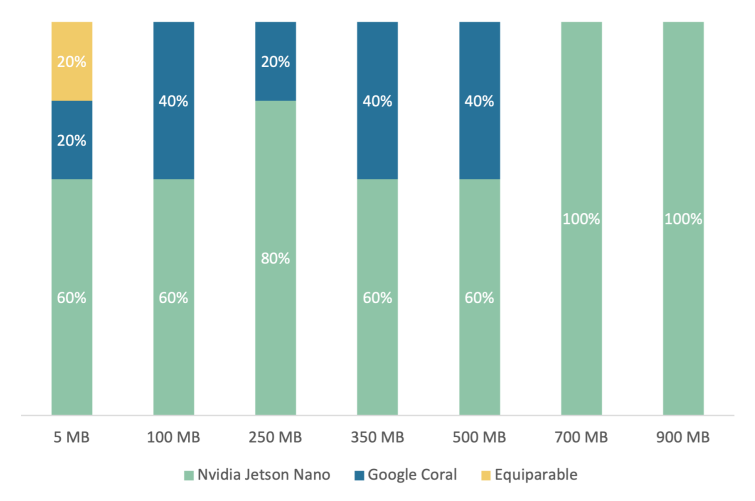

Figura 3.27: Gráfico de barras para hardware más óptimo para el entrenamiento en función del tamaño de archivo

Como se puede apreciar, el hardware *Nvidia Jetson Nano* es más rápido en una gran mayoría de los casos, llegando al 80 % de las veces para archivos de 250 MB y al 100 % para archivos de 700 MB y 900 MB. En este último caso, cabe destacar que es el más rápido porque es el único que es capaz de procesar el entrenamiento de ese tamaño de archivos, ya que el hardware *Google Coral* no puede entrenar archivos superiores a 500 MB por restricciones computacionales.

En la realización de pruebas, se observa que para el entrenamiento, el hardware *Jetson Nano* es el más rápido en todos los tamaños de archivo, mientras que para la realización del test, el hardware *Google Coral* es más rápido en el 60 % de los casos para archivos de 100 MB y 250 MB.

Por el contrario, en lo que al test se refiere, en la Fig. 3.28 puede apreciarse que mientras para el entrenamiento el hardware *Nvidia Jetson Nano* es en todos los tamaños de archivo el mayoritario en cuánto a rapidez, a la hora de realizar el test, para archivos de 100 MB y 250 MB es el hardware *Google Coral* el que es más rápido en el 60 % de los casos.

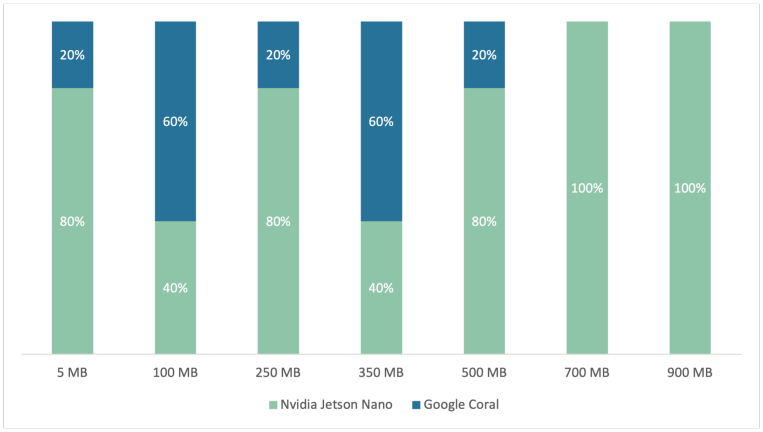

Figura 3.28: Gráfico de barras que muestra el hardware más rápido para el test en función del tamaño de archivo

Tras realizar la experimentación y analizar los resultados obtenidos, se han sacado las conclusiones descritas a continuación.

3.3. Conclusiones

Habiendo concluido la experimentación con conjuntos de datos de diferentes tamaños, hay que señalar que la placa *Google Coral* no es capaz de procesar conjuntos de datos de más de 500 MB, ni para entrenar ni para realizar el test. En algunos casos, incluso este límite es mayor, ya que por ejemplo con el algoritmo *K Nearest Neighbor*, a la hora de realizar el test solo es capaz de procesar archivos iguales o inferiores a 100 MB.

En resumen, se concluye que el hardware *Jetson Nano* es el más rápido en la mayoría de los algoritmos, excepto en las redes neuronales, donde el hardware *Google Coral* es superior en velpcidad de procesamiento debido a su Unidad de Procesamiento por Tensores (TPU). Sin embargo, hay casos puntuales en los que el hardware *Google Coral* ofrece un mejor rendimiento, como con el algoritmo *Logistic Regression* y archivos de tamaño alrededor de 350 MB.

De forma general, todos los algoritmos rinden mejor en el hardware de la *Nvidia Jetson Nano*, excepto las redes neuronales. Esto se debe a que el hardware *Google Coral* está diseñado para realizar inferencia de redes neuronales. Es la unidad de procesamiento por tensores (TPU) la que hace que esta placa sea realmente potente con las redes neuronales. Es un circuito integrado de aplicación específica construido para ser un acelerador de inteligencia artificial [174].

Como se ha mencionado, aunque por norma general el hardware *Nvidia Jetson Nano* sea más rápido, siguen habiendo algunos casos puntuales en los que *Google Coral* tiene un mejor rendimiento. Por ejemplo, con el algoritmo *Logistic Regression*, el hardware *Nvidia Jetson Nano* es en la mayoría de veces más rápido, excepto con archivos que rondan los 350 MB. En este caso, tanto a la hora de entrenar como a la hora de realizar el test, *Google Coral* ofrece un mejor rendimiento.

Otra de las conclusiones que se puede sacar con toda esta experimentación, es que el tamaño del archivo que se quiera procesar también influye en cuál va a ser el dispositivo *Edge Computing* que más rápido lo procese. En el caso del algoritmo *K Nearest Neighbor*, con archivos pequeños menores a 100 MB, el hardware *Google Coral* es más rápido, sin embargo, para archivos superiores, sería el hardware *Nvidia Jetson Nano* el que más rápido actuaría.

La experimentación realizada con el algoritmo *Gaussian Naive Bayes* demuestra que otra de las variables que entran en juego a la hora de procesar archivos en estos hardware es el tipo de procesamiento que se va a realizar: entrenamiento o test. Ya que, para el algoritmo mencionado, si lo que se quiere es realizar el entrenamiento, es el hardware *Google Coral* el más veloz. Sin embargo, a la hora de realizar el test, ambos hardware obtienen resultados similares.

En conclusión, a partir del análisis precedente, hay varios factores que influyen en el tiempo de procesamiento a la hora de correr modelos de inteligencia artificial en dispositivos *Edge Computing*: el

También se encontró que el tamaño del archivo y el tipo de procesamiento (entrenamiento o test) son factores que influyen en el tiempo de procesamiento, y se presenta una tabla resumen para facilitar la elección del hardware adecuado para cada caso. Además, se destaca que el Google Coral no es capaz de procesar conjuntos de datos mayores a 500 MB, y en algunos casos solo puede procesar archivos de hasta 100 MB.

tipo de procesamiento (entrenamiento o test), el tamaño del archivo y el algoritmo que se va a utilizar. Es por eso que la combinación de estas 3 variables es la clave a la hora de poder elegir qué hardware procesaría más rápido el modelo objetivo para cada caso. Con el fin de facilitar la elección de hardware para cualquier caso determinado, a continuación se muestra la tabla resumen de los modelos de comportamiento para cada hardware-tamaño de archivo-algoritmo. Para conocer el tiempo aproximado que va a tardar un experimento en cada uno de los hardware, únicamente habrá que sustituir el tamaño de archivo por la variable *size* en el modelo correspondiente al algoritmo y placa seleccionados y el resultado será el tiempo de procesamiento en segundos.

En las Tablas 3.6 y 3.7 tenemos el resumen de los modelos de comportamiento para el hardware Jetson Nano.

Tabla 3.6: Modelos de comportamiento para entrenamiento Jetson Nano

MODELOS DE ENTRENAMIENTO JETSON NANO			
	$t(size) = a_i * size^{b_i} + c_i$		
	a_i	b_i	c_i
Random Forest	1,976	1,145	0
Logistic Regression	0,007	11,486	3,473
K Nearest Neighbor	9,94e	5,562	0
Neural Network	0,035	11,245	1,982
Gaussian Naive Bayes	5,163e	2,159	-3,416

Tabla 3.7: Modelos de comportamiento para test Jetson Nano

MODELOS DE TEST JETSON NANO			
	$t(size) = a_i * size^{b_i} + c_i$		
	a_i	b_i	c_i
Random Forest	0,012	1,288	0,505
Logistic Regression	1,201	1,01	-2,158
K Nearest Neighbor	0,0004	3,043	0
Neural Network	0,005	0,858	0
Gaussian Naive Bayes	1,345e-14	5,122	0,627

En la Tabla 3.8 y Tabla 3.9 tenemos los resúmenes de los modelos de comportamiento para el hardware Google Coral.

MODELOS DE ENTRENAMIENTO GOOGLE CORAL

	$t(size) = a_i * size^{b_i} + c_i$		
	a_i	b_i	c_i
Random Forest	9,53	0,995	-75,66
Logistic Regression	0,438	0,803	-1,162
K Nearest Neighbor	2,36e	1,841	0,026
Neural Network	0,093	0,918	0
Gaussian Naive Bayes	4,001e-27	10,01	0

Tabla 3.8: Modelos de comportamiento para entrenamiento Google Coral

MODELOS DE TEST GOOGLE CORAL

	$t(size) = a_i * size^{b_i} + c_i$		
	a_i	b_i	c_i
Random Forest	0,058	1,076	0,727
Logistic Regression	3,431	0,833	-9,134
K Nearest Neighbor	0,052	2,04	0
Neural Network	0,001	0,964	0,034
Gaussian Naive Bayes	0,002	1,057	0

Tabla 3.9: Modelos de comportamiento para test Google Coral

Modelos Edge 4

4.1. Experimentación

4.1 Experimentación 95
4.1.1 Conjunto de datos 96
4.1.2 Procesamiento de los datos 100
4.2 Experimentación conjunto de datos "Bosch" 103
4.3 Experimentación conjunto de datos de pintado 104
4.3.1 Resultados modelo Esmalte 108
4.3.2 Resultados modelo Apresto 110
4.3.3 Resultados modelo Apresto y Esmalte unificados 110
4.3.4 Comparativa resultados de precisión entre los modelos de Esmalte, Apresto y ambos modelos unificados 112
4.4 Conclusiones . . . 112

El objetivo de esta experimentación es validar la eficacia y precisión de modelos *Edge Computing* que representan subprocesos dentro del ámbito industrial, y compararlos con el modelo tradicional de *Cloud Computing*.

Como se ha podido observar en la experimentación del capítulo anterior, por lo general, cuanto más pequeño es el conjunto de datos que se va a procesar, más rápida es la respuesta del modelo tanto a la hora de entrenar como a la hora de realizar el test. El problema está, en que como se ha mencionado anteriormente, se ha producido un aumento considerable de los datos que se recogen en los últimos años [9], y esto implica que debido a la digitalización masiva de las empresas, las líneas de producción recogen miles de variables por segundo [6]. Como consecuencia, los gemelos digitales que representan los procesos industriales están compuestos por conjuntos de datos de gran volumen, pero existen sectores cuyas líneas de producción demandan respuestas en tiempo real, como el de la automoción, donde por cada coche fabricado se recogen más de 1000 variables sólo en el proceso de pintado [177]. Entonces, ¿Qué se puede hacer?

Con el fin de poder contibuir a encontrar una solución para este sector y poder procesar los modelos de inteligencia artificial en tiempo real buscando

El objetivo de esta experimentación es comparar la eficacia y precisión de los modelos de Edge Computing con el modelo tradicional de Cloud Computing en el ámbito industrial. Dado que la digitalización masiva de las empresas ha llevado a un aumento considerable de los datos que se recogen, los gemelos digitales que representan los procesos industriales están compuestos por conjuntos de datos de gran volumen.

una alternativa a la arquitectura *Cloud Computing* (que incrementa el tiempo de respuesta debido a las latencias existentes en el envío de datos [27]), se ha dividido el conjunto de datos que representa un proceso entero en conjuntos de datos más pequeños, denominados subprocesos. En lugar de crear un modelo que represente toda la cadena de producción, se quiere crear pequeños modelos que representen subprocesos de la línea de producción. Una vez creados, se validará la eficacia de los pequeños modelos generados comprobando que no sea inferior al modelo del proceso completo.

Sin embargo, hay sectores como el de la automoción que demandan respuestas en tiempo real. Para abordar este desafío, se han dividido los conjuntos de datos que representan un proceso completo en subprocesos más pequeños, creando pequeños modelos que representan estos subprocesos. La eficacia de estos pequeños modelos se validará comparándolos con el modelo del proceso completo para garantizar que no sean inferiores.

4.1.1. Conjunto de datos

Esta experimentación se ha llevado a cabo dos veces con dos conjunto de datos diferentes:

- Conjunto de datos de carácter público de la empresa Bosch [175] (utilizado en la experimentación del capítulo anterior).
- Conjunto de datos de una línea de fabricación de vehículos que representa el proceso de pintado.

Conjunto de datos 'Bosch'

Este conjunto de datos, como bien se ha indicado en el anterior capítulo, lo componen los datos adquiridos durante el proceso de fabricacion de piezas. Este proceso lo componen 4 líneas de producción (L0, L1, L2, L3), por lo que se ha dividido el conjunto de datos por líneas de producción, es decir, se han generado 4 conjuntos de datos, uno por cada línea.

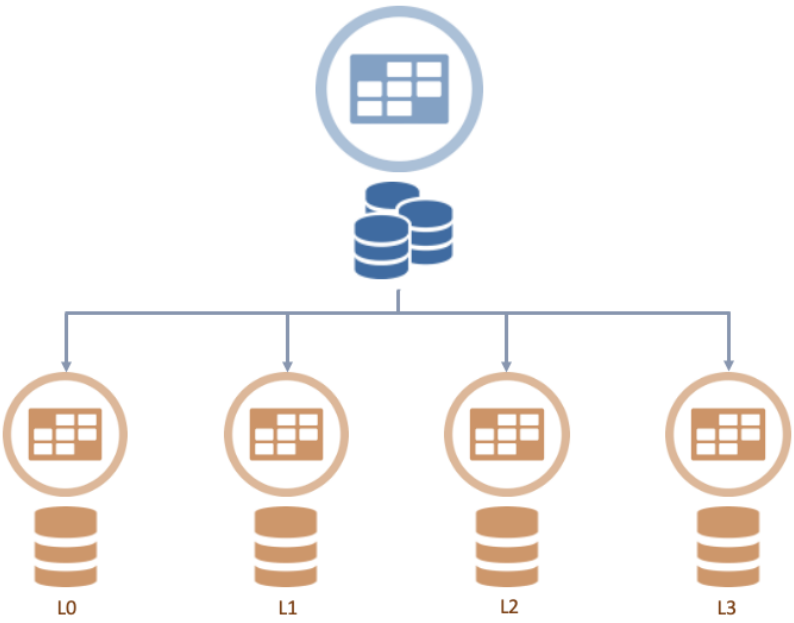

Figura 4.1: Conjunto de datos "Bosch" dividido en líneas de producción

Para entrenar los modelos de inteligencia artificial con cada uno de los conjuntos de datos, la etiqueta que se ha utilizado para todos es la misma: la que indica si pasará o no el control de calidad.

Conjunto de datos 'Pintado de vehículos' Este conjunto de datos se ha obtenido del proceso de pintado de una planta de fabricación de vehículos. Estos datos, se han tenido que pre-procesar debido a que son datos captados directamente de los sensores que existen durante el proceso. Por cada vehículo, se recogen una serie de variables a medida que va avanzando por la línea. A continuación se listan los sensores existentes:

- 9 sensores de temperatura en horno de esmalte
- 2 sensores de humedad en la cabina de pintura
- 2 sensores de humedad en la salida de la cabina de pintura
- 1 sensor de humedad externa
- Presion en la cabina (Pa) de pintado
- 2 sensores de temperatura en la cabina de pintura

La experimentación se realizó dos veces con diferentes conjuntos de datos: el primero es el conjunto de datos público de la empresa Bosch utilizado en el capítulo anterior, que representa el proceso de fabricación de piezas dividido en 4 líneas de producción (L0, L1, L2, L3), y el segundo es un conjunto de datos de una línea de fabricación de vehículos que representa el proceso de pintado. En el caso del conjunto de datos de Bosch, se generaron 4 conjuntos de datos separados por línea de producción.

- 2 sensores de temperatura a la salida de la cabina
- Temperatura externa
- 5 sensores de velocidad del aire horizontal en la cabina de pintura
- 4 sensores de velocidad del aire vertical en la cabina de pintura
- 2 sensores de humedad en la salida de la cabina de apresto
- 1 sensor de humedad en la zona de la cabina de apresto
- 1 sensor de presión diferencial de la zona exterior de apresto
- 1 sensor de presión diferencial en la zona limpia de apresto
- 2 sensores de temperatura de aire frio aportado por el enfriador en la zona de apresto
- 7 sensores de temperatura en la entrada del horno de apresto
- 1 sensor de temperatura en la entrada del incinerador de apresto
- 1 sensor de temperatura ambiente del incinerador de apresto
- 2 sensores de temperatura a la salida de los hornos de apresto
- 1 sensor de temperatura a la salida de aire nuevo de cortinas en apresto
- 1 sensor de temperatura a la salida de la chimenea de apresto
- 1 sensor de temperatura a la salida del incinerador en apresto
- 1 sensor de temperatura en la zona de apresto
- 1 sensor de velocidad de aire horizontal blow-off en la zona de apresto
- 1 sensor de velocidad de aire horizontal flash-off en la zona de apresto

- 1 sensor de velocidad de aire horizontal manual
- 1 sensor de velocidad de aire vertical automático
- 1 sensor de velocidad de aire vertical manual

En el conjunto de datos de la línea de fabricación de vehículos que representa el proceso de pintado, se recogen numerosas variables por cada coche en el proceso de pintado. Además, si el coche no pasa el control de calidad de pintado, vuelve a retrabajarse y recoge nuevos valores de todos los sensores.

Cada vez que un coche pasa por el proceso de pintado, como se puede observar en el listado anterior, se generan todas estas variables en este único proceso. Además, si una vez llegado al punto de control de calidad de pintado, no lo supera, el coche vuelve a retrabajarse, por lo que volvería a pasar por toda la línea, y por tanto, recoger valores de todos los sensores de nuevo. Esto puede pasar hasta 3 veces, por lo que por cada coche, existe la posibilidad de que haya uno, dos o tres valores para cada sensor arriba listado.

Además de los datos de los sensores, también se han recogido en el conjunto de datos las características de cada vehículo:

- Color de la carrocería
- Nº de rondas
- Modelo
- Longitud

Por otro lado, como etiqueta de salida, se ha cogido el tipo de defecto que detecta el control de calidad final, existendo 50 defectos posibles.

Como se ha podido observar, el nº de variables que se recogen por vehículo es elevado, y se puede llegar a triplicar en caso de que el vehículo tenga que retrabajarse 2 veces. En el marco de esta experimentación, este conjunto de datos se ha dividido en dos:

- Subproceso de esmalte: se han cogido las variables que tienen que ver con el proceso de esmaltado y se han añadido las características de cada vehículo.
- Subproceso de apresto: lo componen las variables que tienen que ver con apresto y se han añadido las características de cada vehículo.

Se han recogido también las características de cada vehículo y el tipo de defecto detectado en el control de calidad final. Se ha dividido el conjunto de datos en dos subprocesos: esmalte y apresto, para crear modelos más pequeños y representativos de cada subproceso.

Por lo tanto, de un conjunto de datos principal en el que había variables tanto de esmalte como de apresto, se han creado dos conjuntos de datos, uno para cada subproceso (Fig. 4.2).

Figura 4.2: Conjunto de datos de Pintado de vehículos dividido en los subprocesos de esmalte y apresto

4.1.2. Procesamiento de los datos

El conjunto de datos de pintado de vehículos ha requerido un preprocesamiento en el que se han pivotado los datos para tener una fila por vehículo y las columnas corresponden a los valores de cada sensor. Mientras que el conjunto de datos de Bosch solo se dividió en líneas de producción.

Para poder llevar a cabo la experimentación, ambos conjuntos de datos han tenido que ser preprocesados. En el caso del conjunto de datos de 'Bosch', únicamente se ha tenido que dividir el conjunto de datos en líneas de produccioón. Sin embargo, para el conjunto de datos de pintado de vehículos, se ha tenido que realizar un exhaustivo pre-procesado de datos: en primer lugar, se han

pivotado los datos de manera que queden los valores de los sensores en columnas, ya que antes había una fila por cada par sensor-vehículo. Por lo tanto, se ha conseguido tener un conjunto de datos donde cada fila representa un vehículo y donde las columnas indican los valores de cada sensor para el vehículo correspondiente.

Conjunto de datos original

Vehículo ID	Sensor ID	Valor
Vehiculo_ID_1	Sensor_id_1	X
Vehiculo_ID_1	Sensor_id_2	Y
Vehiculo_ID_1	Sensor_id_3	Z
...	...	...
Vehiculo_ID_N	Sensor_ID_M	K

Figura 4.3: Estructura del conjunto de datos original

En la Fig. 4.3 se puede ver la etructura que tiene el conjunto de datos original. Después de realizar la pivotación de los datos, la estructura de datos resultanse se observa a continuación en la Fig. 4.4.

Conjunto de datos pivotado

Vehículo ID	Sensor ID_1	Sensor_ID_2	Sensor_ID_3	...	Sensor_ID_M
Vehiculo_ID_1	X_1	Y_1	Z_1	...	K_1
Vehiculo_ID_2	X_2	Y_2	Z_2	...	K_2
Vehiculo_ID_3	X_3	Y_3	Z_3	...	K_3
...	...	...	...	...	...
Vehiculo_ID_N	X_N	Y_N	Z_N	...	K_N

Figura 4.4: Estructura del conjunto de datos pivotado

Una vez se han obtenido los conjuntos de datos tanto de apresto como de esmalte pivotados, se ha procedido a añadir las columnas que indican las características del vehículo (color, modelo, longitud) y el nº de rondas que ha realizado.

Figura 4.5: Estructura del conjunto de datos transformado

Por lo tanto, en la Fig. 4.5 se describe la estructura del conjunto de datos final. Cabe destacar, que esa estructura corresponde tanto para el conjunto de datos de apresto como para el conjunto de datos de esmalte.

Una vez se ha conseguido tener los datos en la estructura deseada, se ha realizado la normalización de los mismos.

La normalización de los datos es un enfoque de preprocesamiento que consiste en escalar o transformar los datos para que la contribución de cada característica sea igual. La calidad de los datos es crucial para el éxito de los algoritmos de aprendizaje automático en la obtención de un modelo predictivo generalizado del problema de clasificación. Muchos estudios han demostrado la importancia de la normalización de los datos para mejorar su calidad y, posteriormente, el rendimiento de los algoritmos de aprendizaje automático.

Normalización de los datos

La normalización de los datos es uno de los enfoques de preprocesamiento en el que los datos se escalan o transforman para que la contribución de cada característica sea igual [178]. El éxito de los algoritmos de aprendizaje automático depende de la calidad de los datos para obtener un modelo predictivo generalizado del problema de clasificación. La importancia de la normalización de los datos para mejorar su calidad y, posteriormente, el rendimiento de los algoritmos de aprendizaje automático se ha presentado en muchos estudios [179] [180] [181].

4.2. Experimentación conjunto de datos "Bosch"

En la experimentación, se han procesado subconjuntos de datos de diferentes líneas de producción utilizando diferentes algoritmos de aprendizaje automático, incluyendo *Random Forest*, *Logistic Regression*, *K Nearest Neighbors*, *Neural Network* y *Gaussian Naive Bayes*.

Una vez dividido el conjunto de datos original en subconjuntos de datos, cada uno de ellos hace referencia a una línea de producción, se ha procedido a realizar la experimentación. Para ello, se han procesado los subconjuntos de datos con diferentes familias de algoritmos para comprobar su eficacia y poder compararla con la eficacia del conjunto completo de datos. Los algoritmos utilizados han sido *Random Forest*, *Logistic Regression*, *K Nearest Neighbors*, *Neural Network* y *Gaussian Naive Bayes*. Las tablas 4.1 y 4.2 muestran los resultados de eficacia con cada algoritmo.

COMPARACIÓN DE MODELOS

	L0 + L1 + L2 + L3 (%)	L0 (%)	L1 (%)
Random Forest	98,30	98,39	98,32
Logistic Regression	98,16	98,26	98,26
K Nearest Neighbor	98,17	97,94	97,98
Neural Network	91,11	95,66	98,24
Gaussian Naive Bayes	61,82	75,83	98,30

Tabla 4.1: Comparativa de eficacia de los modelos creados con los subconjuntos de datos y el modelo creado con el conjunto de datos original.

COMPARACIÓN DE MODELOS

	L0 + L1 + L2 + L3 (%)	L2 (%)	L3 (%)
Random Forest	98,30	98,33	98,68
Logistic Regression	98,16	98,26	98,23
K Nearest Neighbor	98,17	97,97	98,15
Neural Network	91,11	91,52	98,30
Gaussian Naive Bayes	61,82	98,30	95,70

Tabla 4.2: Comparativa de eficacia de los modelos creados con los subconjuntos de datos y el modelo creado con el conjunto de datos original.

A continuación se explican los resultados mostrados en la tablas 4.1 y 4.2 y : la primera columna denominada *L0 + L1 + L2 + L3* hace referencia al

conjunto de datos original completo, ya que contiene los datos de las cuatro líneas de producción. Las siguientes columnas, *L0*, *L1*, *L2* y *L3* hacen referencia a los subconjuntos creados para cada línea de producción. Como se puede apreciar, excepto con el algoritmo *K Nearest Neighbors*, los algoritmos ofrecen un mejor resultado cuando los modelos se crean con los subconjuntos de datos, es decir, los modelos obtienen una mejor precisión cuando se entrenan con los datos de cada línea de producción por separado. Aun así, para el algoritmo *K Nearest Neighbors*, la mayor pérdida de precisión se produce en la línea 0 (*L0*) y equivale al 0,23 %, por lo que podría considerarse como despreciable. Por otro lado, cabe destacar que con el algoritmo Gaussian Naive Bayes, la mejora que se consigue en cuanto a precisión cuando se entrenan los modelos con los conjuntos de datos pequeños es importante, llegando a pasar de un 61.82 % de precisión con el modelo completo a una precisión de 98.30 % con el conjunto de datos de la línea 1.

Las tablas 4.1 y 4.2 muestran los resultados de eficacia de cada algoritmo, y se observa que, excepto en el caso de *K Nearest Neighbors*, los modelos entrenados con los subconjuntos de datos ofrecen un mejor resultado que los modelos creados con el conjunto completo de datos.

Además, se destaca que el algoritmo Gaussian Naive Bayes obtiene una mejora significativa en la precisión al entrenarse con los conjuntos de datos de las líneas de producción en lugar del conjunto completo, llegando a una precisión del 98.30 % en el conjunto de datos de la línea 1.

4.3. Experimentación conjunto de datos de pintado

Al realizar la experimentación en este caso, debido al gran volumen de variables, sólo las redes neuronales daban resultados de precisión prometedores. El resto, no llegaba ni a un 40 % de precisión, por lo que la experimentación se ha centrado en analizar los resultados de los modelos con diferentes arquitecturas de redes neuronales. Además de las diferentes arquitecturas, también se han realizado comparativas con diferentes optimizadores, ya

En este caso, se ha llevado a cabo una experimentación en la que se han utilizado diferentes modelos para analizar un gran volumen de variables. Sin embargo, solo las redes neuronales han mostrado resultados de precisión prometedores, mientras que el resto de modelos no superaba el 40 % de precisión. Por tanto, la experimentación se ha centrado en analizar diferentes arquitecturas de redes neuronales y optimizadores para reducir el error de la red. Se han llevado a cabo comparativas para evaluar los resultados obtenidos con diferentes combinaciones de arquitecturas y optimizadores.

que, la función de estos es reducir el error de la red optimitzando los valores [182]

Optimizadores de redes neuronales
La importancia del optimizador en las redes neuronales es vital, ya que se encargan de encontrar los valores óptimos de la función de pérdida calculando el gradiente de la función de coste y generando nuevos pesos mediante el uso de algoritmos de búsqueda metaheurísticos [182].

Para minimizar el error de la función de costa, lo que hace es modificar cada peso de la red en la dirección negativa del gradiente [183]:

$$W_{t+1} = W_t + df(coste)/dW * lr \quad (4.1)$$

La elección del optimizador es fundamental en las redes neuronales, ya que es el encargado de encontrar los valores óptimos de la función de pérdida mediante el cálculo del gradiente de la función de coste y la generación de nuevos pesos mediante algoritmos de búsqueda metaheurísticos. El optimizador ajusta cada peso de la red en la dirección negativa del gradiente para minimizar el error de la función de costo. El factor de entrenamiento, o "*lr*", es multiplicado por el vector de gradiente para acelerar la convergencia de la función de costo hacia su mínimo.

Donde aparece el término llamado "*lr*" que hace referencia al factor de entrenamiento, y se multiplica por el vector de gradiente con el objetivo de acelerar la convergencia de la función de coste hacia su mínimo [184].

Existen varios optimizadores diferentes, a continuación se describen los utilizados en esta experimentación:

***Stochastic Gradient Descent* (SGD)**
Stochastic Gradient Descent o el descenso de Gradiente Estocástica es una aproximación estocástica de la optimización del descenso gradiente y un método iterativo para minimizar/maximizar una función objetivo, que lucha por descubrir los mínimos o máximos por iteración [185]. El SGD lo hace siguiendo el gradiente negativo del objetivo tras comprobar sólo uno o algunos ejemplos de entrenamiento. En otras palabras, cuando el número de conjuntos de datos de entrenamiento es elevado,

En esta experimentación se han utilizado dos optimizadores para las redes neuronales: el descenso de Gradiente Estocástica (SGD) y Adam.

SGD funciona rápidamente porque no utiliza la totalidad de las instancias de entrenamiento en cada integración, lo que reduce la cantidad de cálculos y mejora la velocidad de cálculo [186]. Además, el SGD puede ajustar dinámicamente la estimación de las matrices de primer y segundo orden del gradiente de cada parámetro según la función de pérdida. Así, se puede reducir el riesgo de que el modelo converja al óptimo local. Teniendo en cuenta estas ventajas, especulamos que el uso de SGD puede superar el coste computacional y conducir a una rápida convergencia.

El SGD es una aproximación estocástica de la optimización del descenso gradiente y utiliza solo uno o algunos ejemplos de entrenamiento en cada iteración, lo que reduce la cantidad de cálculos y mejora la velocidad de cálculo.

El SGD se puede detallar de la siguiente manera [187]:

En primer lugar, se asigna un vector de ceros al peso W_1 y a continuación, se selecciona de manera aleatoria una muestra de entrenamiento (x_{it}, y_{it}) de todo el conjunto de entrenamiento, donde $i_t \in (1, , m)$ es el objetivo de la muestra de entrenamiento seleccionada en la iteración t. La función objetivo es:

$$min(W) = \frac{\lambda}{2}||W||^2 + f(W, (x_{it}, y_{it})) \qquad (4.2)$$

En segundo lugar, se calcula el gradiente en función de la ecuación 4.2 y por lo tanto, el gradiente puede expresarse mediante:

$$\Delta_t = \lambda W_t - \alpha_t y_{it} x_{it} \qquad (4.3)$$

donde:

$\alpha_t = 1, \quad$ si $\quad y_{it}(W_t, x_{it}) < 1$

$\alpha_t = 0$, para otros casos

La fórmula de la matriz de los pesos, W, es la siguiente:

$$W_{t+1} = W_t - \eta\alpha \tag{4.4}$$

donde:
$\eta_t = \frac{1}{\lambda t}$

Por consiguiente, se puede obtener una matriz de pesos W actualizada basada en las fórmulas 4.2 y 4.3 mediante:

$$W_{t+1} = (1 - \frac{1}{t})W_t + y_{it}x_{it} \tag{4.5}$$

En la práctica, la función 4.5 se utiliza para encontrar mínimos o máximos por iteración.

Adam

Adam es un método de optimización de SGD que mide los índices de aprendizaje adaptables para cada parámetro [188]. Es una mezcla de RMSProp y Momentum, y la operación de actualización considera la variante del gradiente suavizado y proporciona un mecanismo de corrección del sesgo [189] [190]. El método es realmente eficaz cuando se trabaja con conjuntos de datos grandes, ya que, reduce los costes de computación debido a que necesita menos memoria de ejecución y es invariable al cambio de diagonal del gradiente [188]. Este algoritmo se utiliza para acelerar el algoritmo de descenso de gradiente teniendo en cuenta la "media ponderada exponencialmente" de los gradientes. El uso de promedios hace que el algoritmo

Por otro lado, Adam mide los índices de aprendizaje adaptables para cada parámetro y utiliza la "media ponderada exponencialmente" de los gradientes para acelerar el algoritmo de descenso de gradiente.

converja hacia los mínimos a un ritmo más rápido [191]:

$$W_{t+1} = W_t - \alpha m_t \tag{4.6}$$

$$m_t = \beta m_{t-1} + (1 - \beta)[\frac{\delta L}{\delta W_t}] \tag{4.7}$$

donde:

m_t = agregado de gradientes en el momento t [actual] (inicialmente, $m_t = 0$)
m_{t-1} = agregado de gradientes en el momento t-1 [anterior]
Wt = pesos en el momento t
$Wt + 1$ = pesos en el momento t+1
α_t = tasa de aprendizaje en el momento t
δL = derivada de la función de pérdida
δW_t = derivada de los pesos en el momento t
β = Parámetro medio cambiante (const. 0.9)

Ambos optimizadores son efectivos para trabajar con grandes conjuntos de datos, reduciendo los costes de computación al utilizar menos memoria de ejecución. Otros optimizadores se probaron pero no pudieron ejecutarse debido a problemas computacionales.

En la experimentación, se han utilizado estos dos optimizadores (SGD y Adam) porque funcionan muy bien con grandes conjuntos de datos, reduciendo los costes de computación al utilizar menos memoria de ejecución [188]. Se ha probado a utilizar otros optimizadores, pero no se ha podido ejecutar por problemas computacionales.

4.3.1. Resultados modelo Esmalte

A continuación, en las Tablas 4.3 y 4.4 se muestran los resultados de la experimentación.

Capas ocultas	Adam (%)	SGD (%)
(256, 64)	77,09	76,39
(256, 80)	77,24	76,24
(256, 96)	76,32	75,54
(256, 112)	77,01	75,85
(256, 128)	75,93	77,24
(256, 144)	76,55	76,47
(256, 160)	76,47	76,55
(256, 176)	77,32	75,93
(256, 192)	78,10	76,63
(256, 208)	76,86	76,24
(256, 224)	77,32	76,24
(256, 240)	76,86	74,92
(256, 256)	77,40	75,85

Tabla 4.3: Resultados de precisión del modelo de Esmalte con diferentes configuraciones

Capas ocultas	Adam (%)	SGD (%)
(272, 256)	77,94	76,47
(288, 256)	76,63	76,93
(304, 256)	76,39	75,62
(256, 272)	76,24	75,62
(272, 272)	77,09	76,63
(288, 272)	76,70	76,01
(288, 272)	76,70	76,01
(304, 272)	76,47	75,15
(272, 288)	77,01	76,32
(288, 288)	76,55	75,93
(304, 288)	77,17	76,24
(288, 304)	75,77	76,70
(304, 304)	76,86	75,54

Tabla 4.4: Resultados de precisión del modelo de Esmalte con diferentes configuraciones

Las tablas 4.3 y 4.4muestra los resultados de una experimentación que utilizó diferentes configuraciones de redes neuronales con los optimizadores Adam y SGD. Los resultados muestran que la configuración de capas (256, 192) con Adam como optimizador obtuvo la mayor precisión del 78.10 %, lo que sugiere que una mayor cantidad de capas no siempre conduce a mejores resultados.

Como se puede observar, tal y como se ha comentado en el punto anterior, la experimentación se ha llevado a cabo con redes neuronales de diferentes configuraciones. Es por eso, que para cada combinación de capas ocultas, se ha utilizado tanto el oprimizador *Adam*, como el *SGD*.

Con los resultados a la vista, cabe destacar, que no por tener más capas se obtienen mejores resultados,

ya que la configuración que mejores resultados ha dado es la que tiene *(256, 192)* capas y utilizado como optimizador *Adam*. Esta configuración ha obtenido un 78.10 % de precisión, la más alta frente al resto de configuraciones.

4.3.2. Resultados modelo Apresto

La tabla 4.5 muestra el resumen de los resultados de precisión de la experimentación con el modelo de los datos de Apresto.

Tabla 4.5: Resultados de precisión del modelo de Apresto con diferentes configuraciones

Capas ocultas	Adam (%)	SGD (%)	Capas ocultas	Adam (%)	SGD (%)
(208, 64)	65,10	65,10	(256, 256)	64,58	63,72
(208, 80)	63,54	63,02	(272, 256)	63,54	64,41
(208, 96)	63,71	62,32	(288, 256)	65,45	63,72
(208, 112)	63,19	63,19	(304, 256)	66,15	63,72
(208, 128)	67,01	64,76	(256, 272)	64,24	63,89
(208, 144)	63,71	63,71	(272, 272)	63,37	65,28
(208, 160)	64,06	62,67	(288, 272)	61,11	63,54
(208, 176)	63,72	63,89	(304, 272)	63,37	62,67
(208, 192)	65,62	63,71	(272, 288)	63,37	63,89
(208, 208)	64,24	63,71	(288, 288)	64,24	63,19
(208, 224)	64,76	63,19	(304, 288)	65,97	64,58
(240, 240)	63,02	64,41	(288, 304)	64,41	63,19
(304, 240)	64,24	62,85	(304, 304)	63,37	64,24

La tabla 4.5 presenta los resultados de precisión de la experimentación con el modelo de los datos de Apresto. La configuración con mejor precisión utilizó (208, 128) capas y el optimizador *Adam*, obteniendo una precisión del 67.01 %.

En este caso, la configuración que mejor precisión ha obtenido es la que tiene (208, 128) capas y usa como optimizador *Adam*, con un 67.01 % de precisión.

4.3.3. Resultados modelo Apresto y Esmalte unificados

Como el conjunto de datos de Apresto y de Esmalte pertenecen al proceso de pintado de vehículos, se

ha contemplado la opción de generar un único modelo de 'Pintura' que contenga los datos de ambos subprocesos. Es por eso, que se ha generado este modelo cogiendo ambos conjuntos de datos, para después, poder realizar una comparativa entre los resultados de los modelos generados individualmente y en conjunto. A continuación, en la Tabla 4.6 y Tabla 4.7 se muestran los resultados de precisión del modelo unificado.

Capas ocultas	**Adam (%)**	**SGD (%)**
(288, 64)	26,04	26,04
(288, 80)	28,47	27,43
(288, 96)	28,30	26,91
(288, 112)	26,74	23,61
(288, 128)	26,73	26,39
(288, 144)	25	25,69
(288, 160)	25	24,31
(288, 176)	27,95	27,08
(288, 192)	27,60	25,35
(288, 208)	26,22	25,17
(288, 224)	28,30	28,30
(288, 240)	29,89	27,43
(288, 256)	27,95	26,91

Tabla 4.6: Resultados de precisión del modelo de Apresto y Esmalte unificados con diferentes configuraciones

Capas ocultas	**Adam (%)**	**SGD (%)**
(256, 256)	25,52	25,87
(272, 256)	25,52	26,39
(288, 256)	27,95	26,91
(304, 256)	27,60	27,08
(256, 272)	28,30	27,26
(272, 272)	28,65	26,91
(288, 272)	26,04	26,04
(304, 272)	26,39	26,04
(272, 288)	26,04	26,74
(288, 288)	25,87	26,91
(304, 288)	29,86	26,22
(288, 304)	27,26	25,17
(304, 304)	28,82	28,30

Tabla 4.7: Resultados de precisión del modelo de Apresto y Esmalte unificados con diferentes configuraciones

Como puede apreciarse, en el caso del modelo unificado, la precisión es bastante baja siendo la configuración que mejores resultados da la que tiene (288, 240) capas y utiliza el optimizador *Adam*.

Se ha considerado la posibilidad de crear un modelo de "Pintura" que incluya los datos de los subprocesos de Apresto y Esmalte, ya que ambos pertenecen al proceso de pintado de vehículos. Se generó un modelo unificado utilizando ambos conjuntos de datos y se comparó con los modelos individuales. En las tablas 4.6 y 4.7 se presentan los resultados de precisión del modelo unificado. Sin embargo, la precisión es baja y la configuración que da mejores resultados es la que tiene (288, 240) capas y utiliza el optimizador *Adam*.

4.3.4. Comparativa resultados de precisión entre los modelos de Esmalte, Apresto y ambos modelos unificados

Una vez obtenidos los resultados de precisión de cada caso, la tabla 4.8 refleja la comparativa de precisión de modelos, habiendo cogido el mejor de los resultados de la experimentación para cada caso.

Tabla 4.8: Comparativa de resultados de precisión entre los modelos de Esmalte, Apresto y ambos modelos unificados

Modelo Edge Esmalte (%)	Modelo Edge Apresto (%)	Modelo Apresto y Esmalte (%)
78,10	67,01	39,86

Como se puede ver, la precisión de los modelos cuando se generan por separado es mayor que si se genera un modelo completo que contiene todos los datos, al igual que ha pasado en la experimentación con el conjunto de datos de '*Bosch*'.

La tabla 4.8 presenta una comparativa de precisión entre los modelos generados individualmente para los conjuntos de datos de Apresto y Esmalte, y el modelo unificado que contiene ambos conjuntos. Los resultados indican que la precisión de los modelos individuales es mayor que la del modelo unificado, lo cual también se observó en la experimentación con el conjunto de datos de Bosch.

4.4. Conclusiones

Una vez concluidas ambas experimentaciones, con los resultados de precisión ya analizados, en ambos casos ha sucedido algo similar: los modelos generados con conjuntos de datos de partes del proceso obtienen mejores resultados que modelos grandes que contemplan varios subprocesos.

Como se ha podido ver en el apartado anterior de comparativa de resultados, los modelos *Edge* consiguen una precisión de hasta más de dos veces mejor que el modelo generado con todos los datos, alcanzando un 78,10 % para el modelo de esmalte y un 67,01 % para el modelo de apresto. Por el contrario, cuando se crea un modelo con los datos tanto de apresto como de esmalte, ambos juntos, apenas se consigue una precisión del 30 %. Con todo esto, se puede concluir que en algunos casos, no por tener más datos, es decir, más información, se obtienen mejores resultados en un modelo, ya que, en las experimentaciones llevadas a cabo se ha demostrado lo contrario. Esto sucede, debido a que los datos de un proceso independiente de otro, a la hora de predecir, pueden actuar como 'ruido', como pasa con las líneas de producción en la experimentación del conjunto de datos de '*Bosch*' y como pasa con los subprocesos de esmalte y apresto en la experimentación del conjunto de datos del pintado de vehículos.

En ambas experimentaciones se observó que los modelos generados con conjuntos de datos de partes del proceso obtienen mejores resultados que los modelos más grandes que contemplan varios subprocesos. En la experimentación del conjunto de datos del pintado de vehículos, los modelos Edge consiguen una precisión de hasta más del doble que el modelo generado con todos los datos, mientras que el modelo unificado apenas logra una precisión del 30 %. Esto demuestra que en algunos casos, tener más datos no siempre resulta en un modelo más preciso, ya que los datos de un proceso independiente de otro pueden actuar como ruido al predecir, como se vio en la experimentación del conjunto de datos de Bosch y en la del pintado de vehículos.

Optimización del flujo de datos

5

5.1. Experimentación

5.1 Experimentación 115
5.1.1 Reducción de instancias 118
5.1.2 Reducción de dimensionalidad 119
5.1.3 Pasos de la experimentación 130
5.1.4 Arquitectura . . . 131
5.1.5 Validación 132
5.1.6 Resultados - Reducción de instancias 133
5.1.7 Resultados - Reducción de dimensionalidad 137
5.2 Conclusiones . . . 138

Este capítulo se ha centrado en buscar una solución a escenarios en los que el uso de *Edge Computing* puede verse comprometido, o lo que es lo mismo, se ha centrado en intentar paliar las restricciones existentes de este tipo de entornos, que como se ha podido observar una vez concluidas las experimentaciones del capítulo 3, *Análisis Hardware*, las restricciones tienen tienen relación directa con el volumen de datos que han de proecesar.

Como conclusión de la experimentación realizada en el capítulo 3 acerca del análisis hardware, la principal limitación en los dispositivos *Edge* viene determinada por el tamaño del conjunto de datos a procesar. Sin embargo, debido a la digitalización de las empresas [9], el volumen del conjunto de datos que se recoge cada determinado tiempo sobre la maquinaria suele ser lo suficientemente grande y pesado para procesarlo de manera directa en este tipo de entornos debido a la dimensionalidad y al gran número de instancias.

Por lo tanto, por un lado está la captación masiva de datos, y por otro lado, la restricción de procesar conjuntos de datos de gran volumen en dispositivos *Edge*. Es por eso, que existe la necesidad de buscar una solución para optimizar el flujo de datos entre los dispositivos que capturan el dato (IoT) y los dispositivos que procesan el dato (dispositivos *Edge*). El objetivo final de esta experimentación

En este capítulo se busca una solución a las restricciones de los entornos de Edge Computing, especialmente en relación con el tamaño de los conjuntos de datos a procesar. Se observa que la limitación principal en los dispositivos Edge está determinada por el volumen del conjunto de datos a procesar, lo que resulta en la necesidad de buscar una solución para optimizar el flujo de datos entre los dispositivos IoT y los dispositivos Edge.

por lo tanto, es buscar una técnica adecuada de compresión de datos para que en lugar de tener que procesar el volumen original de los datos en bruto que se recolectan en las máquinas, pueda procesarse un conjunto comprimido pero lo suficientemente representativo para que mantenga la eficacia a la hora de generar modelos de inteligencia artificial. Es decir, partiendo de un voluminoso conjunto de datos con muchas instancias, se quiere generar un conjunto de datos más pequeño que sea capaz de generar modelos de inteligencia artificial con una precisión igual o similar a los generados con el conjunto de datos original.

El objetivo final es encontrar una técnica de compresión de datos adecuada para generar un conjunto de datos más pequeño pero representativo que pueda generar modelos de inteligencia artificial con una precisión igual o similar al conjunto de datos original voluminoso.

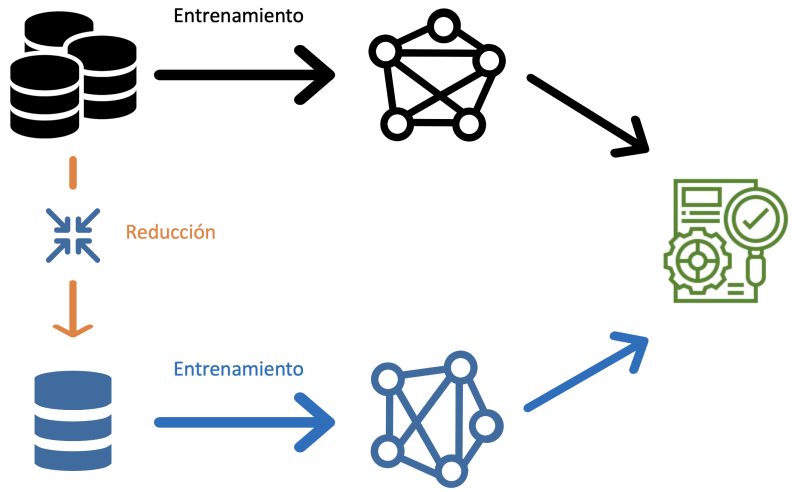

Figura 5.1: Reducción del conjunto de datos con el objetivo de obtener resultados similares

Pero, **¿Cómo se consiguen conjuntos de datos reducidos y, que a la vez, sean representativos?** Para ello, ha sido necesario aplicar nuevos enfoques. Se ha realizado una experimentación en la que se han probado técnicas de compresión de hasta un 90 % de reducción. Es bastante improbables que el modelo del conjunto de datos reducido tenga exactamente la misma precisión que el modelo generado con los datos originales. Sin embargo, en caso de que la precisión sea ligeramente menor,

la pérdida deberá ser insignificante para que la técnica pueda ser validada.

En la experimentación se han utilizado dos enfoques para obtener conjuntos de datos reducidos pero representativos: reducción de instancias y reducción de dimensionalidad. Para la reducción de instancias, se técnica que se ha utilizado es la compresión Bayesiana mediante el uso de redes Bayesianas en el software Weka. Para la reducción de dimensionalidad, se han utilizado autoencoders, que constan de un codificador y un decodificador.

Se han empleado dos enfoques diferentes con el fin de conseguir la reducción de datos: por un lado la reducción del número de instancias y, por otro, la reducción de la dimensionalidad de los datos. Para la reducción de instancias, se ha aplicado el Teorema de Bayes con la compresión Bayesiana, y para la reducción de dimensionalidad, se ha hecho uso de *autoencoders*.

Para la compresión Bayesiana, se han aplicado redes Bayesianas a los conjuntos de datos originales haciendo uso de las bibliotecas de la plataforma de software *Weka*. [1] El número total de salidas de la red Bayesiana varía en función del porcentaje de reducción que se aplique al conjunto de datos original.

1: Weka: *Waikato Environment for Knowledge Analysis* es un software libre con licencia que contiene herramientas de visualización y algoritmos para el análisis de datos y modelado predictivo.

Por otro lado, en cuanto a los *autoencoders*, estos se componen de dos elementos principales: un codificador y un decodificador. Al codificador se le entregará el conjunto de datos original con el objetivo de generar un vector latente, y el decodificador será el encargado de reconstruir los datos originales a partir del vector latente obtenido previamente del codificador. La comparación de los resultados de precisión obtenidos mediante el uso de algoritmos de aprendizaje automático entre los conjuntos de datos originales y los comprimidos demostró que se ha alcanzado el objetivo principal de mantener la tasa de precisión del modelo. De este modo, podría ser un avance para los problemas computacionales que pueden surgir al procesar grandes volúmenes de datos y ayudaría a proporcionar respuestas más rápidas [11].

La comparación de los resultados entre los conjuntos de datos originales y los comprimidos ha mostrado que se ha alcanzado el objetivo de mantener la precisión del modelo, lo que podría ayudar a resolver problemas computacionales al procesar grandes volúmenes de datos y proporcionar respuestas más rápidas.

A continuación, se muestra la experimentación realizada para cada uno de los enfoques.

5.1.1. Reducción de instancias

En esta sección se describe el proceso de reducción de instancias mediante el uso de redes bayesianas. Este proceso se lleva a cabo con el fin de reducir el número de filas del conjunto de datos original. Se utiliza la plataforma de software libre Weka para aplicar las redes bayesianas y el número de *outputs* generados por la red depende del grado de reducción que se aplique al conjunto de datos original.

Con el objetivo de obtener una reducción del conjunto de datos en número de filas, se han aplicado redes bayesianas al conjunto de datos original utilizando las librerías de la plataforma de software libre *Weka*. El número de *outputs* generados por la red bayesiana varía en función del grado de reducción que se aplique al conjunto de datos original.

Comportamiento del algoritmo

El algoritmo de compresión bayesiana sigue varios pasos para llevar a cabo la reducción de instancias. En primer lugar, se verifica que cada instancia tenga una clase de predicción unitaria para poder ser gestionada por las librerías de Weka. A continuación, se crea un clasificador bayesiano a partir de los datos disponibles y se ordenan los nodos adecuadamente. Se genera un diccionario para almacenar la información y se calcula el número de instancias que se deben recoger en función del porcentaje de compresión deseado.

A continuación se detallan los pasos que sigue el algoritmo para realizar la compresión bayesiana de datos durante la presente experimentación:

- Se recogen los datos originales y se comprueba que cada instancia tiene una clase de predicción unitaria. Esta verificación se realiza para determinar si las clases pueden ser gestionadas por las librerías de Weka.
- Para comprimir los datos, a partir de los datos disponibles se crea un clasificador bayesiano. Una vez obtenido,los nodos se ordenan adecuadamente para el sistema de manera que desde la posición de un nodo en la lista hacia la derecha no haya ningún padre de ese nodo. Se genera un diccionario para almacenar la información.
- El número de instancias que se deben recoger se calcula en función del porcentaje de compresión que se desee. A partir del diccionario de nodos, se pasan los nodos ordenados tal

y como fueron devueltos por el método de ordenación. Se obtiene la probabilidad del estado del nodo y se ordenan de menor a mayor. Los estados seleccionados se añaden como instancias generadas como *outputs* y generan un diccionario. Porsteriormente, a partir del diccionario que se ha construido, estos valores se devuelven como un conjunto de instancias, siendo el número de estas equivalente al porcentaje de compresión que se haya querido aplicar.

A partir del diccionario de nodos, se obtiene la probabilidad del estado del nodo y se ordenan de menor a mayor. Los estados seleccionados se añaden como instancias generadas como *outputs* y generan un diccionario. Finalmente, a partir del diccionario que se ha construido, estos valores se devuelven como un conjunto de instancias, siendo el número de estas equivalente al porcentaje de compresión que se haya querido aplicar. La figura 5.2 muestra de manera visual los pasos seguidos por el algoritmo de compresión bayesiana.

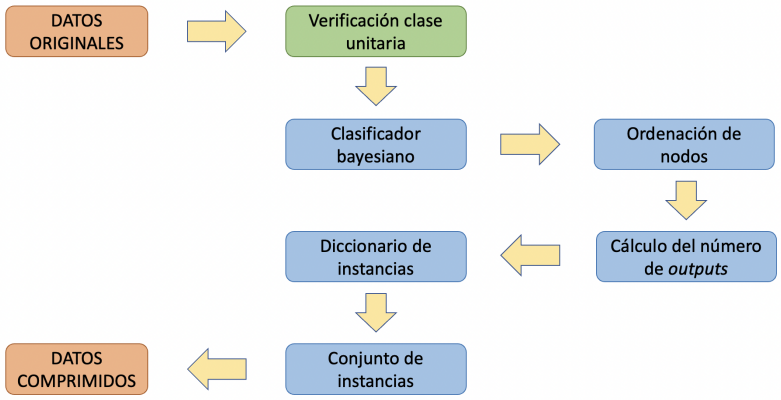

Figura 5.2: Diagrama de flujo de compresión bayesiana.

En la figura 5.2 superior pueden observarse los pasos seguidos por el algoritmo de compresión bayesiana para que partiendo de un conjunto de datos original se obtenga un conjunto de datos comprimido del mismo.

5.1.2. Reducción de dimensionalidad

Para obtener una reducción del conjunto de datos en cuanto a columnas, se ha llevado a cabo utilizando un *Autoencoder*. El *Autoencoder* consta de dos elementos: un codificador y un decodificador. El número de entradas del codificador tiene que coin-

cidir con el número de salidas del decodificador. La figura 5.3 muestra la estructura del *Autoencoder*.

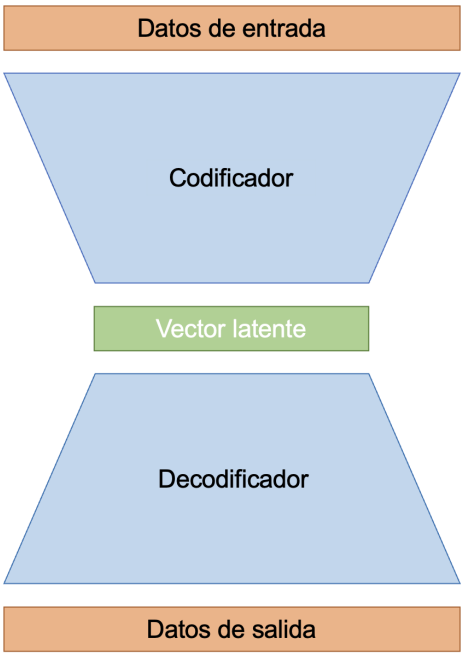

Figura 5.3: Estructura del *Autoencoder*

En cuanto al funcionamiento, el *Autoencoder* lo compone una red neuronal convolucional, y lo primero que ha de hacerse es especificar las capas de la red del codificador y del decodificador. Para este caso concreto se han utilizado capas convolucionales de una dimensionalidad. El número de capas añadidas a ambas partes ha sido diferente, ya que el codificador tiene que comprimir los datos, y el decodificador expandirlos. El número de capas también varía en función de la tasa de reducción que se quiera conseguir, ya que el objetivo es conseguir que el vector latente sea de un tamaño u otro. Por ejemplo, si el objetivo es reducir el conjunto de datos de 24 columnas a 12, el número de capas necesarias será diferente que si el objetivo es reducir el conjunto de datos de 24 columnas a 6. La Figura 5.4 muestra la configuración correspondiente a la reducción de columnas de 24 a 12, es decir, una reducción del 50 %.

Se ha utilizado un Autoencoder para reducir el conjunto de datos en cuanto a columnas. El *Autoencoder* lo compone una red neuronal convolucional y consta de un codificador y un decodificador con el mismo número de entradas y salidas respectivamente. Se han utilizado capas convolucionales de una dimensión en ambos componentes, añadiendo un número diferente de capas en función de la tasa de reducción deseada. Se muestra la configuración correspondiente a una reducción del 50 % de columnas de 24 a 12, del 75 % de columnas de 24 a 6 y del 90 % de 24 a 2.

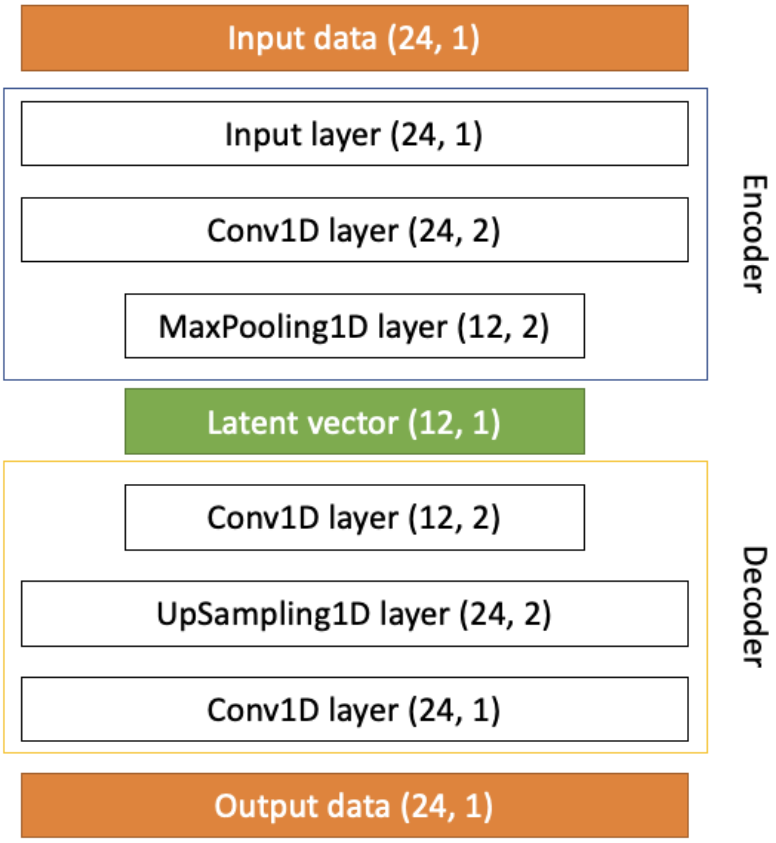

Figura 5.4: Configuración del *Autoencoder* para la reducción de 24 a 12

A continuación, la Figura 5.5, muestra la de la reducción del 75 %, de 24 a 6.

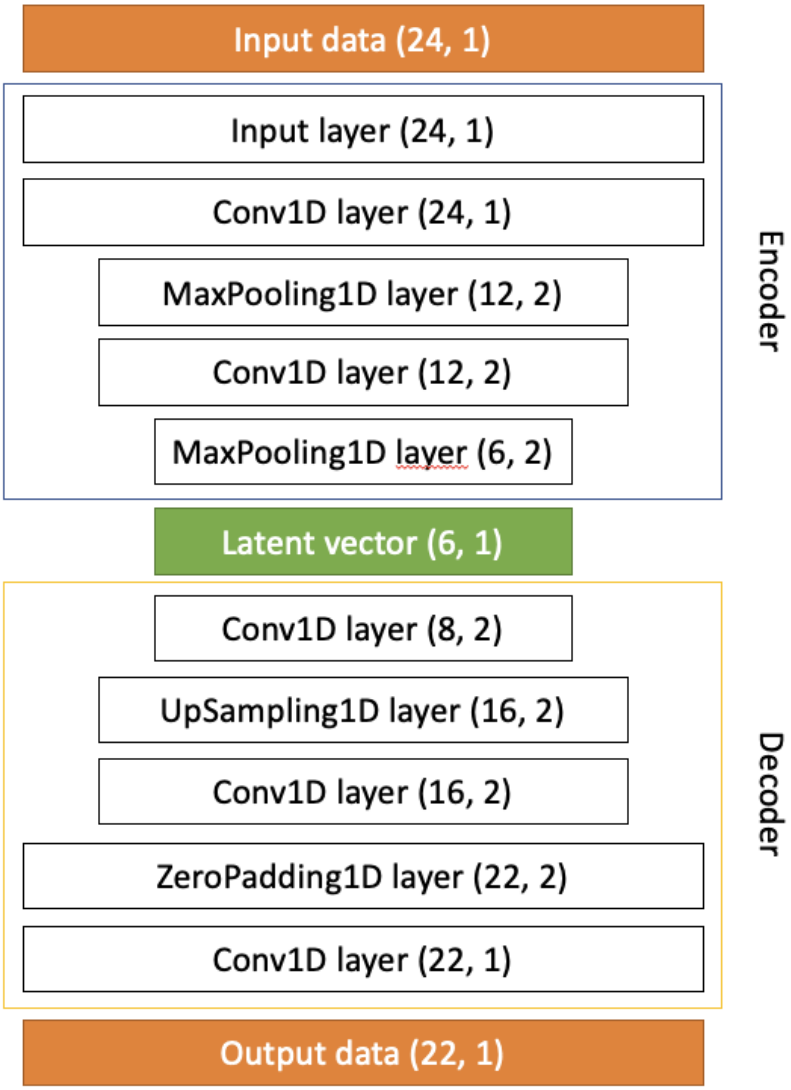

Figura 5.5: Configuración del *Autoencoder* para la reducción de 24 a 6

Por último, se muestra en la Figura 5.6la configuración del autoencoder para la reducción del 90 % para reducir de 24 a 2.

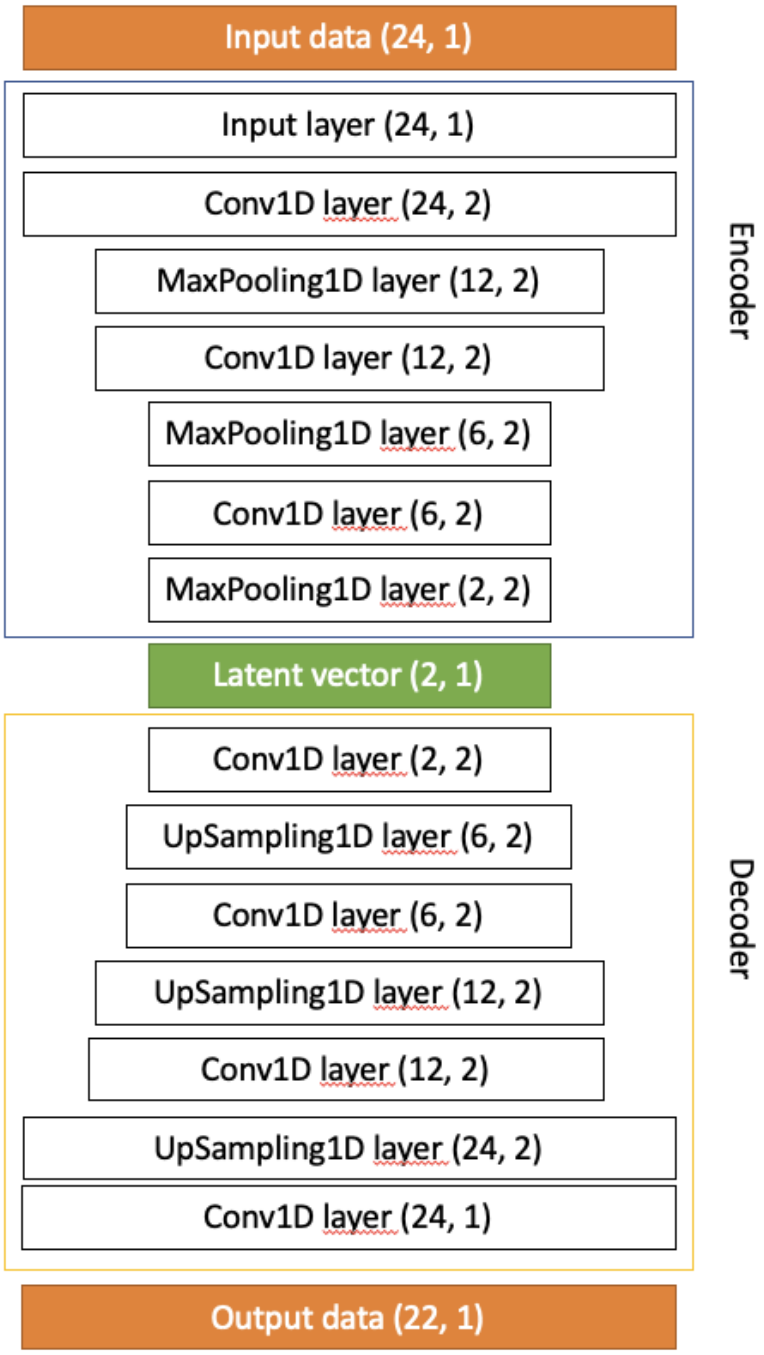

Figura 5.6: Configuración del *Autoencoder* para la reducción de 24 a 2

El siguiente código generado con *Keras* [2] muestra las diferentes configuraciones para cada reducción. La tupla que se especifica entre parentesis indica el tamaño de salida de cada capa. La forma de salida *(None, x, y)* indica el tamaño de la salida de la capa. En este caso, *None* representa el tamaño de la muestra (que puede variar), *x* es la longitud de

2: *Keras:* Es una librería de software de código abierto que ofrece una interfaz de Python para redes neuronales [192].

la secuencia después de aplicar las convoluciones y el pooling, e *y* es el número de canales o características resultantes de las convoluciones. El número de parámetros de una capa representa la cantidad de pesos entrenables que la capa tiene. Estos parámetros se utilizan para aprender y ajustar los filtros convolucionales durante el entrenamiento de la red.

50 % de reducción

El siguiente código muestra la configuración de la red neuronal convolucional para la reducción del 50 % de las columnas:

```
Model: "24to12columns_encoder"
_________________________________________
Layer (type)        Output shape   Param#
=========================================
conv1d (Conv1D)     (None, 24, 2)    8
max_pooling1d
  (MaxPooling1D)    (None, 12, 2)    0
conv1d_1 (Conv1D)   (None, 12, 1)    7

=========================================
Total params: 15
Trainable params: 15
Non-trainable params: 0
_________________________________________
Model: "24to12columns_decoder"
_________________________________________
Layer (type)        Output shape   Param#
=========================================
conv1d_2 (Conv1D)   (None, 12, 2)    8
up_sampling1d
  (UpSampling1D)    (None, 24, 2)    0
```

```
conv1d_3 (Conv1D)   (None, 24, 1)    7

=================================================
Total params: 15
Trainable params: 15
Non-trainable params: 0
_________________________________________________
```

Figura 5.7: Código del *Autoencoder* para la reducción de 24 coumnas a 12

Como podemos ver en la Figura 5.9, en la parte superior del código, en el codificador, los datos de entrada tienen 24 columnas y se pasan a través de una capa convolucional de 1 dimensión. A continuación, se reduce su tamaño a 12 columnas utilizando *Max Pooling* y se pasa a la última capa convolucional.

Por otro lado, el decodificador recibe la salida del codificador, es decir, las 12 columnas, y la pasa a través de una capa convolucional de 1 dimensión, luego la dimensionalidad se incrementa hasta 24 columnas de nuevo utilizando la capa *upsampling* [3] Finalmente, los datos reconstruidos vienen dados por la última capa convolucional que obtiene los datos de la capa de muestreo ascendente anterior.

En el codificador, los datos de entrada con 24 columnas pasan por una capa convolucional de 1 dimensión, se reducen a 12 columnas mediante Max Pooling y se pasan por la última capa convolucional. En el decodificador, las 12 columnas se pasan por una capa convolucional de 1 dimensión, se incrementa la dimensionalidad a 24 columnas mediante una capa de upsampling y se obtienen los datos reconstruidos mediante la última capa convolucional.

3: *Upsampling:* Muestreo ascendente. Es una técnica que produce una aproximación de la secuencia que se habría obtenido muestreando la señal a una frecuencia mayor [193].

75 % de reducción

El siguiente código muestra la configuración de la red neuronal convolucional para la reducción del 75 % de las columnas:

```
Model: "24to6columns_encoder"
_________________________________________________
```

```
Layer (type)        Output shape   Param#
=================================================
conv1d (Conv1D)     (None, 24, 1)    8
max_pooling1d
  (MaxPooling1D)    (None, 12, 2)    0
conv1d_1 (Conv1D)   (None, 12, 2)    14
max_pooling1d_1
  (MaxPooling1D)    (None, 6, 2)     7


=================================================
Total params: 29
Trainable params: 29
Non-trainable params: 0
_________________________________________________
Model: "24to6columns_decoder"
_________________________________________________
Layer (type)        Output shape   Param#
=================================================
conv1d_3 (Conv1D)   (None, 6, 2)     8
up_sampling1d
  (UpSampling1D)    (None, 12, 2)    0
conv1d_4 (Conv1D)   (None, 12, 2)    14
up_sampling1d_1
  (UpSampling1D)    (None, 24, 2)    0
conv1d_5 (Conv1D)   (None, 24, 1)    7

=================================================
Total params: 29
Trainable params: 29
Non-trainable params: 0
_________________________________________________
```

Figura 5.8: Código del *Autoencoder* para la reducción de 24 coumnas a 6

En este caso, se repite el mismo proceso inicial que en la anterior compresión, los datos iniciales se pasan por una capa convolucional de 1 dimensión y posteriormente se reduce su tamaño a 12 columnas a través del max pooling. Se repite el mismo proceso pasando los datos a la capa convolucional de 1 dimensión y reduciendo el tamaño a 6 columnas con el mismo método. Finalmente, los datos de salida de esta capa se pasan a la última capa convolucional. El decodificador aplica una capa convolucional de 1 dimensión a la salida del codificador para luego incrementar la dimensionalidad a 12 columnas utilizando la capa de muestreo ascendente (*upsampling*). El proceso anterior se repite de nuevo con las mismas 2 capas para aumentar la dimensionalidad a 24 columnas y los datos reconstruidos son dados por la última capa convolucional que obtiene los datos de la capa de muestreo ascendente anterior.

En este caso, se sigue un proceso similar al anterior, donde los datos iniciales pasan por una capa convolucional de 1 dimensión y se reducen a 12 y 6 columnas mediante Max Pooling en dos pasos sucesivos. En el decodificador, se aplica una capa convolucional de 1 dimensión a la salida del codificador, se incrementa la dimensionalidad a 12 columnas mediante la capa de upsampling y se repite el proceso para obtener 24 columnas. Los datos reconstruidos son dados por la última capa convolucional.

90 % de reducción

El siguiente código muestra la configuración de la red neuronal convolucional para la reducción del 95 % de las columnas:

```
Model: "24to2columns_encoder"
_________________________________________
Layer (type)        Output shape   Param#
=========================================
conv1d (Conv1D)    (None, 24, 2)     8
max_pooling1d
  (MaxPooling1D)   (None, 12, 2)     0
conv1d_1 (Conv1D)  (None, 12, 2)     14
max_pooling1d_1
  (MaxPooling1D)   (None, 6, 2)      0
```

```
conv1d_2 (Conv1D)  (None, 6, 2)      14
max_pooling1d_2
  (MaxPooling1D)   (None, 2, 2)      0
conv1d_3 (Conv1D)  (None, 2, 1)      7
=================================================
Total params: 43
Trainable params: 43
Non-trainable params: 0
_________________________________________________
Model: "24to12columns_decoder"
_________________________________________________
Layer (type)        Output shape   Param#
=================================================
conv1d_4 (Conv1D)   (None, 2, 2)     8
up_sampling1d
  (UpSampling1D)    (None, 6, 2)     0
conv1d_5 (Conv1D)   (None, 6, 2)     14
up_sampling1d_1
  (UpSampling1D)    (None, 12, 2)    0
conv1d_6 (Conv1D)   (None, 12, 2)    14
up_sampling1d_2
  (UpSampling1D)    (None, 24, 2)    0
conv1d_7 (Conv1D)   (None, 24, 1)    7

=================================================
Total params: 43
Trainable params: 43
Non-trainable params: 0
```

Figura 5.9: Código del *Autoencoder* para la reducción de 24 coumnas a 2

En esta configuración (Figura 5.6), como en las anteriores, los datos de entrada de 24 columnas se pasan por una capa convolucional de 1 dimensión

y luego se reduce su tamaño a 12 columnas utilizando la técnica *Max Pooling*. Después, se repite el mismo proceso para obtener 6 columnas. Estos datos se pasan a la siguiente capa convolucional y la dimensión se reduce a 2 columnas utilizando *Max Pooling* una vez más. Finalmente, los datos de salida de esta capa se pasan a la última capa convolucional. Posteriormente, el decodificador repite el proceso de las configuraciones anteriores aumentando la dimensionalidad con la técnica *upsampling* de 2 a 6 columnas primero, de 6 a 12 después, y finalmente, de 12 a 24.

En esta configuración, se sigue un proceso similar a las anteriores, donde se utiliza una capa convolucional de 1 dimensión para reducir la dimensionalidad de los datos de entrada de 24 a 12 columnas a través de *Max Pooling*. Luego, se repite el mismo proceso para obtener 8 columnas y se pasa a la siguiente capa convolucional. En esta ocasión, la dimensión se reduce a 4 columnas usando nuevamente la técnica *Max Pooling*. Finalmente, los datos de salida de esta capa se pasan a la última capa convolucional. El decodificador aumenta la dimensionalidad utilizando la técnica *upsampling* de 4 a 8 columnas y de 8 a 12 columnas, antes de obtener los datos reconstruidos de 24 columnas utilizando la última capa convolucional.

Comportamiento del algoritmo

A continuación se detallan los pasos realizados por el algoritmo para la reducción de la dimensionalidad durante la experimentación:

El algoritmo de reducción de dimensionalidad se compone de tres pasos principales.

- Los datos originales son revisados para asegurar que no hay valores atípicos que comprometan el aprendizaje de los modelos de inteligencia artificial. La columna de valores categóricos utilizada para la predicción se separa del resto del conjunto de datos para conservar los valores originales.
- Para la reducción de la dimensionalidad se crea un *Autoencoder*, el cual está compuesto por dos elementos, el codificador como entrada y el decodificador como salida del modelo. El modelo debe ser ajustado por los datos de entrenamiento tomados del conjunto de datos original, para hacerlo lo más óptimo posible se deben especificar los *epochs* [4] y el tamaño del lote teniendo en cuenta la longitud de los datos de entrenamiento.

4: *Epochs:* Es un hiperparámetro que define la cantidad de pasadas que el algoritmo de aprendizaje automático ha completado a través de todo el conjunto de datos de entrenamiento [194].

- Los datos pasan por el codificador para obtener el vector latente, que será el conjunto de datos con la reducción de la dimensionalidad. A continuación, la columna categórica se adjuntará al vector latente para poder realizar predicciones con los nuevos datos reducidos.

Primero, se revisan los datos originales para asegurarse de que no haya valores atípicos y se separa la columna categórica utilizada para la predicción.

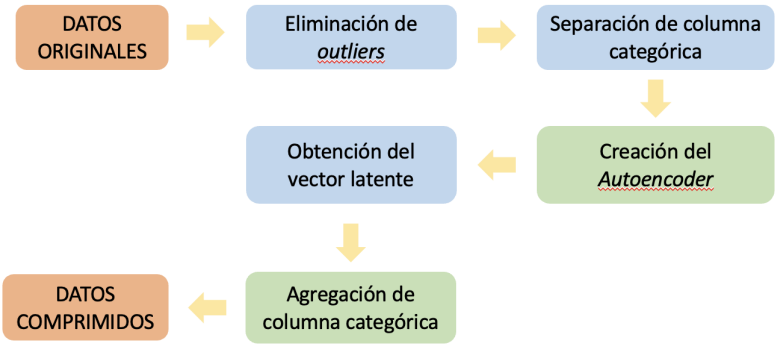

Figura 5.10: Diagrama de compresión bayesiana: paso a paso

Segundo, se crea un autoencoder compuesto por el codificador y el decodificador para reducir la dimensionalidad de los datos de entrada. Se ajusta el modelo utilizando datos de entrenamiento y se especifican los hiperparámetros como epochs y el tamaño del lote. Tercero, los datos se pasan por el codificador para obtener el vector latente, que será el conjunto de datos con la reducción de la dimensionalidad. La columna categórica se adjunta al vector latente para poder realizar predicciones con los nuevos datos reducidos.

Se ha utilizado un conjunto de datos público de Kaggle para la experimentación, que se enfoca en predecir el porcentaje de Silicio en el concentrado de mineral de hierro.

5.1.3. Pasos de la experimentación

Para llevar a cabo la experimentación, se ha tomado un conjunto de datos público de Kaggle, cuyo marco es el paradigma de la industria [195]. El conjunto de datos trata de un proceso industrial que concentra mineral de hierro. El objetivo principal es predecir el porcentaje de Silicio al final del proceso, que se obtiene a partir del concentrado de mineral de hierro y su impureza. Para ello, en la planta existen sensores que miden la temperatura, el pH, el flujo, la densidad, así como todas las variabels continuas del proceso, donde los datos son recogidos cada 20 segundos. Estos datos son recogidos en bruto. Sin embargo, las variables de calidad, como el % de silicio o de mineral de hierro en el contenido, son mediciones de calidad realizadas mediante análisis de laboratorio. Las muestras para el análisis se recogen cada 15 minutos y son

enviadas al laboratorio. De esta manera, el laboratorio da una respuesta sobre el análisis de calidad del flujo del producto cada dos horas. La calidad viene determinada por el concentrado de mineral de hierro. Con el objetivo de dar una respuesta más rapida, se quiere predecir el % de Silicio en el concentrado de mineral de hierro. De esta manera, se podría conocer la calidad evitando las dos horas de análisis de laboratorio.

El conjunto de datos contiene mediciones de sensores y variables continuas del proceso, así como mediciones de calidad realizadas mediante análisis de laboratorio. Para la experimentación, se ha utilizado un Autoencoder para reducir la dimensionalidad de los datos y predecir el % de Silicio en el concentrado de mineral de hierro de manera más rápida.

5.1.4. Arquitectura

Para poder llevar a cabo el propósito de predecir la calidad sin necesidad de enviar las muestras a laboratorio, se ha definido la siguiente arquitectura. Cabe destacar que esta arquitectura se ha diseñado con el objetivo de poder adaptarse a cualquier tipo de proceso. Cuando los datos se recogen, se almacenan en una base de datos con el fin de poder hacer uso de ellos a posteriori a la hora de crear el modelo predictivo. Como se puede ver en la Figura 5.11, con los datos almacenados se realiza la compresión bayesiana en un servidor informático de gran escala. Una vez obtenida la reducción del conjunto de datos, los datos son pasados a los pequeños nodos que están distribuidos en la planta. De esta forma, cada nodo *edge* puede crear modelos *ad-hoc* en función de lo que se quiera predecir en cada uno de ellos de una forma mucho más eficiente y rápida que teniendo todos los datos en bruto.

Se ha diseñado una arquitectura para predecir la calidad del concentrado de mineral de hierro sin necesidad de enviar muestras al laboratorio. Los datos se recogen y se almacenan en una base de datos, y se realiza la compresión bayesiana en un servidor informático. Luego, los datos reducidos se pasan a nodos distribuidos en la planta, donde se pueden crear modelos ad-hoc para predecir diferentes variables. Esta arquitectura se ha diseñado para poder adaptarse a cualquier tipo de proceso.

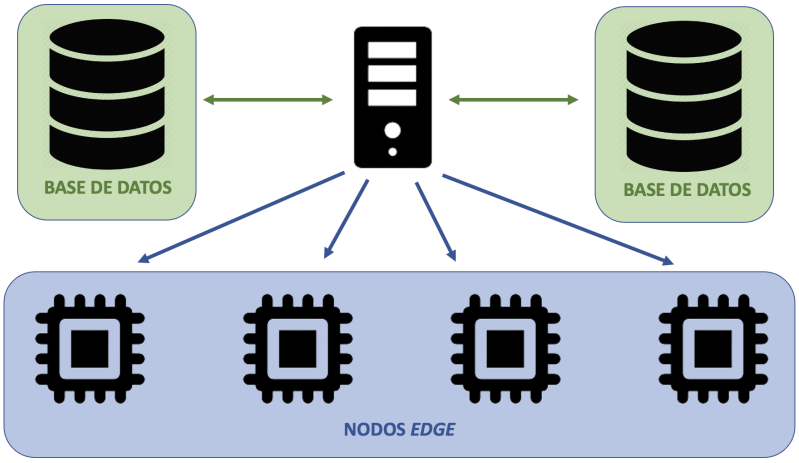

Figura 5.11: Arquitectura nodos *edge*

5.1.5. Validación

Se han utilizado diferentes métodos de búsqueda dentro de la red neuronal para la reducción de instancias mediante la compresión bayesiana y se han aplicado diferentes porcentajes de compresión, incluyendo Hill Climbing y Tree Augmented Network (TAN), con porcentajes de compresión del 25 %, 50 % y 75 %.

En cuanto a la reducción de instancias realizada mediante la compresión bayesiana, se han comprimido los datos utilizando varios métodos de búsqueda diferentes dentro de la red neuronal, y se han aplicado diferentes porcentajes de compresión:

- Algoritmos de búsqueda: Hill Climbing y Tree Augmented Network (TAN).
- Porcentajes de compresión: 25 %, 50 % y 75 %.

Hill Climbing

Hill Climbing es un algoritmo de optimización local que busca la dirección de ascenso/descenso más pronunciada posible desde su posición actual.

Hill Climbing es un algoritmo basado en optimización local. Toma la dirección de ascenso/descenso más pronunciada posible desde su posición actual, por lo cual, le supone un coste computacional mínimo. Como impide volver a una elección anterior, se suele denominar estrategia irreversible [196].

Tree Augmented Network (TAN)

Es una técnica de aprendizaje bayesiano semiautónomo. Al utilizar una estructura de árbol en la que cada atributo depende sólo de la clase y de otro atributo, elimina la suposición de independencia. La clasificación se lleva a cabo mediante un árbol de máxima amplitud ponderada que optimiza la probabilidad de los datos de entrenamiento [197].

Por otro lado, Tree Augmented Network (TAN) es una técnica de aprendizaje bayesiano que utiliza una estructura de árbol en la que cada atributo depende sólo de la clase y de otro atributo, eliminando la suposición de independencia. La clasificación se lleva a cabo mediante un árbol de máxima amplitud ponderada que optimiza la probabilidad de los datos de entrenamiento.

Tras generar los modelos con datos comprimidos, se ha comparado la precisión con los modelos generados con los datos originales.

Se han creado diferentes modelos que emplean diferentes técnicas de clasificación (Bayesian Network K2 *[198]*, Bayesian Network TAN *[154]*, J48 decision tree *[199]*, K-Nearest Neighbor *[200]*, Naive Bayes *[201]*, Random Forest *[202]* y Simple Logistic Regression *[203]*) con el fin de poder realizar un estudio del comportamiento de cada familia de algoritmos.

5.1.6. Resultados - Reducción de instancias

Los resultados de la experimentación de reducción de instancias a través de redes bayesianas se muestran en la Tabla 5.1, Tabla 5.2 y Tabla 5.3. La primera columna de resultados muestra la precisión de cada algoritmo entrenado con los datos originales, es decir, sin realizar ninguna compresión. A continuación, en las columnas posteriores, se muestran los resultados de precisión de los modelos generados con los datos comprimidos al 25 %, 50 % y 75 %. Cada una de las compresiones se ha realizado dos

veces utilizando diferentes algoritmos de búsqueda (*Hill Climbing* y *Tree Augmented Network*), por lo que por cada compresión existen dos resultados de precisión, los de la columna *HC* que corresponden a los modelos comprimidos utilizando el algoritmo de búsqueda *Hill Climbing* y los de la columna *TAN*, que corresponden a aquellos creados utilizando *Tree Augmented Network*.

En cuanto a los resultados de la experimentación de reducción de instancias a través de redes bayesianas, se comparado la precisión de modelos generados con datos originales y datos comprimidos al 25 %, 50 % y 75 % utilizando dos algoritmos de búsqueda diferentes.

A la hora de comparar los resultado de precisión de los modelos comprimidos con los modelos generados con los datos originales, como se puede observar en las tablas (5.1, 5.2, 5.3) la precisión de los modelos comprimidos es casi similar a la de los datos originales, pero con un 75 % menos de datos usados. La diferencia de precisión es muy pequeña, e incluso en algunos casos, la precisión de los modelos generados con datos comprimidos es mayor que la de los datos originales. Por ejemplo, en la Tabla 5.1, generando un modelo *BayesNet TAN* y empleando para entrenarlo los datos comprimidos un 25 % utilizando *TAN*, la precisión del modelo ha sido de un 93.98 % frente al 93.93 % de precisión que tiene el modelo entrenado con los datos en bruto. Además, para el mismo algoritmo, cuando se entrena el modelo con los datos comprimidos en un 75 %, la precisión es de 93.14 %. Esto indica que incluso después de reducir el conjunto de datos un 75 %, es decir, que el volumen final es un 25 % del volumen original, la precisión del modelo solo ha variado un 0.79 %.

Por otro lado, el algoritmo *Random Forest* obtiene muy buenos resultados con los datos comprimidos. Con un 99.95 % de precisión utilizando los datos en bruto, la precisión de los modelos comprimidos es 99.89 % con el 25 % comprimido, 99.83 % con el

50 % comprimido y 99.46 % con una compresión del 75 %. Lo que significa que la diferencia de precisión entre el modelo entrenado con los datos en bruto, y el modelo comprimido en un 75 % sería solo del 0.49 %. Los algoritmos *J48*, *KNN* y *Bayesnet TAN* también han dado muy buenos resultados con los datos en bruto. Con lo que respecta a la compresión en estos algoritmos, el algoritmo que mayor pérdida de precisión sufre es el *KNN* cuando se comprime un 25 %. Sin embargo, con ese mismo ratio de compresión, el algoritmo Bayesnet Tan mejora su precisión un 0.05 %. Estas pérdidas de precisión podrían ser consideradas como insignificantes, ya que para el algoritmo *Bayesnet TAN* la pérdida con la compresión del 50 % y 75 % es de 0.13 % y de 0.72 % respectivamente; y para el algoritmo *J48* es de 0.79 % y 1,61 %.

La mayoría de los algoritmos probados obtuvieron una precisión casi similar a la de los datos originales, incluso con una reducción del 75 % en el conjunto de datos. El algoritmo Random Forest fue el que obtuvo los mejores resultados, con una precisión cercana al 99 % incluso con una compresión del 75 %.

Aunque el resto de algoritmos (*Bayesnet K2*, *Naive Bayes*, *Simple Logistic*) no alcanzan una precisión tan alta con los datos en brutos (73,66 %, 52 %, 59,36 %), la pérdida de precisión con los datos comprimidos es igualmente insignificante, con una pérdida máxima del 1,37 % cuando se comprime al 75 % y se aplica para crear el modelo el algoritmo *Bayesnet K2*. Adicionalmente, se ha hecho un análisis también acerca de cuál es el algoritmo de búsqueda utilizado para la compresión de los datos (*Hill Climbing* o *Tree Augmented Network*) que mejores resultados genera posteriormente en la precisión de los modelos. Se ha llegado a la conclusión de que el éxito de un algoritmo u otro depende del % de compresión que se aplique al conjunto de datos. Cuando se trata de una compresión del 25 %, el *Tree Augmented Network* es el algoritmo de búsqueda que en el 85 % de las veces comprime los datos

También se ha analizado el impacto del algoritmo de búsqueda en la precisión de los modelos y se ha concluidi que su éxito depende del porcentaje de compresión aplicado.

de forma más representativa que el algoritmo *Hill Climbing*. Sin embargo, cuando la compresión es del 75 %, el algoritmo *Hill Climbing* es el que obtiene mejores resultados en un 85 % de los casos. En cambio, cuando los datos en bruto se comprimen al 50 %, no se ve de forma clara que algoritmo funciona mejor, ya que *Hill Climbing* funciona mejor en el 57 % de los casos, mientras que *Tree Augmented Network* en el 43 %.

Tabla 5.1: Resultados con 25 % de compresión

	Without compression	25 % compressed	
		HC	TAN
Bayesnet K2	73,66 %	73,46 %	73,57 %
Bayesnet TAN	93,93 %	93,08 %	93,98 %
J48	99,41 %	97,86 %	99,01 %
KNN(5)	96,85 %	91,55 %	94,83 %
Naive Bayes	52,00 %	52,02 %	52,10 %
Ranfom Forest	99,95 %	99,46 %	99,89 %
Simple Logistic	59,36 %	59,55 %	59,45 %

Tabla 5.2: Resultados con 50 % de compresión

	Without compression	50 % compressed	
		HC	TAN
Bayesnet K2	73,66 %	73,13 %	73,21 %
Bayesnet TAN	93,93 %	93,80 %	93,73 %
J48	99,41 %	98,59 %	98,69 %
KNN(5)	96,85 %	93,95 %	93,75 %
Naive Bayes	52,00 %	51,95 %	52,05 %
Ranfom Forest	99,95 %	99,83 %	99,78 %
Simple Logistic	59,36 %	59,43 %	59,37 %

	Without compression	75 % compressed	
	Accuracy	*HC*	*TAN*
Bayesnet K2	73,66 %	72,29 %	72,28 %
BayesNet TAN	93,93 %	93,13 %	93,14 %
J48	99,41 %	97,80 %	97,78 %
KNN (5)	96,85 %	91,48 %	91,45 %
Naive Bayes	52,00 %	51,96 %	51,94 %
Random Forest	99,95 %	99,46 %	99,45 %
Simple Logistic	59,36 %	59,50 %	59,45 %

Tabla 5.3: Resultados con 75 % de compresión

5.1.7. Resultados - Reducción de dimensionalidad

En las tablas 5.4 y 5.5 se muestran los resultados de la experimentación de la reducción de dimensionalidad. Al igual que en la experimentación de la reducción de instancias del apartado anterior, en la columna de la izquierda sale reflejada la precisión de cada uno de los algoritmos entrenados con los datos en bruto, y en las columnas posteriores, la precisión con las compresiones del 50 %, 75 % y 90 %. Cabe destacar que los resultados obtenidos en esta experimentación no son tan prometedores como en el caso de la reducción de instancias. En el caso del algoritmo *Naive Bayes*, no solo no pierde precisión cuando se comprime hasta en un 75 %, sino que incluso mejora la precisión del modelo en casi un 1 %. Cuando la compresión es de un 50 %, la precisión del modelo mejora un 1,39 %.

Las tablas 5.4 y 5.5 muestran los resultados de la experimentación de la reducción de dimensionalidad, donde se comparan las precisiones de varios algoritmos entrenados con los datos en bruto y con diferentes niveles de compresión (50 %, 75 % y 90 %). Los resultados indican que la reducción de dimensionalidad no es tan efectiva como la reducción de instancias.

Después del algoritmo *Naive Bayes*, el algoritmo que menos precisión pierde con una compresión del 75 % es el *Simple Logistic* (1,79 %), seguido del *KNN* (13,75 %).

Los algoritmos que mejores resultados ofrecen cuando se hace una compresión del 50 % son *KNN*,

El algoritmo *Naive Bayes* mejora su precisión en un 1 % cuando se comprime en un 75 % y en un 1,39 % cuando se comprime en un 50 %, mientras que el *Simple Logistic* y *KNN* son los algoritmos que menos precisión pierden con una compresión del 75 %.

Naive Bayes y *Simple Logistic*. El algoritmo *KNN* y el *Simple Logistic* con una compresión del 50 % tienen una pérdida de precisión de sólo el 1.75 % y 1.61 %, respectivamente, mientras que el algoritmo *Naive Bayes* mejora la precisión en un 1.39 % para este ratio de compresión.

En cuando a los resultados con la compresión del 90 %, la pérdida no es despreciable y el modelo dejaría realizar su función.

Tabla 5.4: Resultados con 50 % y 75 % de compresión

	Without compression	50 % compressed	75 % compressed
Bayesnet K2	73,66 %	57,62 %	54,45 %
Bayesnet TAN	93,93 %	64,33 %	62,91 %
J48	99,41 %	86,66 %	78,46 %
KNN(5)	96,85 %	95,10 %	83,10 %
Naive Bayes	52 %	53,39 %	52,98 %
Random Forest	99,95 %	94,72 %	85,00 %
Simple Logistic	59,36 %	57,75 %	57,57 %

Tabla 5.5: Resultados con 90 % de compresión

	Without compression	90 % compressed
Bayesnet K2	73,66 %	39,53 %
Bayesnet TAN	93,93 %	41,61 %
J48	99,41 %	40,91 %
KNN(5)	96,85 %	36,12 %
Naive Bayes	52 %	38,46 %
Ranfom Forest	99,95 %	NA
Simple Logistic	59,36 %	38,91 %

En cuanto a la compresión del 50 %, los algoritmos *KNN*, *Naive Bayes* y *Simple Logistic* ofrecen los mejores resultados, con una pérdida de precisión de sólo el 1,75 %, 1,39 % y 1,61 %, respectivamente. Sin embargo, la precisión disminuye significativamente cuando se reduce la dimensionalidad en un 90 %, lo que hace que el modelo no pueda cumplir su función.

5.2. Conclusiones

Esta investigación se ha centrado en demostrar que a través de diferentes técnicas de compresión de datos se puede llegar a obtener conjuntos de datos reducidos que representan las características del conjunto de datos original.

Se han aplicado compresiones bayesianas y autoencoders a los datos originales con el objetivo de poder generar pequeños modelos 'Edge Computing'.

Esta investigación demuestra que es posible obtener conjuntos de datos reducidos que representen las características del conjunto de datos original mediante técnicas de compresión de datos. Se han aplicado compresiones bayesianas y *autoencoders* a los datos originales con el objetivo de generar modelos *Edge Computing*.

Como resultado, tal y como se ha demostrado, es posible implementar estos modelos en nodos con una capacidad computacional limitada gracias a la compresión de los datos, ya que siguen generando resultados similares a los modelos creados con los datos en bruto. Además, se ha demostrado que en algunos casos, cuando se aplican ciertos algoritmos, el conjunto de datos comprimido genera mejores resultados que el conjunto de datos original.

Cabe destacar que para el algoritmo Bayesnet TAN en el caso de reducción de instancias (compresión bayesiana) y para el algoritmo Naive Bayes para la reducción de dimensionalidad (autoencoder), el modelo mejora su precisión un 0.05 % y 1.39 % respectivamente con los datos comprimidos un 25 %.

En el caso de la compresión bayesiana, al utilizarse el algoritmo Bayesnet, la pérdida de precisión de los modelos que aplican este algoritmo puede considerarse insignificante incluso cuando la compresión de los datos es de un 75 %. Por otro lado, en cuanto al uso de *autoencoders* para la reducción de los conjuntos de datos, se ha concluido que funciona bien únicamente para casos muy concretos. Por ejemplo, cuando se utiliza la capa de muestreo ascendente (*upsampling)* para la reconstrucción del decodificador con el objetivo de reducir la dimensionalidad en un 50 % funciona bien, ya que no hay que añadir capas de relleno con ceros para la reconstrucción. Sin embargo, cuando hay que añadirlas a la salida, distorsiona el resultado, ya que

Se concluyó que la compresión bayesiana funciona bien con una pérdida de precisión insignificante incluso cuando la compresión de los datos es del 75 %, y que los *autoencoders* funcionan bien en casos muy concretos.

no deja que la granularidad [5] sea tan buena como en la compresión bayesiana y genera resultados no deseados. Gracias a la metodología propuesta en esta experimentación en cuanto a la compresión de los datos a través de redes bayesianas, permite comprimir conjuntos de datos con el porcentaje de reducción deseado, consiguiendo de esta manera que sea una metodología *ah-hoc*.

5: Granularidad: La granularidad en los datos define el nivel de detalle que tiene un conjunto de datos [204].

La metodología propuesta en esta experimentación permite comprimir conjuntos de datos con el porcentaje de reducción deseado.

6 Conclusiones

En este capítulo, se presentarán las conclusiones finales obtenidas a lo largo de las investigaciónes llevadas a cabo en el marco de esta tesis doctoral. Se proporcionará un resumen del análisis detallado del problema principal abordado en la introducción, así como una descripción de los objetivos planteados para solucionarlo. Se discutirán los experimentos realizados, los resultados obtenidos y las validaciones llevadas a cabo. Finalmente, se añadirán posibles líneas de trabajos futuros.

En la introducción de esta tesis doctoral se ha identificado y analizado exhaustivamente el problema principal que ha motivado esta investigación:

Con el auge del Internet de las Cosas (*IoT*) y la digitalización, se ha producido un aumento en los datos manejados por las empresas. La computación en la nube, aunque útil para descentralizar la computación, presenta retos como la latencia en las comunicaciones y la sobrecarga de red debido a la distancia de los centros de datos y al volumen de datos transmitidos. La creciente demanda de procesamiento de datos y la expansión de dispositivos generando datos en el borde de la red requieren soluciones más eficientes que la computación en la nube, lo que ha llevado al surgimiento del concepto de *Edge Computing*.

Esta tesis surge de la necesidad de generar conocimiento a partir de los datos de proceso de un *OEM*[1], ya que, la investigación se llevó a cabo en colaboración con una empresa de automoción cuya

1: OEM: Original Equipment Manufacturer

planta de producción tiene como objetivo fabricar 610 vehículos al día. El proceso de digitalización de esta planta ha permitido recoger una gran cantidad de variables por segundo durante el proceso de fabricación, lo que ha aumentado la necesidad de procesar estos datos de manera eficiente para poder tomar decisiones durante el proceso de producción. Dado que la computación en la nube puede presentar retrasos y las plantas de producción requieren respuestas cada vez más rápidas, esta investigación se enfoca en el potencial del *Edge Computing* para manejar los datos en tiempo real. La meta es desarrollar gemelos digitales que mejoren la eficiencia y la calidad, particularmente en el proceso de pintura de vehículos, una etapa que a menudo es un cuello de botella debido a su complejidad y a los estrictos estándares de calidad requeridos.

Se han explorado las limitaciones y las deficiencias en el estado del arte, destacando las áreas que requerían una mayor atención y los nuevos retos que debían afrontarse. A lo largo de los capítulos precedentes, se ha trabajado en estrecha colaboración con la literatura existente para comprender a fondo el problema y definir objetivos claros y relevantes.

Con el fin de abordar el problema principal, tras identificar los retos presentes en la industria actual y haber realizado un análisis crítico del estado del arte se han establecido una serie de objetivos específicos. Estos objetivos fueron diseñados para proporcionar una estructura clara que permitiera responder a la pregunta de investigación planteada:

¿Puede la combinación de Edge Computing con técnicas

de inteligencia artificial modelizar con suficiente precisión y mejorar el desarrollo de gemelos digitales de partes del proceso de pintura de una planta de producción del sector de la automoción?

Para responder a esta hipótesis, se han establecido los siguientes objetivos que se derivan de la necesidad de acortar los tiempos de cálculo y predicción del resultado del proceso para enfrentar los desafíos tecnológicos que requieren respuestas rápidas, como es el caso del sector de la automoción.

El primer objetivo es comprender a fondo esta nueva tecnología para determinar las restricciones y el comportamiento de cada algoritmo en estos dispositivos durante el entrenamiento y la validación de modelos. Este estudio tiene el propósito de facilitar la selección del hardware a utilizar en función del escenario dado.

En segundo lugar, el objetivo es demostrar que, a pesar de las limitaciones de los dispositivos, pueden facilitar la tarea de descomponer un proceso modelado en la nube en pequeños subprocesos que se modelan en dispositivos *Edge*, no solo manteniendo la eficacia del modelo tradicional, sino también brindando las ventajas correspondientes al uso de esta tecnología.

Finalmente, una vez cumplidos los dos primeros objetivos, y dado que una de las restricciones del *Edge Computing* depende directamente del volumen de datos a procesar, el último objetivo busca comprimir un conjunto de datos de manera que el conjunto resultante pueda generar un modelo que mantenga la precisión y eficacia del modelo creado con los datos originales. Así, en caso de que el volumen de datos para modelar un proceso

industrial sea mayor al que un dispositivo *Edge* puede procesar, en lugar de recurrir a una arquitectura en la nube, pueda utilizar la arquitectura *Edge*.

Para lograr los objetivos planteados, se han llevado a cabo tres investigaciones diferentes, cada una de las cuales se ha desarrollado en un capítulo separado. En cada uno de estos capítulos, se describen los experimentos realizados y las metodologías aplicadas.

En cuanto al estudio de evaluar nuevos hardwares IoT para implementar *Edge Computing* en cadenas de producción para analizar su eficacia a la hora de entrenar y validar modelos, la experimentación ha demostrado que los algoritmos de inteligencia artificial tienen un mejor rendimiento en el hardware *Nvidia Jetson Nano*, con la excepción de las redes neuronales. Para las redes neuronales, el hardware más efectivo es *Google Coral*, pero cabe destacar que el hardware *Google Coral* no puede procesar archivos de más de 500 MB, y en algunos casos, el límite es aún menor. Tal como se ha mencionado, generalmente el hardware *Nvidia Jetson Nano* es más rápido, pero, sin embargo, existen casos específicos en los que *Google Coral* presenta un mejor rendimiento. Un ejemplo de esto es el algoritmo de regresión logística. En la mayoría de los casos, el hardware *Nvidia Jetson Nano* es más rápido, excepto cuando se trabaja con archivos de aproximadamente 350 MB. En estas situaciones, tanto para el entrenamiento como para validación, *Google Coral* ofrece un mejor rendimiento.

Otra conclusión que podemos extraer de esta experimentación indica que el tamaño del archivo a procesar también tiene un impacto en cuál es el

dispositivo *Edge* que más rápido lo procesa. En el caso del algoritmo *K Nearest Neighbor*, cuando se trabaja con archivos pequeños de menos de 100 MB, el hardware *Google Coral* muestra mayor velocidad. Sin embargo, para archivos más grandes, el hardware *Nvidia Jetson Nano* es el que procesa los datos más rápido.

La experimentación realizada con el algoritmo *Gaussian Naive Bayes* revela que otro factor que influye en el procesamiento de archivos en estos dispositivos es el tipo de tarea a realizar: entrenamiento o validación. Para este algoritmo en particular, si el objetivo es realizar el entrenamiento, el hardware *Google Coral* es más rápido. Sin embargo, para procesar una validación, ambos dispositivos obtienen resultados similares en términos de velocidad.

Ciertos factores como el tamaño del archivo y el tipo de procesamiento influyen en qué hardware es más eficiente. Así que, como conclusión final de esta experimentación se obtiene que la elección del hardware más apropiado depende de una combinación de factores: el tipo de procesamiento (entrenamiento o validación), el tamaño del archivo y el algoritmo a utilizar. Para ayudar en esta decisión, se proporciona una tabla resumen con los modelos de comportamiento para cada combinación hardware-tamaño del archivo-algoritmo (Tabla 3.6, Tabla 3.7, Tabla 3.8 y Tabla 3.9).

El análisis ha permitido definir estrategias específicas de Edge Computing y estimar el número de dispositivos requeridos en cada línea de producción. También se crea un modelo de comportamiento para cada combinación de algoritmo y hardware.

Los dispositivos estudiados son *Nvidia Jetson Nano* y *Google Coral*.

Por otro lado, la investigación centrada en validar la eficacia y precisión de los modelos *Edge Computing* que representan subprocesos en el entorno industrial comparándolos con el enfoque tradicional de *Cloud Computing* que contiene todos los datos del proceso, se ha concluido que los modelos *Edge* han alcanzado una precisión hasta dos veces superior. Estos resultados nos llevan a concluir que en algunos casos, tener más datos, es decir, más información, no necesariamente se traduce en mejores resultados para un modelo. De hecho, las experimentaciones demuestran lo contrario. Este fenómeno se explica porque los datos de un proceso específico pueden actuar como "ruido" al predecir otros procesos, como ha sucedido con las dos experimentaciones realizadas en esta línea de investigación.

Una vez concluidas las dos investigaciones anteriores, debido a que una de las conclusiones sacadas ha sido que las restricciones de los dispositivos *Edge* tienen relación directa con el volumen de datos que han de proecesar, la tercera investigación se ha basado en buscar soluciones para superar las limitaciones presentes en los dispositivos *Edge*. La necesidad de optimizar el flujo de datos entre dispositivos IoT y dispositivos *Edge* ha impulsado la búsqueda de una solución adecuada. El objetivo de esta tercera investigación ha sido encontrar una técnica de compresión de datos que permita generar modelos de inteligencia artificial (utilizando los datos comprimidos) con una precisión igual o similar a modelos creados con los datos sin comprimir. De esta manera, se ha querido mitigar el

problema de procesar grandes conjuntos de datos recolectados en bruto por las máquinas.

Los resultados de las experimentaciones realizadas en esta línea de investigación demuestran que es factible implementar modelos de inteligencia artificial en nodos con capacidad computacional limitada gracias a la compresión de datos, ya que siguen generando resultados similares a los modelos basados en los datos sin comprimir. Además, en algunos casos, la precisión de los modelos generados con los datos comprimidos es incluso mejor que la de los modelos basados en los datos originales.

Con el fin de conseguir la reducción de datos, se han empleado dos enfoques diferentes: por un lado la reducción del número de instancias (filas) y, por otro, la reducción de la dimensionalidad de los datos (columnas). Para la reducción de instancias, se ha aplicado el Teorema de Bayes con la compresión Bayesiana, y para la reducción de dimensionalidad, se ha hecho uso de *autoencoders*.

Una vez realizadas las experimentaciones es importante destacar que en algunos casos, a la hora de generar modelos de inteligencia artificial con una compresión del 25 % de los datos, los modelos mejoran la precisión en un 0,05 % cuando se hace una reducción de instancias y un 1,39 % cuando se reduce la dimensionalidad. Por otro lado, cuando se realiza una compresión de un 75 % con la compresión bayesiana, la pérdida de precisión en los modelos puede considerarse indignificante.

Además, se ha concluido que el uso de autoencoders para la reducción de conjuntos de datos funciona bien solo en casos muy específicos. Por ejemplo, cuando se utiliza la técnica de "upsam-

pling" en la capa de reconstrucción del decodificador para reducir la dimensionalidad en un 50 %, el resultado es satisfactorio, ya que no es necesario agregar capas de relleno con ceros. Sin embargo, cuando es necesario agregar dichas capas para la reconstrucción, se observan distorsiones y la granularidad de los datos no es tan precisa como en la compresión bayesiana, lo que genera resultados no deseados.

Se han llevado a cabo diferentes técnicas de validación para respaldar los resultados obtenidos. Cada experimentación ha tenido su método de validación. Para el análisis hardware, utilizando un conjunto de datos amplio como punto de partida, se han generado de manera aleatoria diferentes subconjuntos de datos de diversos tamaños. Posteriormente, cada uno de estos subconjuntos ha sido ejecutado en los dispositivos hardware empleados para el análisis. Este proceso se ha repetido en tres ocasiones para cada combinación de tamaño del subconjunto de datos y hardware utilizado.

Para la validación de la eficacia de modelos *Edge* (modelo de un subproceso) frente a modelos completos que contemplan datos de varios subprocesos, la validación se ha hecho comparando la precisión de unos con otros. Esta técnica ha permitido validar la eficacia de los modelos *Edge*.

Por último, la validación de la eficacia de los conjuntos comprimidos para generar modelos de inteligencia artificial se ha realizado de una manera muy similar a la anterior experimentación de los modelos *Edge*. En este caso, se ha comparado la precisión de los modelos generados con los datos originales con los modelos generados con los datos comprimidos.

A pesar de los avances significativos logrados en esta investigación, existen áreas que aún requieren un mayor desarrollo y estudio. Como trabajo futuro, se recomienda investigar las siguientes direcciones:

- En cuanto a la optimización del flujo de datos mediante la compresión de los mismos, sería interesante combinar ambas técnicas en el mismo conjunto de datos para reducir tanto el número de instancias como el número de dimensiones, o al menos encontrar una técnica diferente que combine ambas reducciones.
- Al analizar un proceso de compresión, es esencial considerar tanto la eficiencia energética como la eficiencia temporal para identificar áreas de mejora. La eficiencia energética es un factor clave para reducir costos y lograr la sostenibilidad ambiental, mientras que la eficiencia temporal es importante para maximizar la productividad y minimizar el tiempo de inactividad. Al analizar ambos aspectos, es posible detectar posibles cuellos de botella en el proceso de compresión y encontrar oportunidades para optimizar el rendimiento. Un análisis exhaustivo de la eficiencia energética y temporal en el proceso de compresión puede ser crucial para lograr un proceso más eficiente y rentable en el futuro.

Bibliografía

[1] Vivekanand Gopalkrishnan et al. «Big data, big business: bridging the gap». En: *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*. 2012, págs. 7-11 (vid. pág. 1).

[2] Lane Thames y Dirk Schaefer. «Software-defined cloud manufacturing for industry 4.0». En: *Procedia cirp* 52 (2016), págs. 12-17 (vid. pág. 1).

[3] Heiner Lasi et al. «Industry 4.0». En: *Business & information systems engineering* 6.4 (2014), págs. 239-242 (vid. págs. 1, 6).

[4] Daniele Miorandi et al. «Internet of things: Vision, applications and research challenges». En: *7* 10.7 (2012), págs. 1497-1516 (vid. pág. 1).

[5] Oleksandr S Vyshnevskyi. «Impact of digitalization on industry: problems of definition in EU countries». En: *Economy of industry* 1 (89) (2020), págs. 31-44 (vid. pág. 1).

[6] Joakim Björkdahl. «Strategies for digitalization in manufacturing firms». En: *California Management Review* 62.4 (2020), págs. 17-36 (vid. págs. 1-4, 13-15, 95).

[7] Volvo. *New digital core for our manufacturing plants*. https://www.volvocars.com/intl/news/industrial/New-digital-core-for-our-manufacturing-plants/?navFromLatest=false. [Online; accessed 09-November-2022] (vid. pág. 2).

[8] Alberto Moreno. «La apuesta por la digitalización del sector del automóvil». En: *California Management Review* 62.4 (2020), págs. 17-36 (vid. pág. 3).

[9] Computing redaction. *El volumen de datos en las empresas crece un 569 % en dos años*. https://www.computing.es/analytics/noticias/1113253046201/volumen-de-datos-empresas-crece-569-dos-anos.1.html. [Online; accessed 03-April-2021] (vid. págs. 4, 95, 115).

[10] Gopika Premsankar, Mario Di Francesco y Tarik Taleb. «Edge computing for the Internet of Things: A case study». En: *IEEE Internet of Things* 5.2 (2018), págs. 1275-1284 (vid. págs. 5, 6, 8).

[11] Keyan Cao et al. «An overview on edge computing research». En: *IEEE access* 8 (2020), págs. 85714-85728 (vid. págs. 5, 7-11, 117).

[12] Weisong Shi y Schahram Dustdar. «The promise of edge computing». En: *Computer5* 49.5 (2016), págs. 78-81 (vid. pág. 5).

[13] Mahadev Satyanarayanan et al. «The case for vm-based cloudlets in mobile computing». En: *IEEE pervasive Computing* 8.4 (2009), págs. 14-23 (vid. pág. 5).

[14] X. Z. Zhang W. S. Shi e Y.F. Wang. «Edge Computing: State-of-the-art and future directions». En: *J. Comput. Res. Develop.* 56 (2019), págs. 1-21 (vid. pág. 7).

[15] Weisong Shi et al. «Edge computing-an emerging computing model for the internet of everything era». En: *Journal of computer research and development* 54.5 (2017), págs. 907-924 (vid. pág. 7).

[16] Weisong Shi et al. «Edge computing: Vision and challenges». En: *IEEE internet of things journal* 3.5 (2016), págs. 637-646 (vid. pág. 7).

[17] Mahadev Satyanarayana. *Carnegie Group University Professor of Computer Science*. https://csd.cmu.edu/people/faculty/mahadev-satyanarayanan. [Online; accessed 09-November-2022] (vid. pág. 8).

[18] Mahadev Satyanarayanan. «The emergence of edge computing». En: *Computer* 50.1 (2017), págs. 30-39 (vid. pág. 8).

[19] F. Liu Z. M. Zha y Z. P. Cai. «Edge computing: Platforms, Applications and challenges». En: *Journal of computer research and development* 55.2 (2018), págs. 327-337 (vid. pág. 8).

[20] Xuehai Hong y Yang Wang. «Edge computing technology: development and countermeasures». En: *Strategic Study of Chinese Academy of Engineering* 20.2 (2018), págs. 20-26 (vid. pág. 8).

[21] Xiang Sun y Nirwan Ansari. «EdgeIoT: Mobile edge computing for the Internet of Things». En: *IEEE Communications Magazine* 54.12 (2016), págs. 22-29 (vid. pág. 8).

[22] Arwa Alrawais et al. «Fog computing for the internet of things: Security and privacy issues». En: *IEEE Internet Computing* 21.2 (2017), págs. 34-42 (vid. pág. 8).

[23] Jiawen Kang et al. «Privacy-preserved pseudonym scheme for fog computing supported internet of vehicles». En: *IEEE Transactions on Intelligent Transportation Systems* 19.8 (2017), págs. 2627-2637 (vid. pág. 8).

[24] Carla Mouradian et al. «A comprehensive survey on fog computing: State-of-the-art and research challenges». En: *IEEE communications surveys & tutorials* 20.1 (2017), págs. 416-464 (vid. pág. 8).

[25] Z. M. Xu e Y. F. Tian. «The History and application of cloud computing». En: *Inf. Recording Mater* 19.8 (2018), págs. 66-67 (vid. pág. 9).

[26] Lei Jiao et al. «Cloud-based computation offloading for mobile devices: State of the art, challenges and opportunities». En: *2013 Future Network & Mobile Summit* (2013), págs. 1-11 (vid. pág. 9).

[27] Ivan Stojmenovic. «Fog computing: A cloud to the ground support for smart things and machine-to-machine networks». En: *2014 Australasian telecommunication networks and applications conference (ATNAC)*. IEEE. 2014, págs. 117-122 (vid. págs. 9, 96).

[28] Sami Yangui et al. «A platform as-a-service for hybrid cloud/fog environments». En: *2016 IEEE international symposium on local and metropolitan area networks (LANMAN)*. IEEE. 2016, págs. 1-7 (vid. pág. 9).

[29] Xiaoqing Zhu et al. «IMPROVING VIDEO PERFORMANCE WITH EDGE SERVERS IN THE FOG COMPUTING ARCHITECTURE.» En: *Intel Technology Journal* 19.1 (2015) (vid. pág. 9).

[30] Qinglin Qi y Fei Tao. «A smart manufacturing service system based on edge computing, fog computing, and cloud computing». En: *IEEE Access* 7 (2019), págs. 86769-86777 (vid. págs. 9, 10).

[31] Linna Ruan et al. «Priority-based residential energy management with collaborative edge and cloud computing». En: *IEEE Transactions on Industrial Informatics* 16.3 (2019), págs. 1848-1857 (vid. pág. 10).

[32] Tian Wang et al. «A secure IoT service architecture with an efficient balance dynamics based on cloud and edge computing». En: *IEEE Internet of Things Journal* 6.3 (2018), págs. 4831-4843 (vid. pág. 10).

[33] M Shamim Hossain y Ghulam Muhammad. «Emotion recognition using secure edge and cloud computing». En: *Information Sciences* 504 (2019), págs. 589-601 (vid. pág. 10).

[34] Michael Armbrust et al. *Above the clouds: A berkeley view of cloud computing*. Inf. téc. Technical Report UCB/EECS-2009-28, EECS Department, University of California . . ., 2009 (vid. pág. 10).

[35] Kevin Ashton et al. «That 'internet of things' thing». En: *22* 22.7 (2009), págs. 97-114 (vid. pág. 13).

[36] Friedemann Mattern y Christian Floerkemeier. «From the Internet of Computers to the Internet of Things». En: *From active data management to event-based systems and more*. Springer, 2010, págs. 242-259 (vid. pág. 13).

[37] Siemens. *Industria 4.0: cuando los datos son la nueva materia prima*. https://new.siemens.com/mx/es/compania/eventos/hannover-messe-2018/industry-4-0--when-data-is-the-new-raw-material-2018.html. [Online; accessed 02-October-2019]. 2018 (vid. pág. 15).

[38] Nancy Velásquez, Elsa Estevez y Patricia Pesado. «Cloud Computing, Big Data and the Industry 4.0 Reference Architectures». En: *Journal of Computer Science and Technology* 18.03 (2018), e29-e29 (vid. pág. 16).

[39] Dimitris Mourtzis et al. «The role of simulation in digital manufacturing: applications and outlook». En: *International journal of computer integrated manufacturing* 28.1 (2015), págs. 3-24 (vid. pág. 16).

[40] Noriyuki Takahashi, Hiroyuki Tanaka y Ryutaro Kawamura. «Analysis of process assignment in multi-tier mobile cloud computing and application to edge accelerated web browsing». En: *2*. IEEE. 2015, págs. 233-234 (vid. pág. 16).

[41] Stefania Sardellitti, Gesualdo Scutari y Sergio Barbarossa. «Joint optimization of radio and computational resources for multicell mobile-edge computing». En: *IEEE Transactions on Signal and Information Processing over Networks* 1.2 (2015), págs. 89-103 (vid. pág. 17).

[42] Cunhua Pan et al. «Multicell MIMO communications relying on intelligent reflecting surfaces». En: *IEEE Transactions on Wireless Communications* 19.8 (2020), págs. 5218-5233 (vid. pág. 17).

[43] Ian Goodfellow et al. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016 (vid. pág. 17).

[44] Shan Ullah y Deok-Hwan Kim. «Benchmarking Jetson Platform for 3D Point-Cloud and Hyper-Spectral Image Classification». En: *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE. 2020, págs. 477-482 (vid. pág. 17).

[45] Nvidia. *Jetson Nano Developer Kit*. https://developer.nvidia.com/embedded/jetson-nano-developer-kit. [Online; accessed 20-May-2020]. 2020 (vid. págs. 17, 60).

[46] Tao Peng et al. «Evaluating the power efficiency of visual slam on embedded gpu systems». En: *2019 IEEE National Aerospace and Electronics Conference (NAECON)*. IEEE. 2019, págs. 117-121 (vid. pág. 17).

[47] Nvidia. *Jetson AGX Xavier Developer Kit*. https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit. [Online; accessed 20-May-2020]. 2020 (vid. pág. 17).

[48] Nvidia. *Jetson TX2 Module*. https://developer.nvidia.com/embedded/jetson-tx2. [Online; accessed 20-May-2020]. 2020 (vid. pág. 17).

[49] Jongmin Jo, Sucheol Jeong y Pilsung Kang. «Benchmarking GPU-Accelerated Edge Devices». En: *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE. 2020, págs. 117-120 (vid. pág. 17).

[50] Artiom Basulto-Lantsova et al. «Performance comparative of OpenCV Template Matching method on Jetson TX2 and Jetson Nano developer kits». En: *202*. IEEE. 2020, págs. 0812-0816 (vid. pág. 17).

[51] D Randall Wilson y Tony R Martinez. «Instance pruning techniques». En: *ICML*. Vol. 97. 1997. 1997, págs. 400-411 (vid. pág. 18).

[52] Yann LeCun, John S Denker y Sara A Solla. «Optimal brain damage». En: *Advances in neural information processing systems*. 1990, págs. 598-605 (vid. pág. 18).

[53] Eric Nalisnick, Anima Anandkumar y Padhraic Smyth. «A scale mixture perspective of multiplicative noise in neural networks». En: *arXiv preprint arXiv:1506.03208* (2015) (vid. pág. 18).

[54] Song Han et al. «EIE: efficient inference engine on compressed deep neural network». En: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2016, págs. 243-254 (vid. pág. 18).

[55] Christos Louizos, Karen Ullrich y Max Welling. «Bayesian compression for deep learning». En: *Advances in Neural Information Processing Systems*. 2017, págs. 3288-3298 (vid. pág. 19).

[56] Darryl Lin, Sachin Talathi y Sreekanth Annapureddy. «Fixed point quantization of deep convolutional networks». En: *International Conference on Machine Learning*. 2016, págs. 2849-2858 (vid. pág. 19).

[57] Wenlin Chen et al. «Compressing convolutional neural networks». En: *arXiv preprint arXiv:1506.04449* (2015) (vid. pág. 19).

[58] benchmarks.ai. *CIFAR-10*. https://benchmarks.ai/cifar-10. [Online; accessed 01-July-2021] (vid. pág. 19).

[59] Nat Dilokthanakul et al. «Deep unsupervised clustering with gaussian mixture variational autoencoders». En: *arXiv preprint arXiv:1611.02648* (2016) (vid. pág. 19).

[60] John M McNamara, Richard F Green y Ola Olsson. «Bayes' theorem and its applications in animal behaviour». En: *Oikos* 112.2 (2006), págs. 243-251 (vid. pág. 20).

[61] K Ming Leung. «Naive bayesian classifier». En: *Polytechnic University Department of Computer Science/Finance and Risk Engineering* 2007 (2007), págs. 123-156 (vid. págs. 20, 21).

[62] James Joyce. «Bayes' theorem». En: (2003) (vid. pág. 20).

[63] Benjamin Graham. «Fractional max-pooling». En: *arXiv preprint arXiv:1412.6071* (2014) (vid. pág. 23).

[64] Xuan Li et al. «Guided autoencoder for dimensionality reduction of pedestrian features». En: *Applied Intelligence* 50.12 (2020), págs. 4557-4567 (vid. pág. 23).

[65] Claude Sammut y Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011 (vid. pág. 23).

[66] N Adeen, M Abdulazeez y D Zeebaree. «Systematic review of unsupervised genomic clustering algorithms techniques for high dimensional datasets». En: *Technol. Reports Kansai Univ* 62.3 (2020), págs. 355-374 (vid. pág. 24).

[67] Dildar Masood Abdulqader, A Mohsin Abdulazeez y Diyar Qader Zeebaree. «Machine learning supervised algorithms of gene selection: A review». En: *Machine Learning* 62.03 (2020), págs. 233-244 (vid. págs. 24, 29).

[68] Shereen S Sadiq, Adnan Mohsin Abdulazeez y Habibollah Haron. «Solving multi-objective master production schedule problem using memetic algorithm». En: *Indonesian Journal of Electrical Engineering and Computer Science* 18.2 (2020), págs. 938-945 (vid. pág. 24).

[69] G Kishor Kumar, P Viswanath y A Ananda Rao. «Ensemble of randomized soft decision trees for robust classification». En: *Sādhanā* 41.3 (2016), págs. 273-282 (vid. pág. 25).

[70] Ye Li et al. «Outsourced privacy-preserving C4. 5 decision tree algorithm over horizontally and vertically partitioned dataset among multiple parties». En: *Cluster Computing* 22.1 (2019), págs. 1581-1593 (vid. pág. 25).

[71] Leo Breiman. «Random forests». En: *Machine learning* 45.1 (2001), págs. 5-32 (vid. pág. 25).

[72] Nasiba Mahdi Abdulkareem, Adnan Mohsin Abdulazeez et al. «Machine learning classification based on Radom Forest Algorithm: A review». En: *International Journal of Science and Business* 5.2 (2021), págs. 128-142 (vid. pág. 25).

[73] Itamar Reis, Dalya Baron y Sahar Shahaf. «Probabilistic random forest: A machine learning algorithm for noisy data sets». En: *The Astronomical Journal* 157.1 (2018), pág. 16 (vid. pág. 25).

[74] Busra Ozgode Yigin, Oktay Algin y Gorkem Saygili. «Comparison of morphometric parameters in prediction of hydrocephalus using random forests». En: *Computers in Biology and Medicine* 116 (2020), pág. 103547 (vid. pág. 26).

[75] Mónica Lizares Castillo. «Comparación de modelos de clasificación: regresión logıstica y árboles de clasificación para evaluar el rendimiento académico». En: (2017) (vid. pág. 26).

[76] Javier Jesús Espinosa-Zúñiga. «Aplicación de algoritmos Random Forest y XGBoost en una base de solicitudes de tarjetas de crédito». En: *Ingenierıa, investigación y tecnologıa* 21.3 (2020) (vid. págs. 26, 28, 29).

[77] Danielle Denisko y Michael M Hoffman. «Classification and interaction in random forests». En: *Proceedings of the National Academy of Sciences* 115.8 (2018), págs. 1690-1692 (vid. pág. 27).

[78] Lev V Utkin, Maxim S Kovalev y Frank PA Coolen. «Imprecise weighted extensions of random forests for classification and regression». En: *Applied Soft Computing* 92 (2020), pág. 106324 (vid. pág. 27).

[79] Liliya Demidova y Mariya Ivkina. «Defining the Ranges Boundaries of the Optimal Parameters Values for the Random Forest Classifier». En: *2019 1st International Conference on Control Systems, Mathematical Modelling, Automation and Energy Efficiency (SUMMA)*. IEEE. 2019, págs. 518-522 (vid. pág. 27).

[80] Mohan L Kolhe et al. «Advances in data and information sciences». En: *Proceedings of ICDIS* 1 (2017) (vid. págs. 27, 29).

[81] Ossama Abdel-Hamid et al. «Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition». En: *2012 IEEE international conference on Acoustics, speech and signal processing (ICASSP)*. IEEE. 2012, págs. 4277-4280 (vid. pág. 27).

[82] Er. Anhilasha E. Goel. «Random Forest: A review». En: *Computer Science Engineering GZSCCET*. IJARCSSE. 2017, págs. 251-257 (vid. pág. 28).

[83] Fulgencio Cánovas-Garcıa et al. «Modification of the random forest algorithm to avoid statistical dependence problems when classifying remote sensing imagery». En: *Computers & Geosciences* 103 (2017), págs. 1-11 (vid. pág. 29).

[84] Anjaneyulu Babu Shaik y Sujatha Srinivasan. «A brief survey on random forest ensembles in classification model». En: *International Conference on Innovative Computing and Communications*. Springer. 2019, págs. 253-260 (vid. pág. 29).

[85] Steven L Salzberg. *C4. 5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993*. 1994 (vid. págs. 30, 31).

[86] Thales Sehn Korting. «C4. 5 algorithm and multivariate decision trees». En: *Image Processing Division, National Institute for Space Research–INPE Sao Jose dos Campos–SP, Brazil* (2006), pág. 22 (vid. págs. 30, 31).

[87] Viv Bewick, Liz Cheek y Jonathan Ball. «Statistics review 14: Logistic regression». En: *Critical care* 9.1 (2005), págs. 1-7 (vid. págs. 32, 41).

[88] Jack V Tu. «Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes». En: *Journal of clinical epidemiology* 49.11 (1996), págs. 1225-1231 (vid. pág. 32).

[89] David W Hosmer Jr, Stanley Lemeshow y Rodney X Sturdivant. *Applied logistic regression*. Vol. 398. John Wiley & Sons, 2013 (vid. págs. 32, 34).

[90] Gary King y Langche Zeng. «Logistic regression in rare events data». En: *Political analysis* 9.2 (2001), págs. 137-163 (vid. pág. 32).

[91] David W Hosmer y Stanley Lemeshow. «Applied Logistic Regression. John Wiley & Sons». En: *New York* (2000) (vid. pág. 32).

[92] Nicolas Bacaër. «Verhulst and the logistic equation (1838)». En: *A short history of mathematical population dynamics*. Springer, 2011, págs. 35-39 (vid. pág. 33).

[93] Raymond Pearl y Lowell J Reed. «On the rate of growth of the population of the United States since 1790 and its mathematical representation». En: *Proceedings of the national academy of sciences* 6.6 (1920), págs. 275-288 (vid. pág. 33).

[94] Ernest Yeboah Boateng y Francis T Oduro. «Predicting microfinance credit default: a study of Nsoatreman rural bank, Ghana». En: *Journal of Advances in Mathematics and Computer Science* 26.1 (2018), págs. 1-9 (vid. págs. 33, 39).

[95] David A Hensher y William H Greene. «The mixed logit model: the state of practice». En: *Transportation* 30.2 (2003), págs. 133-176 (vid. pág. 33).

[96] David W Hosmer, Stanley Lemeshow y Rodney X Sturdivant. «The multiple logistic regression model». En: *Applied logistic regression* 1 (1989), págs. 25-37 (vid. pág. 33).

[97] Stanton A Glantz, Bryan K Slinker y Torsten B Neilands. *Primer of applied regression & analysis of variance, ed*. Vol. 654. McGraw-Hill, Inc., New York, 2001 (vid. pág. 34).

[98] Robert P Burns y Richard Burns. *Business research methods and statistics using SPSS*. Sage, 2008 (vid. pág. 34).

[99] David W Hosmer, Borko Jovanovic y Stanley Lemeshow. «Best subsets logistic regression». En: *Biometrics* (1989), págs. 1265-1270 (vid. pág. 34).

[100] L López y JL Sánchez. «Discriminant methods for radar detection of hail». En: *Atmospheric Research* 93.1-3 (2009), págs. 358-368 (vid. pág. 34).

[101] Melek Acar Boyacioglu, Yakup Kara y Ömer Kaan Baykan. «Predicting bank financial failures using neural networks, support vector machines and multivariate statistical methods: A comparative analysis in the sample of savings deposit insurance fund (SDIF) transferred banks in Turkey». En: *Expert Systems with Applications* 36.2 (2009), págs. 3355-3366 (vid. pág. 34).

[102] S Karp. «The Problem of Media Economics: Value Equations Have Radically Changed». En: *Retrieved on May* 10 (2009), pág. 2009 (vid. págs. 34, 39).

[103] Gary King, Michael Tomz y Jason Wittenberg. «Making the most of statistical analyses: Improving interpretation and presentation». En: *American journal of political science* (2000), págs. 347-361 (vid. pág. 34).

[104] N Srivastava. «A logistic regression model for predicting the occurrence of intense geomagnetic storms». En: *Annales geophysicae*. Vol. 23. 9. Copernicus GmbH. 2005, págs. 2969-2974 (vid. pág. 34).

[105] Xiaoqian Jiang, Robert El-Kareh y Lucila Ohno-Machado. «Improving predictions in imbalanced data using pairwise expanded logistic regression». En: *AMIA annual symposium proceedings*. Vol. 2011. American Medical Informatics Association. 2011, pág. 625 (vid. pág. 34).

[106] Phil Reed y Yaqionq Wu. «Logistic regression for risk factor modelling in stuttering research». En: *Journal of fluency disorders* 38.2 (2013), págs. 88-101 (vid. págs. 34, 39).

[107] Chi-Yueh Wang y Walton M Hancock. «MINIMIZING SPLITS IN THE AUTOMOTIVE STAMPING PROCESS BY LOGISTIC REGRESSION». En: *Quality engineering* 9.4 (1997), págs. 653-663 (vid. pág. 34).

[108] R Norman et al. «A comparison of peak vs cumulative physical work exposure risk factors for the reporting of low back pain in the automotive industry». En: *Clinical biomechanics* 13.8 (1998), págs. 561-573 (vid. págs. 34, 35).

[109] Douglas W Kononen, Carol AC Flannagan y Stewart C Wang. «Identification and validation of a logistic regression model for predicting serious injuries associated with motor vehicle crashes». En: *Accident Analysis & Prevention* 43.1 (2011), págs. 112-122 (vid. págs. 34, 35).

[110] Khalid S Khan, Patrick FW Chien y Linga S Dwarakanath. «Logistic regression models in obstetrics and gynecology literature». En: *Obstetrics & Gynecology* 93.6 (1999), págs. 1014-1020 (vid. pág. 34).

[111] Yongdai Kim, Sunghoon Kwon y Seuck Heun Song. «Multiclass sparse logistic regression for classification of multiple cancer types using gene expression data». En: *Computational Statistics & Data Analysis* 51.3 (2006), págs. 1643-1655 (vid. pág. 34).

[112] Peter Howell y Stephen Davis. «Predicting persistence of and recovery from stuttering by the teenage years based on information gathered at age 8 years». En: *Journal of Developmental & Behavioral Pediatrics* 32.3 (2011), págs. 196-205 (vid. pág. 34).

[113] Kenneth J Ottenbacher et al. «A review of two journals found that articles using multivariable logistic regression frequently did not report commonly recommended assumptions». En: *Journal of clinical epidemiology* 57.11 (2004), págs. 1147-1152 (vid. pág. 35).

[114] GH Hall y AP Round. «Logistic regression—explanation and use». En: *Journal of the Royal College of Physicians of London* 28.3 (1994), pág. 242 (vid. pág. 35).

[115] Guo-Wen Sun, Thomas L Shook y Gregory L Kay. «Inappropriate use of bivariable analysis to screen risk factors for use in multivariable analysis». En: *Journal of clinical epidemiology* 49.8 (1996), págs. 907-916 (vid. pág. 35).

[116] Ralf Bender y Ulrich Grouven. «Logistic regression models used in medical research are poorly presented». En: *BMJ* 313.7057 (1996), pág. 628 (vid. pág. 35).

[117] Ernest Yeboah Boateng y Daniel A Abaye. «A review of the logistic regression model with emphasis on medical research». En: *Journal of data analysis and information processing* 7.4 (2019), págs. 190-207 (vid. págs. 35, 37, 38, 40-42).

[118] Jan Salomon Cramer. «The origins of logistic regression». En: (2002) (vid. pág. 37).

[119] Walter W Hauck Jr y Allan Donner. «Wald's test as applied to hypotheses in logit analysis». En: *Journal of the american statistical association* 72.360a (1977), págs. 851-853 (vid. pág. 38).

[120] Jason W Osborne. «Bringing balance and technical accuracy to reporting odds ratios and the results of logistic regression analyses». En: *Best practices in quantitative methods* (2008), págs. 385-389 (vid. pág. 38).

[121] Paul S Levy y Kristine Stolte. «Statistical methods in public health and epidemiology: a look at the recent past and projections for the next decade». En: *Statistical Methods in Medical Research* 9.1 (2000), págs. 41-55 (vid. pág. 39).

[122] K Chien et al. «A prediction model for type 2 diabetes risk among Chinese people». En: *Diabetologia* 52.3 (2009), págs. 443-450 (vid. pág. 39).

[123] Hyeoun-Ae Park. «An introduction to logistic regression: from basic concepts to interpretation with particular attention to nursing domain». En: *Journal of Korean Academy of Nursing* 43.2 (2013), págs. 154-164 (vid. pág. 39).

[124] Ralf. BENDER. «Statistical methods in public health and epidemiology: a look at the recent past and projections for the next decade.» En: *Statistical Methods in Medical Research* 9.1 (2000), págs. 41-55 (vid. pág. 40).

[125] Kristine. LEVY Paul S.; STOLTE. «Introduction to the use of regression models in epidemiology.» En: *Cancer Epidemiology* (2009), págs. 179-195 (vid. pág. 40).

[126] Thomas Oommen, Laurie G Baise y Richard M Vogel. «Sampling bias and class imbalance in maximum-likelihood logistic regression». En: *Mathematical Geosciences* 43.1 (2011), págs. 99-120 (vid. pág. 40).

[127] Alan Agresti. «Building and applying logistic regression models». En: *Categorical data analysis* (2007), págs. 211-66 (vid. pág. 40).

[128] Scott Menard. *Applied logistic regression analysis*. 106. Sage, 2002 (vid. pág. 41).

[129] Rosa Arboretti Giancristofaro y Luigi Salmaso. «Model performance analysis and model validation in logistic regression». En: *Statistica* 63.2 (2003), págs. 375-396 (vid. pág. 42).

[130] Frank E Harrell Jr, Kerry L Lee y Daniel B Mark. «Multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors». En: *Statistics in medicine* 15.4 (1996), págs. 361-387 (vid. pág. 42).

[131] Joseph S Meyer et al. «Estimating uncertainty in population growth rates: jackknife vs. bootstrap techniques». En: *Ecology* 67.5 (1986), págs. 1156-1166 (vid. pág. 42).

[132] Robin T Vollmer. «Multivariate statistical analysis for pathologists: Part I, The logistic model». En: *American journal of clinical pathology* 105.1 (1996), págs. 115-126 (vid. pág. 43).

[133] Leif E Peterson. «K-nearest neighbor». En: *Scholarpedia* 4.2 (2009), pág. 1883 (vid. págs. 43, 44, 72).

[134] Bernard W Silverman y M Christopher Jones. «E. fix and jl hodges (1951): An important contribution to nonparametric discriminant analysis and density estimation: Commentary on fix and hodges (1951)». En: *International Statistical Review/Revue Internationale de Statistique* (1989), págs. 233-238 (vid. pág. 43).

[135] Thomas Cover y Peter Hart. «Nearest neighbor pattern classification». En: *IEEE transactions on information theory* 13.1 (1967), págs. 21-27 (vid. pág. 43).

[136] Keinosuke Fukunaga y L Hostetler. «K-nearest-neighbor Bayes-risk estimation». En: *IEEE Transactions on Information Theory* 21.3 (1975), págs. 285-293 (vid. pág. 44).

[137] Sahibsingh A Dudani. «The distance-weighted k-nearest-neighbor rule». En: *IEEE Transactions on Systems, Man, and Cybernetics* 4 (1976), págs. 325-327 (vid. pág. 44).

[138] T Bailey y JAIN AK. «A NOTE ON DISTANCE-WEIGHTED K-NEAREST NEIGHBOR RULES.» En: (1978) (vid. pág. 44).

[139] Sergio Bermejo y Joan Cabestany. «Adaptive soft k-nearest-neighbour classifiers». En: *Pattern Recognition* 33.12 (2000), págs. 1999-2005 (vid. pág. 44).

[140] Adam Jóźwik. «A learning scheme for a fuzzy k-NN rule». En: *Pattern Recognition Letters* 1.5-6 (1983), págs. 287-289 (vid. pág. 44).

[141] James M Keller, Michael R Gray y James A Givens. «A fuzzy k-nearest neighbor algorithm». En: *IEEE transactions on systems, man, and cybernetics* 4 (1985), págs. 580-585 (vid. pág. 44).

[142] Aman Kataria y MD Singh. «A review of data classification using k-nearest neighbour algorithm». En: *International Journal of Emerging Technology and Advanced Engineering* 3.6 (2013), págs. 354-360 (vid. pág. 44).

[143] David W Aha, Dennis Kibler y Marc K Albert. «Instance-based learning algorithms». En: *Machine learning* 6.1 (1991), págs. 37-66 (vid. pág. 44).

[144] Liangxiao Jiang et al. «Survey of improving k-nearest-neighbor for classification». En: *Fourth international conference on fuzzy systems and knowledge discovery (FSKD 2007)*. Vol. 1. IEEE. 2007, págs. 679-683 (vid. págs. 44-46).

[145] Aman Singh y Babita Pandey. «An euclidean distance based KNN computational method for assessing degree of liver damage». En: *2016 International Conference on Inventive Computation Technologies (ICICT)*. Vol. 1. IEEE. 2016, págs. 1-4 (vid. págs. 44, 48).

[146] Belur V Dasarathy. «Nearest neighbor (NN) norms: NN pattern classification techniques». En: *IEEE Computer Society Tutorial* (1991) (vid. pág. 44).

[147] Sana Ansari et al. «Diagnosis of liver disease induced by hepatitis virus using artificial neural networks». En: *2011 IEEE 14th international multitopic conference*. IEEE. 2011, págs. 8-12 (vid. pág. 45).

[148] Michel Verleysen y Damien François. «The curse of dimensionality in data mining and time series prediction». En: *International work-conference on artificial neural networks*. Springer. 2005, págs. 758-770 (vid. pág. 45).

[149] Tom M Mitchell y Tom M Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997 (vid. pág. 45).

[150] Ron Kohavi y George H John. «Wrappers for feature subset selection». En: *Artificial intelligence* 97.1-2 (1997), págs. 273-324 (vid. pág. 46).

[151] Pat Langley y Stephanie Sage. «Induction of selective Bayesian classifiers». En: *Uncertainty Proceedings 1994*. Elsevier, 1994, págs. 399-406 (vid. pág. 46).

[152] L Jiang et al. «Evolutional naive bayes». En: *Proceedings of the 2005 International Symposium on Intelligent Computation and its Application, ISICA*. 2005, págs. 344-350 (vid. pág. 46).

[153] Eui-Hong Sam Han, George Karypis y Vipin Kumar. «Text categorization using weight adjusted k-nearest neighbor classification». En: *Pacific-asia conference on knowledge discovery and data mining*. Springer. 2001, págs. 53-65 (vid. pág. 46).

[154] Nir Friedman, Dan Geiger y Moises Goldszmidt. «Bayesian network classifiers». En: *Machine learning* 29.2 (1997), págs. 131-163 (vid. págs. 46, 133).

[155] Zhexue Huang. «A fast clustering algorithm to cluster very large categorical data sets in data mining.» En: *Dmkd* 3.8 (1997), págs. 34-39 (vid. pág. 47).

[156] Michael J Greenacre. «Theory and applications of correspondence analysis». En: (1984) (vid. pág. 47).

[157] Craig Stanfill y David Waltz. «Toward memory-based reasoning». En: *Communications of the ACM* 29.12 (1986), págs. 1213-1228 (vid. pág. 47).

[158] Aye Aye Thant, Soe Moe Aye y Myanmar Mandalay. «Euclidean, Manhattan and Minkowski Distance Methods For Clustering Algorithms». En: *Int. J. Sci. Res. Sci. Eng. Technol* 7 (2020) (vid. págs. 48, 49).

[159] Yann LeCun, Yoshua Bengio y Geoffrey Hinton. «Deep learning». En: *nature* 521.7553 (2015), págs. 436-444 (vid. pág. 49).

[160] Alex Krizhevsky, Ilya Sutskever y Geoffrey E Hinton. «Imagenet classification with deep convolutional neural networks». En: *Communications of the ACM* 60.6 (2017), págs. 84-90 (vid. pág. 49).

[161] Jacob Devlin et al. «Bert: Pre-training of deep bidirectional transformers for language understanding». En: *arXiv preprint arXiv:1810.04805* (2018) (vid. pág. 49).

[162] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Inf. téc. Cornell Aeronautical Lab Inc Buffalo NY, 1961 (vid. pág. 49).

[163] Kunihiko Fukushima y Sei Miyake. «Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position». En: *Pattern recognition* 15.6 (1982), págs. 455-469 (vid. pág. 49).

[164] Yann LeCun, Yoshua Bengio et al. «Convolutional networks for images, speech, and time series». En: *The handbook of brain theory and neural networks* 3361.10 (1995), pág. 1995 (vid. pág. 49).

[165] David H Hubel y Torsten N Wiesel. «Receptive fields, binocular interaction and functional architecture in the cat's visual cortex». En: *The Journal of physiology* 160.1 (1962), pág. 106 (vid. pág. 49).

[166] Daniel J Felleman y David C Van Essen. «Distributed hierarchical processing in the primate cerebral cortex.» En: *Cerebral cortex (New York, NY: 1991)* 1.1 (1991), págs. 1-47 (vid. pág. 49).

[167] Demis Hassabis et al. «Neuroscience-inspired artificial intelligence». En: *Neuron* 95.2 (2017), págs. 245-258 (vid. pág. 49).

[168] Guangyu Robert Yang y Xiao-Jing Wang. «Artificial neural networks for neuroscientists: A primer». En: *Neuron* 107.6 (2020), págs. 1048-1070 (vid. págs. 50-53).

[169] Yann LeCun et al. «Gradient-based learning applied to document recognition». En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324 (vid. pág. 50).

[170] Frank Rosenblatt. «The perceptron: a probabilistic model for information storage and organization in the brain.» En: *Psychological review* 65.6 (1958), pág. 386 (vid. pág. 51).

[171] Kurt Hornik, Maxwell Stinchcombe y Halbert White. «Multilayer feedforward networks are universal approximators». En: *Neural networks* 2.5 (1989), págs. 359-366 (vid. pág. 52).

[172] Herbert Robbins y Sutton Monro. «A stochastic approximation method». En: *The annals of mathematical statistics* (1951), págs. 400-407 (vid. pág. 52).

[173] David E Rumelhart, Geoffrey E Hinton y Ronald J Williams. «Learning representations by back-propagating errors». En: *nature* 323.6088 (1986), págs. 533-536 (vid. pág. 52).

[174] Google. *Dev Board*. `https://coral.ai/products/dev-board`. [Online; accessed 20-April-2020]. 2020 (vid. págs. 60, 90).

[175] Bosch. *Reduce manufacturing failures*. `https://www.kaggle.com/c/bosch-production-line-performance/data`. [Online; accessed 20-April-2021] (vid. págs. 60, 96).

[176] Jia Li et al. «Deep convolutional neural network based ECG classification system using information fusion and one-hot encoding techniques». En: *Mathematical Problems in Engineering* 2018 (2018) (vid. pág. 61).

[177] Nerea Gómez Larrakoetxea. private interview. 2020 (vid. pág. 95).

[178] Dalwinder Singh y Birmohan Singh. «Investigating the impact of data normalization on classification performance». En: *Applied Soft Computing* 97 (2020), pág. 105524 (vid. pág. 102).

[179] Jorge Sola y Joaquin Sevilla. «Importance of input data normalization for the application of neural networks to complex industrial problems». En: *IEEE Transactions on nuclear science* 44.3 (1997), págs. 1464-1468 (vid. pág. 102).

[180] SC Nayak, Bijan B Misra e Himansu Sekhar Behera. «Impact of data normalization on stock index forecasting». En: *International Journal of Computer Information Systems and Industrial Management Applications* 6.2014 (2014), págs. 257-269 (vid. pág. 102).

[181] Seo Young Kim, Jae Won Lee y Jong Sung Bae. «Effect of data normalization on fuzzy clustering of DNA microarray data». En: *BMC bioinformatics* 7.1 (2006), págs. 1-14 (vid. pág. 102).

[182] S Vani y TV Madhusudhana Rao. «An experimental approach towards the performance assessment of various optimizers on convolutional neural network». En: *2019 3rd international conference on trends in electronics and informatics (ICOEI)*. IEEE. 2019, págs. 331-336 (vid. pág. 105).

[183] Léon Bottou et al. «Stochastic gradient learning in neural networks». En: *Proceedings of Neuro-Nımes* 91.8 (1991), pág. 12 (vid. pág. 105).

[184] Luis Velasco. *Optimizadores en redes neuronales profundas: un enfoque práctico*. https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5. [Online; accessed 30-Nov-2022] (vid. pág. 105).

[185] Martin Zinkevich et al. «Parallelized stochastic gradient descent». En: *Advances in neural information processing systems* 23 (2010) (vid. pág. 105).

[186] Jing Yang y Guanci Yang. «Modified convolutional neural network based on dropout and the stochastic gradient descent optimizer». En: *Algorithms* 11.3 (2018), pág. 28 (vid. pág. 106).

[187] Moritz Hardt, Ben Recht y Yoram Singer. «Train faster, generalize better: Stability of stochastic gradient descent». En: *International conference on machine learning*. PMLR. 2016, págs. 1225-1234 (vid. pág. 106).

[188] Sebastian Bock, Josef Goppold y Martin Weiß. «An improvement of the convergence proof of the ADAM-Optimizer». En: *arXiv preprint arXiv:1804.10587* (2018) (vid. págs. 107, 108).

[189] Zhigen Fei et al. «A new short-arc fitting method with high precision using Adam optimization algorithm». En: *Optik* 212 (2020), pág. 164788 (vid. pág. 107).

[190] Timothy Dozat. «Incorporating nesterov momentum into adam». En: (2016) (vid. pág. 107).

[191] Geeks for geeks. *Intuition of Adam Optimizer*. https://www.geeksforgeeks.org/intuition-of-adam-optimizer/. [Online; accessed 30-Nov-2022] (vid. pág. 108).

[192] Nikhil Ketkar. «Introduction to keras». En: *Deep learning with Python*. Springer, 2017, págs. 97-111 (vid. pág. 123).

[193] Qi Shan et al. «Fast image/video upsampling». En: *ACM Transactions on Graphics (TOG)* 27.5 (2008), págs. 1-7 (vid. pág. 125).

[194] Jason Brownlee. «What is the Difference Between a Batch and an Epoch in a Neural Network». En: *Machine Learning Mastery* 20 (2018) (vid. pág. 129).

[195] EduardoMagalhãesOliveira. *Quality Prediction in a Mining Process*. https://www.kaggle.com/edumagalhaes/quality-prediction-in-a-mining-process. [Online; accessed 15-March-2020] (vid. pág. 130).

[196] Eduardo Morales Manzanares. *Hill-Climbing*. 2004. URL: https://ccc.inaoep.mx/~emorales/Cursos/Busqueda/node23.html (visitado 20-03-2020) (vid. pág. 132).

[197] Fei Zheng y Geoffrey I. Webb. «Tree Augmented Naive Bayes». En: *Encyclopedia of Machine Learning*. Ed. por Claude Sammut y Geoffrey I. Webb. Boston, MA: Springer US, 2010, págs. 990-991. DOI: 10.1007/978-0-387-30164-8_850 (vid. pág. 133).

[198] Boaz Lerner y Roy Malka*. «Investigation of the K2 algorithm in learning Bayesian network classifiers». En: *Applied Artificial Intelligence* 25.1 (2011), págs. 74-96 (vid. pág. 133).

[199] Manish Mathuria. «Decision tree analysis on j48 algorithm for data mining». En: *Intrenational Journal of Advanced Research in Computer Science and Software Engineering* 3.6 (2013) (vid. pág. 133).

[200] Laszlo Kozma. «k Nearest Neighbors algorithm (kNN)». En: *Helsinki University of Technology* (2008) (vid. pág. 133).

[201] Gurneet Kaur y Er Neelam Oberai. «A review article on Naive Bayes classifier with various smoothing techniques». En: *International Journal of Computer Science and Mobile Computing* 3.10 (2014), págs. 864-868 (vid. pág. 133).

[202] Adele Cutler, D Richard Cutler y John R Stevens. «Random forests». En: *Ensemble machine learning*. Springer, 2012, págs. 157-175 (vid. pág. 133).

[203] Todd G Nick y Kathleen M Campbell. «Logistic regression». En: *Topics in biostatistics* (2007), págs. 273-301 (vid. pág. 133).

[204] Witold Pedrycz y Shyi-Ming Chen. *Information granularity, big data, and computational intelligence*. Vol. 8. Springer, 2014 (vid. pág. 140).