

**UNIVERSIDAD ALFONSO X EL SABIO**  
**ESCUELA DE INGENIERÍA, ARQUITECTURA Y DISEÑO**  
**MÁSTER UNIVERSITARIO EN INGENIERÍA AERONÁUTICA**  
**(MUN)**



**TRABAJO FIN DE MÁSTER**

**MODELOS DE APRENDIZAJE AUTOMÁTICO PARA  
MANTENIMIENTO PREDICTIVO**

**ANDRÉS VICENTE MARTÍNEZ**

**CONVOCATORIA SEPTIEMBRE 2024**





**UNIVERSIDAD ALFONSO X EL SABIO**

**ESCUELA DE INGENIERÍA, ARQUITECTURA Y DISEÑO**

**MÁSTER UNIVERSITARIO EN INGENIERÍA AERONÁUTICA**

**MODELOS DE APRENDIZAJE AUTOMÁTICO PARA  
MANTENIMIENTO PREDICTIVO**

**ALUMNO:** Andrés Vicente Martínez

**N.P.:** 854052

**DIRECTOR DEL TRABAJO:** N/A

**TUTOR ACADÉMICO:** Marcos Antonio Rodríguez Jiménez

**FECHA DE PRESENTACIÓN:** DD/09/2025

**BREVE DESCRIPCIÓN:**

El siguiente proyecto trata sobre las aplicaciones del aprendizaje automático, comúnmente conocido como Machine Learning, en actividades relacionadas con el mantenimiento aeronáutico.

El proyecto replica dos estudios recientes en el ámbito del mantenimiento predictivo.

**FIRMA DEL ALUMNO**



## Resumen

Este proyecto presenta el estudio de los principales algoritmos de Aprendizaje Automático (Machine Learning) aplicados al mantenimiento predictivo de aeronaves y equipos aeronáuticos. Con el objetivo de demostrar su aplicabilidad práctica, se tomaron como referencia dos artículos científicos que emplean algoritmos de clasificación.

En primer lugar, se replicó un estudio enfocado en la clasificación mediante Aprendizaje Supervisado. Este es utiliza diferentes algoritmos para separar un conjunto de señales, en función de si estas muestran síntomas de fallo o funcionamiento normal. En este primer estudio se entrenaron diferentes modelos y se comparan sus resultados y métricas de rendimiento.

En el segundo estudio, se implementó una metodología que combina reducción de dimensionalidad con modelos de clasificación probabilística. En esta parte del proyecto se investigaron diferentes estrategias para replicar los resultados de la bibliografía, empleando métodos de optimización iterativos que permiten encontrar la combinación óptima de parámetros de un autoencoder. Finalmente, se implementó un modelo de mezclas gausianas, para clasificar una serie de motores según su estado de salud, validando los resultados obtenidos con los del estudio original. Los datos utilizados en este caso proceden de una serie de simulaciones de un motor turbofán, obtenidas y publicadas por la NASA.

**Palabras clave:** mantenimiento predictivo (PdM), aprendizaje automático, aprendizaje supervisado, aprendizaje no supervisado, autoencoder (AE), modelos de mezcla gaussiana (GMM), vida útil remanente (RUL).



## **Dedicatoria**

A mis padres, que una vez más me han apoyado en todos los aspectos para que pueda perseguir todas mis metas.

También a mis amigos de Múnich, que han estado ahí durante todo el verano, animándome a encontrar la motivación en los días en los que parece no haberla, y al Dr. Slawomir Szrama, que desde la Universidad de Poznan respondió a mis dudas en pleno agosto.

Este trabajo debe ser también un recordatorio para tí mismo de la capacidad que tienes para cumplir todo lo que te propongas.



## Índice

<b>1 INTRODUCCIÓN</b>	<b>1</b>
1.1 Mantenimiento aeronáutico y la necesidad del mantenimiento predictivo . . . . .	2
1.2 Costes de mantenimiento de aeronaves civiles . . . . .	3
1.2.1 Coste de mantenimiento de aeronaves militares . . . . .	5
1.3 Evolución de las estrategias de mantenimiento: Correctivo, Preventivo y Predictivo . . . . .	7
1.3.1 Mantenimiento reactivo . . . . .	8
1.3.2 Mantenimiento preventivo . . . . .	8
1.3.3 Mantenimiento predictivo . . . . .	9
1.3.4 Mantenimiento prescriptivo . . . . .	11
1.4 Machine Learning en mantenimiento predictivo . . . . .	12
1.4.1 Fundamentos del Machine Learning . . . . .	12
1.4.2 Aplicaciones actuales en la industria aeronáutica . . . . .	15
1.5 Estado del arte: ML y PdM . . . . .	17
<b>2 OBJETIVOS Y MOTIVACIÓN</b>	<b>21</b>
2.1 Objetivos del proyecto . . . . .	21
2.2 Justificación y relevancia del estudio . . . . .	21
2.3 Estructura del proyecto . . . . .	22
<b>3 Modelos de Machine Learning para mantenimiento predictivo</b>	<b>23</b>
3.1 Modelos supervisados (Random Forest, SVM, Redes Neuronales, etc.) . . . . .	24
3.1.1 Regresión lineal . . . . .	25
3.1.2 Regresión Logística . . . . .	26
3.1.3 K-Nearest Neighbours . . . . .	27
3.1.4 Support Vector Machines (SVMs) . . . . .	28
3.1.5 Decision Tree (DT) y Random Forest (RF) . . . . .	30
3.2 Redes Neuronales . . . . .	31
3.3 Modelos no supervisados . . . . .	32
3.3.1 Clustering . . . . .	33
3.3.2 Reglas de asociación . . . . .	34
3.3.3 Reducción de la dimensionalidad . . . . .	35
3.4 Modelos híbridos y su potencial en el mantenimiento predictivo . . . . .	36

3.5 Reinforcement Learning . . . . .	37
3.6 Errores de ajuste de modelos . . . . .	37
3.6.1 Overfitting . . . . .	37
3.6.2 Underfitting . . . . .	38
3.7 Métricas de rendimiento del modelo . . . . .	38
3.7.1 Exactitud (Accuracy) . . . . .	39
3.7.2 Matriz de confusión . . . . .	39
3.7.3 Precisión . . . . .	40
3.7.4 Recall . . . . .	40
3.7.5 F1-Score . . . . .	40
3.7.6 Índice de Silhouette . . . . .	41
3.8 Validación Cruzada . . . . .	42
3.9 Algoritmos de Optimización . . . . .	42
3.9.1 Gradiente Descendente . . . . .	42
3.9.2 RMSProp, AdaGrad, momentum y Adam . . . . .	43
3.9.3 Grid Search . . . . .	46
3.10 Implementación computacional en Python . . . . .	47
<b>4 Experimentación y Resultados</b>	<b>49</b>
4.1 Caso 1: Modelos Supervisados en Tareas de Clasificación . . . . .	49
4.1.1 Metodología . . . . .	49
4.1.2 Descripción del dataset . . . . .	50
4.1.3 Preprocesamiento y división del dataset . . . . .	52
4.1.4 Entrenamiento de los modelos . . . . .	53
4.1.5 Evaluación de precisión, error y robustez del modelo . . . . .	53
4.1.6 Visualización de resultados . . . . .	55
4.2 Caso 2: Modelos No Supervisados en tareas de clasificación . . . . .	57
4.2.1 Metodología . . . . .	59
4.2.2 Descripción del dataset . . . . .	60
4.2.3 Preprocesamiento y división del dataset . . . . .	64
4.2.4 Entrenamiento del Autoencoder . . . . .	64
4.2.5 Consolidación de la matriz de entrada y clustering por GMM . . . . .	70
4.2.6 Comparación de los tres casos . . . . .	82
<b>5 Conclusiones y Futuros Trabajos</b>	<b>87</b>

5.1 Principales resultados obtenidos . . . . .	87
5.1.1 Resultados del Caso Práctico 1 . . . . .	87
5.1.2 Resultados del Caso 2 . . . . .	89
5.2 Contribuciones del estudio . . . . .	92
5.3 Limitaciones y posibles mejoras . . . . .	92
5.4 Aplicaciones futuras y líneas de investigación . . . . .	93
<b>Bibliografía</b>	<b>95</b>



## Lista de Figuras

1.1	Ciclo de vida de una aeronave desde su creación hasta su retirada (Fuente: [1]). . . . .	2
1.2	asa de accidentes y víctimas mortales por millón de despegues en la flota internacional hasta 2022 (Fuente: FAA [2]). . . . .	3
1.3	Desglose de gastos de las principales aerolíneas españolas, desde 2010 hasta 2020 (Fuente: OTLE [3]). . . . .	4
1.4	Desglose de gastos de aerolíneas a nivel global (Fuente: IATA [4]). . . . .	5
1.5	Representación visual de los diferentes gastos de un programa militar según su naturaleza (Fuente: Aviation Week [7]). . . . .	6
1.6	Clasificación general de las principales estrategias de mantenimiento aeronáutico (Fuente: [8]). . . . .	7
1.7	Diagrama de flujo con las principales fases del mantenimiento predictivo (Fuente: elaboración propia). . . . .	10
1.8	Comparación de procesos de modelado: programación explícita, ML superficial y Deep Learning (Fuente: [23]). . . . .	14
1.9	Resumen esquemático del proceso de mantenimiento predictivo (Fuente: [14]). . . . .	16
1.10	Ejemplo simplificado de Gemelo Digital de un avión comercial (Fuente: [29]). . . . .	18
1.11	Equipo físico y metodología propuesta por Aminzadeh et al. en el análisis de compresores de aire (Fuente: [32]). . . . .	20
3.1	Clasificación general de algoritmos de ML (Fuente: Analytics Yogi [37]). . . . .	23
3.2	Ejemplo de datos supervisados (Fuente: [22]). . . . .	24
3.3	Errores homocedásticos y no homocedásticos (Fuente: [42]). . . . .	25
3.4	Representación gráfica de la función sigmoide (Fuente: elaboración propia) . . . . .	27
3.5	Ejemplo de clasificación por k-NN con $k = 5$ . (Fuente: Freie Universität Berlin [46]). . . . .	28
3.6	Principio de funcionamiento de los SVMs con dos clases (Fuente: [47]). . . . .	29
3.7	Árbol de decisión categórico aplicado al mantenimiento predictivo de un motor eléctrico genérico (Fuente: elaboración propia). . . . .	30
3.8	Ejemplo de una capa de red neuronal (Fuente: [52]). . . . .	31
3.9	Diagrama de funcionamiento una neurona en una NN con función de activación sigmoide (Fuente: elaboración propia). . . . .	32
3.10	Dendrograma de clustering jerárquico mediante el método divisivo (Fuente: IBM [54]). . . . .	33
3.11	Distribuciones gaussianas superpuestas en un GMM (Fuente: Universidad de Shanghai Jiao Tong [55]). . . . .	34

3.12 Representación gráfica de una red neuronal en forma de autoencoder (Fuente: AI Planet [58]). . . . .	36
3.13 Representación gráfica del overfitting y underfitting en tareas de regresión y clasificación (Fuente: Mathworks [64]). . . . .	38
3.14 Matriz de confusión con los conceptos de precisión y recall (Fuente: [22]). . .	40
3.15 Esquema gráfico del funcionamiento del gradiente descendente (Fuente: Analytics Yogi [37]) . . . . .	43
4.1 Ejemplos gráficos de distribuciones con distinta skewness y kurtosis (Fuente: elaboración propia). . . . .	52
4.2 Salida de la función <code>df.info()</code> en Python. . . . .	52
4.3 Matrices de confusión de los tres modelos supervisados (Fuente: elaboración propia). . . . .	53
4.4 Distribución cruzada de variables una vez clasificados los datos (Fuente: elaboración propia). . . . .	56
4.5 (a) Esquema simplificado del motor turbofán [76]. (b) Diagrama de flujo del paso del aire y combustible por las diferentes estaciones del mismo [77] . . .	58
4.6 Resumen esquemático de la metodología del caso 2 . . . . .	59
4.7 Información sobre el dataset <code>train_FD001.txt</code> en Python . . . . .	62
4.8 Función ReLU (Fuente: elaboración propia). . . . .	66
4.9 Evolución del error con el número de épocas para tres configuraciones distintas del AE (Fuente: elaboración propia). . . . .	68
4.10 Evolución del error con el número de épocas para las dos mejores configuraciones	69
4.11 Análisis de Componentes Principales de la clasificación con Agregación Estadística (Fuente: elaboración propia). . . . .	73
4.12 Ejemplo que muestra el concepto de Post Padding con ceros en una estructura de datos (Fuente: elaboración propia). . . . .	75
4.13 Análisis de Componentes Principales de la clasificación con Zero Padding (Fuente: elaboración propia). . . . .	77
4.14 PCA de los resultados obtenidos por Lodygowski et al. para los datos del CMAPSS (Fuente: [36]). . . . .	78
4.15 Análisis de Componentes Principales según clasificación con GMM usando Fixed Length Sampling (Fuente: elaboración propia). . . . .	80
4.16 PCA de los resultados obtenidos por Lodygowski et al. para los datos de motores reales (Fuente: [36]). . . . .	81
4.17 RUL por Cluster en el caso de Agregación Estadística . . . . .	84
4.18 RUL por Cluster en el caso de Fixed Length Sampling . . . . .	85

4.19 RUL por Cluster en el caso de Zero Padding . . . . .	86
5.1 Dispersión y Distribución de los elementos respecto a la Varianza y la Skewness	88



## Lista de Tablas

1.1	Resumen de diferencias entre IA, ML y DL . . . . .	13
4.1	Ejemplo de registros del dataset Bill Authentication con sus cuatro características y la clase asociada. . . . .	51
4.2	Decision Tree: Precisión, Recall y F1-score por clase . . . . .	54
4.3	Random Forest: Precisión, Recall y F1-score por clase . . . . .	54
4.4	K-Nearest Neighbors (k=5) Precisión, Recall y F1-score por clase . . . . .	54
4.5	Comparación de modelos con validación cruzada de 5 folds . . . . .	54
4.6	Comparación entre resultados experimentales y los reportados en el trabajo de Ouadah et al. . . . .	55
4.7	Parámetros de salida del motor turbofán simulados en CMAPSS. . . . .	61
4.8	Descripción estadística del dataset FD001 . . . . .	63
4.9	Parámetros óptimos del autoencoder según Lodygowski et al. para el set de datos de motores reales . . . . .	65
4.10	Comparativa entre configuraciones de autoencoder . . . . .	69
4.11	Composición en capas del Autoencoder . . . . .	70
4.12	Representación del dataset tras agregación estadística, usando media, desviación típica y valor máximo para cada variable latente. . . . .	72
4.13	Distribución de motores por cluster según clasificación GMM usando Agregación Estadística . . . . .	73
4.14	Comparación de medias por cluster de las variables más discriminantes en el dataset de agregación estadística . . . . .	74
4.15	Ejemplo de zero-padding con un máximo de 10 ciclos por motor. . . . .	76
4.16	Distribución de motores por cluster según clasificación GMM usando Zero Padding . . . . .	76
4.17	Distribución de motores según el número de ciclos registrados . . . . .	79
4.18	Distribución de motores por cluster según clasificación GMM usando Fixed Length Sampling . . . . .	80
4.19	Comparación de resultados de clustering GMM con distintas matrices de entrada	82
4.20	Estadísticas por cluster en Fixed Length Sampling . . . . .	84
4.21	Estadísticas por cluster en Fixed Length Sampling . . . . .	85
4.22	Estadísticas por cluster en Zero Padding . . . . .	86
5.1	Comparación porcentual de las métricas de cada modelo respecto al kNN . .	87
5.2	Comparación de la estructura del autoencoder de Lodygowski para motores reales y el obtenido para CMAPSS. . . . .	90



## **Tabla de Acrónomos y Símbolos**

AE	Autoencoder
(A)NN	(Artificial) Neural Network
CMAPSS	Commercial Modular Aero-Propulsion System Simulation
CNN	Convolutional Neural Network
DL	Deep Learning
DT	Decission Tree
DTw	Digital Twin
FLS	Fixed Length Sampling
GMM	Gaussian Mixture Model
HI	Health Index
IA	Inteligencia Artificial
IoT	Internet of Things
k-NN	k-Nearest Neighbors
LSTM	Long Short-Term Memory
ML	Machine Learning
MRO	Maintenance Repair and Overhaul
PCA	Principal Component Analysis
PdM	Predictive Maintenance
RCM	Reliability Centered Maintenance
ReLU	Rectified Linear Unit
RF	Random Forest
RL	Reinforcement Learning
RUL	Remaining Useful Life
SVM	Support Vector Machine
UAV	Unmanned Aereal Vehicle
ZP	Zero Padding



## 1 INTRODUCCIÓN

El mantenimiento de una aeronave es uno de los pilares fundamentales para facilitar su apropiado uso durante el ciclo de vida del producto. Esta actividad es fundamental para el desarrollo de las operaciones de forma segura, ya que en el ámbito aeronáutico es imprescindible la detección de fallos previos al funcionamiento erróneo de los sistemas. Además, los programas de revisión de aeronaves son una parte clave del modelo de negocio de empresas como aerolíneas, debido a la importancia de mantener una disponibilidad de flota lo más elevada posible. Por otra parte, en el ámbito militar resulta también clave la absoluta disposición de los sistemas necesarios en caso de necesidad de despliegue.

Según el momento en el que se aplican las correcciones necesarias, las estrategias de mantenimiento se pueden clasificar en dos grandes grupos. Por un lado está el mantenimiento reactivo, en el que se actúa una vez aparecen los fallos. En el extremo opuesto se encuentra el mantenimiento proactivo en el que la aeronave es reparada anticipándose al funcionamiento indeseable.

Estos métodos han ido evolucionando a lo largo de la historia de la aviación, debido a los principales avances tecnológicos de la industria. En este contexto, el desarrollo tan acelerado de la Inteligencia Artificial en las últimas décadas ha favorecido la transformación de los métodos de mantenimiento hacia la parte proactiva-predictiva. El estilo predictivo busca aplicar medidas anticipadas en el momento óptimo, reduciendo costes y aumentando la disponibilidad de los equipos. Estas medidas deben realizarse manteniendo un margen de seguridad que no ponga en peligro la seguridad del vuelo.

En este proyecto se presentará una visión general de la evolución del mantenimiento hasta llegar a la rama predictiva. Se revisará la importancia de las reparaciones y los procesos asociados a las mismas en programas aeronáuticos, incluyendo los costes de los programas y los desafíos existentes a día de hoy. Asimismo, se verá cómo la Inteligencia Artificial (IA) por medio del aprendizaje automático, más conocido como Machine Learning (ML), ha facilitado su adopción. Las secciones dedicadas esta disciplina recorren los principales algoritmos de predicción usados en la industria, con ejemplos detallados sobre su funcionamiento e implementación. Aquí se detallan también en las etapas más importantes de un proyecto de IA que emplea Aprendizaje Automático. Finalmente, se replicarán varios artículos recientes que utilizan ML aplicado al mantenimiento predictivo, aportando explicaciones detalladas sobre los datos, metodologías y algoritmos usados en la bibliografía. Estos serán llevados a la práctica mediante su implementación en un código de Python. A partir de este código se generarán una series de resultados en formas de gráficas y métricas de evaluación de los modelos.

## 1.1 Mantenimiento aeronáutico y la necesidad del mantenimiento predictivo

El mantenimiento es una de las fases fundamentales en el ciclo de vida de un producto. En general, el ciclo de vida del producto está compuesto por diferentes etapas, en las que se encuentran la planificación, diseño y desarrollo, producción y montaje, seguidos del despliegue y mantenimiento y acabando con la retirada del producto cuando su vida útil finaliza.

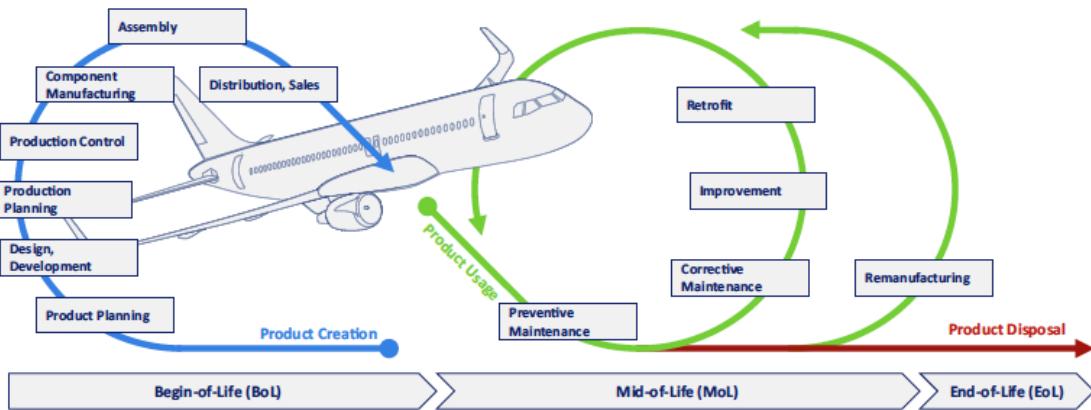


Figura 1.1. Ciclo de vida de una aeronave desde su creación hasta su retirada (Fuente: [1]).

Como se ve en la Figura 1.1, existen varias fases necesarias para la puesta en servicio del producto, previas a las tareas de inspección. Sin embargo, en el caso de una aeronave, una vez que esta comienza a operar, las tareas asociadas al mantenimiento, reparación y revisión mayor (MRO, Maintenance, Repair and Overhaul) pasan a ser parte imprescindible del ciclo de vida, ya que estas se producen de manera recurrente durante el uso del vehículo. Como indica K. Moenk en su estudio *Digital twins in aircraft production and MRO: challenges and opportunities*[1], en el marco temporal se da mayor importancia al *Mid-of-Life*, que abarca el mantenimiento durante la fase operativa, frente al *Begin-of-Life*, centrado en diseño y producción.

El mantenimiento aeronáutico es esencial desde el punto de vista de seguridad de las operaciones. Esto es así en cualquier medio de transporte, pero en aviación existe un riesgo adicional, causado por el medio en el que se transportan los vehículos. A diferencia de medios de transporte terrestres, si la aeronave se encuentra en el aire no se puede simplemente estacionar y esperar a que un vehículo especializado acuda en su ayuda. En caso de avería durante el vuelo, la aeronave debe contar con redundancias suficientes para poder continuar su trayecto hasta el aeropuerto más cercano, o en su defecto, realizar un aterrizaje de emergencia. En aeronaves comerciales, un fallo no controlado puede ocasionar consecuencias catastróficas, poniendo en riesgo las vidas de cientos de usuarios. No obstante, gracias a los avances en

seguridad, la aviación se mantiene como el medio de transporte más seguro, con una tasa de accidentes fatales que continúa descendiendo tanto en número de vuelos como en distancia recorrida [2].

Además, el mantenimiento de una aeronave colleva gastos muy elevados. Las instalaciones, equipos, personal cualificado necesario para realizar las tareas y las piezas necesarias para sustituir a las defectuosas suponen altos costes para el operador, ya sea un particular, aerolínea o un ejercito en el caso de vehículos militares.

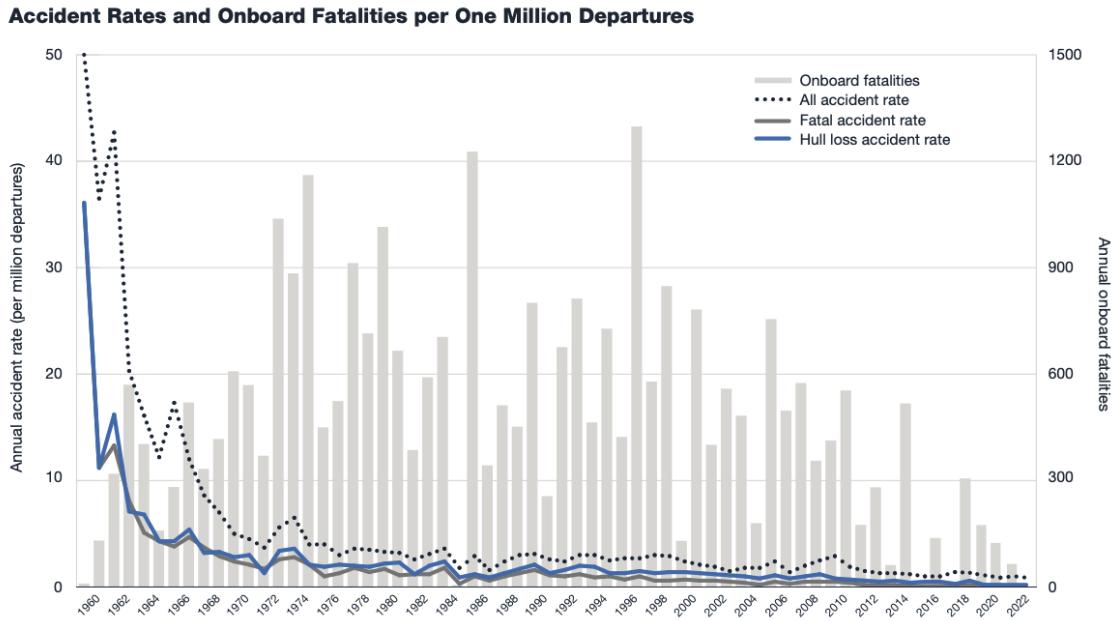


Figura 1.2. tasa de accidentes y víctimas mortales por millón de despegues en la flota internacional hasta 2022 (Fuente: FAA [2]).

## 1.2 Costes de mantenimiento de aeronaves civiles

En un estudio de la OTLE (Observatorio del Transporte y la Logística en España) en colaboración con la Dirección General de Aviación Civil (DGAC) [3] se analizaron los gastos de las principales aerolíneas españolas, obteniéndose los siguientes resultados:

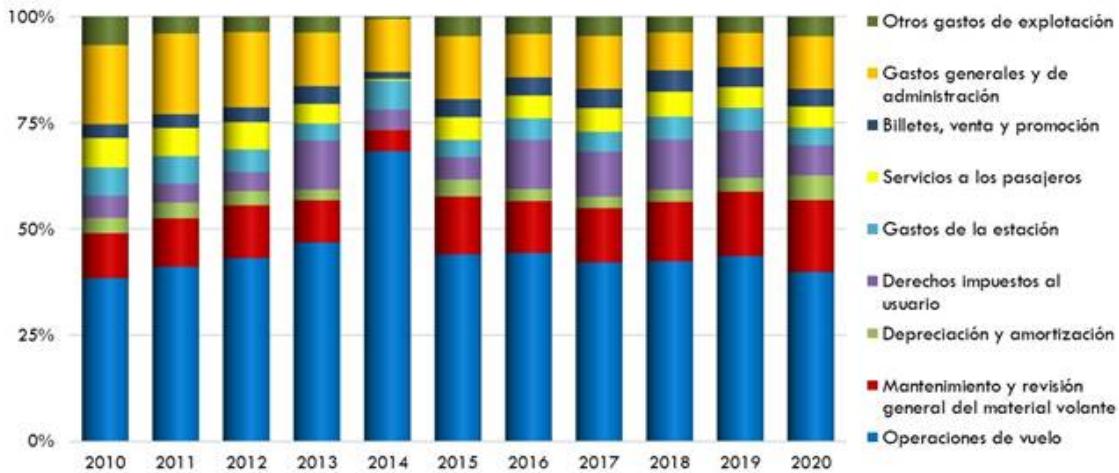


Figura 1.3. Desglose de gastos de las principales aerolíneas españolas, desde 2010 hasta 2020 (Fuente: OTLE [3]).

Cabe destacar que el estudio solo considera datos hasta 2020, año en el que se redujeron en gran medida los costes de las aerolíneas debido a la crisis de la pandemia causada por el COVID-19. Sin embargo, se puede ver cómo el gasto porcentual asociado a las operaciones de vuelo ocupa el primer puesto año a año, significando en torno al 40 % anual (aunque llegó a ser casi un 70 % en 2014). En segundo lugar se sitúa el mantenimiento del material aeronáutico, que representa en 2020 un 17% del total. Estos datos reflejan la importancia del mantenimiento como área de inversión, así como de la necesidad de desarrollar nuevas técnicas para la reducción de costes sin comprometer la calidad de los procesos de reparación y revisión, y en consecuencia, la seguridad de las operaciones.

Si comparamos estas cifras con las ecogidas por la WATS (World Air Transport Statistics) en 2022, los costes asociados al MRO de los equipos son del 8,4 % del total. Los datos difieren ya que el desglose realizado en el estudio de la WATS es distinto al del OTLE, incluyendo nuevas categorías como se puede ver en la figura Figura 1.4.

Además hay que tener en cuenta que estos datos publicados por IATA tienen en cuenta aerolíneas a nivel global. Esto también conlleva grandes diferencias en los porcentajes con respecto al mercado de la aviación civil española.

Sin embargo, se ve cómo los gastos de mantenimiento (en rosa) alcanzan 8,4 puntos porcentuales, manteniéndose así entre los mayores gastos después de los precios de aceite y combustibles (28,7%), otros gastos operacionales (10,16%), depreciación y amortización (9,1%) y los salarios y gastos de las tripulaciones (8,6%) [4].

En cuanto al coste de mantenimiento para una aeronave, no son demasiados los estudios disponibles, ya que las aerolíneas y principales operadores prefieren no公开 estos datos en la mayoría de los casos. Uno de los pocos estudios que se han podido encontrar [5],

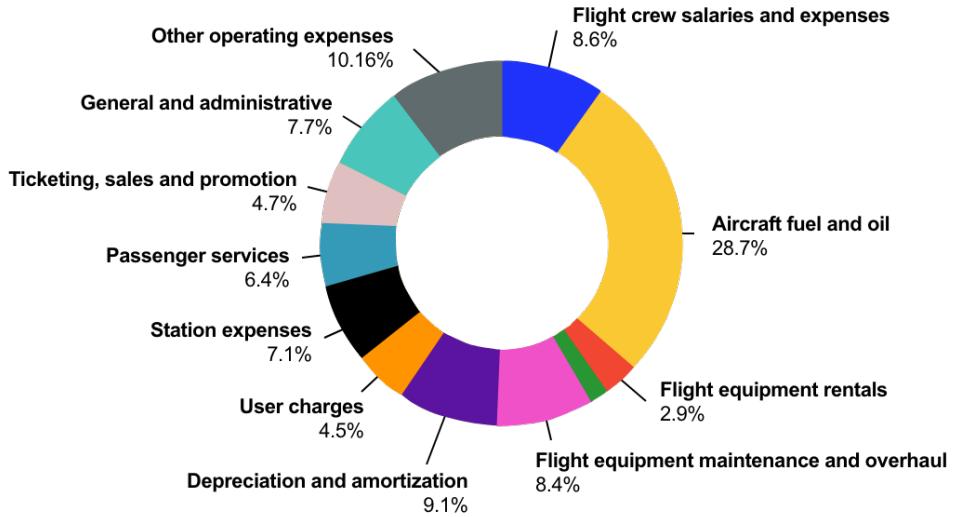


Figura 1.4. Desglose de gastos de aerolíneas a nivel global (Fuente: IATA [4]).

data del año 2006, en el que se estimó que un ciclo de mantenimiento base (unos 9 años de mantenimiento, en los que se incluyen mantenimientos pesados y revisiones rutinarias) de un A320, suponen unas 43 500 horas de mano de obra, equivalentes a unos 3,5 millones de dólares. Extrapolando esta cifra a la vida útil de una aeronave de este estilo (alrededor de los 30 años), los gastos ascienden a unos 12 millones de dólares. Esta suma representa aproximadamente el 10 % del coste unitario de una de estas aeronaves [6].

Además el propio estudio indica que hay gastos que no se han tenido en cuenta, como los mantenimientos en línea o reparaciones mayores imprevistas, así como costes de planificación e ingeniería, por lo que estaríamos hablando de costes bastante más elevados.

### 1.2.1 Coste de mantenimiento de aeronaves militares

Un estudio a destacar fue realizado por la compañía de noticias aeronáuticas Aviation Week, en el que se compararon los costes durante el ciclo de vida completo de varios aviones de combate. Aquí se define el concepto *Through Life Costs*, haciendo referencia al coste completo de los programas militares que se pueden separar en distintas categorías, como se muestra en la Figura 1.5.

Entre otras, se observa cómo el mantenimiento (en la parte inferior derecha) ocupa una gran parte de estos costes. A su vez se enumera una larga lista de gastos asociados al mantenimiento de la aeronave, motores, software, consumibles y repuestos; trabajos de reparación y soporte de ingeniería; publicaciones técnicas y el soporte a la entidad que adquiere el programa. Todos estos aportes económicos son a priori necesarios y difícilmente reducibles. Sin embargo, se puede apreciar también la casilla de operaciones (en azul arriba a la derecha),

en la que se incluye el coste de la gestión del programa. Aquí cobra verdadera importancia la planificación y las técnicas de mantenimiento usadas, donde el mantenimiento predictivo puede desempeñar un papel diferencial, como se detallará más adelante.

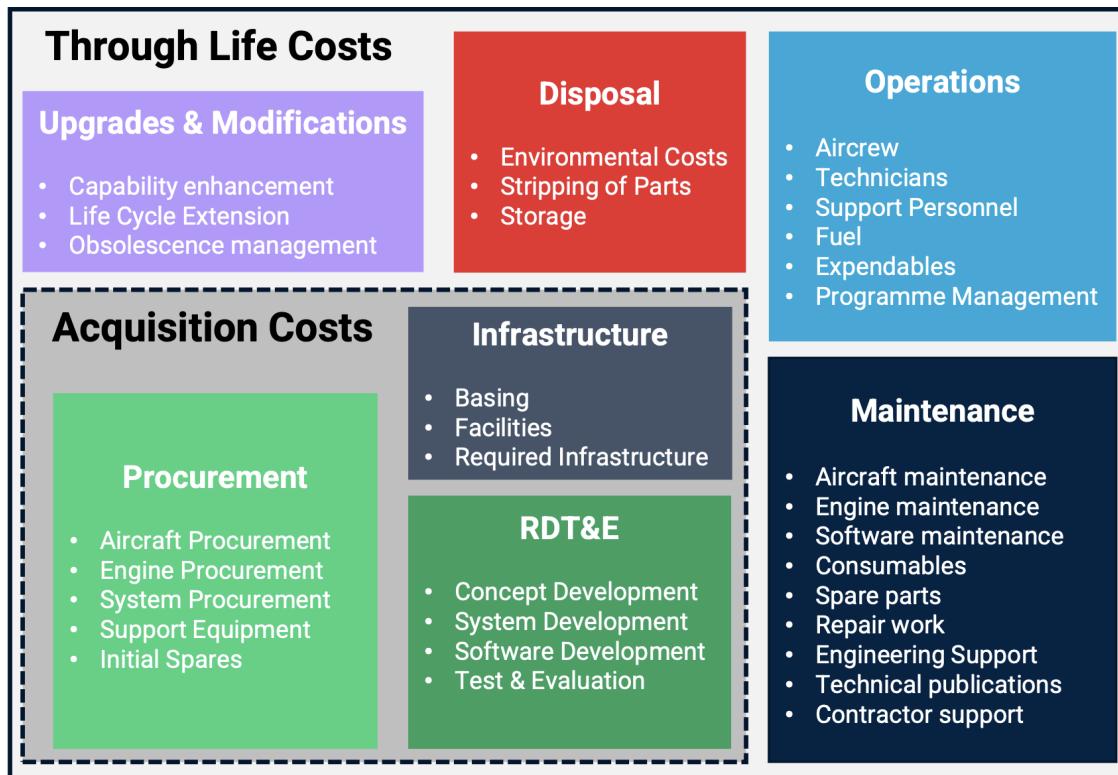


Figura 1.5. Representación visual de los diferentes gastos de un programa militar según su naturaleza (Fuente: Aviation Week [7]).

El artículo anterior explica cómo los gastos asociados al mantenimiento y operaciones constan alrededor del 60 % del total de los programas militares, y cómo estos costes varían según el tipo de aeronave. En aeronaves no tripuladas (Unmanned Aerial Vehicle, UAVs), los costes son alrededor del 58 %, mientras que en aeronaves de ala fija como los cazas convencionales asciende al 64 %. Los costes de mantenimiento son, de media, máximos para helicópteros, rondando el 68 % del coste total.

El estudio se realiza sobre una hipotética flota de 100 aviones de combate iguales, asumiendo una vida media de 37 años por sistema y 200 horas de vuelo al año, para realizar una estimación basada en datos del Departamento de Defensa de los Estados Unidos de América. Además, se calcularon los costes para distintos modelos de aeronaves. De las aeronaves comparadas, la más cara resultó el F35A Lightning II, con unos costes totales del programa de 38,6 mil millones de dólares americanos (38,6 billion USD [7]), de los cuales el 45,2 % corresponden a costes de mantenimiento, para un total de 17 452 millones de dólares, o 29 283 dólares por hora de vuelo. A lo largo de los 37 años del hipotético programa, por tanto, se

gastarían unos 472 millones de euros al año solo en mantener la flota a punto.

En resumen, el estudio muestra el alto coste de los programas militares, resaltando el elevado peso de los costes de mantenimiento durante el ciclo de vida de estas aeronaves. Es importante destacar el alto coste de adquisición de equipos totalmente nuevos, cuando se trata de cazas de combate de última generación, por lo que es indispensable contar con herramientas y programas de mantenimiento eficientes que permitan alargar la vida de los equipos lo máximo posible, siempre dentro de los estándares de seguridad que marcan las regulaciones aeronáuticas.

### 1.3 Evolución de las estrategias de mantenimiento: Correctivo, Preventivo y Predictivo

Desde el inicio de la aviación, las técnicas de mantenimiento han evolucionado en paralelo con el avance del conocimiento, la ingeniería y la tecnología. Históricamente, la industria adoptó inicialmente un enfoque puramente reactivo, al que progresivamente se fueron incorporando estrategias preventivas, posteriormente predictivas y, más recientemente, prescriptivas. Este cambio ha sido impulsado tanto por la complejidad creciente de los sistemas como por la necesidad de optimizar costes, mejorar la disponibilidad y garantizar la seguridad operacional.

Una primera clasificación distingue entre actividades de mantenimiento reactivo (o correctivo) y mantenimiento proactivo, siendo este último el que incluye el mantenimiento preventivo, el basado en condición, el predictivo y el prescriptivo.

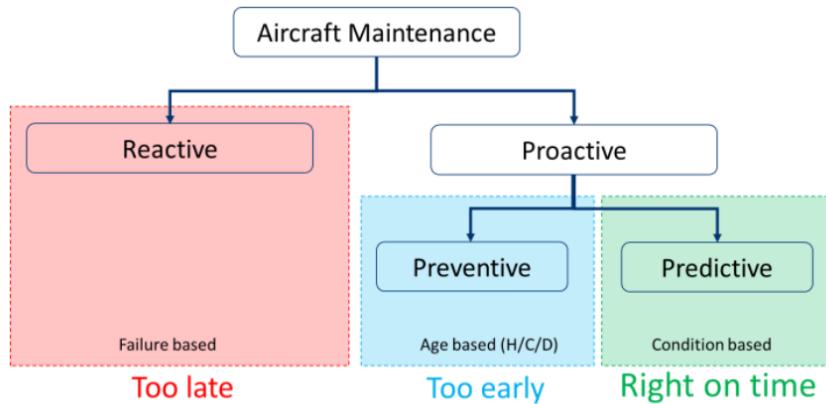


Figura 1.6. Clasificación general de las principales estrategias de mantenimiento aeronáutico (Fuente: [8]).

### **1.3.1 Mantenimiento reactivo**

El mantenimiento reactivo engloba todas aquellas acciones que tienen lugar una vez ha fallado un equipo o sistema. En este caso, el mantenimiento es consecuencia directa del fallo experimentado. Fue la estrategia dominante durante la Revolución Industrial (finales del siglo XVIII y XIX), donde las máquinas, como las de vapor, se reparaban únicamente tras su avería. Este enfoque presenta la ventaja de no incurrir en costes de inspección periódica, pero implica un alto riesgo de paradas imprevistas, costes de reparación elevados y posibles consecuencias de seguridad.

Como es obvio, este tipo de mantenimiento no resulta viable en la aviación, ya que esperar que se produzca una avería para reparar un equipo significaría un riesgo inaceptable para tripulación, pasajeros y otros usuarios de la infraestructura del transporte aéreo. La estrategia reactiva se usó en un marco temporal anterior a la aeronáutica y nunca se aplicó a vehículos aeronáuticos.

### **1.3.2 Mantenimiento preventivo**

La aparición de sistemas más complejos en la década de 1930, como aeronaves monoplano de múltiples motores, y posteriormente, tras la Segunda Guerra Mundial, los motores a reacción presurizado y con equipos electrónicos abundantes, causó la instauración de programas de mantenimiento preventivo basados en revisiones programadas [9]. Bajo este modelo, la fiabilidad se asociaba directamente con la frecuencia del mantenimiento, asumiendo que intervenciones más frecuentes aumentaban la seguridad y la disponibilidad.

Sin embargo, esta filosofía derivó en la realización de intervenciones innecesarias y en un incremento de costes. En la década de 1960, el Maintenance Steering Group (MSG-1), grupo de investigación en materia de mantenimiento, impulsado por la Air Transport Association (ATA), introdujo un enfoque de revisiones sistemáticas inicialmente aplicado al Boeing 747. Este incorporaba principios de ingeniería de seguridad y tareas basadas en la condición de salud de los equipos (lo que se conoce como on-condition maintenance), reduciendo los costes de ciclo de vida. Durante la década de 1970, el MSG-2 añadió la monitorización de condición en aeronaves como el Lockheed L-1011 o el McDonnell Douglas DC-10. Finalmente, en 1978, el informe de Nowlan y Heap [10] sobre Reliability Centered Maintenance (RCM) demostró que la mayoría de fallos en sistemas complejos no siguen un patrón de desgaste, sino que eran de carácter aleatorio.

### **1.3.3 Mantenimiento predictivo**

La evolución tecnológica de los sensores, instrumentación digital y las Tecnologías de la Información y la Comunicación (TICs) en las décadas posteriores dio paso al mantenimiento predictivo (PdM), que combina la monitorización continua del estado de los activos con algoritmos de diagnóstico y prognosis. El objetivo es estimar la vida útil remanente (Remaining Useful Life, RUL) y programar intervenciones justo antes de que ocurra el fallo. La RUL se define como el valor numérico que indica el tiempo restante durante el cual va a poder operar un equipo antes del fallo. En el caso de un equipo aeronáutico, lo más normal es que el RUL venga indicado en horas de vuelo o en ciclos (despegue-terrizaje).

Dentro de este enfoque se distinguen dos niveles:

- Diagnóstico: identificación de fallos incipientes o anomalías mediante el análisis de datos operacionales en tiempo real.
- Predictivo: proyección de la evolución futura del deterioro y predicción del momento más probable de fallo.

El PdM requiere una infraestructura digital avanzada que incluya sensores distribuidos, sistemas de adquisición de datos, algoritmos de aprendizaje automático y elevada capacidad de procesamiento. Su implementación permite una gestión más eficiente del ciclo de vida de los sistemas, reduciendo costes y paradas no planificadas.

La implementación de una estrategia de mantenimiento predictiva se desarrolla siguiendo un conjunto de fases bien definidas que permiten pasar de la simple monitorización de datos a la toma de decisiones fundamentadas [11]. En primer lugar, se lleva a cabo la adquisición de datos, lo que implica la instalación de sensores y sistemas de monitorización capaces de registrar variables relevantes como vibraciones, temperatura, presión o consumo energético. A continuación, se realiza el preprocesamiento de estos datos, etapa en la que se filtra el ruido, se corrijen valores atípicos y se normalizan las señales para garantizar que la información se encuentre en un formato adecuado para el análisis.

La tercera fase corresponde a la extracción de características, en la que se calculan métricas o indicadores representativos del estado del sistema. Aquí debe identificarse qué características son las más significativas en la evolución del sistema, desechariendo aquellas que no aportan demasiada información (como variables constantes a lo largo de todas las mediciones realizadas).

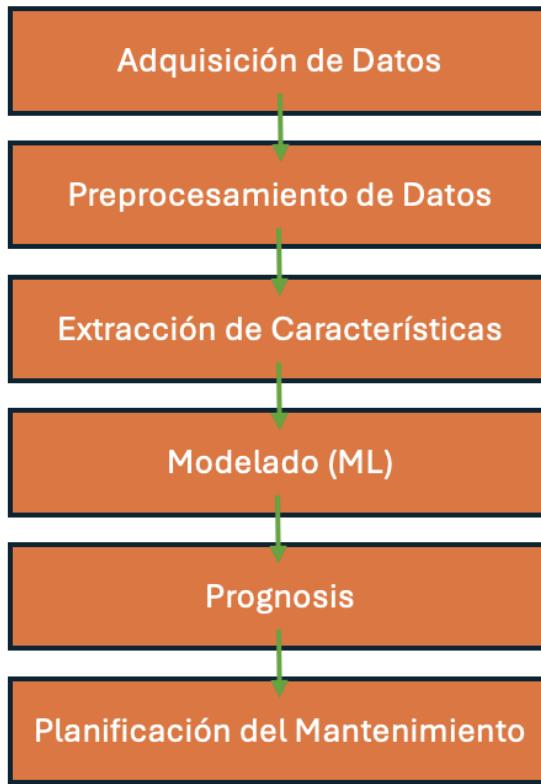


Figura 1.7. Diagrama de flujo con las principales fases del mantenimiento predictivo (Fuente: elaboración propia).

Posteriormente, se desarrolla el modelado, que puede basarse en modelos físicos, estadísticos o en algoritmos de aprendizaje automático entrenados para identificar patrones de degradación y relacionarlos con el comportamiento operativo del activo, como los que se explicarán en la sección Sección 3. El siguiente paso se conoce como prognosis, en el que se estima la vida útil remanente (Remaining Useful Life, RUL) y se predicen posibles fallos con el objetivo de anticiparse a los eventos críticos. Finalmente, se llega a la planificación del mantenimiento, donde se programan intervenciones de forma que se maximice el tiempo de operación del activo y se minimicen las paradas imprevistas.

Entre las principales ventajas del mantenimiento predictivo se encuentra la posibilidad de aumentar la disponibilidad de los equipos al reducir significativamente las paradas no planificadas. La monitorización continua y el análisis avanzado permiten además alargar la vida útil de los componentes, ya que las intervenciones se realizan únicamente cuando existe una evidencia objetiva de deterioro. Esto contribuye a un uso más eficiente de los recursos de mantenimiento y de los repuestos, lo que repercute positivamente en los costes operativos y en la fiabilidad general del sistema.

Por otra parte, la implantación de esta estrategia presenta ciertas limitaciones. La in-

versión inicial en sensores, sistemas de adquisición y procesamiento de datos, así como en la capacitación del personal, puede ser considerable. Además, la fiabilidad de las predicciones depende en gran medida de la calidad y representatividad de los datos, lo que exige un control riguroso de la instrumentación y de las condiciones de operación. La integración de este tipo de sistemas en infraestructuras existentes también puede ser compleja, especialmente en entornos industriales con una gran variabilidad en las condiciones de trabajo, lo que añade un desafío adicional en la implementación práctica [12].

En definitiva, el PdM constituye una estrategia de gran potencial para mejorar la fiabilidad y optimizar costes, aunque su funcionamiento depende de una adecuada calidad de datos, infraestructura tecnológica y capacitación del personal.

#### **1.3.4 Mantenimiento prescriptivo**

Como sección adicional, encontramos una nueva técnica que se conoce como mantenimiento prescriptivo. Esta es una estrategia moderna que integra las capacidades del mantenimiento predictivo con información contextual sobre el entorno operativo, restricciones logísticas y objetivos estratégicos. De este modo, no solo anticipa fallos, sino que también propone acciones óptimas que equilibren disponibilidad, coste, sostenibilidad y seguridad [13].

La evolución completa puede describirse como:

1. **Descriptivo:** análisis retrospectivo de datos históricos.
2. **Diagnóstico:** identificación en tiempo real de causas de fallo.
3. **Predictivo:** estimación de la evolución futura del deterioro.
4. **Prescriptivo:** recomendación activa de acciones considerando múltiples restricciones.

A pesar de sus ventajas potenciales, el mantenimiento prescriptivo se encuentra todavía en una fase de adopción temprana en la mayoría de los sectores industriales. Su implementación suele concentrarse en industrias con activos de alto valor y operaciones críticas, como la aviación, la energía o la manufactura avanzada, donde el retorno de inversión justifica el coste y la complejidad del sistema. Las principales barreras identificadas incluyen la falta de estandarización de datos y la necesidad de una infraestructura digital avanzada. También destaca la complejidad de integración con los sistemas de gestión existentes. No obstante, este concepto ha experimentado un gran impulso en los últimos años, impulsado por el progreso de métodos de análisis avanzados, la inteligencia artificial y el Internet de las Cosas (Internet of Things, IoT), que han facilitado su adopción [14].

## 1.4 Machine Learning en mantenimiento predictivo

Como se ha expuesto en la sección anterior, el mantenimiento predictivo requiere no solo de una infraestructura de monitorización avanzada, sino también de herramientas analíticas capaces de transformar los datos recogidos en información útil para la toma de decisiones. En este contexto, el aprendizaje automático (Machine Learning, ML) se ha consolidado como un elemento clave para extraer patrones, realizar pronósticos precisos y optimizar la planificación de intervenciones. Su capacidad para aprender de datos históricos y adaptarse a condiciones operativas cambiantes lo convierte en un componente esencial de los sistemas modernos de prognosis y gestión de la salud de activos en entornos complejos como la aviación [15].

Esta disciplina es la encargada de desarrollar modelos eficaces que sean capaces de alimentarse de los datos proporcionados por los sensores incorporados a los equipos, para obtener información sobre su estado de salud y estimar la necesidad de mantenimiento.

### 1.4.1 Fundamentos del Machine Learning

El aprendizaje automático o Machine Learning es un subcampo de la inteligencia artificial que se centra en el desarrollo de algoritmos capaces de mejorar su rendimiento en una tarea determinada a partir de la experiencia. Una definición ampliamente aceptada es la propuesta en 1997 por Tom M. Mitchell, quien establece lo siguiente:

Un programa de computadora aprende de la experiencia  $E$  con respecto a una clase de tareas  $T$  y una medida de rendimiento  $P$ , si su rendimiento en las tareas de  $T$ , medido por  $P$ , mejora con la experiencia  $E$ . [16]

De forma más reciente, el propio Mitchell junto con M. Jordan describen el aprendizaje automático como un conjunto de métodos que utilizan la experiencia, normalmente en forma de datos, para mejorar el rendimiento predictivo o la toma de decisiones en un amplio rango de dominios [17]. El objetivo fundamental del ML es entrenar a sistemas informáticos para que puedan "aprender" sobre una serie de inputs, y poder llegar a un resultado sin estar explícitamente programados para ello [18]. En este contexto, se utilizan datos existentes o datasets para que los algoritmos puedan predecir resultados futuros en base a estos datos históricos.

Aunque los términos inteligencia artificial (IA) y aprendizaje automático (Machine Learning, ML) suelen utilizarse de forma intercambiable, no son equivalentes. La IA engloba un conjunto amplio de técnicas que permiten a las máquinas imitar las capacidades humanas para resolver tareas complejas, interpretar lenguaje natural y tomar decisiones en entornos inciertos [19]. El ML constituye un subcampo específico de la IA centrado en el desar-

rollo de algoritmos que mejoran su rendimiento a partir de datos, sin estar programados explícitamente para cada tarea.

Además, es común encontrar también el término Aprendizaje Profundo o Deep Learning (DL). Dentro del ML, el Deep Learning representa un enfoque avanzado basado en redes neuronales profundas, capaces de aprender representaciones jerárquicas de los datos y procesar información compleja como imágenes, audio o texto de manera altamente eficiente [20]. Las redes neuronales, como se explicará más adelante, consisten en la conexión entre múltiples nodos denominados neuronas, que transforman entradas (inputs) en salidas (outputs) mediante operaciones matemáticas sencillas. Este tipo de algoritmos reciben el nombre de Red Neuronal Artificial (Artificial Neural Network, ANN), ya que tratan de representar el funcionamiento del cerebro humano [21]. Las ANNs serán definidas en mayor detalle en la Sección 3.

Concepto	Definición
Inteligencia Artificial (IA)	Conjunto amplio de técnicas que permiten a máquinas imitar o superar capacidades humanas, como la toma de decisiones, el razonamiento o la interpretación del lenguaje natural. <b>Ejemplos:</b> sistemas expertos, procesamiento de lenguaje natural, planificación autónoma.
Machine Learning (ML)	Subcampo de la IA que desarrolla algoritmos capaces de aprender de datos y mejorar su rendimiento en tareas específicas sin estar programados explícitamente para cada caso. <b>Ejemplos:</b> clasificación de fallos, predicción de demanda, detección de anomalías.
Deep Learning (DL)	Subcampo del ML basado en redes neuronales profundas (DNN), capaces de aprender representaciones jerárquicas y extraer características automáticamente de datos complejos. <b>Ejemplos:</b> reconocimiento de imágenes, procesamiento de audio, traducción automática.

Tabla 1.1. RESUMEN DE DIFERENCIAS ENTRE IA, ML Y DL

Cuando una red neuronal artificial (ANN) contiene una gran cantidad de capas ocultas, se denomina red neuronal profunda (Deep Neural Network, DNN). El campo del aprendizaje profundo estudia las DNN y, en general, cualquier modelo que incorpore capas profundas de operaciones; sin embargo, en el uso cotidiano, el término Deep Learning se aplica con frecuencia incluso a redes neuronales poco profundas [22].

Después de ver las diferencias entre inteligencia artificial, aprendizaje automático y aprendizaje profundo, es útil explicar cómo cambia el procedimiento para construir un modelo

según el enfoque que se use. En mantenimiento predictivo, esta elección influye en cuánto trabajo manual se necesita para preparar los datos y en la capacidad del modelo para aprender directamente de la información disponible. La Figura Figura 1.8 muestra de forma visual estas diferencias y ayudará a entender mejor los métodos que se presentarán más adelante.

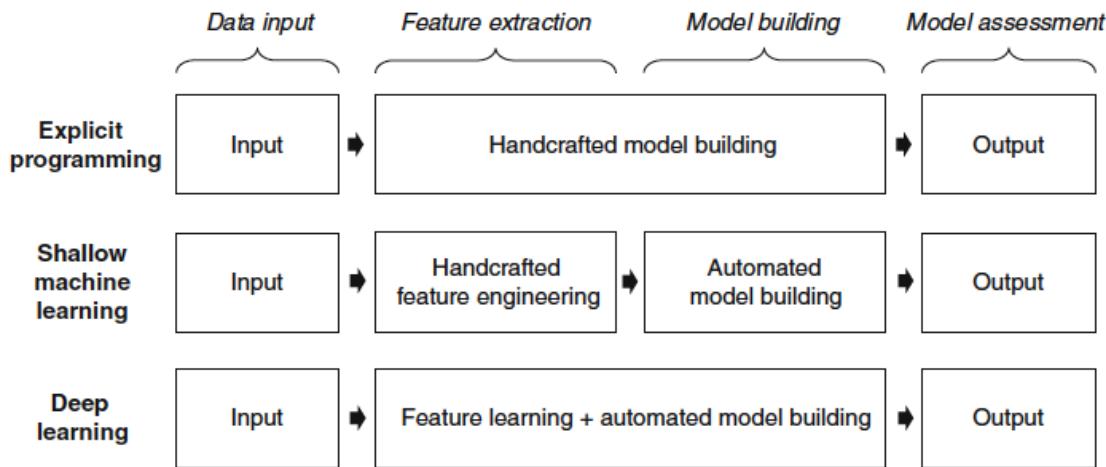


Figura 1.8. Comparación de procesos de modelado: programación explícita, ML superficial y Deep Learning (Fuente: [23]).

Aquí se observa que todos los enfoques comparten un esquema general de datos de entrada y salida, pero difieren en la forma en que procesan y transforman esos datos:

- Programación explícita: el modelo es completamente definido por expertos. Tanto la extracción de características como la construcción del modelo se realizan manualmente.
- Aprendizaje automático superficial (shallow ML): requiere ingeniería de características manual, que consiste en seleccionar o crear las variables más relevantes a partir de los datos disponibles. La construcción del modelo se automatiza mediante algoritmos de ML, que se describirán más adelante.
- Deep Learning: automatiza tanto la extracción de características como el modelado. Las redes neuronales profundas pueden aprender representaciones jerárquicas directamente a partir de datos crudos como imágenes o texto, sin intervención manual [23].

En cuanto a su relación con el mantenimiento predictivo, el aprendizaje automático ha mostrado un auge en las últimas décadas debido a diferentes factores que han facilitado su implementación en la industria [24]. Entre los principales catalizadores se encuentran:

- Expansión del IoT para recopilar grandes cantidades de datos: el desarrollo de sensores novedosos y su uso a gran escala en sistemas complejos facilita la recolección masiva

de datos sobre estos sistemas. En el contexto del mantenimiento predictivo, contar con sets de datos extensos es clave para la implementación y el entrenamiento de algoritmos de predicción eficientes y precisos.

- Incremento del poder computacional: con las recientes mejoras en procesadores de los equipos que se usan para realizar las simulaciones, es posible ejecutar algoritmos complejos que procesen grandes cantidades de datos en el menor tiempo posible, reduciendo el tiempo de entrenamiento y facilitando una más rápida visualización de los resultados.
- Nuevas herramientas de almacenamiento y procesamiento de datos: haciendo uso de la computación en la nube, los usuarios pueden almacenar, procesar y analizar estos datos en tiempo real.
- Acceso a las herramientas: la gran cantidad de herramientas disponibles y sobre todo de código abierto como las principales librerías de Python (scikit-learn, numpy, etc.) favorecen la aplicación del ML en empresas de cualquier tipo.
- Exigencia creciente de eficiencia operativa en la industria: la mayor competitividad entre empresas y la exigente demanda de los sectores que requieren mantenimiento constante ha favorecido la búsqueda de técnicas que aumenten la vida útil de los equipos, reduciendo costes y parando los equipos cuando es estrictamente necesario, además del menor tiempo posible. En este ámbito el Machine Learning favorece todos estos requisitos de mantenimiento, convirtiéndose en una herramienta clave para cumplirlos.

#### **1.4.2 Aplicaciones actuales en la industria aeronáutica**

La aplicación de técnicas de Machine Learning en la industria aeronáutica abarca un amplio espectro de casos, desde la mejora de la seguridad operacional hasta la optimización del mantenimiento. Un ejemplo destacado fuera del ámbito del mantenimiento es el desarrollado por el proyecto europeo AIDA (Ash Ingestion Detection for Aircraft) [25], que creó un sistema autónomo para la detección en tiempo real de ceniza volcánica en aeronaves. Utilizando un láser pulsado de luz polarizada y técnicas ópticas avanzadas, el sistema es capaz de diferenciar partículas de ceniza de otras usualmente encontradas en el ambiente, como arena o polvo, con una elevada precisión de detección. Esta capacidad de discriminación es fundamental para prevenir daños en motores y sistemas neumáticos derivados de la ingesta de ceniza, un riesgo crítico en zonas afectadas por actividad volcánica. La implementación de este tipo de soluciones basadas en aprendizaje automático demuestra cómo el análisis avanzado de datos puede mejorar la toma de decisiones en vuelo y la seguridad de la operación.

La misma lógica de extracción de valor a partir de datos complejos puede aplicarse al mantenimiento predictivo, como demuestra el caso de KLM Royal Dutch Airlines en colaboración con la Universidad Técnica de Delft. En este proyecto se desarrollaron modelos de DL

para anticipar fallos en las válvulas de aire de sangrado de los Boeing 747, un componente esencial para el sistema de presurización y climatización de cabina. Mediante el análisis de datos históricos de sensores y registros de mantenimiento, los algoritmos identificaron patrones asociados a fallos inminentes, permitiendo planificar el reemplazo de las válvulas antes de que se produjera una avería en servicio [26]. Este tipo de aplicación muestra el potencial del Machine Learning para optimizar la disponibilidad de la flota, reducir costes de mantenimiento y minimizar interrupciones operativas.

Como se ha explicado anteriormente, para la implementación de procesos de mantenimiento predictivo en la aeronáutica que incluyan ML, se requieren sistemas complejos de recolección de datos, que cuenten con sensores que extraigan los datos de un sistema. Como se ve en la figura Figura 1.9, estos datos se utilizan para estimar el estado del sistema mediante el modelo de Machine Learning, que además debe introducir simulaciones de cargas (basadas también en estimaciones a partir de lecturas de sensores). Este conjunto formado por la recolección de datos, modelo de ML y su posterior aplicación real, muestran el marco de trabajo de cualquier sistema de mantenimiento basado en prognosis.

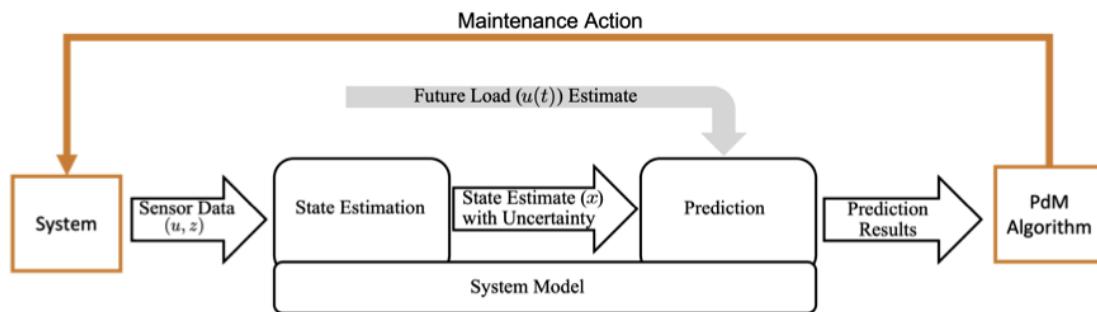


Figura 1.9. Resumen esquemático del proceso de mantenimiento predictivo (Fuente: [14]).

A pesar del creciente volumen de datos disponible gracias a la implantación de la industria 4.0 en el sector aeronáutico, uno de los principales retos para el desarrollo de técnicas de mantenimiento predictivo es la escasa disponibilidad de conjuntos de datos reales. Operadores y fabricantes suelen mantener esta información bajo estrictas medidas de confidencialidad, ya que puede revelar detalles sensibles sobre el funcionamiento interno y la degradación progresiva de los sistemas. Por motivos de seguridad y competitividad, los datos operacionales completos rara vez se comparten de forma pública, lo que limita la investigación y validación de modelos en escenarios reales.

Como alternativa, se han desarrollado entornos de simulación que permiten generar datos sintéticos para investigación. Entre ellos destaca el generador de datos CMAPSS (Commercial Modular Aero-Propulsion System Simulation) de la NASA [27]. Este entorno simula el funcionamiento de motores turbofán bajo diferentes perfiles operativos y condiciones de degradación, generando series temporales de variables como temperaturas y presiones en dis-

tintas estaciones del motor. Cada registro incluye un número de ciclo asociado al instante de medición, así como la vida útil remanente, indicada como el número de ciclos restantes a los que se puede someter el motor antes del fallo.

El carácter sintético de CMAPSS permite que sus datos se distribuyan abiertamente, fomentando el desarrollo y la comparación de algoritmos en la comunidad científica. De hecho, la Prognostics and Health Management (PHM) Society organiza anualmente una competición centrada en la predicción del RUL a partir de este conjunto de datos, introduciendo variaciones en cada edición para evaluar modelos en diferentes condiciones y contextos. Este enfoque ha convertido este set de datos en uno de los recursos más utilizados para la investigación en mantenimiento predictivo aeroespacial.

Además de la limitación que supone la escasez de datos operacionales reales, el trabajo de Teubert para la NASA [14] menciona otros obstáculos que dificultan la aplicación de estas técnicas en aviación. Por un lado, se destaca la dificultad de conseguir que los modelos mantengan un rendimiento fiable en entornos tan variables y exigentes como los que se dan en operación real. A esto se suma el reto de cumplir con los procesos de certificación y validación exigidos por las normativas de seguridad, que pueden retrasar considerablemente su entrada en servicio. Por último, la ausencia de criterios estandarizados y métricas unificadas que permitan cuantificar con precisión el beneficio económico y operativo de estas soluciones complica la justificación de las inversiones necesarias para su adopción.

## **1.5 Estado del arte: ML y PdM**

Hasta ahora se han recorrido los principales aspectos de las dos disciplinas que marcan las bases de este proyecto. En esta sección se presentan avances técnicos notables y varias publicaciones recientes tanto del ámbito del mantenimiento predictivo como de modelos de aprendizaje automático novedosos y la aplicación conjunta de ambos. Se abordan, entre otros, el concepto de Digital Twin (DTw), así como modificaciones disruptivas de algoritmos basados en redes neuronales que han obtenido una gran popularidad en los últimos años. En los últimos años, el uso de técnicas digitales avanzadas en el sector aeronáutico ha crecido de manera notable, para la optimización de procesos y para la gestión del ciclo de vida de las aeronaves. Entre estas tecnologías destacan los Digital Twins (DTw), que permiten reproducir virtualmente el comportamiento de un sistema físico, y las novedosas soluciones de aprendizaje automático aplicadas al mantenimiento predictivo. A continuación se presentan algunas líneas de trabajo recientes que ilustran el avance de estas metodologías.

## Gemelos digitales en aviación

Un gemelo digital es una representación virtual de un sistema físico que combina modelos matemáticos, datos de sensores y herramientas de análisis para reproducir y predecir su comportamiento en diferentes condiciones. En la industria aeronáutica este enfoque posibilita un seguimiento continuo del estado de componentes y sistemas, permitiendo la detección temprana de desviaciones respecto al comportamiento esperado. Estudios como el de Xiong y Wang (2022) [28] identifican cómo tecnologías clave la fusión de datos, la integración con sistemas ciberfísicos y el uso de plataformas IoT, con aplicaciones que abarcan desde la fase de diseño, fabricación y operaciones. Esta tecnología también es aplicable al mantenimiento predictivo, ya que los modelos virtuales pueden actualizarse de forma automática a medida que reciben nuevos datos, facilitando la evaluación de salud y prognosis en tiempo real .

## Estandarización de DTw para mantenimiento predictivo

En otro estudio realizado por Ma, Flanigan y Begés (2024) se analizan los retos actuales en el uso de DTw para mantenimiento predictivo. Como se ha mencionado en la sección anterior, la ausencia de marcos estandarizados que guíen el desarrollo e implementación de este tipo de tecnologías tan recientes supone un gran obstáculo a la adopción de las mismas. Investigaciones recientes han propuesto hojas de ruta basadas en requisitos funcionales y de información. Estas analizan casos existentes y señalan carencias en aspectos como la escalabilidad, la interpretabilidad y la capacidad de generalización de los modelos [29]. Este tipo de guías resulta fundamental de cara a la adopción consistente de los gemelos digitales en entornos operativos reales.

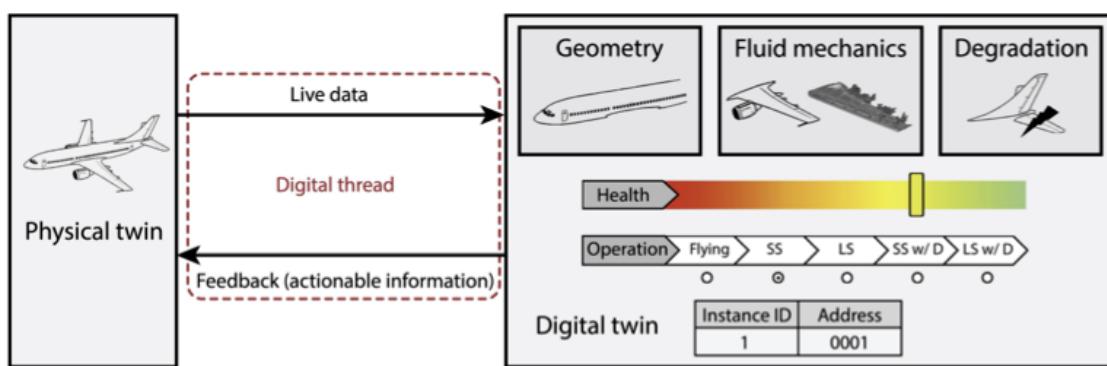


Figura 1.10. Ejemplo simplificado de Gemelo Digital de un avión comercial (Fuente: [29]).

## Modelos ML y DL aplicados a motores aeronáuticos

Un estudio reciente realizado por Hermawan et al. (2020) [30] presenta un enfoque avanzado de mantenimiento predictivo aplicado a motores aeronáuticos mediante técnicas de Deep Learning. El objetivo principal del trabajo es estimar con precisión la vida útil remanente de los motores, una tarea crítica debido a la baja tolerancia a fallos de estos sistemas. En este artículo se propone una arquitectura híbrida que combina redes neuronales convolucionales (Convolutional Neural Network, CNN) y redes neuronales con memoria temporal (Long Short Term Memory, LSTM), que los autores denominan CLSTM, con el fin de aprovechar las capacidades de las CNN para extraer características relevantes de los datos de entrada y de las LSTM para manejar datos temporales. Los resultados experimentales muestran que esta arquitectura híbrida mejora significativamente la exactitud en comparación con métodos tradicionales. En la evaluación de rendimiento del modelo se indica:

“...the proposed algorithm achieved 99.60 % accuracy, while the CNN and LSTM achieved 89.33% and 97.32%, respectively.” [30]

Este enfoque demuestra el potencial de las redes neuronales profundas para prever fallos de motores y optimizar la planificación del mantenimiento, contribuyendo a una mayor seguridad y eficiencia operativa en el sector aeronáutico.

Otros artículos aún más recientes como el de Hasib et al. (2023) evalúan y comparan distintos algoritmos de Machine Learning y Deep Learning para la predicción de fallos en motores de avión, siguiendo los pasos de Hermawan. Este estudio destaca cómo las redes neuronales recurrentes profundas muestran un rendimiento particularmente alto en el tratamiento de datos temporales secuenciales y en los obtenidos por sensores de motor. En este contexto, se vuelven a usar redes LSTM para retener información temporal. Además, se emplean otras estructuras de redes neuronales como LSTM bidireccionales (Bi-LSTM), que procesan la secuencia tanto en sentido directo como inverso, capturando dependencias pasadas y futuras de los datos. Otras metodologías propuestas incluyen Gated Recurrent Units (GRU) simplifican la estructura de las LSTM, utilizando menos parámetros y facilitando un entrenamiento más rápido. En este estudio, las arquitecturas basadas en estas redes superaron con frecuencia el 95% de precisión en la estimación de la vida útil remanente. Además, se identificó un creciente interés por integrar técnicas de interpretabilidad como LIME, que permiten comprender mejor las decisiones del modelo y facilitan su validación en entornos operativos [31].

Cabe destacar que ambos proyectos de investigación utilizan el mismo set de datos de la NASA CMAPSS que se describió anteriormente. Esto refuerza la capacidad de generación de datos sintéticos realistas del simulador de la NASA para este tipo de fines académicos. Posteriormente, en la Sección 4 se presentará también un caso práctico utilizando esta colección de datos.

## Caso real en mantenimiento predictivo de compresores industriales

Un estudio reciente de Aminzadeh et al. (2025) detalla en un entorno industrial cómo se integran tecnologías de adquisición avanzada de datos, IoT y algoritmos de aprendizaje automático para mejorar los procesos de mantenimiento predictivo de compresores de aire. El sistema combina sensores, procesamiento en la nube y bases de datos estructuradas para monitorear parámetros críticos como presión, temperatura y vibración. Se implementó un modelo que, mediante inferencia automatizada, reduce la intervención humana y minimiza errores operativos al anticipar posibles fallos. Esta solución demuestra aplicabilidad en unidades industriales pequeñas, siendo implementable como software automatizado de fácil uso en contextos reales [32].

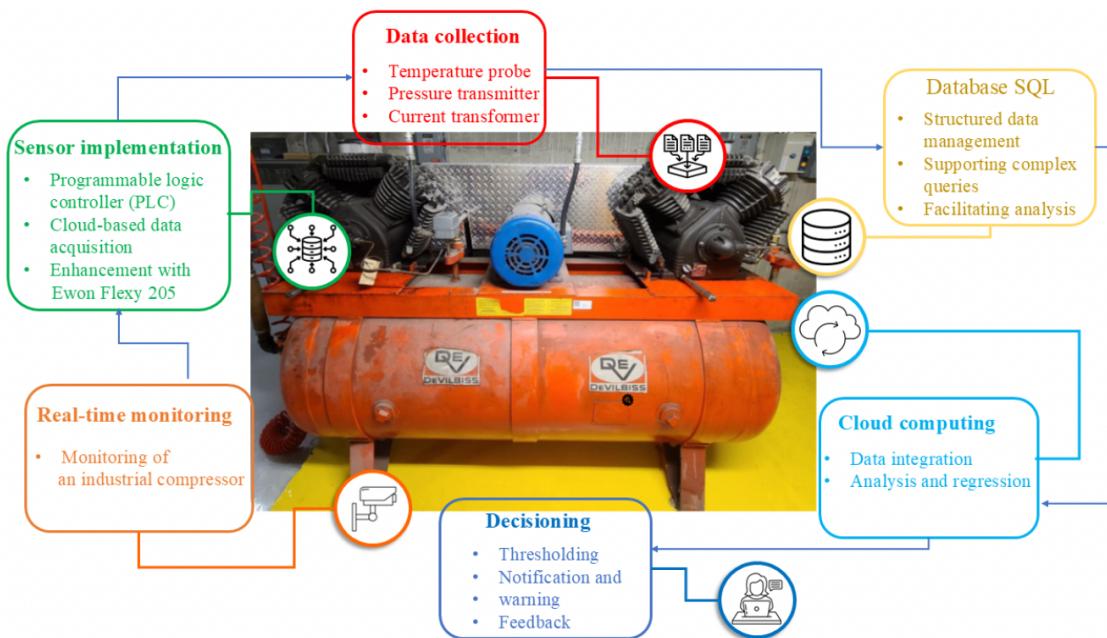


Figura 1.11. Equipo físico y metodología propuesta por Aminzadeh et al. en el análisis de compresores de aire (Fuente: [32]).

## 2 OBJETIVOS Y MOTIVACIÓN

### 2.1 Objetivos del proyecto

El propósito de este Trabajo de Fin de Máster (TFM) es explicar de forma concisa y completa un proyecto de Machine Learning aplicado al mantenimiento predictivo. Para ello, se debe en primer lugar presentar un contexto detallado de las disciplinas que componen esta iniciativa. Para que el lector adquiera una base sólida de la temática. Se recorrerán también los principales algoritmos de aprendizaje automático utilizados en la industria y se describirá una metodología general que debe seguirse para llevar a cabo un proyecto de estas características.

En cuanto a la parte práctica, la principal meta de esta tesis consiste en aplicar estos conocimientos teóricos a varios artículos científicos, intentando replicar los procedimientos de la bibliografía para demostrar su aplicabilidad y su fiabilidad. Se explicarán en detalle los sets de datos seleccionados y su relación con los modelos a utilizar, ya que ambos deben ser escogidos cuidadosamente para la obtención de soluciones coherentes. Finalmente, se analizarán los resultados obtenidos mediante una serie de métricas de rendimiento y la comparación y validación con los estudios originales. Para cada artículo replicado se pretende realizar algún tipo de aportación propia, explicando los procedimientos aplicados con fuentes bibliográficas que justifiquen su elección.

Por otra parte, se pretende proporcionar una revisión bibliográfica extensa, relativa a este tipo de algoritmos de ML, explorando los avances más destacados y relevantes a nivel de investigación y aplicaciones en la industria. Además, se desarrollará extensamente sobre la importancia del mantenimiento en la Industria Aeroespacial, centrándonos en el mantenimiento predictivo, explicando sus bases, tipos, ventajas e inconvenientes y su uso junto a otro tipo de tecnologías, como los Gemelos Digitales.

### 2.2 Justificación y relevancia del estudio

La elección de la temática del proyecto se basó en la elección disciplinas de gran relevancia en la actualidad de la industria aeronáutica y que fueran a su vez de interés para el autor. En este aspecto, las técnicas asociadas tanto al Mantenimiento Predictivo como al Aprendizaje Automático han experimentado un crecimiento exponencial en las últimas décadas.

La inclinación del autor por estos dos temas es existente desde hace ya varios años, como se refleja en la redacción de su Trabajo de Fin de Grado *Machine Learning Approaches to Solve Boundary Layer Problems* [33] (Métodos de Aprendizaje Automático para Resolver Problemas de Capa Límite), desarrollado y redactado en 2023 en la Universidad Carlos III de Madrid. La continuación de esta temática viene motivada por los mediáticos y tan notorios

avances recientes de los últimos dos años en el ámbito de la Inteligencia Artificial.

La parte más aeronáutica del TFM trata el mantenimiento predictivo como campo fundamental para las principales compañías del sector, por los diferentes motivos explicados en la Sección 1.1. Las empresas más grandes de la industria cuentan ya con procesos e iniciativas consolidadas relacionadas con el PdM [34], lo que refuerza la relevancia de este proyecto final.

En cuanto a la parte relacionada con el aprendizaje automático, es evidente que esta es más ajena al contenido habitual de un máster en Ingeniería Aeronáutica. No obstante, este trabajo expone la necesidad de aplicación de algoritmos de ML en las principales disciplinas aeronáuticas, y su influencia en la industria y en el mercado laboral en el presente y en los próximos años.

### **2.3 Estructura del proyecto**

En la sección anterior se ha presentado el contexto del proyecto, introduciendo las claves del PdM y del ML. Se habla también del nivel de implementación en la industria de ambas disciplinas, culminando con una revisión bibliográfica sobre el Estado del Arte.

A continuación, la Sección 3 explica detalladamente los algoritmos más comunes del ámbito del aprendizaje automático. Este capítulo cubre aprendizaje supervisado, no supervisado y redes neuronales. Además, se resumen también errores típicos en el entrenamiento de un modelo y las principales métricas de rendimiento que se emplearán posteriormente para evaluar nuestros resultados.

Posteriormente se replican dos artículos científicos en la Sección 4. Este apartado cuenta con dos subsecciones diferenciadas, en las que se desarrolla tanto metodología y resultados para cada uno de los estudios. En primer lugar, la Sección 4.1 expone un estudio de clasificación binaria, basado en aprendizaje supervisado. Este estudio, basado en la publicación *Selecting an appropriate supervised machine learning algorithm for predictive maintenance* de A. Oudah [35] compara el uso de distintos modelos para clasificar señales de vibración. Seguidamente, en la Sección 4.2 se analiza un estudio muy reciente de T. Lodygowski y S. Szrama: *Unsupervised Classification and Remaining Useful Life Prediction for Turbofan Engines Using Autoencoders and Gaussian Mixture Models: A Comprehensive Framework for Predictive Maintenance* [36]. Este artículo, publicado en 2025, aplica mecanismos de clasificación a una serie de datos no etiquetados pertenecientes a datos simulados de un motor turbofán procedentes del CMAPSS (comentado previamente en la Sección 1.4).

Finalmente, la Sección 5 cierra el TFM con un comentario general en forma de conclusiones sobre los resultados obtenidos. Se hace también balance de los objetivos alcanzados, y se listan posibles mejoras a ser implementadas en potenciales futuros trabajos.

### 3 Modelos de Machine Learning para mantenimiento predictivo

Como se ha visto en la Sección 1.4, el mantenimiento predictivo es posible gracias a la combinación de conocimientos sobre los equipos y la lectura y análisis de grandes cantidades de datos mediante técnicas de Machine Learning. La combinación de estas disciplinas permite anticipar fallos y optimizar la disponibilidad de sistemas críticos.

Los diferentes algoritmos de aprendizaje automático permiten construir modelos capaces de aprender patrones a partir de datos históricos y de operación, sin necesidad de definir de forma explícita todas las reglas del sistema. Dependiendo de la naturaleza del problema y de los datos disponibles, los algoritmos se pueden clasificar en distintos grupos.

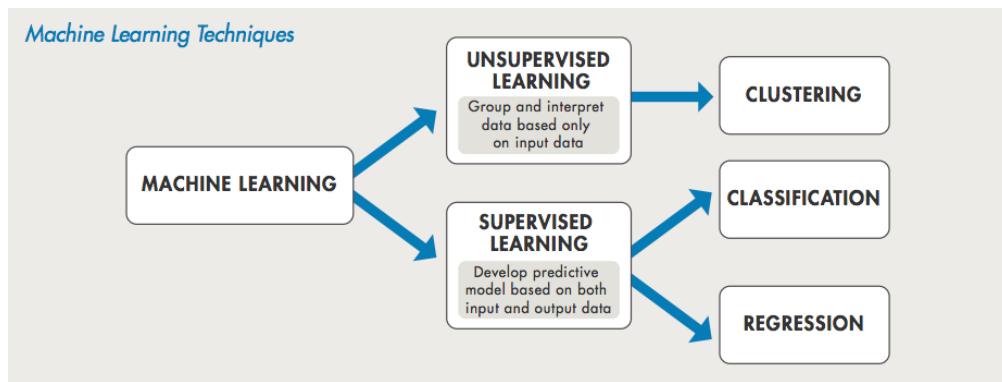


Figura 3.1. Clasificación general de algoritmos de ML (Fuente: Analytics Yogi [37]).

En el aprendizaje supervisado, el modelo se entrena con datos etiquetados para predecir variables de interés, como el estado de salud de un componente o su vida útil remanente. El aprendizaje no supervisado, por su parte, identifica estructuras y relaciones ocultas en datos sin etiquetar, como la detección de grupos de comportamiento o patrones anómalos. Finalmente, las redes neuronales ofrecen un marco flexible y potente para capturar relaciones no lineales y de alta complejidad, siendo capaces de aproximar funciones y reconocer patrones en entornos donde otros métodos resultan insuficientes [38]. La combinación de estos enfoques permite desarrollar sistemas de prognosis más robustos, adaptables y precisos, aumentando la fiabilidad y seguridad en el ámbito aeronáutico.

En esta sección se describen los diferentes tipos de algoritmos de ML usados en la industria, con ejemplos de aplicación y diagramas detallados. Además, se explican varios algoritmos de optimización y de simplificación de problemas complejos, que componen la base de los casos prácticos que se realizarán en la Sección 4.

### 3.1 Modelos supervisados (Random Forest, SVM, Redes Neuronales, etc.)

Los modelos de aprendizaje supervisado son algoritmos que aprenden a partir de un conjunto de datos etiquetados. Por un lado se tienen unos datos de entrada, de los que se alimenta el modelo. Estos deben convertirse en una serie de datos de salida, que son aquellos que se quiere que el modelo aprenda a predecir [39]. Por tanto, la base de los modelos supervisados es obtener una función que relacione inputs y outputs, logrando así predecir los outputs de nuevos datos.

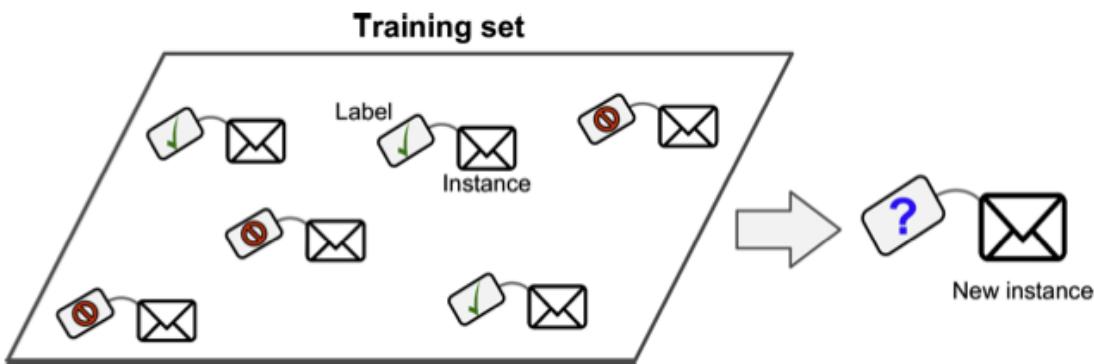


Figura 3.2. Ejemplo de datos supervisados (Fuente: [22]).

Un ejemplo sencillo es el que usa A. Géron, que se muestra en la figura Figura 3.2. El diagrama muestra una serie de correos electrónicos previamente clasificados con un indicador al que se le llama *label*. En este caso, este indicador muestra si el correo es spam o no. Esta clasificación se da en función de distintos parámetros o propiedades de los correos electrónicos que se conocen como *features* o características. Estos podrían ser el remitente, el asunto, determinadas frases del contenido o enlaces que se identifiquen como fraudulentos. A partir de estos datos, el modelo debe ser capaz de estimar con una determinada probabilidad si un nuevo correo es o no es spam. Este tipo de datos con sus etiquetas correspondientes suelen usarse cuando se tienen bases de datos ya etiquetadas, por ejemplo, con registros de fallos en equipos documentados por grandes fabricantes o compañías de mantenimiento. Como expone A. Baradan en su artículo dedicado al mantenimiento predictivo de motores eléctricos, los datos fueron tomados de la AMA Industrial Company, un conocido fabricante de utensilios de soldado [40]. El conjunto de datos incluía variables del motor (temperaturas, intensidades, corrientes, etc.), acompañados de un indicador de salud que clasificaba el estado del motor como saludable, experimentando un fallo crítico o en un condición intermedia que requiere un mantenimiento.

Estos modelos realizan dos tareas fundamentales, la clasificación y la regresión. A continuación se definen los principales algoritmos para modelos supervisados.

### 3.1.1 Regresión lineal

La regresión lineal es uno de los métodos más comunes para estimar el valor de una variable dependiente a partir de una o varias variables independientes. Si solo hay una variable explicativa se habla de regresión lineal simple, y cuando hay varias, de regresión lineal múltiple. Este tipo de algoritmo funciona mejor cuanto mayor es la relación lineal entre variables [41].

El método se basa en construir una recta de mejor ajuste que minimiza la suma de los errores cuadráticos entre los valores observados  $y_i$  y los estimados  $\hat{y}_i$ :

$$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (3.1)$$

donde  $SSE$  es la suma de los errores cuadráticos, y  $(y_i - \hat{y}_i)^2$  representa el error individual para cada observación.

En la regresión lineal simple, la ecuación es:

$$\hat{y} = \alpha + \beta x \quad (3.2)$$

donde  $\alpha$  es el intercepto y  $\beta$  la pendiente que mide el cambio medio en  $\hat{y}$  por unidad de cambio en  $x$ . Para la regresión múltiple, la ecuación se generaliza a:

$$\hat{y}_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_k x_{ki} \quad (3.3)$$

donde cada coeficiente  $\beta_j$  indica la dirección y magnitud de la relación entre la variable  $x_j$  y la respuesta:  $\beta_j > 0$  implica correlación positiva,  $\beta_j < 0$  negativa y  $\beta_j = 0$  ausencia de correlación. El subíndice  $k$  indica el número de variables explicativas.

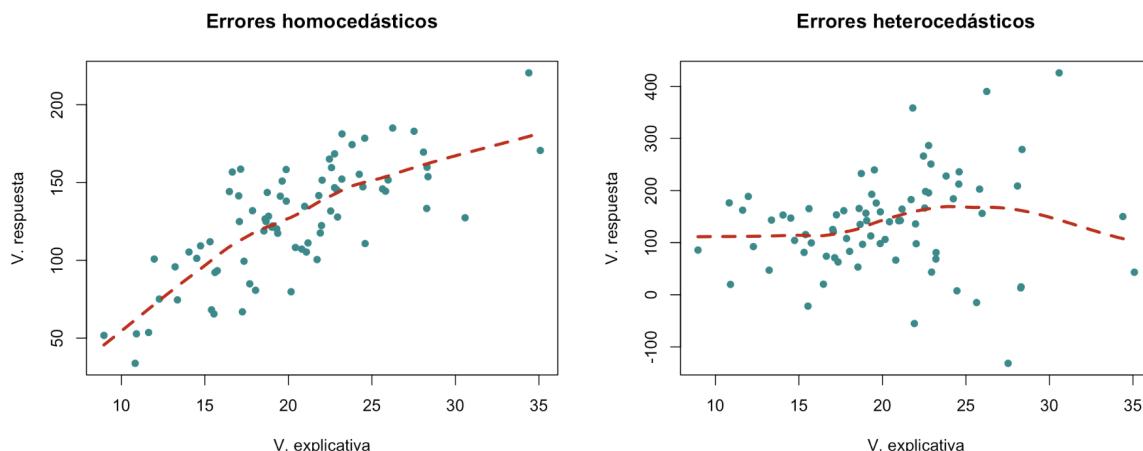


Figura 3.3. Errores homocedásticos y no homocedásticos (Fuente: [42]).

Para que el modelo sea válido, se asumen las siguientes hipótesis [43]:

- Linealidad: debe existir una relación aproximadamente lineal entre las variables explicativas y la respuesta.
- Normalidad de los errores: los residuos deberían seguir una distribución aproximadamente normal, con la mayoría cercanos a cero y pocos valores extremos (outliers).
- Homocedasticidad: la varianza de los errores debe ser constante a lo largo del rango de predicción. Esta se verifica observando que la dispersión de los residuos no aumente ni disminuya sistemáticamente con los valores predichos, como se muestra en la Figura 3.3.

### 3.1.2 Regresión Logística

A diferencia de la regresión lineal, que estima valores continuos, la regresión logística se utiliza para tareas de clasificación [44].

La regresión logística calcula la probabilidad de pertenencia a una clase de un punto de datos en un entorno etiquetado. El funcionamiento de este algoritmo se basa en combinar linealmente un vector de características  $\mathbf{x}$  con una serie de pesos  $\mathbf{w}$  y un factor de sesgo  $b$  siguiendo la formulación  $z = \mathbf{w} \cdot \mathbf{x} + b$ . De esta manera, el algoritmo trata de asignar el valor de los pesos, que pueden ser positivos y negativos, para dar mayor o menor importancia a cada atributo según su influencia en el resultado final. Una vez se ha obtenido el valor  $z$ , este se pasa por una función sigmoide:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (3.4)$$

que mapea cualquier valor real a un valor entre cero y uno. El valor de salida de la función sigmoide representa la probabilidad de que el punto evaluado pertenezca a una clase. Esta función es muy útil ya que es de carácter no lineal y su elevada pendiente en torno a  $x = 0$  la hace especialmente adecuada para tareas de clasificación. Además es una función continua y diferenciable, lo que la convierte en una función óptima para su uso en redes neuronales **Han1995Sigmoid**.

Si se considera una clase  $y$  binaria (puede ser igual a cero o a uno), entonces se calcula  $P(y = 1|x)$ . Por tanto, existe un valor límite o frontera de decisión, correspondiente al valor de probabilidad límite para considerar que un punto pertenece a la categoría 0 o a la categoría 1 [45]. Normalmente se fija un valor límite de 0.5, de forma que se obtiene:

$$\text{decisión}(x) = \begin{cases} 1, & \text{si } P(y = 1|x) > 0.5 \\ 0, & \text{en cualquier otro caso} \end{cases} \quad (3.5)$$

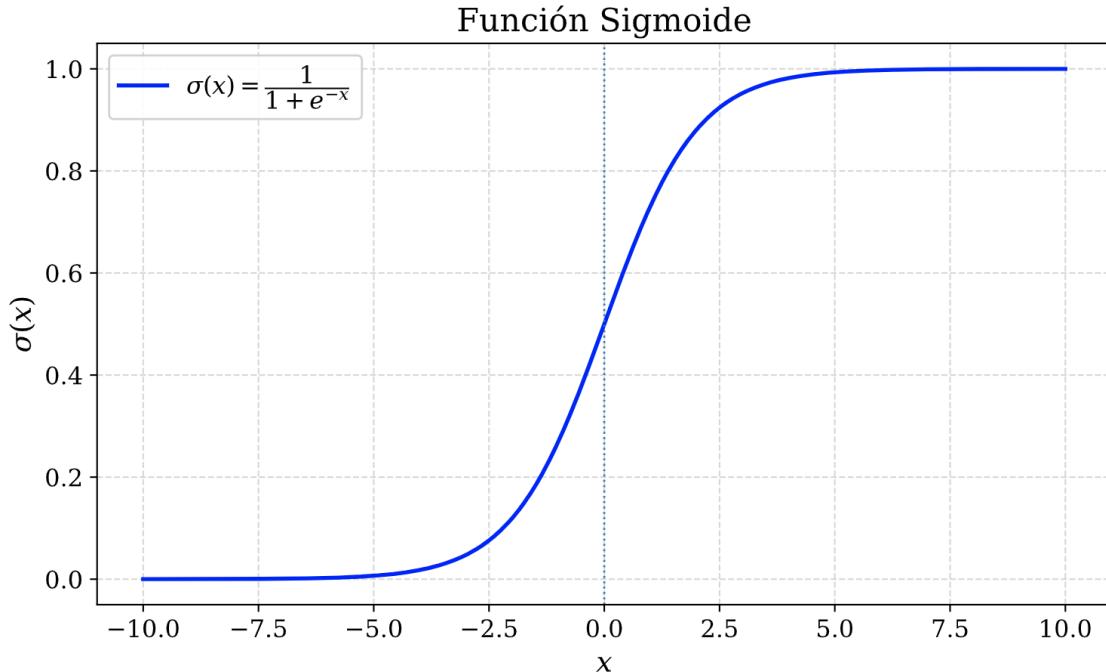


Figura 3.4. Representación gráfica de la función sigmoide (Fuente: elaboración propia)

El algoritmo busca la mejor combinación que de los pesos  $w$ . Esto se logra mediante algoritmos de optimización como el gradiente descendente y similares, que serán explicados en detalle en secciones posteriores.

### 3.1.3 K-Nearest Neighbours

El algoritmo *k-Nearest Neighbours* (*k*-NN, cuya traducción significa los *k* vecinos más próximos) es uno de los métodos más sencillos de aprendizaje supervisado. Este consiste en clasificar un nuevo punto en una nube de datos en el espacio según la categoría de los puntos que le rodean. Este es muy útil ya que puede utilizarse para clasificación multiclas.

El usuario especifica el número *k* de vecinos a considerar, y el algoritmo asigna la clase del nuevo punto en función de las categorías mayoritarias entre los *k* puntos más cercanos que le rodean. Se pueden usar diferentes medidas para identificar los vecinos más próximos, aunque lo más común es tomar la distancia euclídea:

$$d(p, q) = \|q - p\|_2 = \sqrt{\sum_{i=1}^n (q_i - p_i)^2} \quad (3.6)$$

donde *p* y *q* son los puntos entre los que se calcula la distancia y *n* es el número de propiedades que se comparan. La figura Figura 3.5 ilustra como se clasifican 4 puntos en un

entorno de elementos con 3 clases posibles, cuando se comparan únicamente 2 propiedades (características o *features*, cuyos valores vienen representados en los ejes).

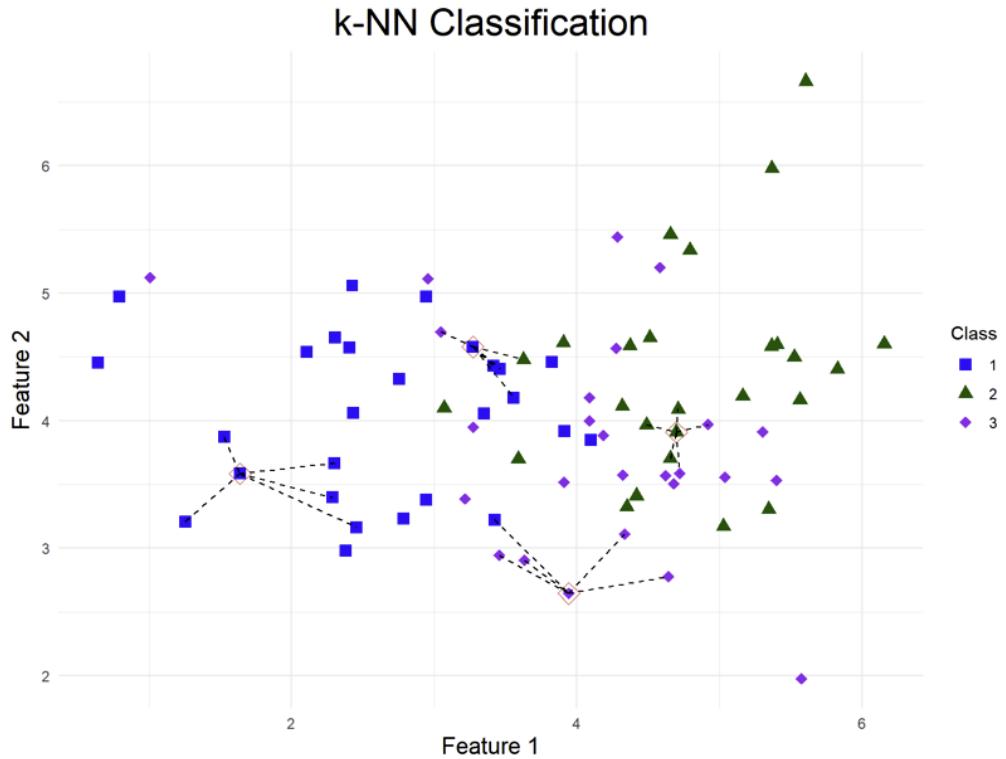


Figura 3.5. Ejemplo de clasificación por k-NN con  $k = 5$ . (Fuente: Freie Universität Berlin [46]).

A pesar de la simplicidad del método y su capacidad de adaptación a un amplio rango de problemas de clasificación, este presenta desventajas tales como su sensibilidad a atributos irrelevantes, y al aumento de dimensionalidad. Además el factor  $k$  debe elegirse cuidadosamente para que la clasificación no se vea afectada por valores atípicos (*outliers*) o ruido existente en el conjunto de datos [46].

### 3.1.4 Support Vector Machines (SVMs)

Las máquinas de soporte vectoriales (SVMs, por sus siglas en inglés), son un tipo de algoritmo supervisado que se caracteriza por su capacidad para encontrar una frontera de decisión óptima entre puntos en el espacio pertenecientes a distintas clases. El concepto de funcionamiento de los SVMs se ve gráficamente en la siguiente figura:

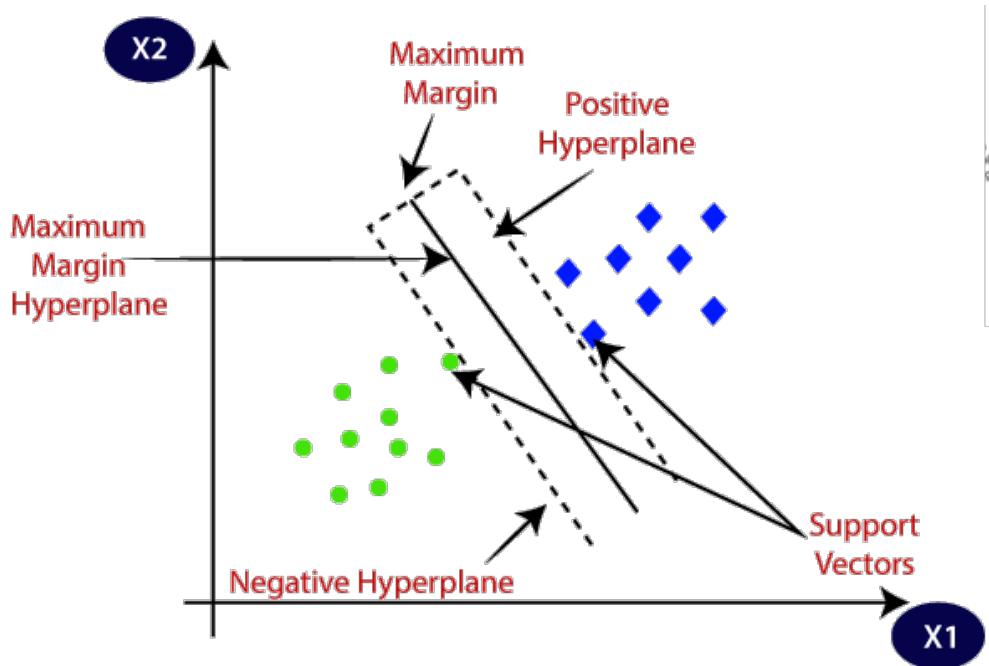


Figura 3.6. Principio de funcionamiento de los SVMs con dos clases (Fuente: [47]).

Aquí se define un hiperplano, como delimitación óptima entre los datos de las distintas clases. El principio de funcionamiento es sencillo: se definen una serie de vectores de soporte, que pasan por los puntos de una clase más próximos a los de la categoría opuesta. A partir de estos vectores de soporte, se busca la dirección que mejor separa, es decir, que maximiza la distancia de separación (margin) entre ambas clases, de forma que cuando se incluyan nuevos puntos estos sean asignados con mayor probabilidad a la categoría correcta [48].

La principal ventaja de este método es su capacidad para funcionar proyectando los datos originales a subespacios de mayores dimensiones, de forma que si una barrera es difícil de definir por la posición inicial de los puntos, se puede transformar el espacio para obtener una delimitación más clara. Esto ocurre normalmente para datos no lineales, y la estrategia de transformación se conoce como *kernel trick*.

Los SVMs alcanzan altos niveles de precisión si han sido definidos correctamente, con la desventaja de que requieren datos etiquetados para su entrenamiento. Esto los hace particularmente eficaces en problemas de clasificación binaria. En el contexto de mantenimiento predictivo, son muy útiles si se poseen datos históricos de equipos con diferentes características e indicaciones de funcionamiento correcto o malfuncionamiento [49].

### 3.1.5 Decision Tree (DT) y Random Forest (RF)

Los árboles de decisión aprenden una serie de reglas a partir de los atributos de un conjunto de datos. El objetivo de este tipo de algoritmos es predecir el valor de una variable a partir de una secuencia de decisiones sobre el resto de características. De esta forma, se puede entender un árbol de decisiones como una aproximación por partes. A mayor número de decisiones, mejor aproximará el modelo el valor de la variable buscada [50].

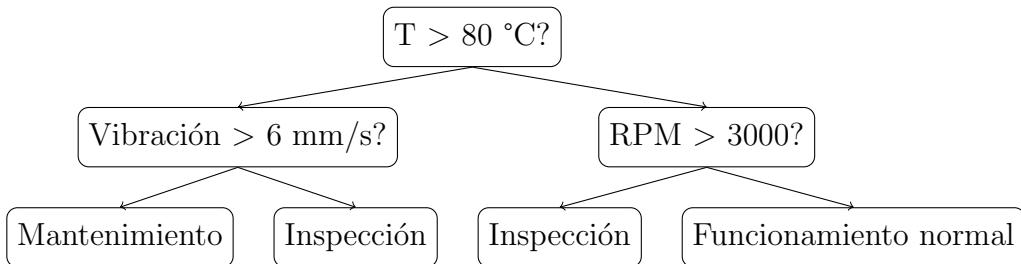


Figura 3.7. Árbol de decisión categórico aplicado al mantenimiento predictivo de un motor eléctrico genérico (Fuente: elaboración propia).

Estos algoritmos se pueden usar tanto para clasificación como para regresión, soportan tanto datos numéricos como categóricos y requieren una preparación de datos casi nula. Además el coste computacional una vez entrenado es bajo (logarítmico en el número de datos). Su principal ventaja es que se pueden interpretarse gráficamente.

Sin embargo, son modelos inestables ya que una pequeña variación puede causar el cambio de las decisiones alterando por completo el resultado y no funcionan bien en problemas que requieren extrapolación. También presentan riesgo de overfitting (cuando el modelo aprende a clasificar demasiado bien los datos de entrenamiento, pero no es eficaz clasificando nuevos datos) debido a ruido en los datos o valores atípicos, especialmente cuando el árbol es muy profundo o complejo.

Por otra parte, los Random Forests (RFs) corresponden a una combinación de árboles de decisión, agregando mayor aleatoriedad al proceso de decisión. En estos algoritmos, cada árbol presenta condiciones de clasificación distintas. Se realizan tantas predicciones individuales como árboles existentes y luego simplemente se computa la media de los resultados para tareas que requieran regresión, o se escoge la categoría más repetida en el caso de procesos de clasificación.

En contraste con los DTs, los RFs son más complejos de representar, pero reducen considerablemente problemas de *overfitting*, por lo que generalizan mejor para nuevos datos, ofreciendo un comportamiento más robusto frente a datos ruidosos o mediciones inexactas [51].

### 3.2 Redes Neuronales

Las redes neuronales (Neural Networks, NNs) son otro tipo de algoritmo que se basa en el funcionamiento del cerebro humano. Este tipo de modelos no clasifica dentro del aprendizaje supervisado, ni del no supervisado, ya que se puede usar indistintamente en cualquiera de estos ámbitos. Las NNs se han popularizado mucho por su capacidad de adaptación a tareas muy variadas. Estas consisten en una serie de elementos interconectados denominados neuronas, que realizan un gran número de operaciones sencillas, transformando unos datos de entrada en salidas procesadas.

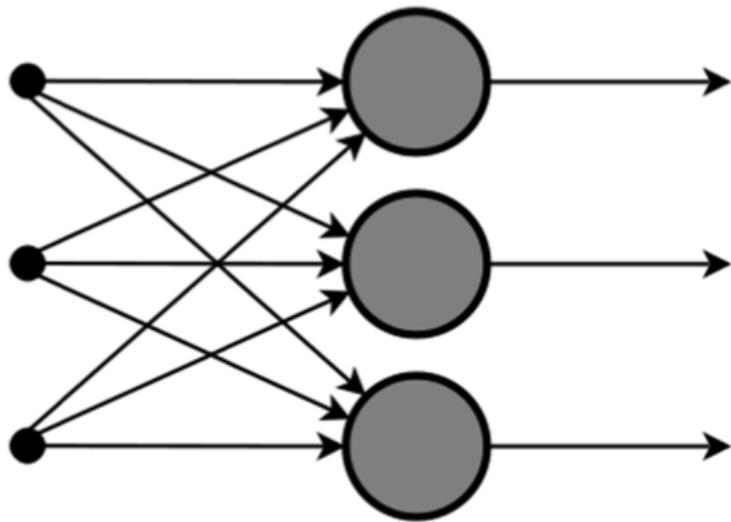


Figura 3.8. Ejemplo de una capa de red neuronal (Fuente: [52]).

Las redes neuronales se caracterizan por el gran número de parámetros que pueden manejar y la abundante cantidad de nodos que soportan. Como se ve en la Figura 3.9, cada neurona recibe un valor de entrada proveniente de otra neurona o de un punto inicial. Este input se modifica mediante una combinación lineal de parámetros denominados pesos de las neuronas, y finalmente se pasan por una función de activación, como puede ser la función sigmoidal explicada en el Sección 3.1.2 u otras como ReLU (Rectified Linear Unit) o la tangente hiperbólica.

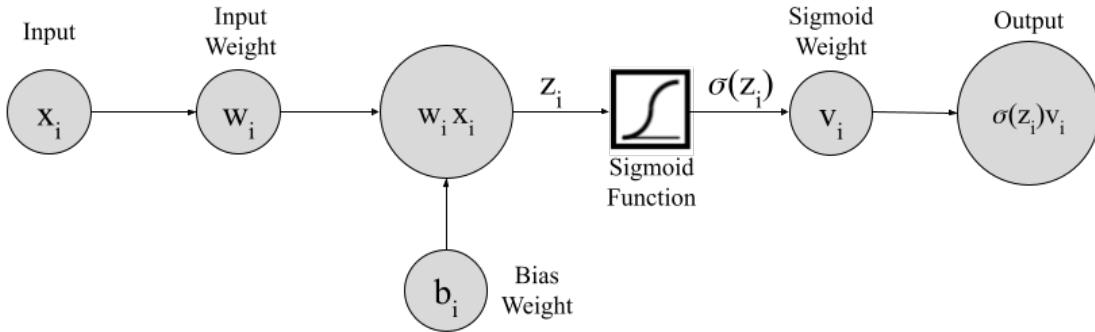


Figura 3.9. Diagrama de funcionamiento una neurona en una NN con función de activación sigmoide (Fuente: elaboración propia).

Esta construcción se suele usar conjuntamente con algoritmos de optimización, para minimizar alguna función de coste definida por el usuario que permita obtener la representación más fiel de una ecuación compleja. En esencia, las redes neuronales usan la *backpropagation* (retropropagación), mediante la cual calculan los gradientes del error respecto a los parámetros, actualizando el peso de las neuronas en cada iteración para minimizar una función de coste. La NN identifica patrones en los datos de entrada y usando un buen algoritmo y unos buenos parámetros de modelización es capaz de generalizar a datos no vistos anteriormente para resolver problemas complejos [52].

Generalmente, una red neuronal agrupa neuronas en capas (layers), teniendo siempre una capa de entrada y salida, y en ocasiones capas intermedias denominadas capas ocultas. El número de neuronas de cada capa es fácilmente programable, lo que permite modificar la dimensionalidad del problema, según las particularidades de cada caso. Estos usos de redes neuronales para modificación de la dimensionalidad se encuentran en soluciones como *autoencoders*, que serán explicados en detalle en secciones posteriores.

### 3.3 Modelos no supervisados

Los modelos no supervisados son aquellos que trabajan con datos sin información de salida o etiquetas asociadas. Estos modelos se utilizan principalmente para agrupar datos en categorías (*clustering*), detectar anomalías o descubrir relaciones ocultas entre los atributos de un conjunto de datos.

Si bien el aprendizaje supervisado ha demostrado ser efectivo en aplicaciones industriales, su aplicabilidad suele estar limitada por la escasez de datos etiquetados en entornos industriales. El aprendizaje no supervisado representa una alternativa muy útil, ya que permite modelar los datos sin necesidad de retroalimentación explícita basada en etiquetas.

Como ejemplo real de un estudio basado en mantenimiento predictivo usando apren-

dizaje no supervisado, G. Nota et al. utiliza un modelo basado en distribuciones de comportamiento de equipos de maquinaria industrial [53]. El modelo aprende de la distribución de los datos, identificaando desviaciones anómalas y agrupando observaciones en categorías o *clusters* probabilísticos, según la probabilidad de pertenencia a la distribución.

Para el entrenamiento de estos algoritmos, se utilizan principalmente tres técnicas, detalladas a continuación con ejemplos de algoritmos para cada una de ellas.

### 3.3.1 Clustering

El *clustering* se basa en agrupar datos no etiquetados en función de sus similitudes o diferencias en las propiedades observadas. El clustering puede realizarse de diferentes maneras [54]:

- Clustering exclusivo: cada punto puede pertenecer únicamente a un grupo de datos o cluster. Aquí destaca el K-means Clustering como algoritmo principal, en el que se define un valor  $k$  que indica el número de clusters buscados. Para cada cluster se calcula un centroide y cada punto se asigna al clúster con el centroide esté más cercano, usando generalmente la distancia euclídea (véase la Ecuación 3.6).
- Clustering jerárquico: utiliza dos enfoques distintos para separar los datos. Puede ser aglomerativo, comenzando con un cluster por punto, fusionándolos iterativamente; o divisivo, iniciando con todo el conjunto de datos y dividiéndolo progresivamente. Este método no requiere fijar un número de clusters, lo que permite explorar distintos niveles de granularidad. Las divisiones suelen representarse mediante dendrogramas, donde se puede observar las distintas divisiones realizadas en cada paso.

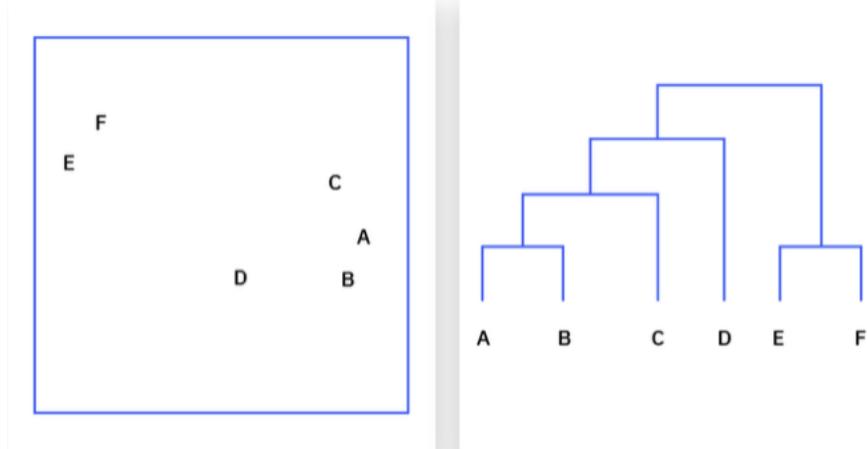


Figura 3.10. Dendrograma de clustering jerárquico mediante el método divisivo (Fuente: IBM [54]).

- Modelos probabilísticos: usa modelos de mezcla gaussiana Mixtos (Gaussian Mixture Models, GMMs) para clasificar un punto acorde a la probabilidad existente de que este pertenezca a un grupo determinado. Los GMMs están formados por varias distribuciones de probabilidad en base a distintas características del set de datos. De esta forma, por ejemplo, si se definen tres funciones de probabilidad, un punto puede pertenecer con un 65 % de probabilidades al cluster 2, con un 25 % al cluster 1, pero solo pertenecerá al grupo 3 en un 10 % de los casos.

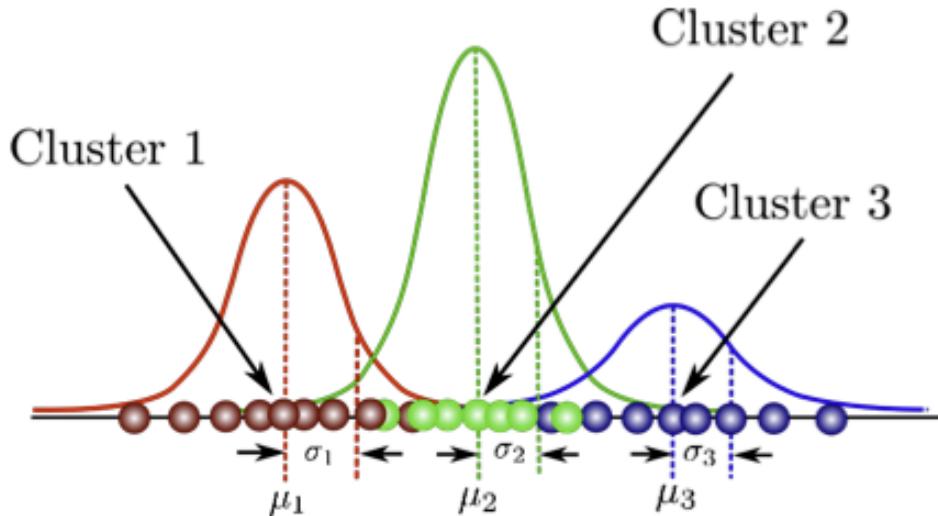


Figura 3.11. Distribuciones gaussianas superpuestas en un GMM (Fuente: Universidad de Shanghai Jiao Tong [55]).

### 3.3.2 Reglas de asociación

Esta técnica es común en aplicaciones móviles que recomiendan productos basado en adquisiciones anteriores o elecciones de compra. Se basa en encontrar relaciones entre variables de varios puntos de un dataset dado para realizar agrupaciones.

El principal algoritmo de este tipo de modelos no supervisados es el algoritmo Apriori. Este algoritmo agrupa objetos según la probabilidad de que se escoja un ítem dada la elección previa de otro ítem. Empresas como Amazon o Spotify entran estos algoritmos uniendo una gran cantidad de datasets, gracias a la gran cantidad de datos que obtienen de las compras o elecciones musicales de sus usuarios. De esta forma realizan recomendaciones en base a las elecciones anteriores del propio usuario y a gustos similares de terceros [54].

### 3.3.3 Reducción de la dimensionalidad

Es un método necesario para datasets en el que el número de atributos es demasiado elevado. Un número elevado de atributos conlleva un número elevado de dimensiones de análisis. Este concepto consiste en reducir la cantidad de atributos de entrada, para un procesamiento más eficiente de los datos, manteniendo únicamente las variables más relevantes. La reducción de la dimensionalidad tiene lugar en el preprocesamiento de datos. Existen varias técnicas de reducción la dimensionalidad:

- Análisis de Componentes Principales (Principal Component Analysis, PCA): identifica las direcciones que captan la máxima varianza de los datos. Esto se hace mediante el cálculo de la matriz de covarianzas, y extrayendo los autovectores (direcciones) y autovalores (varianza de cada una de ellas). De esta forma se pueden seleccionar las principales variables que aportan a la varianza mediante una combinación lineal de las mismas, para después ser proyectadas en las direcciones principales donde los datos muestran mayor dispersión, facilitando, por tanto, su diferenciación y clasificación [56].
- Descomposición en Valores Singulares (Singular Value Decomposition, SVD): utiliza un procedimiento similar pero descomponiendo una matriz formada por las variables de los distintos puntos de la siguiente manera:

$$A = U \Sigma V^T \quad (3.7)$$

Donde  $A$  es la matriz original de tamaño  $mxn$ ,  $U$  es una matriz  $mxm$  con vectores ortogonales cuyas columnas son direcciones en el espacio de las filas,  $\Sigma$  es una matriz diagonal con valores reales positivos llamados valores singulares, que aparecen ordenados de mayor a menor y  $V^T$  es una matriz  $nxn$  cuyas columnas corresponden a las direcciones en el espacio de las columnas. La clave de este método está en eliminar las direcciones que aportan menor varianza, manteniendo las filas y columnas de las  $k$  direcciones más relevantes, eliminando así la influencia del ruido sin perder información esencial [57]. En la práctica, se conservan únicamente los  $k$  valores singulares más grandes, reduciendo así el número de dimensiones del problema.

- Autocodificadores: utilizan redes neuronales para comprimir la representación de los datos. Generalmente utilizan una capa oculta en la red neuronal con menores dimensiones que la entrada, comprimiendo inicialmente los datos. De esta manera, la NN aprende a eliminar el ruido, y reconstruir los datos originales, manteniendo la información esencial. Este método se utiliza mucho en reconstrucción de imágenes, aunque también pueden aplicarse a mantenimiento predictivo, por ejemplo entrenando un modelo para reconstruir señales de funcionamiento normal de un equipo. El sistema tendrá

problemas para reconstruir señales que indiquen un comportamiento anómalo, lo que permite detectar fallos en el sistema [58].

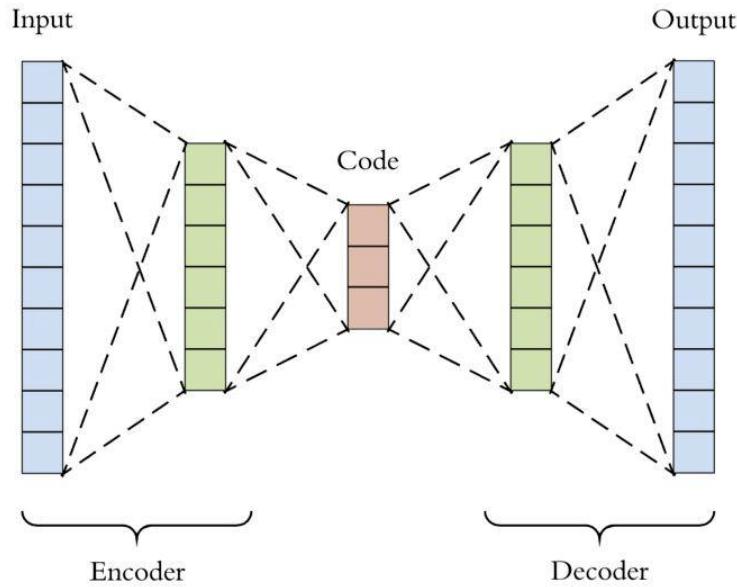


Figura 3.12. Representación gráfica de una red neuronal en forma de autoencoder (Fuente: AI Planet [58]).

### 3.4 Modelos híbridos y su potencial en el mantenimiento predictivo

Existen también modelos semisupervisados, es decir, que solo requieren que parte de los datos vengan etiquetados, reduciendo así el tiempo invertido en añadir una etiqueta (*label*) a cada uno de los datos para entrenar el modelo [59].

Estos modelos utilizan etiquetas en parte de los datos de entrenamiento, aprendiendo el modelo a diferenciar los datos, clasificándolos o agrupándolos en regiones definidas por un hiperplano de decisión. Por esto, es indispensable que los datos etiquetados contengan información útil sobre la tarea a realizar. Por ejemplo, si tratamos de diferenciar entre un avión y un helicóptero, es necesario que los datos con etiquetas sean aviones y helicópteros. Si se incluyeran, por ejemplo, coches en los datos etiquetados, el modelo no obtendría información útil para diferenciar aeronaves.

Un ejemplo aplicado a este tipo de modelos puede verse en el estudio realizado por S. Yoon et al. [60], en el que combinan redes neuronales y modelos generativos para evaluar la vida útil remanente basado en los datos del CMAPSS de la NASA vistos anteriormente[27], donde únicamente se conoce el 1% de los valores del RUL. Para ciertos algoritmos usados, el rendimiento del modelo mejora cuanto menor es la cantidad de datos etiquetados, lo que

significa una gran ventaja cuando se poseen datos crudos captados por sensores que aún no han sido analizados y etiquetados.

Este tipo de algoritmos son óptimos para aplicaciones de mantenimiento, ya que, al menos en aeronáutica, se extraen grandes cantidades de datos de las aeronaves con la implementación de multitud de sensores, pero además los fabricantes y operadores suelen mantener datos históricos de aeronaves previamente usadas.

### **3.5 Reinforcement Learning**

El aprendizaje por refuerzo, más conocido como reinforcement learning, es un enfoque de entrenamiento de algoritmos basado en recompensas. Estos modelos constan de un agente, el cual realiza una tarea específica, recibiendo una recompensa o penalización en función a su desempeño. De esta forma el agente busca maximizar las recompensas obtenidas a largo plazo [61].

Este tipo de modelos es ampliamente utilizado en la industria, ya que no requiere datos etiquetados. Y. Chen y C. Liu (2025) [62] aplicaron este concepto al mantenimiento predictivo de motores turbofán usando redes neuronales complejas basadas en la presencia de múltiples agentes. Cada uno de ellos está enfocado en la minimización de una función de coste. El primer agente penalizaba mantenimientos demasiado tardíos (por seguridad) o demasiado tempranos (por coste), buscando un equilibrio óptimo en la reducción de RUL. El segundo agente trataba de maximizar el intervalo entre inspecciones, reduciendo los costes globales de mantenimiento.

### **3.6 Errores de ajuste de modelos**

Entre los principales errores a la hora de entrenar modelos de ML, encontramos dos errores muy comunes: el sobreajuste y el subajuste, comúnmente nombrados como *overfitting* y *underfitting*. Estos se dan por un exceso o carencia de ajuste de los modelos.

#### **3.6.1 Overfitting**

El overfitting es uno de los eventos más frecuentes que afectan negativamente a la performance de un modelo de aprendizaje automático. Este ocurre cuando un modelo a entrenar presenta tal complejidad, que el algoritmo no aprende patrones generales, sino que memoriza los ejemplos del conjunto de entrenamiento, y los pasos que ha seguido para la obtención de los mejores resultados. Aunque los modelos sobreajustados presentan unas métricas de rendimiento muy positivas, estos resultados se dan únicamente en el conjunto de entrenamiento, sin capacidad

de generalización al conjunto de prueba. Por tanto, cuando a estos modelos se les entregan nuevos datos, estos no son capaces de reproducir los resultados del entrenamiento en la fase de prueba. Como se verá posteriormente, este efecto se puede reducir con medidas aplicables al entrenamiento como la Validación Cruzada (ver Sección 3.8), u otros métodos más sencillos como reducir el valor de ciertos parámetros en algoritmos concretos (número de capas de una red neuronal, profundidad del árbol en un árbol de decisión, etc.), o la eliminación de variables innecesarias. [63]

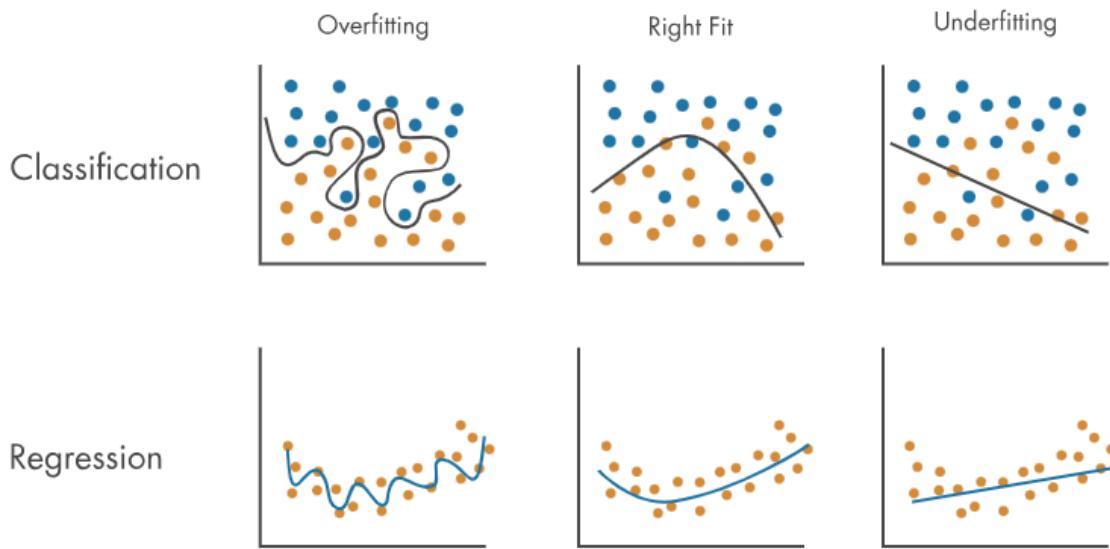


Figura 3.13. Representación gráfica del overfitting y underfitting en tareas de regresión y clasificación (Fuente: Mathworks [64]).

### 3.6.2 Underfitting

El underfitting es un error algo menos común. Al contrario que con el Overfitting, este aparece cuando el modelo es tan sencillo que no es capaz de reconocer la estructura del dataset, generando resultados poco precisos incluso en la fase de entrenamiento. Este puede estar causado por falta de datos para el entrenamiento, el uso de modelos demasiado sencillos para datos complejos o falta de profundidad del modelo por los parámetros usados [63].

## 3.7 Métricas de rendimiento del modelo

Una vez se ha usado un modelo, es importante conocer las métricas que se utilizan para evaluar su rendimiento. La siguiente sección explica las principales formas de evaluar un modelo con sus ventajas e inconvenientes.

### 3.7.1 Exactitud (Accuracy)

Consiste en el porcentaje de aciertos de un modelo de clasificación sobre el total de predicciones realizadas. Matemáticamente:

$$\text{Accuracy} = \frac{n_{correctas}}{n_{total}} \quad (3.8)$$

Esta no es la métrica más adecuada, ya que puede inducir a error: un modelo puede mostrar una exactitud elevada aunque en realidad no esté resolviendo bien la tarea. Por ejemplo, supongamos que se quiere entrenar un modelo que identifique imágenes que representan gatos. Si este y simplemente clasifica todas las imágenes como "No gato", y solo el 10% de las imágenes corresponden a gatos, nuestro modelo obtendrá un 90% de accuracy, aunque no haya aprendido a identificar gatos.

### 3.7.2 Matriz de confusión

La matriz de confusión es útil en clasificadores binarios, en los que existen dos categorías en las que clasificar un ítem. Esta muestra una tabla de dos filas y dos columnas. En las filas, encontramos las clases reales existentes de cada elemento, mientras que en las columnas encontramos las clases predichas por el modelo. La matriz de confusión de un modelo de clasificación nos da información muy útil sobre las decisiones tomadas por el mismo. Según la Figura 3.14, en un modelo entrenado para identificar imágenes que contengan el dígito cinco, se distinguen las siguientes celdas:

- Superior izquierda: 'no cincos' correctamente clasificados. Genéricamente, son los verdaderos negativos (*True Negatives*, *TN*).
- Inferior izquierda: 'cincos' clasificados erróneamente como 'no cincos'. Corresponden a los falsos negativos (*False Negatives*, *FN*).
- Superior derecha: 'no cincos' clasificados incorrectamente como 'cincos'. Representan los falsos positivos (*False Positives*, *FP*).
- Inferior derecha: 'cincos' correctamente clasificados. Corresponden a los verdaderos positivos (*True Positives*, *TP*).

A partir de estas clases, es posible calcular nuevas métricas, como las que se muestran a continuación.

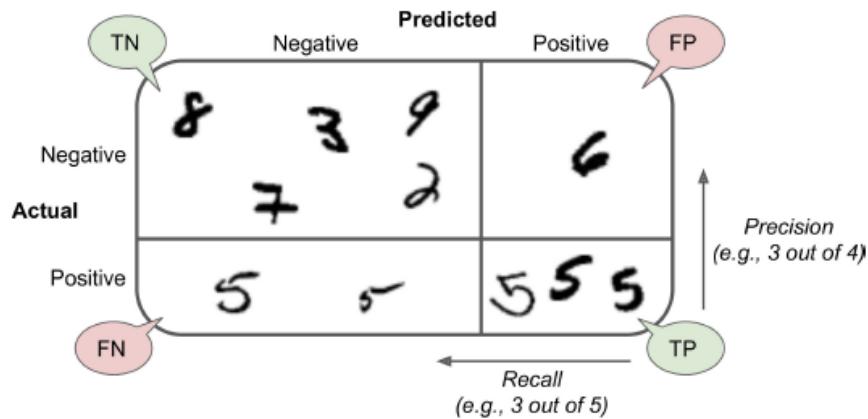


Figura 3.14. Matriz de confusión con los conceptos de precisión y recall (Fuente: [22]).

### 3.7.3 Precisión

La precisión consiste en el número de verdaderos positivos que se identifican sobre el total de positivos detectados. De esta forma:

$$\text{Precision} = \frac{TP}{FP + TP} \quad (3.9)$$

Esta mide la capacidad del modelo de identificar correctamente los ejemplos positivos entre todos los predichos como positivos.

### 3.7.4 Recall

El recall indica cuantos verdaderos positivos se han identificado sobre el total real de valores positivos (falsos negativos más verdaderos positivos), valorando no solo la correcta identificación de positivos, sino también penalizando la no detección de casos reales. Por tanto se tiene:

$$\text{Recall} = \frac{TP}{FN + TP} \quad (3.10)$$

En este caso, el recall nos indica la capacidad del modelo de identificar valores positivos.

### 3.7.5 F1-Score

La puntuación F1 consiste en una métrica que combina Precisión y Recall. Su fórmula es la siguiente:

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3.11)$$

Esta es la media harmónica entre las dos métricas anteriores, por lo que es necesario que tanto precisión como recall sean elevadas para obtener una puntuación F1 elevada [65]. La media harmónica se diferencia de la aritmética en la forma en la que penaliza una diferencia elevada entre dos métricas, si comparamos, por ejemplo, un caso con precisión = 0.9 y recall = 0.2:

$$\bar{X} = \frac{0.9 + 0.2}{2} = 0.55 \quad (3.12)$$

mientras que

$$F_1 = 2 \cdot \frac{0.9 \cdot 0.2}{0.9 + 0.2} = 0.327 \quad (3.13)$$

### 3.7.6 Índice de Silhouette

En el contexto de algoritmos de clustering, el Índice de Silhouette cuantifica la calidad de la separación entre grupos formados por algoritmos tales como el k-Means y GMM. Para ello, utiliza dos distancias diferentes para cada punto de datos clasificado por el algoritmo:

- En primer lugar se calcula la distancia media  $a$  entre el punto en cuestión y el resto de puntos del mismo cluster.
- Se calcula también la distancia media al cluster más cercano distinto al que pertenece el punto. Esta será nombrada como  $b$ .

Con estas medidas, se calcula el Índice de Silhouette:

$$s = \frac{b - a}{\max(a, b)} \quad (3.14)$$

Este índice puede tomar valores entre -1 y 1, con tres puntos clave [66]:

- Un valor cercano a 1 significa que el punto está bien asignado a su grupo. Valores de este tipo se obtienen cuando mayor sea la distancia entre clusters  $b$  y menor sea la distancia intra-cluster  $a$
- Valores en torno al 0 significan que las distancias  $a$  y  $b$  son bastante similares, por lo que el punto podría pertenecer a otro cluster.
- Un índice cercano al -1 indica que el punto probablemente haya sido clasificado incorrectamente, ya que el punto está más cerca del cluster más cercano que de su propio cluster ( $a > b$ ).

Una vez se han hallado los valores para cada uno de los puntos, se computa la media, que indica la métrica global del clasificador. Estas métricas aportan un valor de referencia numérico para la fiabilidad del modelo, aunque pueden ser sustituidas por un análisis visual, tras la reducción dimensional correspondiente si fuera necesario.

### 3.8 Validación Cruzada

La validación cruzada o validación en k-Folds permite evaluar un modelo de ML mediante la división del conjunto de datos en distintas particiones o *folds*. Para ello, se define el valor  $k$  de particiones que se quieren hacer. Una vez obtenidos los folds, se reserva uno de ellos como conjunto de prueba, mientras que el modelo se entrena con los restantes. Este proceso se repite hasta que cada fold haya sido usado una vez para validación. Se obtienen las métricas del modelo y posteriormente se calcula la media y la desviación típica de cada una de ellas.

De esta manera, se obtienen métricas más representativas y se evitan errores de overfitting (ver Sección 3.6), ya que el modelo se entrena y evalúa múltiples veces con diferentes particiones del conjunto de datos [67].

### 3.9 Algoritmos de Optimización

Los modelos que se han visto en las secciones anteriores basados en la reducción de una función de coste requieren un algoritmo de optimización. Estos algoritmos son generalmente procesos iterativos que tratan de buscar la mejor combinación de pesos de neuronas, para obtener el mínimo error de predicción posible. La combinación entre un error reducido y la capacidad de generalización sobre nuevos datos dan como resultado un modelo versátil y eficaz. Aquí explicamos alguno de los algoritmos más populares, así como los que se utilizarán en el caso práctico posteriormente.

#### 3.9.1 Gradiente Descendente

El método del gradiente descendente o Gradient Descent (GD) es el algoritmo más común en la práctica. Este se basa en la existencia de un vector de pesos a optimizar denominado  $\mathbf{w}$ , y la función a minimizar  $J(\mathbf{w})$ . Su principio de funcionamiento se basa en dos pasos sencillos:

1. Se calcula el gradiente de la función de coste respecto a cada parámetro del vector de pesos.
2. Se actualizan los pesos en la dirección opuesta al gradiente, ya que este gradiente indica la dirección de máximo crecimiento, y nosotros buscamos la dirección de mayor descenso.

Matemáticamente, se define como:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla_{\mathbf{w}} J(\mathbf{w}_t) \quad (3.15)$$

donde  $\nabla_{\mathbf{w}} J(\mathbf{w}_t)$  es el gradiente de la función respecto a los distintos parámetros, y  $\eta$  es la tasa de aprendizaje o *learning rate*. Este último parámetro controla el tamaño del paso, de forma que si es muy pequeño la convergencia es lenta pero la optimización más robusta, mientras que un valor demasiado alto puede provocar divergencia, al sobrepasar el mínimo en cada iteración [20].

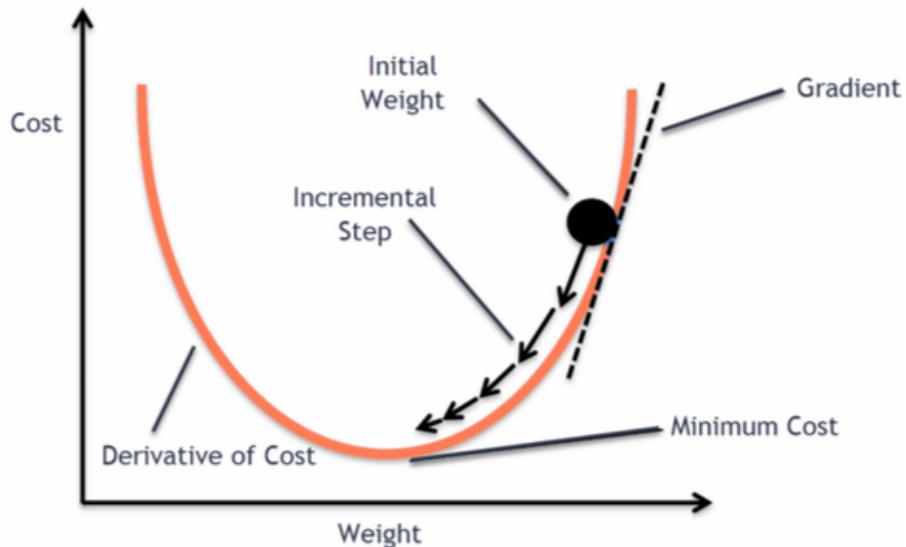


Figura 3.15. Esquema gráfico del funcionamiento del gradiente descendente (Fuente: Analytics Yogi [37])

Este algoritmo tiene variaciones numerosas basadas en pequeñas modificaciones del método original.

### 3.9.2 RMSProp, AdaGrad, momentum y Adam

En el contexto de la optimización de modelos de ML, y en particular en el entrenamiento de redes neuronales, existen múltiples algoritmos que buscan mejorar la convergencia del descenso por gradiente. Entre ellos, **AdaGrad**, **RMSProp**, **Momentum** y **Adam** son ampliamente utilizados debido a su capacidad para adaptar la tasa de aprendizaje de forma individual para cada parámetro del modelo.

## AdaGrad

El algoritmo AdaGrad (*Adaptive Gradient Algorithm*) ajusta dinámicamente la tasa de aprendizaje de cada parámetro en función de la magnitud acumulada de los cuadrados de los gradientes a lo largo del tiempo [68]. Esto resulta especialmente útil en problemas con gradientes dispersos, ya que contiene la evolución de parámetros que ya han sufrido grandes cambios, favoreciendo la actualización de parámetros asociados a gradientes menos frecuentes o de menor magnitud. La actualización de AdaGrad se define como:

$$r_t = r_{t-1} + g_t^2 \quad (3.16)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{r_t} + \varepsilon} g_t \quad (3.17)$$

donde  $g_t = \mathbf{w}J(\mathbf{w}_t)$  es el gradiente en el instante  $t$ ,  $r_t$  es el acumulador de gradientes al cuadrado,  $\alpha$  es la tasa de aprendizaje inicial y  $\varepsilon$  es un valor pequeño para evitar divisiones por cero. La ecuación Ecuación 3.17 muestra como la tasa de aprendizaje se modifica con la raíz del acumulador de gradientes cuadrado, disminuyendo el término cuanto mayor sea la acumulación en un determinado tiempo  $t$ . La principal limitación de este optimizador es que  $r_t$  crece indefinidamente, lo que provoca que la tasa de aprendizaje tienda progresivamente a valores muy pequeños, pudiendo dificultar la convergencia.

## RMSProp

EL algoritmo RMSProp (*Root Mean Square Propagation*) surge como una modificación de AdaGrad. Este cambia la suma acumulada  $r_t$  por una media móvil de los gradientes al cuadrado, lo que evita que la tasa de aprendizaje decaiga demasiado rápido [69]:

$$s_t = \rho s_{t-1} + (1 - \rho) g_t^2 \quad (3.18)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{s_t} + \varepsilon} g_t \quad (3.19)$$

donde  $\rho$  es el factor de decrecimiento exponencial, típicamente cercano a 0.9. Esta modificación permite que RMSProp se adapte mejor a problemas de larga duración y datos no estacionarios.

## Momentum

El método de momentum es una extensión del gradiente descendente diseñada para acelerar la convergencia y suavizar las oscilaciones en la dirección del gradiente [70]. La idea es mantener una velocidad acumulada que representa una media móvil de los gradientes pasados, de forma que las direcciones coherentes en el tiempo se vean reforzadas y las direcciones ruidosas se atenúen.

La formulación estándar de *momentum* es:

$$m_t = \mu m_{t-1} - \alpha g_t \quad (3.20)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + m_t \quad (3.21)$$

donde:

- $g_t$ : gradiente de la función de pérdida respecto a los parámetros en la iteración  $t$ .
- $m_t$ : velocidad acumulada (momentum) en el instante  $t$
- $\alpha$ : tasa de aprendizaje.
- $\mu$ : coeficiente de momentum, típicamente cercano a 0.9.

De esta manera los gradientes que mantienen la misma dirección durante varias iteraciones acumulan más velocidad, acelerando el aprendizaje en esa dirección, mientras que los gradientes que no parecen modificarse demasiado se amortiguan.

## Adam

Finalmente, el algoritmo Adam (Adaptive Moment Estimation) se crea combinando las ventajas de RMSProp con la técnica de Momentum, que consiste en mantener una media móvil exponencial de los gradientes. Adam calcula dos estimaciones: el primer momento (media de gradientes) y el segundo momento (media de gradientes al cuadrado), aplicando además una corrección de sesgo para compensar la inicialización en cero [71]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (3.22)$$

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2) g_t^2 \quad (3.23)$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{s}_t = \frac{s_t}{1 - \beta_2^t} \quad (3.24)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{s}_t} + \varepsilon} \quad (3.25)$$

donde  $\beta_1$  y  $\beta_2$  son los factores de estabilización para el primer y segundo momento, respectivamente.

Es importante destacar el papel de la corrección de sesgo representado por la Ecuación 3.24. En las primeras iteraciones, tanto  $m_t$  como  $s_t$  tienden a estar sesgados hacia cero, ya que inicialmente  $m_0 = 0$  y  $s_0 = 0$ . Este sesgo es más acusado cuanto más altos son los valores de  $\beta_1$  y  $\beta_2$ , pues la media móvil exponencial necesita más pasos para “cargarse” de valores reales. Al dividir por los factores  $(1 - \beta_1^t)$  y  $(1 - \beta_2^t)$ , se corrige este sesgo, obteniendo estimaciones  $\hat{m}_t$  y  $\hat{s}_t$  que representan de forma más precisa el promedio real de gradientes y gradientes al cuadrado, incluso al inicio del entrenamiento.

Adam hereda de RMSProp el uso de una media móvil de gradientes al cuadrado para escalar la tasa de aprendizaje (nótese cómo la Ecuación 3.23 es matemáticamente equivalente a la Ecuación 3.18) y del algoritmo de Momentum la acumulación suavizada de gradientes (Ecuación 3.22 es matemáticamente equivalente a la Ecuación 3.20), lo que lo hace robusto y eficiente en problemas con ruido y gradientes dispersos [69]. Este será utilizado posteriormente en uno de nuestros casos prácticos, al ser elegido por el autor del estudio replicado como optimizador de una red neuronal con numerosos parámetros, caso para el que es especialmente útil este método.

### 3.9.3 Grid Search

Con el objetivo de obtener los mejores resultados en nuestros modelos, existen distintas técnicas para optimizar los mismos basadas en la modificación de parámetros del modelo. En cada uno de los algoritmos de ML intervienen una serie de parámetros que modifican el funcionamiento de los mismos. El rendimiento del modelo dependerá, además de la calidad de los datos de entrada, de la combinación de los parámetros internos del modelo y su correcto ajuste acorde a los datos a analizar.

En las librerías de Python que se usarán para obtener los modelos de ML, existen gran cantidad de funciones cuyos parámetros pueden ser ajustados manualmente por el usuario. Sin embargo, encontrar la combinación óptima de parámetros es un proceso generalmente tedioso, puesto que en la mayoría de casos la lista de parámetros es extensa, generándose así una enorme cuantía de combinaciones distintas posibles.

La función Grid Search lleva a cabo un proceso iterativo en el que se modifican uno a uno

una lista de parámetros, realizando los cálculos necesarios para evaluar un modelo en todas y cada una de las combinaciones posibles. Para ello, el usuario define una lista de parámetros a tener en cuenta, así como un rango (en caso de valores numéricos) o una lista de categorías (para parámetros categóricos).

La función evalúa el modelo en toda esta lista de parámetros, obteniendo una métrica determinada también especificada por el usuario, que se usará para comparar el rendimiento del modelo con las diferentes combinaciones de parámetros. Finalmente, la función escogerá la combinación que mejor puntuación obtiene en la métrica dada y mostrará la lista de parámetros del mejor conjunto.

Además, la función es compatible con una validación cruzada de k-Fold, de manera que es posible combinar la optimización paramétrica con la evaluación del modelo con un número determinado de subconjuntos o *subsets* del conjunto de datos original [72].

La herramienta de Grid Search con Cross Validation puede encontrarse en la librería `sklearn` (**Scikit-Learn**) con el comando `GridSearchCV`.

### 3.10 Implementación computacional en Python

El código desarrollado para replicar los estudios de la Sección 4, ha sido íntegramente programado en Python. El uso de esta herramienta se justifica por multitud de razones de peso. En primer lugar, la herramienta es gratuita y de código abierto, y cuenta con una comunidad muy fuerte, contando con foros y webs muy explicativas, que pueden ser consultadas para la resolución de problemas. Además, este lenguaje de programación cuenta con un alto nivel de implementación en la industria, así como librerías variadas con funciones ya creadas [73]. Las principales funciones de visualización y cálculos matemáticos genéricos son:

- **Numpy**: incluye multitud de operadores para cálculo vectorial y matricial.
- **Pandas**: Permite la carga, modificación y análisis de grandes cantidades de datos organizados en dataframes. Esto es una parte crítica de los casos prácticos, ya que su uso determinará la lectura de los datos por parte de los modelos de ML.
- **Matplotlib y Seaborn**: estas librerías de carácter visual facilitan la creación de gráficos explicativos. Con ellas se crearán figuras que serán muy útiles para visualizar los resultados obtenidos.

Para la implementación de código basado en inteligencia artificial y en concreto machine learning, existen librerías muy destacadas que facilitan la implementación rápida y sencilla de diferentes modelos y herramientas que se han visto en esta sección. Las principales son :

- **Scikit-Learn:** incluye multitud de atajos para el preprocesamiento de datos como normalizadores así como recursos para el entrenamiento y validación de los modelos de ML.
- **TensorFlow y Keras:** se basan en funciones para la creación, entrenamiento y evaluación de redes neuronales. Estas librerías incluyen los algoritmos de optimización como Adam, y la implementación de funciones de pérdida como el error cuadrático medio.

## 4 Experimentación y Resultados

En las secciones anteriores se han descrito los principales algoritmos y modelos de ML, así como su aplicación al mantenimiento predictivo. Una vez establecido el contexto y presentadas las herramientas disponibles, se muestran a continuación una serie de ejemplos prácticos que ilustran su aplicación.

Para la parte experimental, se replicaron dos artículos académicos que tratan los principales algoritmos supervisados y no supervisados. A estos se añadieron procedimientos innovadores, métricas de rendimiento adicionales y representaciones gráficas que facilitan la interpretación de resultados. De esta forma se obtiene una visión global de los métodos empleados y una mejor comparación de los resultados.

La sección se organiza de la siguiente manera:

- En la Sección 4.1 se replica un estudio de clasificación binaria que utiliza modelos supervisados.
- En la Sección 4.2 se analiza un caso con modelos no supervisados, incluyendo reducción dimensional y clasificación probabilística.

Ambos estudios son independientes entre sí, y se comparan únicamente con las referencias originales. Cada caso introduce su propia metodología, describe los datos empleados, así como el procedimiento usado y los resultados obtenidos.

### 4.1 Caso 1: Modelos Supervisados en Tareas de Clasificación

Para este primer estudio nos basamos en el artículo *Selecting an appropriate supervised machine learning algorithm for predictive maintenance* [35] de A. Ouadah, Z. Ghomari, L. y S. Nedjma. En el estudio original se compara el rendimiento de los principales algoritmos supervisados, a partir de unas métricas obtenidas al consultar varias fuentes bibliográficas. Aquí se expresa cómo, en general, para tareas de clasificación, los algoritmos más destacados son Random Forest y k-Nearest Neighbours, seguidos de cerca por los Árboles de Decisión (Decision Trees).

Se usarán por tanto estos tres algoritmos, comparando los resultados obtenidos entre ellos, además de validándolos con respecto a la referencia.

#### 4.1.1 Metodología

Este estudio consiste en el entrenamiento de modelos supervisados para la clasificación binaria de señales. Como se verá a continuación, se presenta un set de datos con una serie de

mediciones y características asociadas a cada una de ellas. Además, cada medición contiene una etiqueta binaria.

Los modelos entrenados deberán predecir el valor de la clase a la que pertenece una medición cuando no se le muestra la clase.

El procedimiento para este primer caso es sencillo, y se resume en los siguientes pasos:

1. Análisis y normalización los datos disponibles.
2. Selección de los parámetros de entrenamiento.
3. Entrenamiento de los modelos e introducción de validación en k-Folds.
4. Test con datos no vistos previamente, cuya etiqueta se oculta.
5. Validación de resultados y obtención de métricas.

Finalmente se realizará un análisis comparativo de los resultados obtenidos.

#### **4.1.2 Descripción del dataset**

De cara a la comparación algorítmica, se usó un dataset público llamado *Bill Authentication*, basado en datos simulados. Este no se basa en mediciones de sistemas reales sobre los que se trabaja el mantenimiento predictivo, sino que fue seleccionado por su simplicidad y las características que ofrece. Según se expone en el artículo:

We chose this data because it included very important parameters in the vibration analysis [35].

El archivo de datos consiste en un .csv fácilmente accesible desde Python para su análisis. La estructura del dataset consiste en 1372 elementos con mediciones de *varianza*, *skewness*, *kurtosis* y *entropía*. Si bien los datos no provienen de un sistema real, las variables que presentan las mediciones son análogas a las empleadas en análisis de vibraciones, como se muestra en otros estudios (U. Farooq, 2024 [74]).

Observando una muestra del conjunto de datos, se obtiene lo siguiente:

Tabla 4.1. EJEMPLO DE REGISTROS DEL DATASET BILL AUTHENTICATION CON SUS CUATRO CARACTERÍSTICAS Y LA CLASE ASOCIADA.

Variance	Skewness	Curtosis	Entropy	Class
3.6216	8.6661	-2.8073	-0.4470	0
4.5459	8.1674	-2.4586	-1.4621	0
3.8660	-2.6383	1.9242	0.1065	0
3.4566	9.5228	-4.0112	-3.5944	0
0.3292	-4.4552	4.5718	-0.9888	0

Las características disponibles según el dataset son las siguientes:

- **Varianza:** hace referencia al grado de dispersión de los datos. Una alta varianza conlleva una gran variación con respecto al valor medio en una distribución normal.
- **Skewness:** se refiere a la asimetría de la distribución. Una skewness hacia la derecha (positiva) indica que la distribución se concentra en los valores bajos y viceversa. Si la distribución es simétrica, la skewness será cero.
- **Curtosis:** mide la concentración de la distribución en torno a los extremos, en comparación con una distribución normal. Una curtosis elevada indica mayor concentración de valores cerca de la media y colas más pesadas, lo que puede reflejar presencia de datos atípicos.
- **Entropía:** mide el grado de aleatoriedad o incertidumbre de la señal, lo que refleja la dificultad de realizar predicciones precisas.

El valor de la etiqueta viene dado por la característica *Class*, que corresponde con una clasificación binaria de ceros y unos. Como se verá posteriormente, la etiqueta *Class* es binaria: el valor 0 representa un estado normal, mientras que el valor 1 indica un estado anómalo.

Se entiende por tanto, que estos datos pueden asemejarse a mediciones periódicas de un equipo, para las que se registran las métricas anteriores.

El objetivo es, por tanto, tratar de predecir en base a las cuatro variables disponibles, si la fila corresponde a la clase 0 o a la clase 1. Una inspección rápida con el comando `df.info()` de Pandas muestra información relevante sobre nuestro conjunto de datos o *dataframe* (`df` se utiliza comúnmente como abreviación de *dataframe* en Python). La Figura 4.2 muestra que no existen valores nulos y que todas las variables son de clase `float64` (número decimal de 64 bits), a excepción de la columna *Class* es `int64` (número entero de 64 bits, aunque podría especificarse como variable booleana).

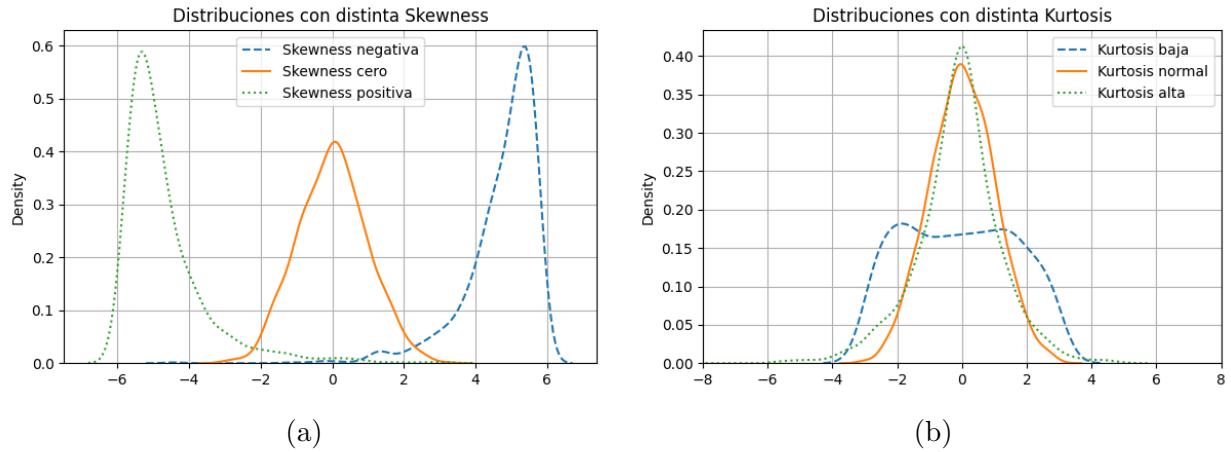


Figura 4.1. Ejemplos gráficos de distribuciones con distinta skewness y kurtosis (Fuente: elaboración propia).

Data columns (total 5 columns):				
#	Column	Non-Null Count	Dtype	
0	Variance	1372 non-null	float64	
1	Skewness	1372 non-null	float64	
2	Curtosis	1372 non-null	float64	
3	Entropy	1372 non-null	float64	
4	Class	1372 non-null	int64	

Figura 4.2. Salida de la función `df.info()` en Python.

Es importante resaltar que el conjunto de datos es de acceso público [75], lo que facilita enormemente su uso y reproducción en estudios académicos. En contraste, la mayoría de los conjuntos de datos reales forman parte del conocimiento interno de empresas privadas, que prefieren mantenerlos como información reservada para su propio análisis.

#### 4.1.3 Preprocesamiento y división del dataset

El siguiente paso consiste en evaluar la división que se realizará en el dataset para su evaluación. En primer lugar, ya que en el estudio de Ouadah no se menciona la división realizada, se decidió entrena con un 70 % de los datos, reservando el 30 % para el testeo.

Además, se normalizará usando la función de `sklearn StandardScaler`. De esta forma, todas las variables se escalan restando su media  $\bar{x}$  y dividiendo entre su desviación típica  $s$ :

$$x_{escalado} = \frac{x - \bar{x}}{s} \quad (4.1)$$

Este escalado produce una distribución con media cero y varianza unitaria [72].

#### 4.1.4 Entrenamiento de los modelos

Cada uno de los tres modelos se entrenó por separado, manteniendo en la mayoría de los casos los parámetros por defecto de Scikit-Learn, salvo las excepciones indicadas.

- Decision Tree
  - Función: `sklearn.tree DecisionTreeClassifier`
  - Resto de variables por defecto
- K-Nearest Neighbors
  - Función: `sklearn.neighbors KNeighborsClassifier`
  - $K = 5$
- Random Forest
  - Función `sklearn.ensemble RandomForestClassifier`
  - Número de árboles:  $N = 100$

#### 4.1.5 Evaluación de precisión, error y robustez del modelo

Los resultados obtenidos son los siguientes:

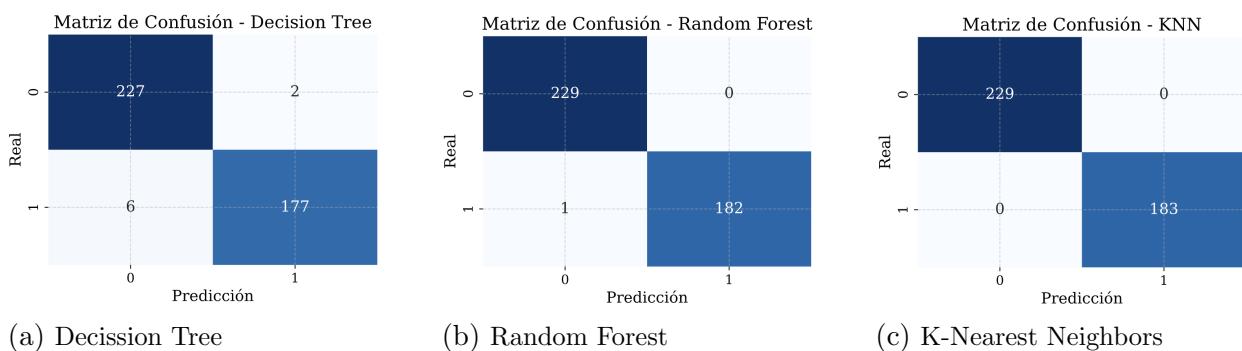


Figura 4.3. Matrices de confusión de los tres modelos supervisados (Fuente: elaboración propia).

Con estas matrices, se obtienen las siguientes métricas:

Clase	Precisión	Recall	F1-score
0	0.97	0.99	0.98
1	0.99	0.97	0.98

Tabla 4.2. DECISION TREE: PRECISIÓN, RECALL Y F1-SCORE POR CLASE

Clase	Precisión	Recall	F1-score
0	1.00	1.00	1.00
1	1.00	0.99	1.00

Tabla 4.3. RANDOM FOREST: PRECISIÓN, RECALL Y F1-SCORE POR CLASE

Clase	Precisión	Recall	F1-score
0	1.00	1.00	1.00
1	1.00	1.00	1.00

Tabla 4.4. K-NEAREST NEIGHBORS (K=5) PRECISIÓN, RECALL Y F1-SCORE POR CLASE

Como se ve, el k-NN con consigue clasificar correctamente cada uno de los items, logrando métricas perfectas. El RF clasificó erróneamente un 1 como un 0, obteniendo un recall de 0.99 para la clase 1. Por último, el modelo de DT realizó algunos errores puntuales, con puntuaciones F1 en torno al 98% para cada clase. Los modelos logran en los tres casos predicciones muy acertadas. Las exactitudes (*accuracies*) de los tres modelos fueron del 100% para RF y k-NN, mientras que para el DT alcanzó el 98%.

Para comprobar la validez de las métricas se aplicó **validación cruzada** con 5 particiones (k-fold), obteniéndose los resultados globales mostrados en la Tabla Tabla 4.5.

Modelo	Accuracy	Precisión	Recall	F1-score
Decision Tree	$0.9811 \pm 0.0071$	$0.9756 \pm 0.0087$	$0.9820 \pm 0.0120$	$0.9787 \pm 0.0080$
Random Forest	$0.9956 \pm 0.0015$	$0.9903 \pm 0.0032$	$1.0000 \pm 0.0000$	$0.9951 \pm 0.0016$
KNN	$0.9985 \pm 0.0018$	$0.9967 \pm 0.0040$	$1.0000 \pm 0.0000$	$0.9984 \pm 0.0020$

Tabla 4.5. COMPARACIÓN DE MODELOS CON VALIDACIÓN CRUZADA DE 5 FOLDS

Tras la validación cruzada, se comprueba que nuestros modelos fueron eficaces en las 5 versiones, afianzando la veracidad de las métricas. La puntuación  $F_1$  más alta se dio en el k-NN, seguido del RF y del DT. Además cabe destacar como la métrica del recall obtuvo una media del 100 % tanto en k-NN como en RF, lo que significa que estos modelos no obtienen falsos negativos, aunque si presentan algún falso positivo.

El artículo de Ouadah, tiene en cuenta métricas poco apropiadas para evaluar modelos de

clasificación, como el error medio absoluto (*Mean Absolute Error*, MAE) y el error cuadrático medio (*Mean Squared Error*, MSE), que son más útiles en modelos de regresión. Comparando el dato de accuracy, se ve como nuestros resultados experimentales obtienen un valor ligeramente mayor. Además, los tiempos de entrenamiento resultan también mucho menores en nuestros modelos.

	Accuracy		Tiempo de Predicción (s)	
	Referencia	Experimental	Referencia	Experimental
Decision Tree	0.976	0.981	0.864	0.0002
Random Forest	0.984	0.996	4.245	0.0096
KNN	0.989	0.998	3.251	0.0035

Tabla 4.6. COMPARACIÓN ENTRE RESULTADOS EXPERIMENTALES Y LOS REPORTADOS EN EL TRABAJO DE OUADAH ET AL.

Las diferencias sobre todo notables en el tiempo de entrenamiento pueden darse por la capacidad de procesador del equipo usado para las simulaciones, aunque también podrían explicarse porque en el artículo el cronómetro incluyera procesos adicionales como la lectura de datos, mientras que en este trabajo se midió únicamente el tiempo de evaluación.

Los datos de rendimiento son por tanto semejantes, aunque algo mayores para nuestro modelo.

#### 4.1.6 Visualización de resultados

Para contrastar los resultados obtenidos con los datos originales del dataset, se obtuvieron unas gráficas en las que se muestran las principales diferencias entre clases, acorde a las distintas variables obtenidas.

Usando la librería `seaborn` y especificando un número  $n = 300$  de muestras a ser representadas (para obtener una representación más limpia), se obtuvo la gráfica múltiple que se presenta en la Figura 4.4.

En rojo aparece representada la distribución de puntos etiquetados con 0, y en azul los etiquetados con 1. De las gráficas de la diagonal se puede extraer cómo se distribuyen los diferentes atributos del conjunto de datos. En este caso específico, con el número de muestras tomadas, los puntos quedan distribuidos de tal forma que se produce una separación clara de los mismos, especialmente en las gráficas de las variables de varianza y skewness. En este caso, los puntos con varianzas más altas y mayor skewness corresponden a los datos clasificados como cero, mientras que los valores de entropía y curtosis muestran distribuciones más similares, con picos más altos para los "ceros".

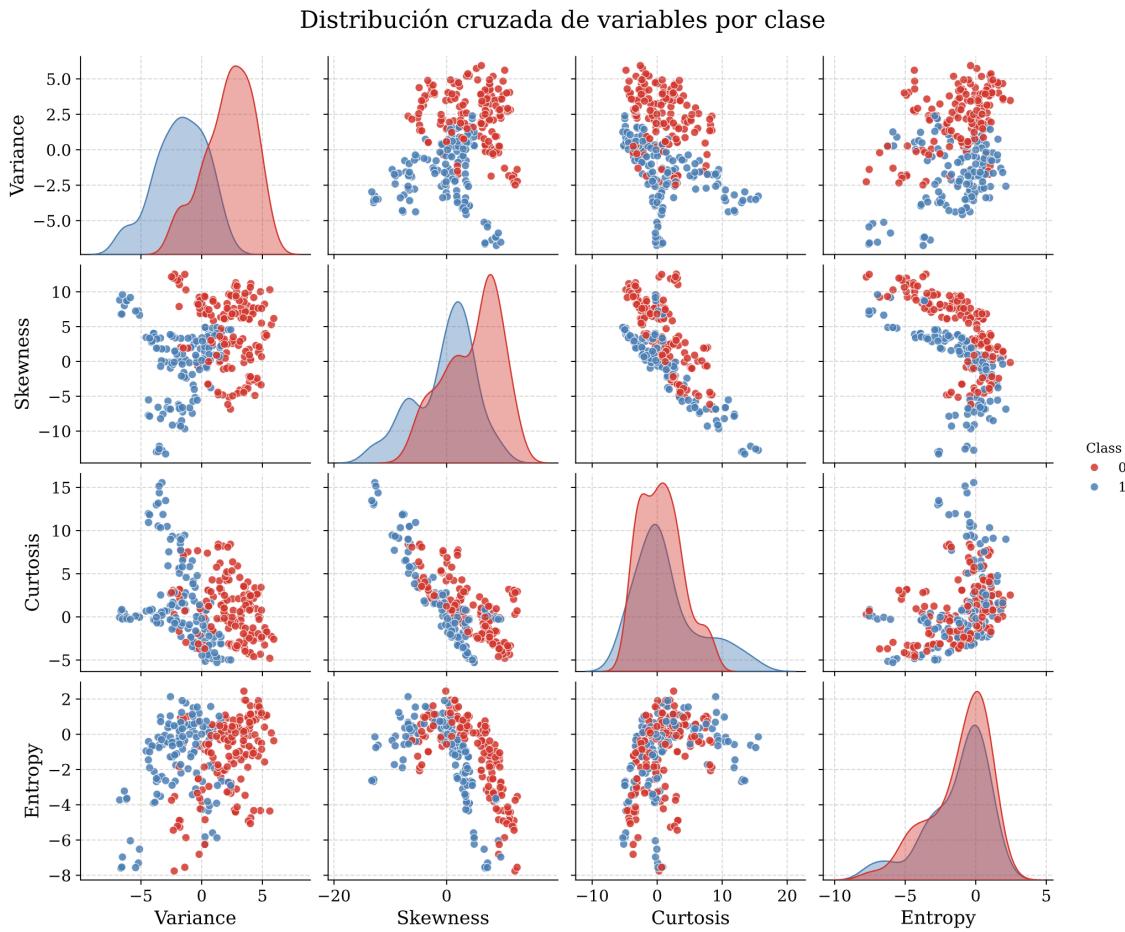


Figura 4.4. Distribución cruzada de variables una vez clasificados los datos (Fuente: elaboración propia).

El resto de gráficas muestran las distribuciones de una variable en función de otra. En particular, se aprecia una separación clara en la gráfica de Skewness vs Varianza (primera columna de la segunda fila) donde se muestra la combinación de distribuciones estas dos variables, quedando los puntos clasificados como "uno" a la izquierda y los clasificados como "cero" a la derecha, a excepción de algunos puntos que se salen del comportamiento habitual.

Es notable como con una separación tan fácilmente visible, los datos de entrada ya estaban generalmente muy polarizados, por lo que no supone dificultad para el modelo la obtención de valores altos de precisión y otras métricas.

## 4.2 Caso 2: Modelos No Supervisados en tareas de clasificación

Para este segundo caso se utiliza como referencia el estudio realizado por T. Lodygowski y S. Szrama, *Unsupervised Classification and Remaining Useful Life Prediction for Turbofan Engines Using Autoencoders and Gaussian Mixture Models: A Comprehensive Framework for Predictive Maintenance* [36]. En este estudio se utiliza un proceso de clasificación que combina una reducción de dimensionalidad mediante autocodificadores (AutoEncoders, AE) seguido de la clasificación probabilística por medio de Modelos de Mezcla Gaussiana (Gaussian Mixture Models, GMMs) para la clasificación de equipos aeronáuticos en distintos grupos según sus características de funcionamiento.

En el artículo original, se utilizaron varios conjuntos de datos: un dataset real y uno simulado, ambos basados en mediciones de motores turbofán. Como ocurrió en el Caso 1, el dataset real pertenece a una empresa que colaboraba con el estudio por lo que este no es de uso público. Por tanto, se utilizarán los datasets basados en datos simulados del CMAPSS [27]. Este es un conjunto de datos popularmente usados para estudios relacionados con prognosis.

El CMAPSS (Commercial Modular Aero-Propulsion System Simulation) consiste en una herramienta de simulación desarrollada por la NASA que genera datos de vuelo realistas de un motor turbofán comercial, tal y como se describe en la documentación oficial del dataset. La herramienta es capaz de simular un vuelo ascendente hasta unos 35000 pies (12000 metros aproximadamente) y descensos posteriores a nivel del mar. Cada dataset representa una serie de vuelos en secuencia temporal, de forma que se produce un deterioro constante del motor a lo largo del tiempo. Además, durante algunas de las simulaciones se introduce algún tipo de fallo de forma que la vida útil del motor se reduce considerablemente más rápido a partir de ese momento. La herramienta CMAPSS de la NASA fue ideada para generar datos sintéticos que pudieran ser utilizados en competiciones de prognosis y para estudios de desarrollo e investigación, supliendo la falta de datos públicos de vuelo de aeronaves compartidos por los fabricantes u operadores.

Este conjunto de datos es interesante para el estudio por su complejidad, ya que incluye un total de 21 parámetros relevantes del funcionamiento del motor turbofán simulado. El turbofán consiste en las siguientes estaciones:

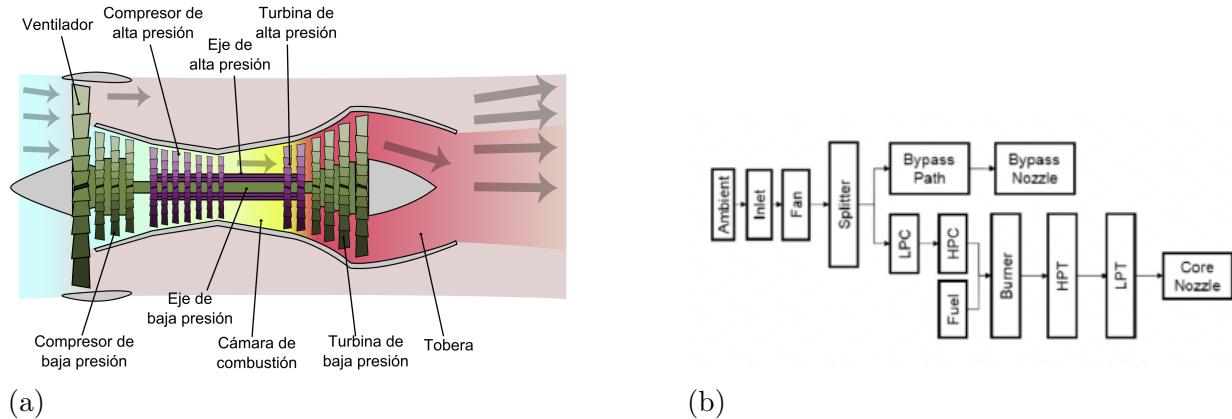


Figura 4.5. (a) Esquema simplificado del motor turbofán [76]. (b) Diagrama de flujo del paso del aire y combustible por las diferentes estaciones del mismo [77]

- Ventilador (*Fan*): realiza la entrada del aire. A partir de aquí parte del aire pasa al núcleo de motor y el resto por el conducto de bypass.
- Compresor de baja presión (*Low Pressure Compressor*, *LPC*): aumenta la presión del aire que entra al núcleo después del ventilador.
- Compresor de alta presión (*High Pressure Compressor*, *HPC*): realiza la segunda etapa de compresión antes de la cámara de combustión.
- Cámara de combustión (*Combustion Chamber*, *CC*): aquí se mezcla el aire con el combustible a altas temperaturas.
- Turbina de alta presión (*High Pressure Turbine*, *HPT*): encargada de usar la energía extraída de los gases calientes para accionar el *HPC*, al que va conectado por medio del eje de alta presión.
- Turbina de baja presión (*Low Pressure Turbine*, *LPT*): al igual que la *HPT*, extrae energía de los gases para accionar tanto el *LPC* como el ventilador. *LPT*, *HPT* y ventilador se conectan mediante el eje de baja presión.
- Toberas (*Nozzles*): existen dos diferentes, una para el aire que pasa por el bypass y otra para el aire que pasa por el core. Expulsa los gases a altas velocidades para generar el empuje que mueve la aeronave.

Aunque los detalles técnicos del motor no son objeto de este estudio, se incluyen aquí como referencia para entender las variables disponibles en el dataset.

#### 4.2.1 Metodología

Este segundo caso consta de una metodología más compleja, ya que se reaaliza una utilización secuencial de distintos modelos, que se entrena con varios datasets.

El conjunto de datos sintéticos provenientes de la simulación CMAPSS consta de cuatro datasets de entrenamiento y cuatro conjuntos de testeo. La principal diferencia entre los sets de entrenamiento y los de test es la vida útil remanente al final de la simulación de cada motor. Mientras que en los sets de entrenamiento todos los motores simulados finalizan su simulación cuando su degradación es tal que el valor del RUL es igual a cero, los conjuntos de prueba interrumpen la simulación cuando el RUL aún es mayor que cero, sin llegar al fallo completo. Además estos cuatro conjuntos de testeo vienen acompañados de los datos de Remaining Useful Life (RUL) de cada motor cuando se pausa su simulación, en un archivo separado.

El primer paso consiste en la preparación de los datos previo al análisis, de cara a obtener unos datos de entrada adecuados para su procesamiento mediante redes neuronales.

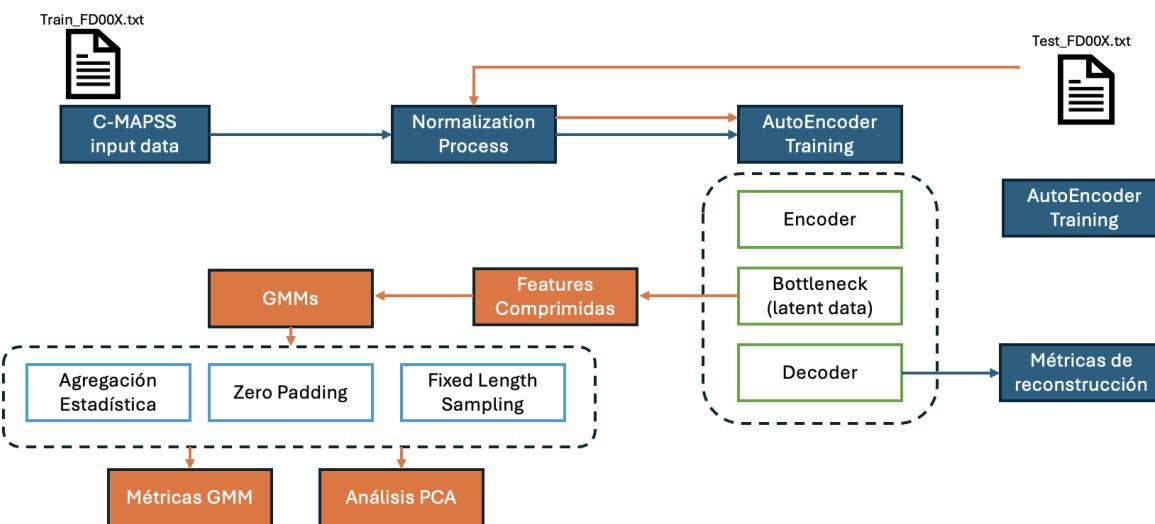


Figura 4.6. Resumen esquemático de la metodología del caso 2

Una vez normalizados los datos pasamos a entrenar nuestro Autoencoder con los sets de entrenamiento. De esta forma, se espera que el autoencoder aprenda el proceso de degradación completo de los motores.

Hasta este punto se utiliza el dataset de entrenamiento, y a partir de aquí se pasa a utilizar el set de testeo para evaluar nuestro modelo de autoencoder. Este nuevo flujo de información viene representado en la Figura 4.6 por flechas de color naranja. Los datos de testeo (archivos Test\_F00X.txt) se pasarán por el Autoencoder ya entrenado. En este paso se obtendrá la versión comprimida de los datos, denominado **espacio latente**, es decir, un

conjunto de datos que contenga la información más relevante.

De este espacio latente correspondiente a la parte del core del autoencoder se extraen las propiedades que se emplean como entrada a un GMM, que clasifica los motores en tres estados finales: ‘saludable’, ‘degradado’ y ‘fallo’. Esta clasificación no supervisada se llevará a cabo mediante el uso de modelos de mezcla gaussiana.

De los cuatro sets de datos de entrenamiento y test disponibles, en nuestro caso se usan únicamente los primeros (**Train\_F001.txt** y **Test\_F001.txt**, con su correspondiente **RUL\_001.txt**). Como veremos, el dataset de entrenamiento contiene información de 100 motores distintos, a lo largo de diferentes periodos de tiempo denominados ciclos. De cara a la clasificación de cada uno de los motores, se agrupan todos los ciclos en un único vector por motor, de manera que la clasificación se realiza una sola vez por motor y no en cada ciclo individual. De esta manera, el GMM clasificará teniendo en cuenta toda la evolución temporal del motor, etiquetando todos los ciclos conjuntamente.

Esta necesidad de agrupación introducirá una nueva cuestión a resolver en nuestro estudio, puesto que cada motor estudiado consta de un número N de ciclos diferente en su simulación. Es necesario asegurar que el GMM reciba un vector de entrada de 100 filas (una por cada motor) y n columnas, correspondientes a las características de los motores a lo largo de los distintos ciclos. Para resolver esta tarea se plantearon 3 estructuras diferentes, pero que responden al mismo objetivo anteriormente explicado: la agrupación estadística, en la que se calculan distintas métricas de la evolución temporal de cada motor, siendo estas las que se tienen en cuenta para la clasificación; el Zero Padding, que consiste en llenar los vectores de los motores con menor número de ciclos con ceros hasta que alcancen la dimensión del motor con mayor número de ciclos; y el Fixed Length Sampling, en el que se seleccionan los últimos N ciclos de la evolución de cada motor, teniendo solo estos en cuenta para la clasificación de salud.

El estudio original llevó a cabo un último paso final, en el que se entrenaba una red neuronal LSTM (Long Short Term Memory) con la que se trató de predecir el valor restante del RUL para cada uno de los motores del set de entrenamiento, comparándolo con los valores del RUL de las simulaciones que vienen dados en los ficheros adicionales. Este paso no se abordó en el presente trabajo y se propone como línea de investigación futura.

#### **4.2.2 Descripción del dataset**

Como se ha explicado anteriormente, los datos del CMAPSS vienen organizados en una carpeta que consiste en 4 casos, denominados FD001, FD002, FD003 y FD004. Cada uno de ellos contiene datos de aviones en vuelos sucesivos. En cada archivo **train** y **test**, cada una de las líneas corresponde a un ciclo (un vuelo completo) y en cada una de ellas se pueden ver

26 columnas, que según el paper de referencia de la NASA representan lo siguiente:

- Una primera columna con un número que identifica el motor.
- Una segunda columna indicadora del número de ciclo.
- Tres columnas que muestran condiciones de operación no conocidas.
- Las 21 columnas restantes corresponden a parámetros de salida de la simulación CMAPSS, que emulan las mediciones de sensores reales del motor turbofán.

Los 21 parámetros utilizados en el estudio son los siguientes [77]:

Símbolo	Descripción [Unidades]
T2	Temperatura total a la entrada del fan [°R]
T24	Temperatura total a la salida del LPC [°R]
T30	Temperatura total a la salida del HPC [°R]
T50	Temperatura total a la salida del LPT [°R]
P2	Presión total a la entrada del fan [psia]
P15	Presión total en el conducto bypass [psia]
P30	Presión total a la salida del HPC [psia]
Nf	Velocidad física del fan [rpm]
Nc	Velocidad física del núcleo [rpm]
epr	Relación de presión del motor (P50/P2)
Ps30	Presión estática a la salida del HPC [psia]
phi	Relación flujo de combustible / Ps30 [pps/psi]
NRf	Velocidad corregida del fan [rpm]
NRc	Velocidad corregida del núcleo [rpm]
BPR	Bypass Ratio
farB	Relación combustible-aire en el quemador
htBleed	Entalpía de purga (bleed)
Nf_dmd	Velocidad del fan demandada [rpm]
PCNfR_dmd	Velocidad corregida del fan demandada [rpm]
W31	Purga de refrigeración HPT [lbm/s]
W32	Purga de refrigeración LPT [lbm/s]

Tabla 4.7. PARÁMETROS DE SALIDA DEL MOTOR TURBOFÁN SIMULADOS EN CMAPSS.

Como se ha mencionado anteriormente, el CMAPSS tiene en cuenta cierta degradación de cada motor en cada ciclo y en alguno de estos ciclos se introducen fallos genéricos que

producen una degradación mayor. Además, se define un índice de salud del motor (Health Index, HI) acotado entre 1 y 0. El HI consiste en una combinación de varios parámetros de operación. Este toma el valor 1 cuando el motor está en condiciones nominales y decrece progresivamente hasta 0 al alcanzarse el fallo [77].

Anteriormente se han mencionado los diferentes ficheros de datos existentes para cada uno de los casos. Adicionalmente, se ha comentado que para el siguiente caso usaremos únicamente los datos presentes en el caso *FD001*. Los datos usados inicialmente son los que encontramos en el dataset de entrenamiento (*Train\_F001.txt*), por lo que se realiza el primer análisis sobre estos mismos datos. Tras una primera inspección de los datos del fichero de entrenamiento FD001 tal y como se realizó en el caso práctico anterior, se obtiene la siguiente información:

Data columns (total 26 columns):				
#	Column	Non-Null Count	Dtype	
0	unit_number	20631	non-null	int64
1	time_in_cycles	20631	non-null	int64
2	op_setting_1	20631	non-null	float64
3	op_setting_2	20631	non-null	float64
4	op_setting_3	20631	non-null	float64
5	sensor_1	20631	non-null	float64
6	sensor_2	20631	non-null	float64
7	sensor_3	20631	non-null	float64
8	sensor_4	20631	non-null	float64
9	sensor_5	20631	non-null	float64
10	sensor_6	20631	non-null	float64
11	sensor_7	20631	non-null	float64
12	sensor_8	20631	non-null	float64
13	sensor_9	20631	non-null	float64
14	sensor_10	20631	non-null	float64
15	sensor_11	20631	non-null	float64
16	sensor_12	20631	non-null	float64
17	sensor_13	20631	non-null	float64
18	sensor_14	20631	non-null	float64
19	sensor_15	20631	non-null	float64
20	sensor_16	20631	non-null	float64
21	sensor_17	20631	non-null	int64
22	sensor_18	20631	non-null	int64
23	sensor_19	20631	non-null	float64
24	sensor_20	20631	non-null	float64
25	sensor_21	20631	non-null	float64
dtypes: float64(22), int64(4)				

Figura 4.7. Información sobre el dataset *train\_FD001.txt* en Python

En este caso se da, como era de esperar, que el número de ciclos y el número del motor son números enteros. En cuanto a los parámetros del motor, encontramos que los parámetros

17 y 18 también son enteros, mientras que del resto de parámetros son decimales.

Además, aplicando aquí la función `df.describe()` se obtienen los siguientes datos:

Variable	Media	Desv. Típica	Mínimo	Máximo	Constante
op_setting_1	0,000	0,002	-0,0087	0,0087	No
op_setting_2	0,000	0,0003	-0,0006	0,0006	No
op_setting_3	100,000	0,000	100,000	100,000	Sí
sensor_1	518,670	0,000	518,670	518,670	Sí
sensor_2	642,681	0,500	641,210	644,530	No
sensor_3	1590,523	6,131	1571,040	1616,910	No
sensor_4	1408,934	9,001	1382,250	1441,490	No
sensor_5	14,620	0,000	14,620	14,620	Sí
sensor_6	21,610	0,000	21,610	21,610	Sí
sensor_7	555,277	1,272	550,500	561,000	No
sensor_8	2388,000	0,000	2388,000	2388,000	Sí
sensor_9	9046,430	22,705	8980,370	9165,500	No
sensor_10	1,300	0,000	1,300	1,300	Sí
sensor_11	47,590	0,014	47,530	47,650	No
sensor_12	521,413	0,738	518,690	523,380	No
sensor_13	2388,096	0,072	2387,880	2388,560	No
sensor_14	8143,753	19,076	8099,940	8293,720	No
sensor_15	8,442	0,038	8,3249	8,5848	No
sensor_16	0,030	0,000	0,030	0,030	Sí
sensor_17	393,211	1,549	388,000	400,000	No
sensor_18	2388,000	0,000	2388,000	2388,000	Sí
sensor_19	100,000	0,000	100,000	100,000	Sí
sensor_20	38,816	0,181	38,140	39,430	No
sensor_21	23,290	0,108	22,8942	23,6184	No

Tabla 4.8. DESCRIPCIÓN ESTADÍSTICA DEL DATASET FD001

De aquí se extrae información muy valiosa, ya que la función `df.describe()` nos muestra el valor medio, desviación típica y los valores mínimo y máximo de cada uno de los parámetros disponibles. De esta forma, si miramos la última columna de la Tabla 4.8, se puede ver cuáles de los parámetros permanecieron constantes durante la simulación, así como aquellos que variaron de forma muy reducida si miramos desviaciones típicas muy bajas. De los 21 parámetros, siete de ellos se mantuvieron constantes durante la simulación, incluso tras producirse el fallo, por lo que pueden considerarse no relevantes a la hora de introducirlos

en el modelo, disminuyendo notablemente el número de dimensiones. Esto se discutirá en mayor detalle más adelante.

#### 4.2.3 Preprocesamiento y división del dataset

El preprocesamiento consistirá simplemente en la normalización de los datos de entrada. Este es un paso muy importante en el análisis, especialmente cuando se entrenan modelos basados en redes neuronales. Se ha demostrado que las redes neuronales pueden ajustar los pesos de las neuronas de forma desigual cuando las variables de entrada muestran escalas de valores muy diferentes, alterando así la certeza de las predicciones del modelo. Cuando los datos no se normalizan, las redes tienden a asignar mayor importancia a las variables con escalas de valor más elevadas, ignorando aquellas que presentan magnitudes más reducidas. Al normalizar los inputs, la red reconoce entradas con escalas de valores similares, tratando los datos de manera equitativa [78].

Para este caso concreto se usó el normalizador de la librería **Scikit-Learn MinMaxScaler**. Este método transforma cada variable para que sus valores queden contenidos en un intervalo definido por dos límites  $a$  y  $b$ . Se decidió usar  $b = 1$  y  $a = 0$ , de forma que el valor más alto de la variable  $x_{max}$  se mapea al valor 1, mientras que el valor más bajo  $x_{min}$  se mapea al valor 0. El resto de valores  $x$  se transforman a su valor normalizado  $x'$  como sigue:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \cdot (b - a) + a \quad (4.2)$$

siguiendo la metodología de Lodygowski, se optó por mantener todas las variables, incluso aquellas prácticamente constantes, dado que posteriormente se incorporarán datasets adicionales con posibles valores distintos. La eliminación temprana de atributos podría generar inconsistencias en la dimensionalidad entre datasets. En caso de que esto ocurriese, la eliminación de ciertas variables para el dataset de entrenamiento puede no ser útil para el dataset de testeo, aunque ambos deben recibir las mismas modificaciones para mantener el número de dimensiones.

#### 4.2.4 Entrenamiento del Autoencoder

De cara al entrenamiento del autoencoder se buscaron una serie de parámetros óptimos que permitieran la compresión de los datos con el menor error posible. A pesar de que Lodygowski muestra una serie de parámetros para los que se obtiene un error mínimo, estos son solo válidos para el dataset real que se analizó en su estudio.

Inicialmente se trató de usar ese mismo método pero tras varios intentos con distintas configuraciones se comprobó que estas configuraciones no resultaban adecuadas para nuestro

caso de estudio. En primer lugar, se evidenció un problema en la compresión de datos en la capa latente (bottleneck) del autoencoder. El set de datos reales del método original consta de unos 86 parámetros, por los 24 disponibles en el dataset del CMAPSS. Los datos óptimos de entrenamiento según Lodygowski fueron:

Parámetro	Valor
Dataset de entrada	Datos reales (86 parámetros)
Número de capas ocultas	4 capas de 128 neuronas (simetría encoder-decoder)
Tamaño de la capa latente	32
Función de activación	ReLU (ocultas), Linear (salida)
Función de pérdida	MSE (Error cuadrático medio)
Optimización	Adam
Tasa de aprendizaje	0.001
Dropout	0.2
Épocas de entrenamiento	400
Criterio de parada	Early stopping con paciencia de 10

Tabla 4.9. PARÁMETROS ÓPTIMOS DEL AUTOENCODER SEGÚN  
LODYGOWSKI ET AL. PARA EL SET DE DATOS DE MOTORES  
REALES

Como se ve, el autor se inclina por el uso del optimizador Adam para el entrenamiento del autoencoder. Como se explicó en el Sección 3.9.2 este algoritmo combina una buena adaptabilidad en la tasa de aprendizaje con estabilidad en la convergencia, lo que es especialmente ventajoso en redes neuronales profundas como la que se muestra en el artículo, donde los gradientes pueden variar mucho entre capas. En el caso del autoencoder, esta característica permite ajustar de forma efectiva cada peso de la red, acelerando el aprendizaje y evitando oscilaciones excesivas en la función de pérdida. A pesar de las modificaciones realizadas para la réplica del método explicado, se optó por mantener Adam como optimizador de nuestro Autoencoder. La tasa de aprendizaje  $\alpha$  se eligió igual a 0.001, mientras que el resto de parámetros se mantuvieron por defecto según vienen configurados en la librería TensorFlow ( $\beta_1 = 0.9$ ,  $\beta_2 = 0.99$  y  $\varepsilon = 10^{-7}$ , ver Ecuación 3.22-3.25).

La red neuronal utiliza como función de activación ReLU y Linear en la salida. La función ReLU (Rectified Linear Unit) se define como  $f(x) = \max(0, x)$  [79]. Esta simplemente transforma cualquier valor negativo a cero, dejando pasar únicamente los valores positivos, que permanecen intactos. Para casos como este, favorece el aprendizaje rápido y estable, evitando el problema de saturación del gradiente que se presenta en funciones como la sigmoide, la cual comprime los valores al rango  $[0, 1]$  y produce gradientes muy pequeños en los extremos, causando la ralentización de la convergencia [80]. Además, al anular todas las entradas neg-

ativas, reduce el número de neuronas activas en cada capa, lo que ayuda a concentrar el aprendizaje en las características más relevantes y a filtrar ruido. La capa de salida usa una función de activación lineal para no delimitar los outputs y reproducir los valores de los parámetros de entrada de forma más realista.

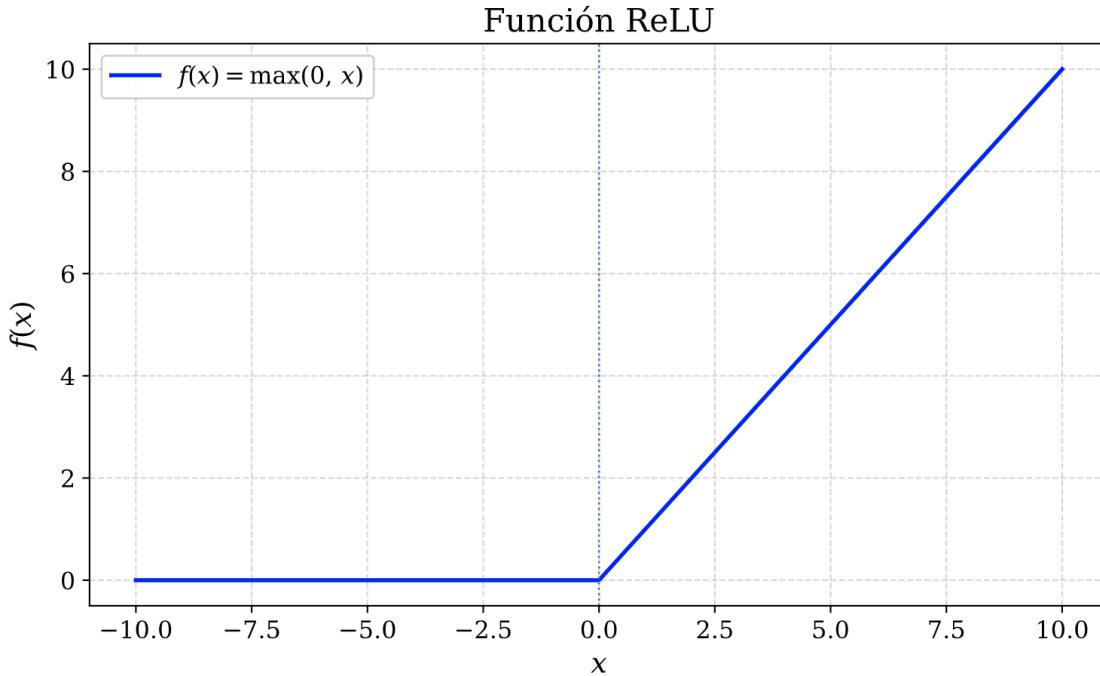


Figura 4.8. Función ReLU (Fuente: elaboración propia).

La Tabla 4.9 muestra también como el tamaño de los datos comprimidos es de 32 dimensiones, el cual es mayor que el de nuestro número de dimensiones original. En este caso, dicha configuración no resulta adecuada, ya que en lugar de reducir la dimensionalidad, estaría expandiendo el número de variables. Esto no nos interesa ya que el objetivo del AE es la reducción dimensional, obteniendo un conjunto de atributos relevantes para la clasificación de los motores.

Por tanto, tras las primeras simulaciones y resultados, se descartó esta estructura de autoencoder y se buscó una nueva formulación con un número menor de dimensiones latentes. Las dimensiones latentes coinciden con el número de neuronas que se utilizan en la capa latente o *bottleneck* de la red de neuronas que componen el Autoencoder.

Por tanto, se inicia aquí un proceso iterativo para obtener un autoencoder capaz de reproducir los datos con la mayor fidelidad posible, con la restricción de que el número de neuronas latentes debe ser menor que 24, correspondiente al número de dimensiones que aparecen en el dataset del CMAPSS. Para ello, se configuró una estructura de parámetros en Python con las que se evaluaba la reconstrucción de los datos en forma de bucle, construyendo

diferentes redes con los parámetros definidos y hallando el error de reconstrucción al final de las iteraciones.

Además de la reducción de neuronas en el cuello de botella, entre las consideraciones tomadas frente a las que realizaron Lodygowski et al. para el caso de motores reales, encontramos las siguientes:

- Se decidió reducir las dos capas de entrada y salida por una única capa en cada extremo, para reducir el número de conexiones y parámetros, ya que nuestro dataset tiene muchos menos parámetros que el dataset de motores reales. Estas capas de entrada y salida también vieron una reducción en su número de neuronas.
- Se introdujo un forzado de las iteraciones en caso de que el algoritmo mostrara convergencia temprana, lo que se conoce como Early Stopping. Este método se programó para pausar las iteraciones cuando se daban 20 iteraciones seguidas sin un descenso significativo del error de reconstrucción.
- Como consecuencia del punto anterior, se redujo el número de iteraciones, ya que prácticamente en ningún caso se llegaba a más de 70 iteraciones.

Los parámetros modificables que se seleccionaron para estas simulaciones fueron el número de neuronas, el número de iteraciones (también llamado número de épocas), número de neuronas en las capas de entrada y salida, parámetro de dropout. Tras probar con diferentes combinaciones de parámetros, se programó el código para generar gráficas con la pérdida de información entre entrada y salida. Esta función de pérdida es la conocida como pérdida de entrenamiento o *Train Loss*. Una comparativa de diferentes combinaciones de parámetros para el entrenamiento puede verse en la siguiente gráfica:

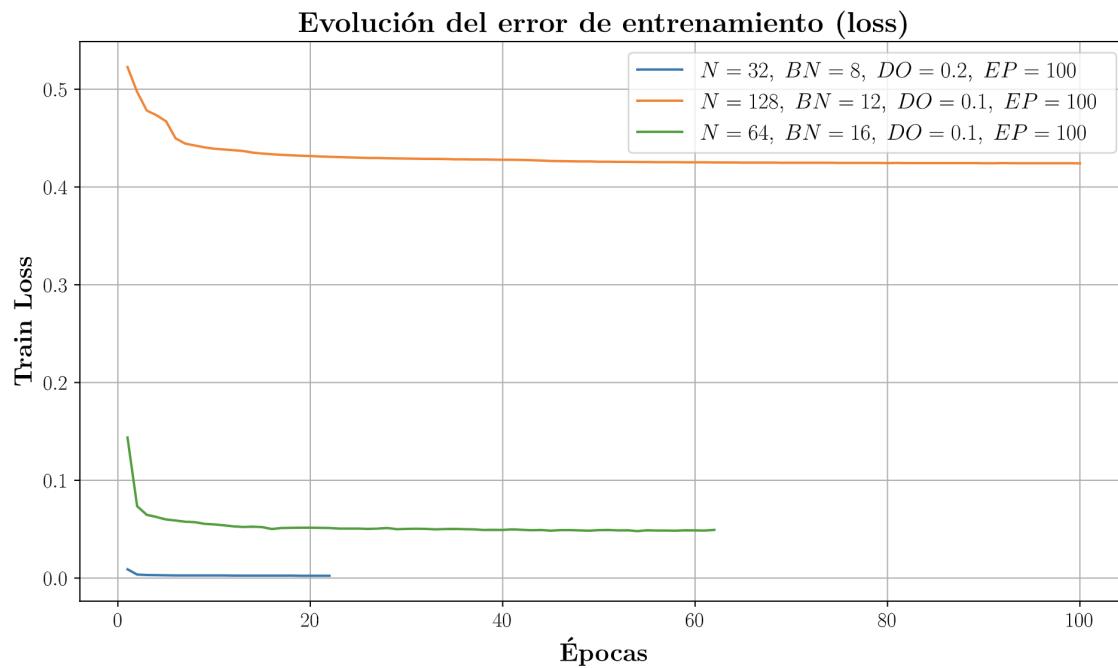


Figura 4.9. Evolución del error con el número de épocas para tres configuraciones distintas del AE  
(Fuente: elaboración propia).

Esta muestra tres progresiones para tres casos con un número máximo de 100 épocas (EP) cada uno en los que varía el número de neuronas de las capas de entrada y salida (N), el número de dimensiones en el bottleneck (BN) y el índice de dropout (DO). Como se observa, a medida que se disminuyó el número de neuronas, y por tanto el de parámetros a optimizar en la red neuronal, el error disminuyó drásticamente. Esto demuestra que debe realizarse una reducción paramétrica acorde a la disminución de parámetros de entrada de nuestro dataset, ya que redes con una cantidad de neuronas demasiado elevadas muestran comportamientos inefficientes para un número bajo de dimensiones del problema.

Los errores (error cuadrático medio) se mostraron mucho menores en casos con menor número de neuronas, presentando también menores disminuciones con respecto a la primera iteración. Además, se puede ver como el número de iteraciones totales del máximo de 100 también se reduce en el caso del entrenamiento con 32 neuronas, que solo llega a las 22 iteraciones, frente a las 62 del caso de las 64 neuronas. Es importante recordar que estos casos no llegan a las 100 iteraciones debido a la aplicación de Early Stopping cuando se obtiene un error muy similar durante 20 iteraciones consecutivas. El entrenamiento con capas de 128 neuronas fue el único en alcanzar las 100 iteraciones, demostrando que el error aún seguía disminuyendo lentamente. La siguiente tabla muestra una comparativa más detallada:

ID	Configuración	Loss inicial	Loss final	Épocas
1	N=32, BN=8, DO=0.2, EP=100	0.009022	0.002402	22
2	N=128, BN=12, DO=0.1, EP=100	0.522723	0.424208	100
3	N=64, BN=16, DO=0.1, EP=100	0.143758	0.049250	62
4	N=32, BN=8, DO=0.1, EP=100	0.167326	0.071754	78

Tabla 4.10. COMPARATIVA ENTRE CONFIGURACIONES DE AUTOENCODER

Por otra parte, comparando el mejor caso anterior con otro con los mismos parámetros pero con un dropout rate de 0.1 en vez de 0.2 (ID 2 contra ID 1), se ve como el primer caso bate al segundo en términos de error.

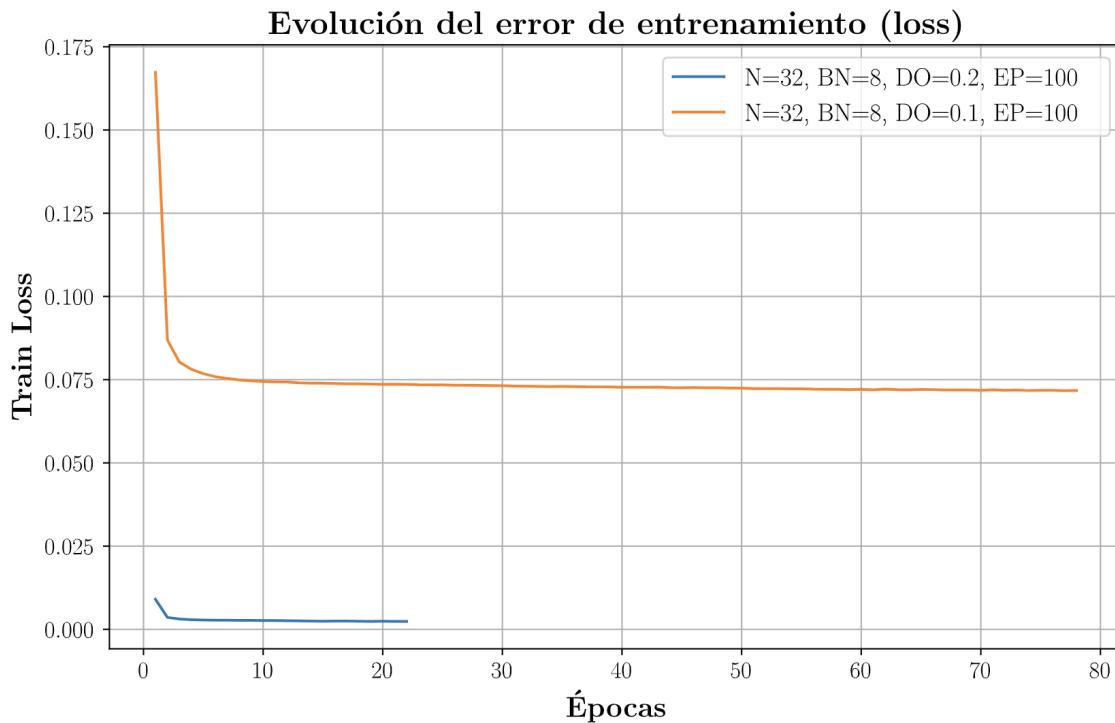


Figura 4.10. Evolución del error con el número de épocas para las dos mejores configuraciones

Aquí se ve como el hecho de desactivar mayor número de neuronas durante cada fase del entrenamiento produce un error inicial y final mucho menor. Incluir un dropout de 0.1 en el caso 4, cuyas características restantes son idénticas al caso 1, produjo una reconstrucción ligeramente pero que el caso 3, en el que se contaban con 64 neuronas y 16 dimensiones de bottleneck. Cabe destacar que todos los datos presentados mostraron errores relativamente bajos. La mejor representación de los datos tras el autoencoder se produjo en el caso 1, para el que contamos con capas de 32 neuronas en el codificador y descodificador, una dimensión

latente de 8 neuronas, un dropout de 0.2 y un entrenamiento que dejó de mejorar a las 22 iteraciones. A partir de ahora todos los datos pasará por este autoencoder correspondiente con la primera fila (ID 1) de la Tabla 4.10.

Usando el comando de Keras `model.summary()`, se aprecia que la red neuronal del autoencoder finalmente presenta la siguiente configuración:

Capa	Tipo	Número de neuronas
1	Input	24
2	Encoder	32
3	Dropout	32
4	Bottleneck	8
5	Decoder	32
6	Output	24

Tabla 4.11. COMPOSICIÓN EN CAPAS DEL AUTOENCODER

Esta es la composición común usada en los diferentes entrenamientos, donde las capas de entrada (input) y salida (output) eran constantes y de dimensiones idénticas a las del problema; las capas de codificación y decodificación eran variables e iguales al parámetro N y la capa latente (bottleneck) corresponde al parámetro variable BN (ver Tabla 4.10). Por último, se observa la localización de la capa de Dropout, donde se produce la desactivación del número de neuronas dado por el índice de dropout. En este caso se optó por colocar la capa de Dropout tras la capa de codificación y previa a la capa latente. De esta forma se desactiva el 20 por ciento de las neuronas de la capa del encoder, forzando a la red a no depender demasiado de ciertas neuronas y distribuyendo la información de forma más equilibrada. Esto genera una representación más robusta de las dimensiones latentes, como se ha demostrado en otros estudios [81].

#### 4.2.5 Consolidación de la matriz de entrada y clustering por GMM

En este paso se introduce el uso de un GMM para la clasificación probabilística de los diferentes motores. Hasta este punto se usó el archivo de entrenamiento `train_FD001.txt` para la construcción del autoencoder. Para este siguiente paso se utilizará el fichero `test_FD001.txt`.

Volviendo a la Figura 4.6, se observa como todo el flujo relacionado con este segundo archivo de test viene representado mediante flechas de color naranja. Como puede verse, inicialmente se alimenta el autoencoder con este archivo de test, cuyo número de variables es idéntica al archivo de entrenamiento. En este caso, nuestro archivo `train_FD001.txt` consta de datos simulados de 100 motores diferentes, con diferente número de ciclos por motor.

Un punto muy importante de esta fase del estudio es que tras pasar por el autoencoder, necesitamos extraer el valor de las dimensiones latentes, que son las relevantes para la clasificación de los diferentes motores. Estas dimensiones latentes contienen la representación comprimida de los datos de entrenamiento, concentrando las 24 variables iniciales en 8 (capa del bottleneck) que captan la información más relevante de los datos.

Para ello se define en el código implementado en Python una función que genera un archivo .h5 correspondiente al modelo del autoencoder. Estos archivos se guardan con distintos nombres, definidos según los parámetros de entrenamiento, de forma que puedan recuperarse posteriormente. Una vez se encontró la mejor combinación explicada en la sección anterior, se importa el modelo y la hoja de datos de entrenamiento. Este nuevo set de datos debe normalizarse tal y como se hizo para el set de entrenamiento (ver Ecuación 4.2) y posteriormente se le aplica la transformación del autoencoder. Además, cada capa del modelo recibe un identificador con el que es posible recuperar la información tal y como aparecía en una capa determinada. Aquí seleccionamos la capa bottleneck, obteniendo la representación latente de los datos de entrenamiento.

Una vez obtenidos estos datos obtenemos un dataset con el mismo número de filas que el original, pero únicamente con 8 columnas. Aquí se muestra la evolución de cada motor a lo largo del tiempo, incluyendo todos sus ciclos, pero con la dimensionalidad reducida al espacio latente. Aquí es donde aparece el siguiente desafío, que consiste en cómo tratar los datos obtenidos para realizar una clasificación única por motor.

La idea planteada por Lodygowski pasa por unir todos los datos de correspondientes a un mismo motor en un único vector. Considerando que en el dataset de datos comprimidos, cada motor tiene:

- Medidas a lo largo de  $N$  ciclos
- 8 dimensiones por ciclo (representación comprimida)

Se busca crear un vector de longitud  $N \times 8$  que agrupe todos los datos de un motor en una misma fila. Una vez se hace esto para cada motor, sabiendo que no todos los motores tienen el mismo número de ciclos, se debe ajustar de alguna manera estas filas para obtener un dataset con una longitud de filas consistente. Para ello, se investigaron las siguientes estrategias que se presentan a continuación, cada una con sus resultados.

El GMM utilizado para la clasificación probabilística usó los parámetros por defecto para este tipo de modelos, asignando manualmente el número de grupos en los que se pretenden clasificar los 100 motores. Este es el mismo clasificador que utiliza Lodygowski en el estudio que se usa de referencia. En su artículo, se definen tres grupos con la intención de asignar cada motor a las categorías: Saludable (*Healthy*), Intermedio (*Middle*) y No Saludable (*Unhealthy*).

## Agragación estadística

Se realizó un primer estudio simplificando las variables de cada motor en un conjunto de estadísticas descriptivas (media, desviación típica y valor máximo) diferenciadas para cada equipo. Este es un caso sencillo, en el que se decidió tener en cuenta la media, desviación típica y el valor máximo de cada variable a lo largo de los ciclos para cada motor.

Esta es una buena solución de cara a la disminución de la dimensionalidad, que ayuda a la interpretación del modelo gausiano. Sin embargo, esta aproximación implica la pérdida completa de la variabilidad temporal, ya que todas las mediciones se reducen a valores estadísticos agregados.

Partimos de nuestros datos transformados y comprimidos tras pasar por el codificador del Autoencoder, que nos da la matriz de 100 motores con  $N$  ciclos por motor y 8 dimensiones correspondientes al espacio latente. De aquí simplemente se programó un código en python que separara cada motor según su identificador, y calculaba las tres estadísticas explicadas anteriormente para cada una de las variables de los motores.

Se obtiene por tanto un vector de 24 columnas para cada motor, organizado como sigue:

**Dataset de Agregación Estadística**

ID	Media			Desv. Típica			Valor Máximo		
1	$\bar{x}_1^{(1)}$	...	$\bar{x}_1^{(8)}$	$\sigma_1^{(1)}$	...	$\sigma_1^{(8)}$	$m_1^{(1)}$	...	$m_1^{(8)}$
2	$\bar{x}_2^{(1)}$	...	$\bar{x}_2^{(8)}$	$\sigma_2^{(1)}$	...	$\sigma_2^{(8)}$	$m_2^{(1)}$	...	$m_2^{(8)}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
100	$\bar{x}_{100}^{(1)}$	...	$\bar{x}_{100}^{(8)}$	$\sigma_{100}^{(1)}$	...	$\sigma_{100}^{(8)}$	$m_{100}^{(1)}$	...	$m_{100}^{(8)}$

Tabla 4.12. REPRESENTACIÓN DEL DATASET TRAS  
AGREGACIÓN ESTADÍSTICA, USANDO MEDIA, DESVIACIÓN  
TÍPICA Y VALOR MÁXIMO PARA CADA VARIABLE LATENTE.

Donde cada ítem  $X_i^{(j)}$  hace referencia a la variable estadística  $X$  de la dimensión latente ( $j$ ), para el motor  $i$ . Cada vector contiene primero los ocho valores medios de cada variable ( $j$ ), seguidos de los ocho valores de desviación típica y los ocho valores máximos de cada una de las dimensiones latentes. Así se obtiene finalmente una matriz de 100 filas por 24 columnas, lo que mantiene una dimensionalidad reducida en comparación con los otros dos enfoques, que serán explicados más tarde.

Una vez obtenida la matriz de entrada uniforme, con mismo número de columnas para cada motor, se pasa a entrar el clasificador GMM, del que se obtienen los siguientes clusters:

Cluster	Número de motores
0	55
1	13
2	32

Tabla 4.13. DISTRIBUCIÓN DE MOTORES POR CLUSTER SEGÚN CLASIFICACIÓN GMM USANDO AGREGACIÓN ESTADÍSTICA

Como se ve en la Tabla 4.13, se obtuvieron tres clusters como se especificó en la configuración del modelo gausiano mixto. El primer cluster agrupó el 55 por ciento de los motores, el segundo el 13 por ciento y el tercero el 32 por ciento restante. Como se ve, se obtiene un total de 100 motores, logrando el objetivo principal de clasificar cada motor una única vez. Una buena manera de observar la clasificación por cluster es la reducción de las 24 variables en dos dimensiones únicas (llamadas  $PCA_1$  y  $PCA_2$ ) mediante análisis de componentes principales (PCA, ver sección Reducción de la dimensionalidad). De esta forma se crean dos variables artificiales como combinación lineal de las 24 variables disponibles en este caso. Estas dos variables capturan la influencia del resto de variables y nos permite obtener un gráfico en función de estas dos, localizando cada motor en un gráfico bidimensional.

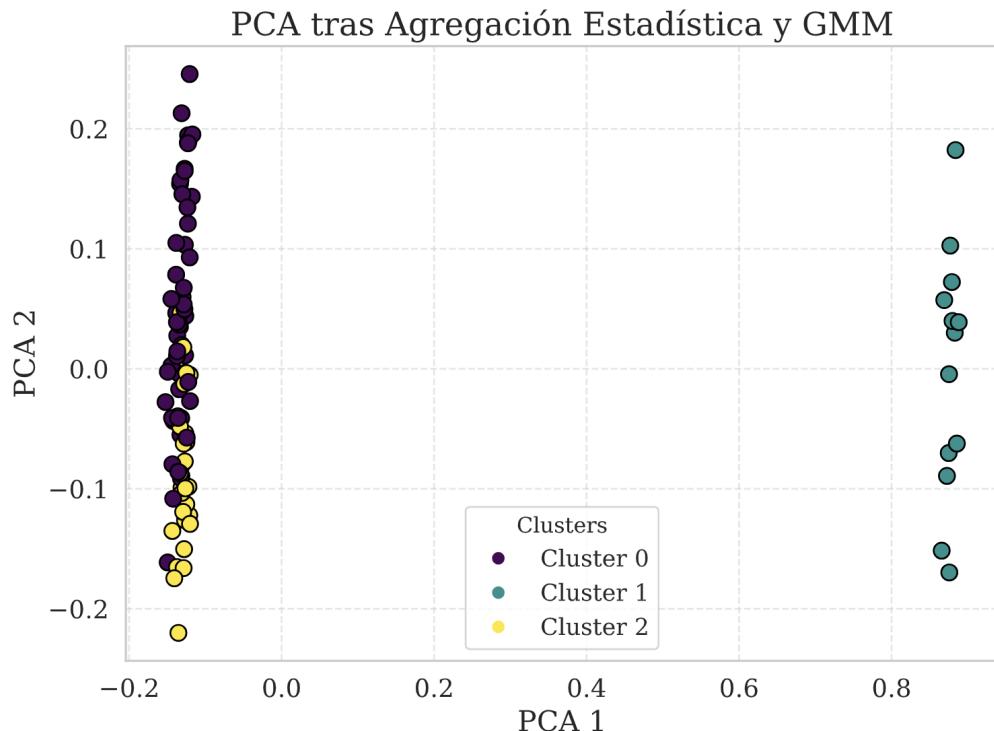


Figura 4.11. Análisis de Componentes Principales de la clasificación con Agregación Estadística (Fuente: elaboración propia).

Además se representó cada motor con el color del cluster correspondiente, pudiendo ver

las diferencias entre clusters de forma visual. En este primer caso, se obtuvieron dos grandes grupos:

- Un grupo mayoritario formado por 88 motores pertenecientes a los clusters 0 y 2, parcialmente superpuestos. Los puntos se agrupan alrededor de  $PCA_1 = -0.1$ . La agrupación es vertical a lo largo de la segunda variable, quedando la mayoría de motores del cluster 0 por encima de los del cluster 2.
- Un grupo más reducido en el otro extremo del gráfico, con puntos únicamente del cluster 1, distribuidos también verticalmente a la altura de  $PCA_1 = 0.9$

Como se ve aquí, la clasificación en torno a los dos componentes principales no mostró resultados demasiado relevantes. Además, buscando semejanzas con los resultados del artículo de Lodygowski et al., no se encontraron parecidos significativos.

Observando la diferenciación mostrada por la gráfica tras aplicar PCA, se puede concluir que el dataset formado mediante agregación estadística reduce la dimensionalidad hasta un punto en el que las clasificaciones se vuelven muy cercanas a binaria, diferenciándose dos grandes grupos en base a alguna variable. Analizando los datos de entrada del GMM junto con la clasificación realizada por el modelo, encontramos los siguientes datos relevantes:

- Los datos correspondientes al valor máximo de la variable 6 clasifican binariamente, obteniendo 1 para todos los datos clasificados como 1 y 0 para todos los puntos de los clusters 0 y 2.
- Lo mismo ocurre con la desviación típica media de la variable 6.

cluster	$m_6$	$\sigma_6$	$\bar{x}_1$	$\bar{x}_7$	$\sigma_1$	$\sigma_7$	$m_1$	$m_7$
0	0.0000	0.0000	0.5309	0.3710	0.2010	0.2230	0.9895	0.9781
1	1.0000	0.0914	0.6257	0.4037	0.1886	0.2393	1.0000	1.0000
2	0.0000	0.0000	0.6389	0.4903	0.1573	0.2050	0.9860	0.9833

Tabla 4.14. COMPARACIÓN DE MEDIAS POR CLUSTER DE LAS VARIABLES MÁS DISCRIMINANTES EN EL DATASET DE AGREGACIÓN ESTADÍSTICA

Esto genera que la variable 6 se convierta en la más relevante en cuanto a la clasificación realizada. La tabla Tabla 4.14 muestra las diferencias fácilmente apreciables entre las variables destacadas  $m_6$  y  $\sigma_6$ , y el resto de variables de las dimensiones 1 y 7 (escogidas

únicamente con el propósito de realizar esta comparación). Como se aprecia, el resto de dimensiones presentan variables estadísticas para las que resulta más complicado obtener un patrón concreto, mientras que en  $m_6$  y  $\sigma_6$  las diferencias entre el cluster 1 y los cluster 0 y 2 convierten la clasificación en una decisión casi binaria, que se traduce posteriormente en la distribución mostrada en la Figura 4.11.

## Zero Padding

El concepto de *Padding* es muy usado en muchas ramas de la ingeniería de software y del aprendizaje automático. El término *Padding* significa literalmente ‘relleno’ y hace referencia a la estrategia de sustituir valores faltantes o incompletos por otros definidos, generalmente ceros u otro valor fijo. Inicialmente, este concepto de padding surge en el contexto de programación de sistemas para obtener un número de bits acorde al especificado en un protocolo de comunicación. Por ejemplo, si una estructura de datos ocupa 12 bits, pero el sistema requiere utilizar dos bytes para este tipo de estructuras, el propio sistema puede llenar con ceros los cuatro bits faltantes para evitar inconsistencias.

DATA STRUCTURE (16 bits)																
Filled Structure														Padding bits		
1	0	1	1	1	0	0	1	0	0	0	0	1	0	0	0	0

Figura 4.12. Ejemplo que muestra el concepto de Post Padding con ceros en una estructura de datos (Fuente: elaboración propia).

Existen muchos tipos diferentes de padding para llenar estructuras de datos. En nuestro caso, se decidió utilizar Zero Post-Padding, que consiste simplemente en sustituir valores vacíos al final de una estructura por ceros. En nuestro caso, como ya se ha visto en los dos casos anteriores, este proceso nos permite igualar la longitud de las cien secuencias de motores, ya que cada una cuenta con un número de ciclos distintos.

En nuestro caso, cada motor  $i$  tiene una secuencia de representaciones comprimidas por el autoencoder  $M_i^{(j)}$  (siendo  $j$  el índice de ciclo). Supongamos que el motor que más ciclos obtiene llega a 10 ciclos. Si un motor dispone de menos observaciones, las posiciones faltantes se llenan con vectores nulos, de manera que todas las filas queden con la misma dimensionalidad. Así, el conjunto se transforma en una matriz homogénea que puede ser utilizada por nuestro modelo GMM, que requiere una entrada de tamaño fijo. Al igual que los casos anteriores, esta estrategia también presenta desventajas. En este caso la principal es que este

relleno introduce tramos sin información física (ceros) condicionando el comportamiento del modelo [82].

	<b>C<sub>1</sub></b>	<b>C<sub>2</sub></b>	<b>C<sub>3</sub></b>	<b>C<sub>4</sub></b>	<b>C<sub>5</sub></b>	<b>C<sub>6</sub></b>	<b>C<sub>7</sub></b>	<b>C<sub>8</sub></b>	<b>C<sub>9</sub></b>	<b>C<sub>10</sub></b>
<b>Motor 1</b>	$M_1^{(1)}$	$M_1^{(2)}$	$M_1^{(3)}$	$M_1^{(4)}$	$M_1^{(5)}$	ZP	ZP	ZP	ZP	ZP
<b>Motor 2</b>	$M_2^{(1)}$	$M_2^{(2)}$	$M_2^{(3)}$	ZP						
<b>Motor 3</b>	$M_3^{(1)}$	$M_3^{(2)}$	$M_3^{(3)}$	$M_3^{(4)}$	$M_3^{(5)}$	$M_3^{(6)}$	$M_3^{(7)}$	$M_3^{(8)}$	ZP	ZP
⋮										
<b>Motor 98</b>	$M_{98}^{(1)}$	$M_{98}^{(2)}$	$M_{98}^{(3)}$	$M_{98}^{(4)}$	$M_{98}^{(5)}$	$M_{98}^{(6)}$	$M_{98}^{(7)}$	$M_{98}^{(8)}$	$M_{98}^{(9)}$	$M_{98}^{(10)}$
<b>Motor 99</b>	$M_{99}^{(1)}$	$M_{99}^{(2)}$	$M_{99}^{(3)}$	$M_{99}^{(4)}$	$M_{99}^{(5)}$	$M_{99}^{(6)}$	$M_{99}^{(7)}$	$M_{99}^{(8)}$	ZP	ZP
<b>Motor 100</b>	$M_{100}^{(1)}$	$M_{100}^{(2)}$	$M_{100}^{(3)}$	$M_{100}^{(4)}$	ZP	ZP	ZP	ZP	ZP	ZP

Tabla 4.15. EJEMPLO DE ZERO-PADDING CON UN MÁXIMO DE 10 CICLOS POR MOTOR.

La tabla Tabla 4.15 muestra un ejemplo simplificado en el que cada **C<sub>i</sub>** representa un ciclo, con sus 8 variables comprimidas. Suponiendo que el motor 98 es el más longevo, llegando a simularse un total de 10 ciclos para él, el resto de líneas correspondientes a los demás motores deben incluir un vector formado por ocho ceros por ciclo, de forma que cada ZP corresponde a uno de estos vectores de ceros de longitud ocho.

Esto era solo un caso simplificado, para mostrar la aplicación del padding en un ejemplo ilustrativo similar a nuestro caso real. En la práctica, el motor con mayor número de simulaciones fue el motor 49, con 303 ciclos. El resto, de motores, con una media de 206 ciclos por motor, se llenaron con ceros hasta alcanzar el número de ciclos máximo. Esto corresponde a una media de 97 ciclos de padding por motor, es decir, un valor promedio de 776 valores nulos añadidos por motor.

Una vez rellenado el dataframe con valores nulos, se obtiene por tanto una matriz rectangular, de 2424 filas y 100 columnas, que se introduce en el GMM, con las propiedades explicadas anteriormente. En el caso de la agregación con Zero Padding, se obtuvieron los siguientes resultados de clasificación:

Cluster	Número de motores
0	45
1	25
2	30

Tabla 4.16. DISTRIBUCIÓN DE MOTORES POR CLUSTER SEGÚN CLASIFICACIÓN GMM USANDO ZERO PADDING

Por medio de esta estrategia, se clasificaron 45 motores en el cluster cero, 25 en el cluster uno y 30 en el cluster 2. Una vez más, se desconoce exactamente qué cluster representa cada

condición, pudiendo visualizar únicamente los motores clasificados en cada uno de ellos.

Una vez más, se aplica PCA para obtener una gráfica en torno a las dos variables más relevantes. De esta forma se visualiza el agrupamiento realizado en torno a estas dos variables. La gráfica PCA obtenida es la siguiente:

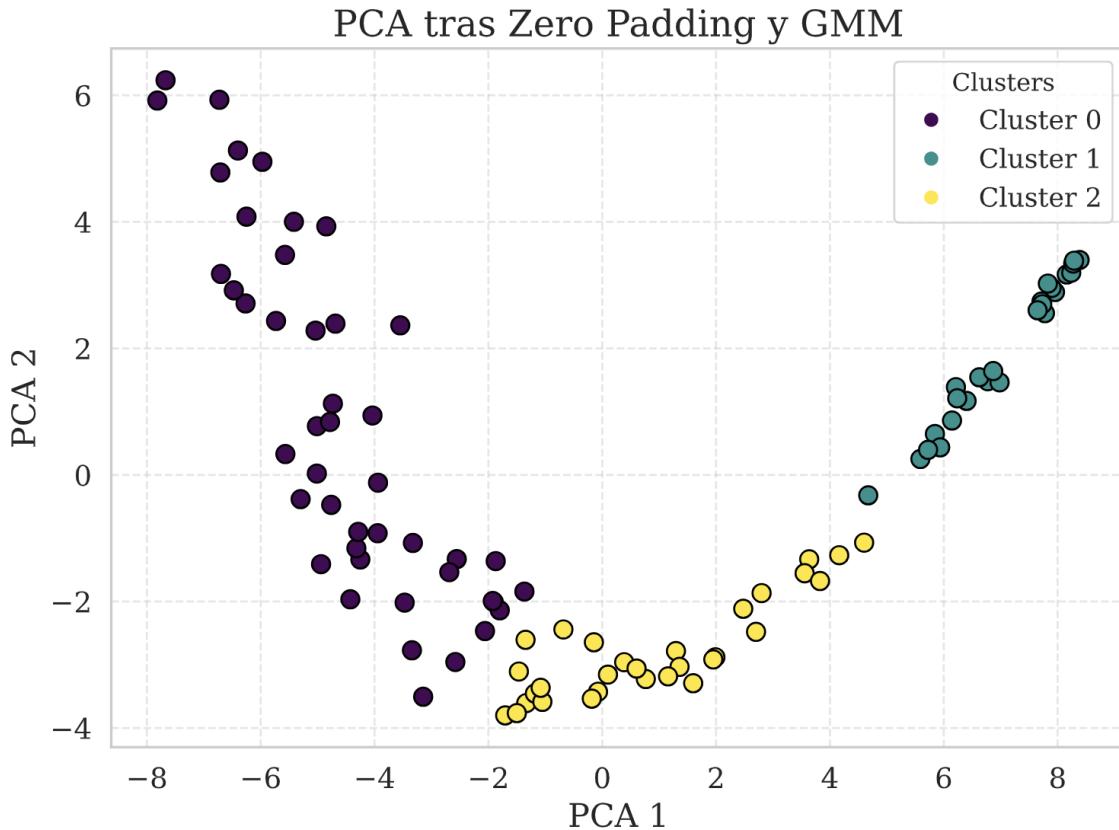


Figura 4.13. Análisis de Componentes Principales de la clasificación con Zero Padding (Fuente: elaboración propia).

El diagrama muestra como los tres grupos se distribuyen a lo largo de las dos dimensiones de la siguiente manera:

- El cluster 0 a lo largo de la dimensión PCA 2 avanzando hacia valores mayores de PCA 1. Este grupo es el más extenso y disperso, sin concentrarse sus puntos en una región concreta.
- El cluster 1 se concentra en el lado derecho de la gráfica, en un cuadrante delimitado casi en su totalidad por  $\text{PCA 1} = [5,8]$  y  $\text{PCA 2} = [0,4]$ . Aquí encontramos el menor número de puntos en una región bastante bien acotada.
- El cluster 2 aparece en la región inferior central de la representación. Su distribución es intermedia entre los dos clusters anteriores, ni tan concentrada como el cluster 1 ni

tan dispersa como el cluster 0.

Del estudio de Lodygowski et al. se puede obtener una gráfica similar, también resultante del estudio del dataset `train_FD001.txt`:

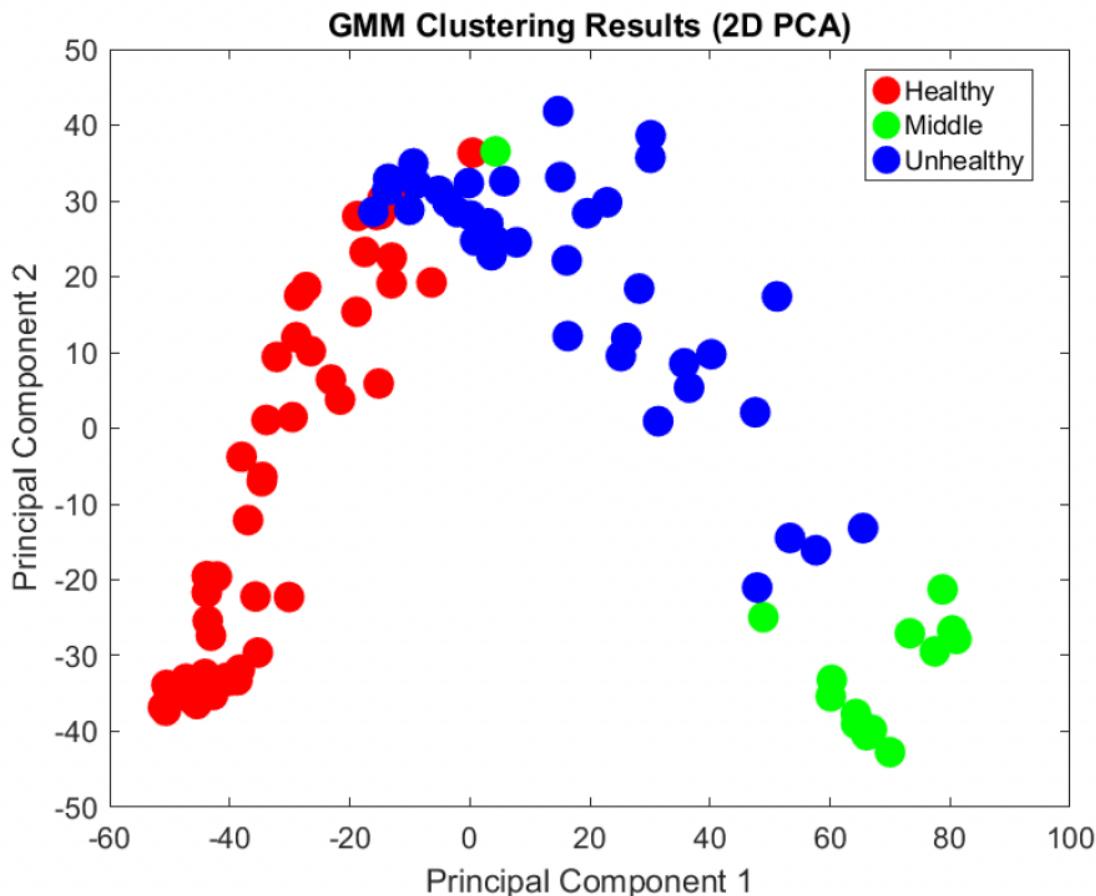


Figura 4.14. PCA de los resultados obtenidos por Lodygowski et al. para los datos del CMAPSS (Fuente: [36]).

Aquí es posible observar como el autor clasifica los diferentes grupos según los estados de salud. La gráfica presenta una forma similar a la de nuestro estudio, tras una inversión con respecto al eje X. Además, se pueden ver también tres grupos con distribuciones similares a las previamente explicadas, siendo el cluster 0 similarmente disperso al catalogado como Unhealthy; el cluster 2, con una dispersión menor, semejante a su categoría Healthy y un pequeño cluster más contenido en una región más delimitada, catalogado como Middle, que se corresponde con nuestro cluster 1. Además, el número de motores que clasificaron para cada cluster en el paper fue de 58 para healthy (45 en nuestro caso), 11 para Middle (25 en nuestro caso) y 31 para Unhealthy (30 en nuestro caso), siendo las cifras significativamente similares en ambos casos, a pesar de las potenciales diferencias en los procesos.

## Fixed Length Sampling

La última estrategia que aplicada para obtener una representación con mismo número de columnas por motor consiste eliminar un número determinado de ciclos de simulación de cada uno de los motores. El Fixed Length Sampling (FLS) o Muestreo de Longitud Fija se basa en recoger únicamente un número idéntico de ciclos para cada motor y concatenarlos en un único vector por motor, teniendo todos estos vectores el mismo tamaño.

En este caso, se decidió tomar los últimos  $k$  ciclos por motor, de manera que se muestre la evolución final de cada uno de los especímenes. El GMM solo tendrá en cuenta estos últimos ciclos, ignorando el resto de ciclos anteriores. Para seleccionar el número de ciclos a tener en cuenta, se debe observar el número mínimo de ciclos experimentados, en el motor con la simulación más corta. En el dataset de entrenamiento, este corresponde al motor 51, con un total de 31 ciclos. En general, los 100 motores presentan un número de ciclos bastante superior a este número mínimo, estando la media como se vio anteriormente en 206 ciclos por motor. Por tanto, se decidió usar como valor fijo una muestra de  $k = 75$  ciclos. Se calculó el número de motores cuya cantidad de ciclos eran menores a 75, obteniendo:

Condición	Número de motores (%)
Menos de 75 ciclos	20 (20.00%)
Menos de 70 ciclos	14 (14.00%)
Menos de 50 ciclos	7 (7.00%)

Tabla 4.17. DISTRIBUCIÓN DE MOTORES SEGÚN EL NÚMERO DE CICLOS REGISTRADOS

Como se puede apreciar, el número de motores cuyo número de repeticiones es menor que 75 es de 20 motores. A pesar de que esta es una cifra significativa, solo el 14 por ciento de los motores contaban con menos de 14 ciclos, mientras que solo 7 de los cien tenían menos que 50. Por tanto, con la intención de mantener la mayor parte de la información posible, se decidió mantener la cifra de 75 ciclos de muestreo.

Una vez fijado el número de ciclos de muestreo, se especificó cómo proceder con cada motor, teniendo en cuenta el número real de ciclos  $c_i$  de cada uno de ellos, de forma que:

- Se mantenían los últimos 75 ciclos si para el motor  $i$  se da que  $c_i > 75$ .
- Se mantienen los  $c_i$  ciclos del motor y se rellenan los restantes hasta  $c_i = 75$  usando la técnica de padding, explicada anteriormente.

De esta forma, sabiendo que el espacio latente es de 8 dimensiones, manteniendo filas de 75 ciclos por motor, se obtiene una matriz  $M_{100 \times 600}$ . Así, cada motor queda representado por

un único vector de dimensión fija de 600 unidades, compatible con la clasificación por GMM. Esta técnica requiere menos relleno con ceros que el método de Padding con ceros explicado previamente, pero tiene la desventaja de que se pierde parte de la evolución temporal temprana de muchos de los motores. La matriz ya formada se pasa por el clasificador GMM, del que se obtienen los siguientes resultados:

Cluster	Número de motores
0	9
1	12
2	79

Tabla 4.18. DISTRIBUCIÓN DE MOTORES POR CLUSTER SEGÚN CLASIFICACIÓN GMM USANDO FIXED LENGTH SAMPLING

En este caso, la clasificación resulta menos balanceada, con una gran concentración en el cluster 2, en el que clasificaron el 79 % de los motores. El cluster 0 recogía solo 9 motores mientras que el cluster 1 obtuvo 12 de ellos. Para evaluar la separabilidad obtenida, se proyectaron los vectores de cada motor con PCA como se hizo en los casos anteriores.

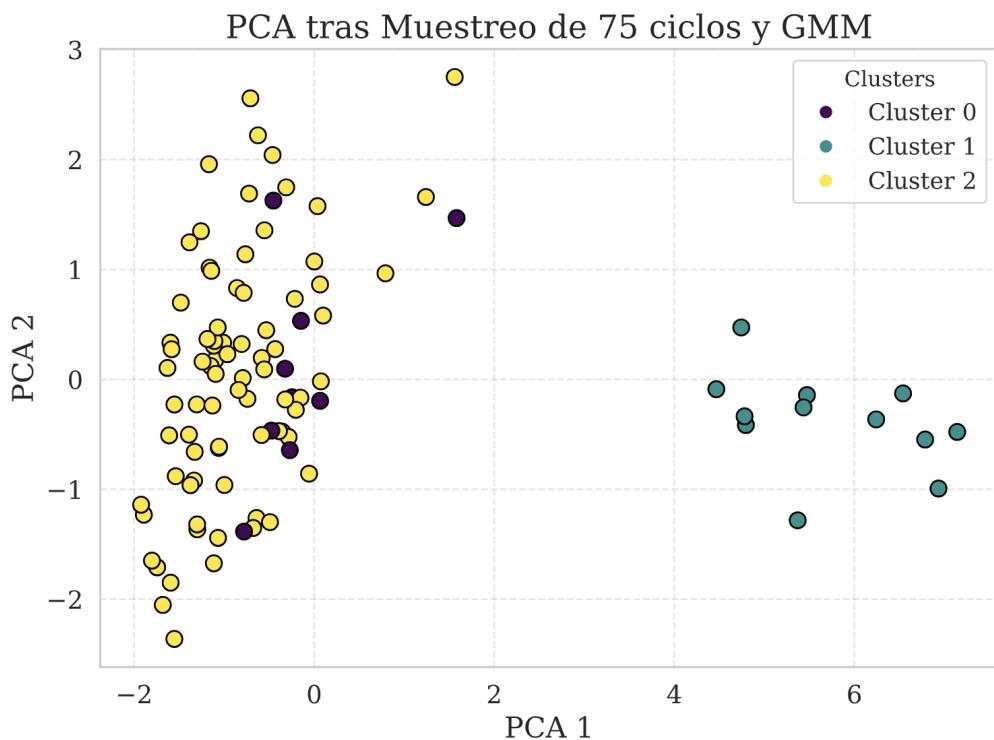


Figura 4.15. Análisis de Componentes Principales según clasificación con GMM usando Fixed Length Sampling (Fuente: elaboración propia).

La figura 4.15 muestra que la mayor parte de los motores se agrupan en un cluster dominante. Se ve como en este caso, respecto a los componentes principales, el clasificador

no consigue separar los clusters 0 y 2, que aparecen solapados en un único grupo disperso a lo largo de la segunda componente, principalmente en el rango  $-2 < PCA_1 < 0$ , salvo algunos outliers. El cluster 1 sí se diferencia claramente de los otros dos, en el extremo opuesto de la gráfica a partir de valores de  $PCA_1 > 4$  y  $-1 < PCA_2 < 1$ .

Como se puede ver, los datos y la forma de los clusters difieren bastante de los de la publicación de Lodygowski y Szrama. Sin embargo, en cierta medida, este resultado se asemeja al clustering que Lodygowski obtuvo para el dataset de motores reales. La agrupación por PCA de los motores reales del estudio original se representa en la Figura 4.16, donde se aprecia como los clusters Unhealthy y Middle están prácticamente superpuestos, con un cluster Healthy algo más independiente.

Este caso no obtuvo resultados relevantes en comparación con el caso anterior de Zero Padding, pero ofrece una alternativa para el proceso de construcción de la matriz de entrada que puede tenerse en cuenta para otro tipo de estudios.

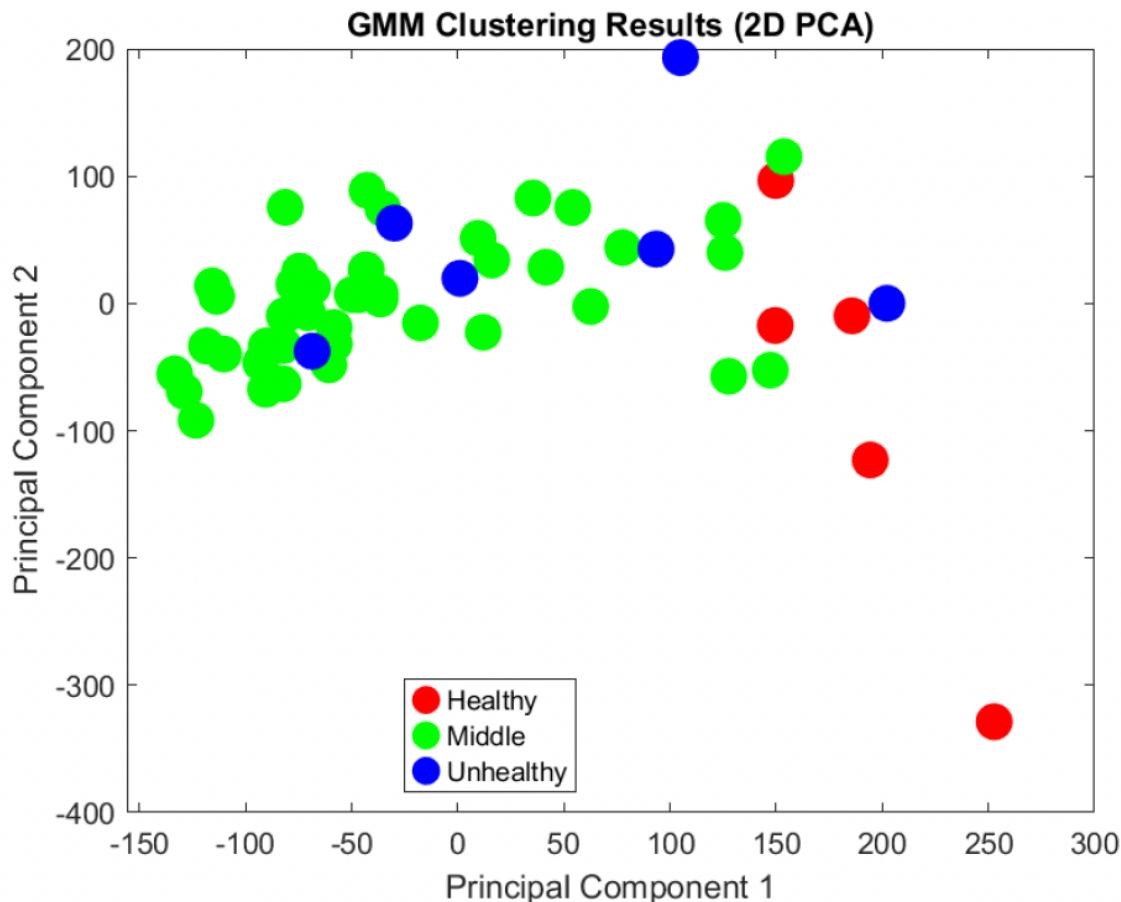


Figura 4.16. PCA de los resultados obtenidos por Lodygowski et al. para los datos de motores reales (Fuente: [36]).

#### 4.2.6 Comparación de los tres casos

En la sección anterior se han expuesto las tres técnicas utilizadas para intentar replicar la clasificación de Lodygowski para el primer dataset de prueba del CMAPSS `test_FD001.txt`. Se aprecian diferencias notables en los tres casos debido a la configuración final del dataset de entrada del GMM. Es importante recordar que en los tres casos se emplearon los mismos datos de salida del autoencoder.

Además de los datos presentados anteriormente, se obtuvo el índice de Silhouette para cada caso. Como se ha explicado anteriormente, en la sección Sección 3.7.6, este indicador muestra el nivel de clasificación realizado por el agrupador, en este caso el GMM. Las principales diferencias apreciadas en los casos anteriores vienen descritas en la Tabla 4.19.

Tabla 4.19. COMPARACIÓN DE RESULTADOS DE CLUSTERING  
GMM CON DISTINTAS MATRICES DE ENTRADA

Criterio de comparación	Agregación estadística	Zero Padding	Fixed Length Sampling
Tamaño de la matriz	100x24	100x2424	100x600
Calidad de PCA	Reducida	Alta	Intermedia
Silhouette Index	0.2582	0.1471	0.0413
Motores en Cluster Dominante	55	45	79
Motores en Cluster Intermedio	32	25	12
Motores en Cluster Minoritario	13	30	9

En primer lugar, se encuentra el tamaño de las matrices de entrada tras seleccionar la estrategia de unificación. En los tres procedimientos, las matrices cuentan con 100 filas, una por cada motor. Además, es importante recordar en este punto que el dataset de entrada al GMM tras el autoencoder cuenta con una fila por ciclo (20631 filas) y 8 columnas, correspondientes a la representación latente de las 24 iniciales. El conjunto de datos de agregación estadística cuenta con la matriz más reducida, siendo esta de dimensión  $100 \times 24$ . Además, sus columnas no corresponden a las columnas latentes originales, si no que se sustituyeron por el valor máximo, la media y la desviación típica de cada una de las 8 variables disponibles. En el segundo caso, se tiene la matriz más grande, cuyo tamaño resultó ser  $100 \times 2424$ , determinado por el motor con mayor número de ciclos (303), multiplicado por las 8 variables latentes. Por último, usando muestreo fijo de 75 ciclos se obtuvo una matriz de  $100 \times 600$ , bastante mayor que en el primer caso, pero con longitud cercana a cuatro veces menor que en el caso de padding.

Como se ha visto anteriormente, las representaciones mediante análisis de componentes principales resultaron muy distintas para cada uno de los casos. Mientras que para el caso de

agregación estadística se agruparon todos los clusters en dos grupos bien separados, habiendo motores de dos clusters diferentes superpuestos, en el dataset que usaba padding se logró distinguir tres grupos bien distribuidos a lo largo de las dimensiones principales. En último lugar, para el caso de fixed length sampling se obtuvo un resultado intermedio, con tres clusters no demasiado diferenciados, pero no tan superpuestos como en agregación estadística.

Pasando a métricas numéricas, el índice de Silhouette nos da alguna indicación importante. Como se ve, el caso que obtuvo mayor puntuación en este indicador fue el de agregación estadística, con un  $s_{stat} = 0.2582$ . Este viene seguido del Zero Padding, con  $s_{zp} = 0.1417$ . En último lugar aparece la construcción formada a partir de los 75 últimos ciclos, con un  $s_{fls} = 0.0413$ . Es importante recalcar que el índice de Silhouette da una estimación de lo correcta que es la clasificación. Un  $s$  cercano a cero, como en el tercer caso, indica que los puntos fácilmente podrían pertenecer a otro cluster, ya que estos se encuentran en la mayoría de casos igual de lejos de los puntos de su grupo que de los del grupo más cercano. Para las tres estrategias se obtuvieron valores positivos, lo que al menos indica que la clasificación no tiene outliers excesivamente lejanos al grupo al que fueron asignados. En comparación con el estudio original, en el que se obtuvo un  $s_{original} = 0.39$  para los datos de CMAPSS, lo que es un 51.05% más alto que en el mejor de nuestros casos.

El set de datos del CMAPSS incluye también un archivo de texto en el que se pueden ver los valorers de la vida útil remanente de los motores al final de cada simulación. Se optó por graficar para cada uno de los tres métodos de construcción de datos de entrada del GMM, como se clasificaron los datos junto con su valor real de RUL. Los resultados obtenidos fueron los siguientes:

- El método de agregación estadística destaca por el pequeño número de motores que entran en el cluster 1, en comparación con los dos restantes. Aún así, se ve como el RUL para los 13 motores del cluster 1 es muy variado, con valores desde 7 a 128 ciclos. El valor medio de este cluster es de 49 ciclos aproximadamente, por lo que se ven varios valores muy lejanos a esta media que podrían ser considerados outliers. Los otros dos clusters corresponden con los que aparecían superpuestos en la Figura 4.11. Desde este punto de vista estadístico, se aprecia como las distribuciones de valores son muy similares, con valores medios del RUL para el cluster 0 y el 2 de 76 y 85 ciclos respectivamente. Ambos agrupan motores que con 7 ciclos de RUL como valor mínimo, y máximos de 137 (0) y 145(2). De estos datos se concluye que el modelo no es capaz de separar correctamente los tres casos, ya que no se logra agrupar valores similares de RUL en ninguna de las clasificaciones.

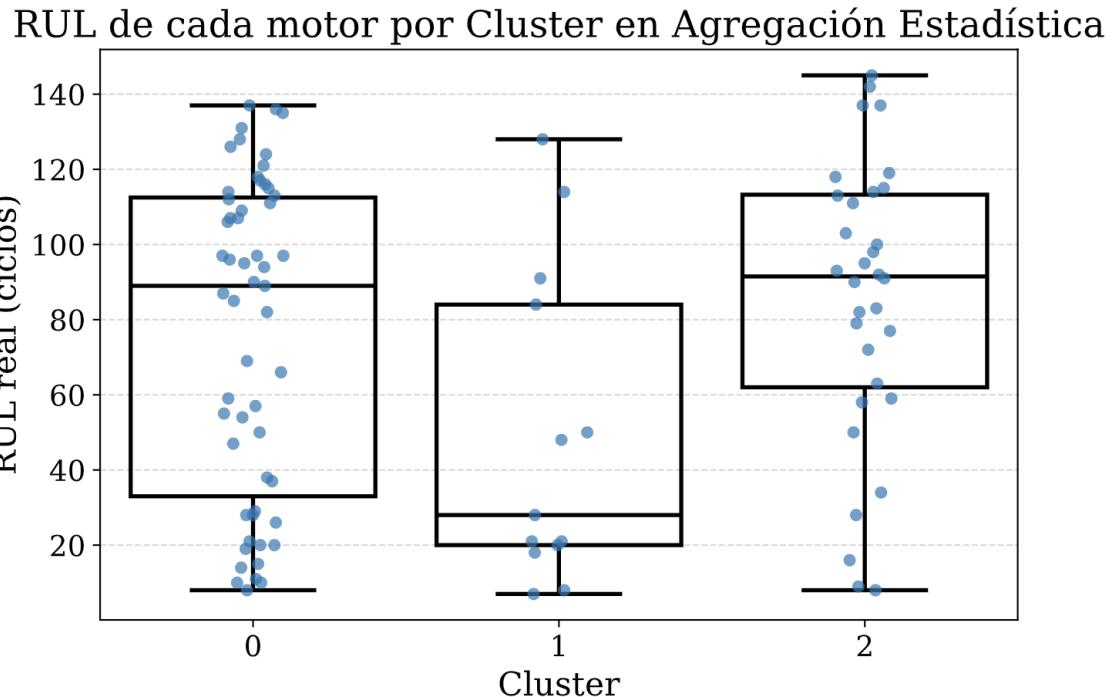


Figura 4.17. RUL por Cluster en el caso de Agregación Estadística

Tabla 4.20. ESTADÍSTICAS POR CLUSTER EN FIXED LENGTH SAMPLING

Cluster	Motores	Media (ciclos)	Mediana (ciclos)	Min (ciclos)	Max (ciclos)
0	55	76.05	89.0	8	137
1	13	49.08	28.0	7	128
2	32	85.34	91.5	8	145

- En segundo lugar, cuando se utilizó Fixed Length Sampling se obtuvo un primer cluster muy minoritario, el cluster 0, cuyos motores poseían valores reales muy dispersos (sólo 9 motores, con RUL entre 28 y 131 ciclos). El cluster 1 obtuvo valores más agrupados, con un RUL medio elevado, en torno a los 116 ciclos. Los ciclos de vida restantes de este grupo formado solo por 12 motores toman valores desde 98 a 145 ciclos. Este cluster podría considerarse mejor separado, aunque el cluster 2 tiene gran cantidad de puntos también en este rango. Esta último grupo contiene la mayoría de puntos, con RUL variable desde 7 ciclos hasta los 137, por lo que una vez más no parece hacer una separación exhaustiva entre estados de salud de los motores.

Tabla 4.21. ESTADÍSTICAS POR CLUSTER EN FIXED LENGTH SAMPLING

Cluster	Motores	Media	Mediana	Min	Max
0	9	83.00	85.0	28	131
1	12	116.33	111.5	98	145
2	79	68.47	77.0	7	137

RUL de cada motor por Cluster en Fixed Length Sampling

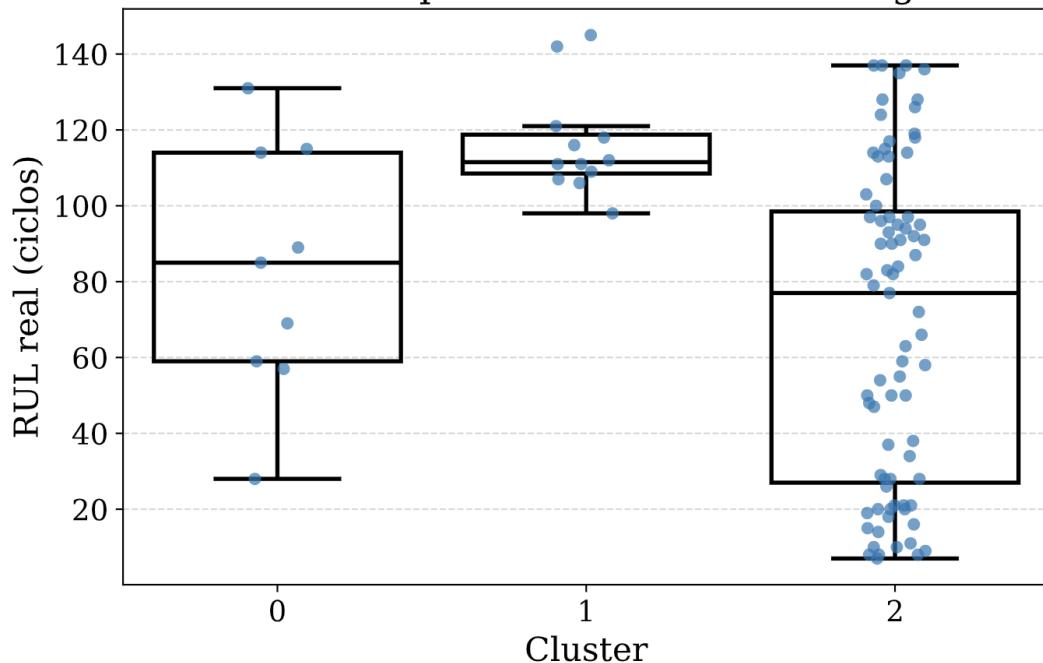


Figura 4.18. RUL por Cluster en el caso de Fixed Length Sampling

- Finalmente, la metodología Zero Padding muestra tres clusters más estructurados. El cluster 0 toma valores medios bajos, de 53.71 ciclos por motor. A pesar de tener valores que alcanzan hasta los 136 ciclos, la concentración de motores con RUL bajo es abundante, con un 42.2% teniendo menos de 40 ciclos restantes. El segundo cluster, al igual que en el caso de muestreo de longitud fija, corresponde a una concentración de motores con RUL alto. Este presenta una media de  $RUL = 118.84$  ciclos y los valores solo oscilan entre 77 y 145 ciclos. Este cluster 1 podría ser una buena definición de motores saludables. Por último, el cluster 2 es más disperso, pero mantiene un valor medio elevado, pero por debajo del cluster 1, de 78 ciclos restantes.

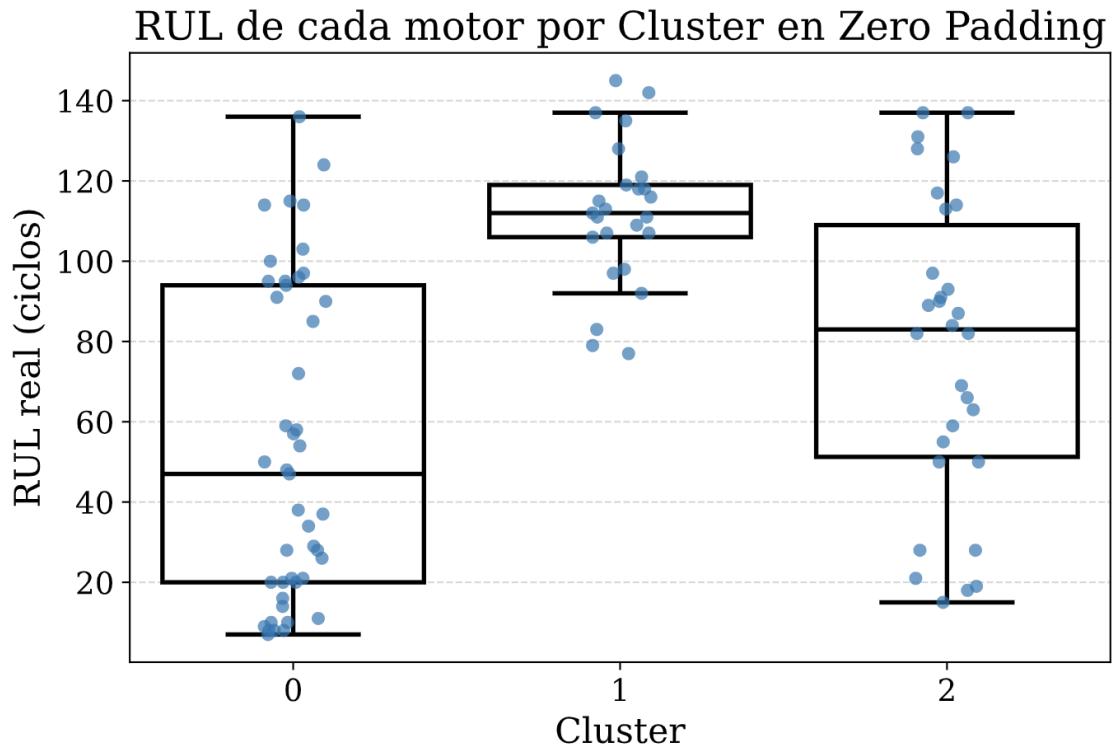


Figura 4.19. RUL por Cluster en el caso de Zero Padding

Tabla 4.22. ESTADÍSTICAS POR CLUSTER EN ZERO PADDING

Cluster	Motores	Media	Mediana	Min	Max
0	45	53.71	47.0	7	136
1	25	111.84	112.0	77	145
2	30	77.97	83.0	15	137

De los tres métodos, el Zero Padding una vez más fue el más resolutivo a la hora de clasificar los motores, sin resultados sobresalientes, pero con indicios claros de coherencia en las agrupaciones realizadas. Esto es un resultado interesante ya que confirma que nuestro procedimiento ha sido capaz de replicar hasta cierto punto la metodología de Lodygowski, a falta de conocer los parámetros exactos utilizados por el estudio referencia.

## 5 Conclusiones y Futuros Trabajos

Tras los resultados obtenidos para los casos prácticos, se han recopilado una serie de conclusiones que indican la calidad del trabajo realizado, así como los puntos débiles y limitaciones encontradas para cumplir los objetivos inicialmente planteados.

En esta sección se exponen los principales descubrimientos obtenidos durante la implementación de este proyecto, así como posibles líneas de trabajos futuros que continúen el estudio.

### 5.1 Principales resultados obtenidos

#### 5.1.1 Resultados del Caso Práctico 1

El primero caso fue el más sencillo de los dos propuestos. Para este primer estudio, los resultados fueron prácticamente idénticos a los de la publicación que de referencia. La principal conclusión, los algoritmos supervisados empleados para clasificación alcanzan un rendimiento muy elevado, alcanzando todos cerca de un 100 % en métricas como exactitud, precisión y recall. Según la Tabla 4.5, el algoritmo k-NN obtuvo los mejores resultados, seguido de cerca por el algoritmo Random Forest. El uso de Decision Trees resultó en métricas ligeramente inferiores, pero muy similares a las de los otros dos casos. En ninguno de los modelos se obtuvieron métricas inferiores al 97 %, siendo la más reducida la precisión del Decision Tree, con un 97.56 %. Comparando estos tres modelos porcentualmente, se obtiene:

Referencia: k-NN				
Modelo	Accuracy (%)	Precisión (%)	Recall (%)	F1-score (%)
Decision Tree	98.26	97.89	98.20	98.03
Random Forest	99.71	99.36	100.00	99.67

Tabla 5.1. COMPARACIÓN PORCENTUAL DE LAS MÉTRICAS DE CADA MODELO RESPECTO AL KNN

Como se observa en el dataset utilizado, no hay diferencias destacables entre los modelos, siendo la métrica más baja del Decision Tree un 98 % del rendimiento del k-NN (obtenido en el F1-score). Se puede concluir de estos resultados que el conjunto de los datos utilizado, contenía features lo suficientemente diferentes en cuanto a sus características para facilitar la distinción entre clases por parte de los modelos.

### Dispersión y distribuciones marginales: Variance vs Skewness

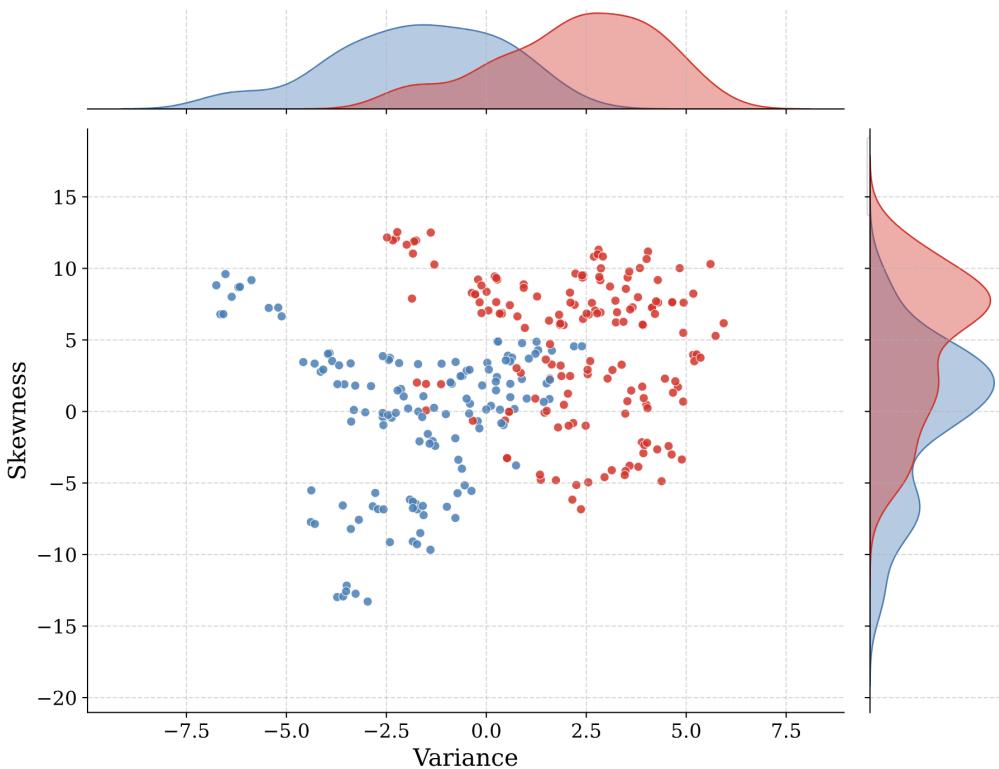


Figura 5.1. Dispersión y Distribución de los elementos respecto a la Varianza y la Skewness

La Figura 5.1 muestra cómo la distribución de las diferentes señales entre puntos en variables como varianza y skewness están bien separadas, a excepción de algún punto que se mezcla con la clase contraria. Los datos de esta gráfica están tomados de los datos etiquetados iniciales, correspondiendo a la clasificación real de cada punto. Como se ha visto, los modelos fallan en la clasificación de algún punto, pero teniendo en cuenta las demás variables, los modelos son capaces de diferenciar las dos clases sin mayor dificultad, confundiendo una cantidad muy reducida de puntos.

Además, estas métricas se obtuvieron tras emplear validación en cinco particiones (folds) diferentes, de manera que el modelo se entrenó múltiples veces con distintos datos, empleando cada una de estas particiones para validar los modelos. Esta es una práctica muy común para evitar overfitting y asegurar que el modelo aprende a diferenciar los datos y no memoriza la información disponible, pudiendo generalizar a nuevos datos que se le presenten.

Cabe destacar que tanto Random Forest como k-NN alcanzaron un valor perfecto de Recall, lo que significa que no se obtuvieron falsos negativos (considerando los 0 como negativo y los 1 como positivo) en la clasificación.

En resumen, este primer caso demuestra la potencia de los algoritmos Decision Tree,

Random Forest y k-Nearest Neighbors para tareas sencillas de clasificación binaria. Este caso es fácilmente aplicable a la identificación de señales de vibración de cualquier tipo de máquina física o componente rotatorio, como se muestra en gran variedad de estudios [83]. La implementación de estos modelos permite una revisión sostenida en el tiempo de sistemas aeronáuticos. Con la debida instalación de una buena infraestructura de adquisición de datos, es posible alimentar estos modelos para que el sistema aprenda de casos históricos, logrando una mejor generalización del algoritmo. Como contrapartida se tiene, como ya se ha mencionado anteriormente, la dificultad de etiquetar todos estos datos, ya que requiere la comprobación definitiva por parte del experto para confirmar cada caso y plasmarlo en el conjunto de datos.

### **5.1.2 Resultados del Caso 2**

En este segundo estudio se han realizado aportaciones valiosas que permiten aproximarse a la metodología seguida por Lodygowski y Szrama en su artículo. La complejidad de este caso es notable, ya que se desconoce el proceso seguido por los autores para el tratamiento de los datos del CMAPSS y llegar a los resultados finales. Si bien es cierto que los datos tanto gráficos como numéricos de nuestro estudio difieren de los originales, se ha conseguido aproximar la solución mediante la iteración sobre varias metodologías.

En primer lugar, nos encontramos con la complicación inicial de definir los parámetros necesarios para describir completamente el Autoencoder para los datos del CMAPSS. La publicación referencia muestra los datos para construir la red neuronal para el dataset de motores reales, el cual no es de carácter público. Por tanto, se observó la estructura empleada en datos reales, para posteriormente adaptarla a los datos del CMAPSS. Mediante la iteración con diferentes configuraciones de neuronas en las capas ocultas, el tamaño de la capa latente (bottleneck) y el valor del dropout, se identificó la estructura más adecuada. Tras añadir varias combinaciones de parámetros y poder comparar sus resultados, se obtuvo la Tabla 4.10. De esta tabla se extrajo que la configuración del autoencoder que minimizaba la función de pérdida tras comprimir y descompimir los datos fue la siguiente:

- Capas de encoder y decoder de 32 neuronas.
- Espacio latente de 8 dimensiones.
- Dropout del 20%.

Este caso más favorable empieza en un valor inicial de pérdida mucho más bajo que el resto, y se mantuvo decreciente antes de un Early Stopping a las 22 iteraciones. El proceso de iteración no mostró gran mejoría, pero el reducido valor de error inicial fue suficiente para

superar las reconstrucciones del resto de configuraciones, como muestra la Figura 4.9. En comparación con el autoencoder utilizado para el set de motores reales, se reduce considerablemente el tamaño de la red, ya que como se explica en el artículo, la construcción se hizo utilizando dos capas de 128 neuronas cada una en entrada (codificador) y salida (decodificador), y una capa latente de 32 elementos. De esta forma se eliminaron dos capas completas, y las restantes redujeron considerablemente la cantidad de nodos. La tabla Tabla 5.2 muestra las diferencias numéricas entre ambas representaciones.

Capa	Tipo	# Neuronas (Real)	# Neuronas (CMAPSS)
1	Input	86	24
2	Encoder	(2x)128	32
3	Espacio Latente	32	8
4	Decoder	(2x)128	32
5	Output	86	24

Tabla 5.2. COMPARACIÓN DE LA ESTRUCTURA DEL AUTOENCODER DE LODYGOWSKI PARA MOTORES REALES Y EL OBTENIDO PARA CMAPSS.

Como es obvio, al necesitar reducción de la dimensionalidad, era necesario modificar el esquema de red neuronal, ya que en el caso de motores reales el espacio latente obtenido era mayor que el número de dimensiones del que se dispone para los datos simulados. Todo este paso se llevó a cabo usando el primero de los cuatro training sets disponibles.

Una vez escogido el Autoencoder y los parámetros que lo describen, se pasa a la clasificación binaria de los motores usando el modelo de mezcla gaussiana. Para esta segunda etapa se encontraron nuevas limitaciones. El dataset utilizado en este caso corresponde al test set número 1. Este nuevo dataset se pasó por el autoencoder y se extrae la versión comprimida de los datos, correspondiente a la capa latente o cuello de botella (bottleneck) de la red. Aquí entra en juego la elección del método de unificación de los datos con los que se alimenta el modelo gaussiano de clasificación. Nuestros resultados mostraron que:

- La metodología que emplea agregación estadística redujo aún más la dimensionalidad de los datos iniciales, pudiendo perder información relevante en la clasificación. Para esta configuración se obtuvieron unos datos visuales por PCA no concluyentes, ya que confundía varios clusters en una misma región del mapa bidimensional. Además, los gráficos de RUL por cluster no identificaron un patrón significativo, con el cluster 0 y 2 mostrando un rango de valores muy elevado, sin diferencias notables. Aún así, se obtuvo el índice de Silhouette más alto de los tres casos, con  $s = 0.2582$ .
- Para el caso de Muestreo Fijo (Fixed Length Sampling), se eligió un número de ciclos fijo  $k = 75$ . Se tomaron los 75 últimos ciclos de cada motor, obviando el resto de

datos temporales anteriores. En esta ocasión, también se pierde parte de la evolución temporal, ya que muchos motores superan los 200 ciclos, llegando el motor más longevo a cubrir 303. Esta estrategia mostró clusters algo más diferenciables, con un cluster 1 que podría representar motores saludables, ya que todos los motores de este grupo presentaban un RUL entre 98 y 145 ciclos. Aún así, el resto de grupos no eran muy diferenciables, y los rangos de RUL contenían prácticamente el espectro completo de posibilidades. Visualmente, este caso no mejoró al anterior, y se obtuvo  $s = 0.0413$ , el peor de los tres casos. Esto indica que gran cantidad de puntos podrían pertenecer a otro cluster.

- En último lugar, el caso de padding con ceros obtuvo los resultados más destacables. Esta técnica se decidió usar como medio para no perder información alguna. Aunque los ceros introducen información vacía al modelo y generan una matriz de entrada de gran tamaño para el clasificador, este enfoque permitió obtener, por primera vez, una representación semejante a la del estudio original. Los clusters diferencian entre motores saludables (cluster 1), 25 motores con una media de 118 ciclos; motores intermedios (cluster 2), con RUL medio de 78 ciclos y motores no saludables (cluster 0), con media de 53 ciclos por motor y más de un 40% de motores en este grupo con menos de 40 ciclos restantes. Además, los resultados visuales también son prometedores, con una gráfica en PCA bidimensional similar a la de la referencia utilizada (Figura 4.13). Por último, se obtuvo un índice de Silhouette  $s = 0.1471$ , que no mejoró al del caso de agregación estadística.

A pesar de las grandes diferencias existentes entre nuestro modelo y el de Lodygowski y Szrama, hay que tener en cuenta que se desconocen por completo las condiciones de preprocesamiento de datos, composición del autoencoder y estrategia utilizada para crear la matriz de entrada al GMM por parte de los autores. Cada una de las elecciones tomadas durante el proceso implementado introduce ligeras variaciones numéricas, que pueden afectar notablemente al resultado final obtenido. Esto remarca la dificultad del estudio realizado, en el que se ha buscado iterativamente la mejor solución para cada uno de estos casos, hasta llegar a la solución mostrada. Se puede afirmar, por tanto, que al conseguir resultados similares a los de la referencia tomada, se ha cumplido uno de los objetivos principales del proyecto.

Además, el proyecto en el que se basa este estudio no se centra en el análisis de los datos que se han usado en nuestro proceso, sino que utiliza los datos del CMAPSS para verificar los resultados de su método aplicado a un dataset de valores reales. En la publicación, también se muestran pasos adicionales que no han podido llevarse a cabo por falta de recursos, pero que dejan el proyecto abierto, con una gran cantidad de potenciales adiciones y mejoras para complementarlo.

En definitiva, este estudio demuestra la aplicabilidad de una metodología que combina

varios algoritmos de Machine Learning, como lo son los Autoencoders y los Modelos de Mezcla Gaussiana, en la clasificación de equipos aeronáuticos como motores turbofán. Este método cuenta con la ventaja de que no requiere datos etiquetados, pudiendo emplearse directamente sobre datos crudos obtenidos desde un sistema de adquisición de datos instalado en el equipo a analizar. Esto reduce el tiempo empleado por mano de obra cualificada en analizar los datos obtenidos, ya que evita el tener que clasificar los datos manualmente, pasando de forma directa al análisis de atributos. Además, la ingeniería de características también se automatiza, siendo el propósito principal de la reducción de la dimensionalidad llevada a cabo por el Autoencoder.

Como conclusión final, se ha logrado exponer la importancia de combinar tecnologías novedosas como el aprendizaje automático a la rama del mantenimiento aeronáutico. Esto se ha hecho, primero de forma teórica por medio de un marco sólido que explica los diferentes modelos de Machine Learning, y posteriormente aplicándolo a ejemplos prácticos donde se muestra como aprovechar la potencia de estos algoritmos.

## 5.2 Contribuciones del estudio

La metodología empleada en ambos casos prácticos se basó directamente en los artículos de A. Ouadah et al. [35] y T. Lodygowski, S. Szrama [36].

Para el caso de aprendizaje supervisado, la normalización de los datos empleada y el uso de validación con cinco particiones es aportación propia (Sección 4.1.3 y Sección 4.1.5). Esto pudo haber sido determinante para la obtención de los resultados, que superan ligeramente a los del caso original. Además, las gráficas de matriz de confusión y de distribución cruzada de las variables (Figura 3.14 y Figura 4.4) también son de creación propia, facilitando la exposición de los resultados de forma visual.

En el caso de aprendizaje no supervisado se introducen como contribuciones el proceso iterativo de búsqueda de parámetros óptimos del autoencoder (Sección 4.2.4) y todo el estudio relacionado con las tres técnicas empleadas para la creación de una matriz rectangular de entrada para el clasificador (Sección 4.2.5). Por último, las figuras que se muestran en la Sección 4.2.6 para comparar la clasificación del GMM con los datos reales de RUL para cada motor también han sido realizadas exclusivamente para este estudio.

## 5.3 Limitaciones y posibles mejoras

La principal complicación que enfrenta nuestro estudio es, como se ha mencionado en varias ocasiones, la falta de datos reales públicos. La opacidad mostrada por los fabricantes, dificulta la aplicación de los algoritmos presentados en este proyecto a casos reales de equipamiento

aeronáutico. Esto es uno de los principales retos de cara a la evolución del mantenimiento predictivo en el ámbito aeronáutico, ya que la disponibilidad de sets de datos reales para fines académicos y de investigación fomentarían en gran medida el progreso colaborativo del PdM. Por el momento, los datos proporcionados por la NASA resultan suficientes para un estudio detallado. Sin embargo, la colaboración con algún fabricante que cediera datos reales, o en su defecto, utilizar algún equipo propio, como un banco de prueba perteneciente a alguna institución académica para la extracción de los datos quedaría como posible mejora de este trabajo.

Además, para el caso de clasificación no supervisada, como se ha explicado, la principal dificultad recae en la ausencia de una guía detallada para el tratamiento de los datos del CMAPSS. Existen multitud de mejoras posibles para el análisis realizado, como una iteración más extensa de cara a la obtención de mejores resultados de reconstrucción del autoencoder, o la búsqueda de nuevas maneras de crear la matriz de entrada del GMM, buscando la menor pérdida o adición de datos posible. Es importante destacar que, como se expresa en el artículo original:

The CMAPSS dataset consists of four sub-datasets (FD001, FD002, FD003, FD004), and the results presented in this paper for CMAPSS are often the mean performance across these sub-datasets. [36]

Si bien se conoce que los datos del CMAPSS están compuestos por cuatro simulaciones distintas, en nuestro estudio solo se utilizó la primera de ellas, por lo que otro posible proceso de mejora que podría acercar los resultados a los de la referencia podría basarse en el uso combinado de todos los datasets, aunque esto conllevaría una ampliación notable de los recursos temporales dedicados a análisis de datos y procesado de los mismos.

## 5.4 Aplicaciones futuras y líneas de investigación

En cuanto a posibles ampliaciones, el primer estudio podría extenderse a otros algoritmos de clasificación supervisada, pudiendo así comparar la calidad de las predicciones realizadas en un rango más amplio de casos. Además, todos estos modelos podrían aplicarse a un conjunto de datos que muestre mayor homogeneidad entre las clases, con el fin de analizar comportamiento de los algoritmos en escenarios más complejos. Esto facilitaría el uso de técnicas como el Grid Search (ver Sección 3.9.3) para obtener la combinación óptima de parámetros para estos algoritmos. La aplicación de esta técnica no tiene mucho sentido en nuestro caso, ya que el margen de mejora de los algoritmos utilizados es prácticamente nulo.

Por otra parte, el segundo caso se cierra omitiendo el paso final llevado a cabo por el estudio que se usó de referencia. Este consiste en la aplicación de redes LSTM para

la predicción numérica del RUL tras la clasificación por GMM. Como se muestra en [36], el empleo de este tipo de red neuronal presenta resultados muy precisos, siendo capaz de predecir con un error muy bajo el número de ciclos restantes de cada motor. Esta parte del análisis no se ha llevado a cabo por falta de recursos temporales, ya que conllevaría el estudio del funcionamiento detallado de las redes Long-Short Term Memory, así como la creación del código y la obtención de los parámetros óptimos para la red neuronal. Este último paso, junto a una mejora sólida de lo realizado hasta ahora, tendría pleno sentido, y aporta un valor significativo al estudio.

Como puntos de expansión del proyecto, se puede investigar la aplicación de los métodos a conjuntos de datos reales. Sería de gran utilidad contar con instalaciones especializadas para la recolección de datos, como, por ejemplo, un banco de pruebas facilitado por la universidad o algún colaborador externo que contribuyera al desarrollo de este estudio. Esto reforzaría la validez de la metodología, al aplicarla sobre una mayor variedad de datos, más aún si estos son datos procedentes de equipos funcionales.

## Referencias

- [1] K. Moenck *et al.*, “Digital twins in aircraft production and MRO: challenges and opportunities,” *CEAS Aeronautical Journal*, vol. 15, no. 4, pp. 1051–1067, 2024. DOI: 10.1007/s13272-024-00740-y.
- [2] Federal Aviation Administration, “Statistical Summary of U.S. Aviation, Calendar Year 2022,” FAA, Washington, D.C., Tech. Rep. FAA-2023-10, 2023, [https://www.faa.gov/sites/faa.gov/files/2023-10/statsum\\_summary\\_2022.pdf](https://www.faa.gov/sites/faa.gov/files/2023-10/statsum_summary_2022.pdf) (vistado el 5 de mayo, 2025).
- [3] Observatorio del Transporte y la Logística en España, “Informe Anual 2024,” MITMA, Tech. Rep., 2025, [https://cdn.transportes.gob.es/portal-web-drupal/OTLE/elementos\\_otle/informe\\_anual\\_2024\\_\(abril\\_2025\).pdf](https://cdn.transportes.gob.es/portal-web-drupal/OTLE/elementos_otle/informe_anual_2024_(abril_2025).pdf) (vistado el 24 de mayo, 2025).
- [4] International Air Transport Association, “Unveiling the biggest airline costs,” IATA, Tech. Rep., 2024, <https://www.iata.org/en/publications/newsletters/iata-knowledge-hub/unveiling-the-biggest-airline-costs/> (visitado el 26 de mayo, 2025).
- [5] Aircraft Commerce, “Maintenance Analysis of the A320 Family,” *Aircraft Commerce*, pp. 45–59, 2006, <https://tinyurl.com/4exr5t5u> (visitado el 1 de junio, 2025).
- [6] Forbes España / EP, “IAG convierte diez opciones de compra del A320neo en un pedido en firme, valorado en 1.100 millones,” *Forbes España*, 2023, <https://forbes.es/ultima-hora/305163/iag-convierte-diez-opciones-de-compra-del-a320neo-en-un-pedido-en-firme-valorado-en-1-100-millones/> (visitado el 8 de mayo, 2025).
- [7] Aviation Week, “Fighter Aircraft Through Life Costs,” Aviation Week Network, Tech. Rep., 2023, <https://tinyurl.com/y9sh6kek> (vistado el 18 de mayo, 2025).
- [8] Arshon Technology Inc., *Predictive Maintenance in Aircraft/Aviation Industry*, <https://arshon.com/blog/predictive-maintenance-in-aircraft-aviation-industry/> (vistado el 29 de mayo, 2025), 2021.
- [9] A. P. Dibsdale and M. Ruotsala, *Aerospace Predictive Maintenance: Fundamental Concepts*. SAE International, 2020, ISBN: 978-0-7680-9427-5.
- [10] F. S. Nowlan and H. F. Heap, “Reliability-Centered Maintenance,” United States Department of Defense, Office of Assistant Secretary of Defense, Washington, D.C., Tech. Rep., 1978, <https://reliabilitywebfiles.s3.amazonaws.com/Reliability+Centered+Maintenance+by+Nowlan+and+Heap.pdf> (visitado el 3 de agosto, 2025).

- [11] J. Lee, F. Wu, W. Zhao, M. Ghaffari, L. Liao, and D. Siegel, “Prognostics and health management design for rotary machinery systems—Reviews, methodology and applications,” *Mechanical Systems and Signal Processing*, vol. 42, no. 1-2, pp. 314–334, 2014. DOI: [10.1016/j.ymssp.2013.06.004](https://doi.org/10.1016/j.ymssp.2013.06.004).
- [12] T. P. Carvalho, F. A. A. M. N. Soares, R. Vita, R. D. Francisco, J. P. Basto, and S. G. S. Alcalá, “A systematic literature review of machine learning methods applied to predictive maintenance,” *Computers & Industrial Engineering*, vol. 137, p. 106024, 2019. DOI: [10.1016/j.cie.2019.106024](https://doi.org/10.1016/j.cie.2019.106024).
- [13] R. Meissner, A. Rahn, and K. Wicke, “Developing prescriptive maintenance strategies in the aviation industry based on a discrete-event simulation framework for post-prognostics decision making,” *Reliability Engineering & System Safety*, vol. 214, p. 107812, 2021. DOI: [10.1016/j.ress.2021.107812](https://doi.org/10.1016/j.ress.2021.107812).
- [14] C. Teubert, A. A. Pohya, and G. Gorospe, “An Analysis of Barriers Preventing the Widespread Adoption of Predictive and Prescriptive Maintenance in Aviation,” NASA Ames Research Center, Tech. Rep. NASA/TM-20230000841, 2023, <https://ntrs.nasa.gov/citations/20230000841> (visitado el 12 de agosto, 2025).
- [15] Z. Cinar, A. N. Nuhu, G. Zeeshan, M. Korhan, S. Asmael, and S. Safaei, “Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0,” *Sustainability*, vol. 12, no. 19, p. 8211, 2020. DOI: [10.3390/su12198211](https://doi.org/10.3390/su12198211).
- [16] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill Science/Engineering/Math, 1997, ISBN: 0070428077.
- [17] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science*, vol. 349, no. 6245, pp. 255–260, 2015. DOI: [10.1126/science.aaa8415](https://doi.org/10.1126/science.aaa8415).
- [18] S. Datta, “Machine Learning in Industry,” in *Management and Industrial Engineering*, J. P. Davim, Ed., Springer, 2022, ISBN: 978-3-031-31132-2. DOI: [10.1007/978-3-030-75847-9](https://doi.org/10.1007/978-3-030-75847-9).
- [19] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th. Pearson, 2021, ISBN: 978-1-292-40117-1.
- [20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016, <https://www.deeplearningbook.org/>, ISBN: 978-0-262-03561-3.
- [21] J. Patterson and A. Gibson, *Deep Learning: A Practitioner’s Approach*. O’Reilly Media, 2017, Primera edición, ISBN: 978-1-491-92157-5.
- [22] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd. O’Reilly Media, 2019, ISBN: 9781492032649.

- [23] C. Janiesch, P. Zschech, and K. Heinrich, “Machine learning and deep learning,” *Electronic Markets*, vol. 31, no. 3, pp. 685–695, 2021. DOI: 10.1007/s12525-021-00475-2.
- [24] Y. Zheng, M. Liu, H. Yu, and Y. Wang, “A General Framework for Predictive Maintenance Using Machine Learning,” *Procedia Computer Science*, vol. 199, pp. 1046–1053, 2022. DOI: 10.1016/j.procs.2022.01.125.
- [25] CORDIS - EU Research, “Ash Ingestion Detection for Aircraft (AIDA) Final Report,” European Comission, Tech. Rep., 2019, <https://cordis.europa.eu/project/id/304788/reporting> (visitado el 4 de agosto, 2025).
- [26] E.T. IJzermans, “Machine Learning for Predictive Maintenance: A Boeing 747 Bleed Air Valves case study,” Delft University of Technology, Tech. Rep., 2019, <https://repository.tudelft.nl/record/uuid:27037d74-d49b-4dfe-bcad-ccf0a0bfd957> (visitado el 4 de agosto, 2025).
- [27] NASA. “CMAPSS Jet Engine Simulated Data.” <https://data.nasa.gov/dataset/cmapss-jet-engine-simulated-data> (visitado el 15 de abril, 2025). (Jul. 14, 2022).
- [28] M. Xiong and H. Wang, “Digital twin applications in aviation industry: A review,” *The International Journal of Advanced Manufacturing Technology*, vol. 121, pp. 5677–5692, 2022. DOI: 10.1007/s00170-022-09717-9.
- [29] S. Ma, K. A. Flanigan, and M. Bergés, “State-of-the-art review and synthesis: A requirement-based roadmap for standardized predictive maintenance automation using digital twin technologies,” *Advanced Engineering Informatics*, vol. 62, p. 102800, 2024. DOI: 10.1016/j.aei.2024.102800.
- [30] A. P. Hermawan, D.-S. Kim, and J.-M. Lee, “Predictive Maintenance of Aircraft Engine using Deep Learning Technique,” *IEEE ICTC 2020 Conference Proceedings*, pp. 1296–1298, 2020. DOI: 10.1109/ICTC49870.2020.9289466.
- [31] A. Al Hasib, A. Rahman, M. Khabir, and M. T. Rouf Shawon, *An Interpretable Systematic Review of Machine Learning Models for Predictive Maintenance of Aircraft Engine*, 2023. DOI: 10.48550/arXiv.2309.13310.
- [32] A. Aminzadeh, “A Machine Learning Implementation to Predictive Maintenance of Industrial Compressors,” *Sensors*, vol. 25, no. 4, p. 1006, 2025. DOI: 10.3390/s25041006.
- [33] A. Vicente, *Machine Learning Approaches to Solve Boundary Layer Problems*, Madrid, España, Sep. 2023.
- [34] J. Holland, “Airbus extends Skywise digital platform with new features,” *MRO Management / Aviation Business News*, Nov. 2022, <https://www.aviationbusinessnews.com/mro/airbus-extends-skywise-digital-platform-with-new-features/> (visitado el 25 de junio, 2025).

- [35] A. Ouadah, L. Zemmouchi-Ghomari, and N. Salhi, “Selecting an appropriate supervised machine learning algorithm for predictive maintenance,” *The International Journal of Advanced Manufacturing Technology*, vol. 119, pp. 4277–4301, 2022. DOI: 10.1007/s00170-021-08551-9.
- [36] T. Łodygowski and S. Szrama, “Unsupervised Classification and Remaining Useful Life Prediction for Turbofan Engines Using Autoencoders and Gaussian Mixture Models: A Comprehensive Framework for Predictive Maintenance,” *Applied Sciences*, vol. 15, no. 14, p. 7884, 2025. DOI: 10.3390/app15147884.
- [37] A. Kumar. “Supervised & Unsupervised Learning Difference.” <https://vitalflux.com/dummies-notes-supervised-vs-unsupervised-learning/> (visitado el 13 de agosto, 2025). (May 2, 2023).
- [38] A. A. Torres-García, C. A. Reyes-García, L. Villaseñor-Pineda, and O. Mendoza-Montoya, Eds., *Biosignal Processing and Classification Using Computational Learning and Intelligence: Principles, Algorithms, and Applications*. Cambridge, MA: Academic Press, Elsevier, 2022, ISBN: 978-0-12-820125-1. DOI: <https://doi.org/10.1016/C2019-0-00985-5>.
- [39] G. M. T. Tchio, J. Kenfack, D. Kassegne, F.-D. Menga, and S. S. Ouro-Djobo, “A Comprehensive Review of Supervised Learning Algorithms for the Diagnosis of Photovoltaic Systems, Proposing a New Approach Using an Ensemble Learning Algorithm,” *Applied Sciences*, vol. 14, no. 5, p. 2072, 2024. DOI: 10.3390/app14052072.
- [40] A. H. Baradaran, “Predictive Maintenance of Electric Motors Using Supervised Learning Models: A Comparative Analysis,” *arXiv Computer Science*, Mar. 20, 2025. DOI: 10.48550/arXiv.2504.03670.
- [41] DATAtab. “Regresión lineal - Tutorial.” <https://datatab.net/tutorial/linear-regression>(visitado el 17 de julio, 2025). (2025).
- [42] J. Santibáñez. “Verificación del supuesto de homocedasticidad.” (2025), [http://sigma.iimas.unam.mx/jasantibanez/Cursos/Ciencias/2018\\_1/08\\_homocedasticidad.html](http://sigma.iimas.unam.mx/jasantibanez/Cursos/Ciencias/2018_1/08_homocedasticidad.html)(visitado el 3 de agosto, 2025). (2018).
- [43] Amazon Web Services. “¿Qué es la regresión lineal?” <https://aws.amazon.com/es/what-is/linear-regression/>(visitado el 17 de julio, 2025). (2025).
- [44] Spiceworks. “Linear regression vs logistic regression.” <https://www.spiceworks.com/tech/artificial-intelligence/articles/linear-regression-vs-logistic-regression/> (visitado el 17 de julio, 2025). (2025).
- [45] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 3rd. Stanford, 2025, Capítulo 5: Logistic Regression. [https://web.stanford.edu/~jurafsky/slp3/ed3book\\_Jan25.pdf](https://web.stanford.edu/~jurafsky/slp3/ed3book_Jan25.pdf).

- [46] Freie Universität Berlin. “Nearest Neighbors Algorithm.” <https://www.geo.fu-berlin.de/en/v/soga-r/Machine-Learning/Nearest-Neighbors-Algorithm/index.html> (visitado el 17 de julio, 2025). (2025).
- [47] N. Bhaskar, B. S. Rani, A. S. Reddy, X. S. A. Shiny, B. Gayathri, and R. V. Reddy, “Evaluation on the Effectiveness of SVM-Based Image Processing Algorithms for the Identification and Classification of Lung Cancer,” in *Proceedings of the 15th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2023)*, ser. Lecture Notes in Networks and Systems, vol. 1243, Cham: Springer, 2025, pp. 532–541. DOI: 10.1007/978-3-031-81080-0\_51.
- [48] Low Code for Advanced Data Science (Medium). “Support Vector Machines (SVM): An intuitive explanation.” <https://medium.com/low-code-for-advanced-data-science/support-vector-machines-svm-an-intuitive-explanation-b084d6238106> (visitado el 17 de julio, 2025). (2025).
- [49] I. Assagaf, A. Sukandi, A. A. Abdillah, and S. Arifin, “Machine Predictive Maintenance by Using Support Vector Machines,” *Recent in Engineering Science and Technology*, vol. 1, no. 2, 2023. DOI: 10.59511/riestech.v1i01.6.
- [50] Scikit-learn developers. “Decision Trees.” <https://scikit-learn.org/stable/modules/tree.html> (visitado el 17 de julio, 2025). (2025).
- [51] F. Musacchio. “Decision Trees vs. Random Forests – What is the Difference?” [https://www.fabriziomusacchio.com/blog/2023-06-22\\_decision\\_trees\\_vs\\_random\\_forests/](https://www.fabriziomusacchio.com/blog/2023-06-22_decision_trees_vs_random_forests/) (visitado el 17 de julio, 2025). (2023).
- [52] M. Islam, G. Chen, and S. Jin, “An Overview of Neural Network,” *American Journal of Neural Networks and Applications*, vol. 5, no. 1, pp. 7–11, 2019, ISSN: 2469-7400. DOI: 10.11648/j.ajnna.20190501.12.
- [53] G. Nota, F. D. Nota, A. Toro, and M. Nastasia, “A Framework for Unsupervised Learning and Predictive Maintenance in Industry 4.0,” *International Journal of Industrial Engineering and Management*, vol. 15, no. 4, pp. 304–319, Oct. 11, 2024. DOI: 10.24867/IJIEM-2024-4-365.
- [54] IBM. “What is clustering?” <https://www.ibm.com/think/topics/unsupervised-learning> (visitado el 17 de julio, 2025). (2025).
- [55] S. Li. “Lecture 12: Gaussian Mixture Models (GMM).” [https://shuaili8.github.io/Teaching/VE445/L12\\_gmm.pdf](https://shuaili8.github.io/Teaching/VE445/L12_gmm.pdf) (visitado el 17 de julio, 2025). (2023).
- [56] Teddy van Jerry. “Why do we want to maximize the variance in Principal Component Analysis?” <https://stats.stackexchange.com/questions/305430/why-do-we-want-to-maximize-the-variance-in-principal-component-analysis> (visitado el 19 de julio, 2025). (2017).

- [57] V. Guruswami *et al.* “Singular Value Decomposition (SVD): Overview.” <https://www.cs.cmu.edu/~venkatg/teaching/CStheory-infoage/book-chapter-4.pdf> (visitado el 19 de julio, 2025). (2009).
- [58] AI Planet. “Auto-codificadores para la reducción de la dimensionalidad.” <https://aiplanet.com/learn/unsupervised-learning-es/introduccion-a-la-reduccion-de-la-dimensionalidad-y-sus-tecnicas/1682/auto-codificadores-para-la-reduccion-de-la-dimensionalidad> (visitado el 19 de julio, 2025). (2025).
- [59] IBM. “Semi-supervised learning.” <https://www.ibm.com/think/topics/semi-supervised-learning> (visitado el 15 de julio, 2025). (2025).
- [60] A. S. Yoon *et al.*, “Semi-supervised Learning with Deep Generative Models for Asset Failure Prediction,” *arXiv Computer Science*, Sep. 4, 2017. DOI: 10.48550/arXiv.1709.00845.
- [61] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018, <http://incompleteideas.net/book/the-book-2nd.html>.
- [62] Y. Chen and C. Liu, “Sequential Multi-objective Multi-agent Reinforcement Learning Approach for Predictive Maintenance,” *arXiv Electrical Engineering and Systems Science*, Feb. 4, 2025. DOI: 10.48550/arXiv.2502.02071.
- [63] J. López. “¿Qué es Overfitting y Underfitting y cómo solucionarlo?” <https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/> (vistado el 25 de julio, 2025). (2020).
- [64] MathWorks. “What is Overfitting?” <https://www.mathworks.com/discovery/overfitting.html> (visitado el 19 de julio, 2025). (2025).
- [65] GeeksforGeeks. “F1 Score in Machine Learning.” <https://www.geeksforgeeks.org/machine-learning/f1-score-in-machine-learning/> (visitado el 23 de julio, 2025). (2025).
- [66] K. R. Shahapure and C. Nicholas, “Cluster Quality Analysis Using Silhouette Score,” in *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2020, pp. 747–748, ISBN: 978-1-7281-8206-3. DOI: 10.1109/DSAA49011.2020.00096.
- [67] J. Brownlee. “A Gentle Introduction to k-fold Cross-Validation.” <https://machinelearningmastery.com/k-fold-cross-validation/> (visitado el 23 de julio, 2025). (2023).
- [68] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011, <https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf>.

- [69] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv Computer Science*, 2016. DOI: 10.48550/arXiv.1609.04747.
- [70] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, <https://proceedings.mlr.press/v28/sutskever13.html>, 2013, pp. 1139–1147.
- [71] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *International Conference on Learning Representations*, 2015. DOI: 10.48550/arXiv.1412.6980.
- [72] scikit-learn developers. “scikit-learn: Open-source Machine Learning in Python.” <https://scikit-learn.org/stable/index.html> (visitado el 22 de julio, 2025). (2025).
- [73] Python Software Foundation and Community. “Python Data Science Libraries Documentation.” <https://docs.python.org/3/library/> (visitado el 24 de julio, 2025). (2025).
- [74] U. Farooq, M. Ademola, and A. Shaalan, “Comparative Analysis of Machine Learning Models for Predictive Maintenance of Ball Bearing Systems,” *Electronics*, vol. 13, no. 2, p. 438, 2024. DOI: 10.3390/electronics13020438.
- [75] R. Palem. “Bill Authentication Dataset.” <https://www.kaggle.com/datasets/anjanisona/bill-authentication> (visitado el 20 de julio, 2025). (2017).
- [76] K. Aainsqatsi. “Turbofan operation - es.svg.” Own work, Wikimedia Commons, licenced under CC BY 2.5 [https://commons.wikimedia.org/wiki/File:Turbofan\\_operation\\_-\\_es.svg](https://commons.wikimedia.org/wiki/File:Turbofan_operation_-_es.svg). (2008).
- [77] A. Saxena, K. Goebel, D. Simon, and N. Eklund, “Damage Propagation Modeling for Aircraft Engine Run-to-Failure Simulation,” *IEEE Aerospace Conference*, pp. 1–9, 2008. DOI: 10.1109/PHM.2008.4711414.
- [78] EITCA Academy. “¿Cuál es el propósito de normalizar los datos antes de entrenar una red neuronal?” <https://es.eitca.org/inteligencia-artificial/> (visitado el julio, 2025). (2024).
- [79] V. Nair and G. E. Hinton, “Rectified linear units improve restricted Boltzmann Machines,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, <http://www.icml2010.org/papers/432.pdf>, 2010, pp. 807–814.
- [80] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, <http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf>, 2011, pp. 315–323.

- [81] Y. Mitsufuji, Y. Matsuo, and S.-i. Amari, “Autoencoders that Don’t Overfit Towards the Identity,” in *Advances in Neural Information Processing Systems*, <https://tinyurl.com/yhwmu5yb>, vol. 33, 2020, pp. 19 283–19 294.
- [82] M. Dwarampudi and N. V. S. Reddy, “Effects of padding on LSTMs and CNNs,” *arXiv Computer Science*, 2019. DOI: 10.48550/arXiv.1903.07288.
- [83] N. E. Bouharrouti, D. Morinigo-Sotelo, and A. Belahcen, “Multi-Rate Vibration Signal Analysis for Bearing Fault Detection in Induction Machines Using Supervised Learning Classifiers,” *Machines*, vol. 12, no. 1, p. 17, 2024. DOI: 10.3390/machines12010017.