Universidad de Cantabria

Facultad de Ciencias

AUTOMATIZACIÓN DE OPERACIONES DE MANTENIMIENTO EN FUNCIÓN DE LA CONDICIÓN

AUTOMATION OF CONDITION-BASED MAINTENANCE OPERATIONS

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Javier Arce del Campo

Directora: Marta Elena Zorrilla Pantaleón

Co-Director: Miguel Romero Núñez

Septiembre - 2025

Resumen

En el marco de la Industria 4.0, el mantenimiento industrial está evolucionando de estrategias reactivas y preventivas hacia enfoques predictivos, apoyados en el análisis de datos en tiempo real y aprendizaje automático.

Este trabajo tiene como objetivo desarrollar e integrar modelos de predicción y detección de anomalías en la plataforma industrial Fast Monitoring, desarrollada por la empresa Soincon, utilizada en entornos de monitorización en tiempo real.

La metodología seguida está basada en el estándar de facto CRISP-DM, abordando el preprocesamiento de los datos, el entrenamiento de los modelos y su posterior validación. Se implementan tres casos de uso principales: (i) estimación del tiempo de vida útil restante de un motor turbofán mediante XGBoost, (ii) predicción iterativa de señales temporales, y (iii) detección de anomalías en señales de naturaleza acumulativa empleando Isolation Forest. Los modelos fueron desarrollados y validados en Python y posteriormente exportados a formato ONNX e integrados en la plataforma de la empresa.

Los resultados obtenidos exponen que los modelos cumplen adecuadamente los criterios definidos demostrando un rendimiento satisfactorio para cada caso de uso analizado. La integración de estos modelos como componentes BPM en Fast Monitoring permite incorporar componentes de predicción y detección de anomalías en flujos industriales y sientan las bases para futuros desarrollos, posibilitando la generación de alertas en tiempo real y optimización de tareas de mantenimiento.

En resumen, el trabajo desarrolla tres modelos predictivos aplicando técnicas de aprendizaje automático y valida su aplicabilidad en escenarios industriales reales, contribuyendo a la adopción de un mantenimiento predictivo más eficiente, preciso y escalable.

Palabras clave: Mantenimiento predictivo, Industria 4.0, Aprendizaje automático, ONNX, Isolation Forest, XGBoost.

Abstract

In the context of Industry 4.0, industrial maintenance is evolving from reactive and preventive strategies towards predictive approaches, supported by real-time data analysis and machine learning.

The objective of this work is to develop and integrate prediction and anomaly detection models into the industrial platform Fast Monitoring, developed by the company Soincon and used in real-time monitoring environments.

The methodology followed is based on the de facto standard CRISP-DM, addressing data preprocessing, model training, and subsequent validation. Three main use cases are implemented: (i) estimation of the remaining useful life (RUL) of a turbofan engine using XGBoost, (ii) iterative prediction of time series signals, and (iii) anomaly detection in cumulative signals employing Isolation Forest. The models were developed and validated in Python, exported to ONNX format, and subsequently integrated into the company's platform.

The results obtained show that the models adequately meet the defined criteria, demonstrating satisfactory performance for each analyzed use case. The integration of these models as BPM components in Fast Monitoring enables the incorporation of prediction and anomaly detection capabilities into industrial workflows and lays the foundations for future developments, such as real-time alert generation and optimization of maintenance tasks.

In summary, this work develops three predictive models using machine learning techniques and validates their applicability in real industrial scenarios, contributing to the adoption of more efficient, accurate, and scalable predictive maintenance.

Keywords: Predictive maintenance, Industry 4.0, Machine learning, ONNX, Isolation Forest, XGBoost.

Índice

Índice de figuras

Índice de cuadros

1. Motivación y objetivos

En este capítulo se contextualiza la temática y motivación del proyecto, y se presentan los objetivos que persigue este trabajo así como la planificación temporal para su consecución. Finalmente, se recoge la organización de la memoria.

1.1. Contexto

Este trabajo de fin de grado se ha desarrollado en el marco de una oferta realizada por la empresa Soincon en el Centro de Orientación e Información de Empleo de la Universidad de Cantabria. La actividad se ha desarrollado de forma presencial, durante un período de seis meses de prácticas extracurriculares dedicadas exclusivamente a llevar a cabo este proyecto.

Soincon[1] se define como una empresa especializada en la digitalización de sistemas a través del desarrollo de soluciones software que mejoran la productividad, calidad, mantenimiento y producción de entornos industriales ayudando al sector en la transición digital hacia la Industria 4.0. En este contexto, el mantenimiento de los sistemas industriales está adquiriendo una relevancia creciente.

De acuerdo a [1] se distinguen tres estrategias de mantenimiento:

1. Mantenimiento correctivo: es la estrategia más básica y no requiere planificación previa. Este método reactivo espera a que el sistema falle para intervenir. En consecuencia, es la estrategia más costosa.

2. Mantenimiento preventivo: es la estrategia más implementada.
   Las tareas de mantenimiento se programan periódicamente basadas en horas o ciclos de operación. Sin embargo, la probabilidad de fallo de una máquina fluctúa a lo largo de su vida útil, luego una vez superado cierto umbral, se asumen los costes de la adopción de ciertas tareas correctivas que pueden ser innecesarias.

3. Mantenimiento predictivo (PdM): es la estrategia que despierta mayor interés en el contexto actual. Busca optimizar el momento de la intervención de mantenimiento, realizándola en el momento correcto, a través de la monitorización de las condiciones mecánicas en ese momento, eficiencia operacional y otros indicadores como los resultados de un análisis de datos históricos de los sistemas.

Las estrategias tradicionales, basadas principalmente en el correctivo, están evolucionando hacia enfoques más avanzados como pueden ser el mantenimiento basado en condiciones operativas y el mantenimiento predictivo, considerados pilares esenciales en esta nueva era industrial [2]. Estos enfoques se sustentan en la capacidad de realizar una monitorización continua de los equipos industriales y del análisis avanzado de los datos extraídos de los mismos, presentando beneficios sustanciales sobre tareas de mantenimiento más correctivas tales como: (i) reducción de número de fallos no esperados, (ii) protección frente a comportamientos erráticos o (iii) mejor planificación de paradas.

El mantenimiento de sistemas industriales se trata de una tarea crucial dado los altos costes tanto económicos como reputacionales que pueden provocar los fallos no detectados. En la actualidad, la mayoría de las industrias continúan confiando en estrategias preventivas pues, generalmente son efectivas y no requieren de un entorno de monitorización en tiempo real del sistema a mantener; sin embargo, el reciente auge de la inteligencia artificial y la minería de datos, ha generado un repunte en el interés (tanto en el ámbito académico como profesional) por la adopción de una estrategia preventiva [1].

[1] https://emisuite.es/

1.2. Objetivos del proyecto

El presente trabajo de fin de grado tiene por objeto desarrollar un módulo para la monitorización de equipos industriales y su mantenimiento predictivo e integrarlo en la plataforma Fast Monitoring (descrita en la Sección 4.1) de la empresa Soincon. El proyecto se desarrollará en tres etapas principales:

1. Construcción de modelos de predicción
   - *a*) Se investigarán en la literatura modelos de predicción y detección de patrones aplicados a mantenimiento basados en condiciones, con el objetivo de determinar los algoritmos más adecuados para cada caso de estudio.
   - *b*) Se trabajará con diversos algoritmos que tengan la posibilidad de ser exportados a un formato compatible con el lenguaje C# para posibilitar su uso en el entorno de trabajo de Fast Monitoring, incluyendo XGboost e Isolation Forest (algoritmos estudiados con mayor profundidad durante la formación previa).
   - *c*) Se definirán criterios de mantenimiento, esto es, se establecerán reglas y umbrales operativos en los modelos para la identificación de situaciones que requieran acciones de mantenimiento predictivo.
2. Integración de los modelos desarrollados en la plataforma Fast Monitoring.
   - *a*) Se integrarán los modelos desarrollados en la plataforma Fast Monitoring, permitiendo que los modelos de predicción sean invocados y ejecutados dentro del entorno de la plataforma.
3. Generación de alertas, avisos y automatización de trabajos de mantenimiento.
   - *a*) Se diseñará un *pipeline* automatizado que conecte la monitorización en tiempo real, la ejecución de los modelos y la generación de acciones de mantenimiento a partir de las predicciones.

1.3. Planificación del proyecto

La planificación del proyecto se ha estructurado en las fases que se muestran en la Figura 1.

- Formación inicial: hace referencia al período dedicado al estudio de bibliografía relacionada con aprendizaje automático y al desarrollo de proyectos de prueba que permitieron adquirir una base teórica y práctica suficiente para abordar los modelos que se desarrollan en este trabajo.
- Memoria: se redacta en paralelo durante todo el desarrollo del proyecto.
- Datos: hace referencia a la extracción y preprocesamiento de los conjuntos de datos empleados en cada modelo.
- Modelado: engloba la selección del algoritmo así como su entrenamiento y validación.
- FM (Fast Monitoring): corresponde a la integración de los modelos en el entorno de la plataforma de la empresa.

Asimismo, el proyecto se divide en tres modelos principales (M1, M2, M3) que representan diferentes casos de uso en el ámbito del mantenimiento industrial.

- M1: detección de anomalías en señales.

- M2: predicción del tiempo de vida útil restante (RUL, de sus siglas en ingés, *Remaining Useful Life*) de un componente industrial.
- M3: continuación de señales.

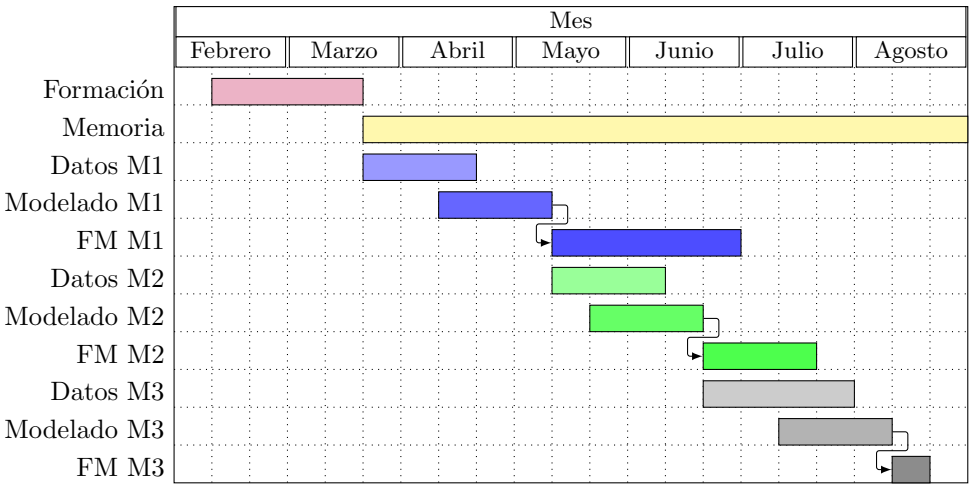

Figura 1: Planificación temporal del proyecto.

En el diagrama se observa que las fases de *Datos* y *Modelado* se solapan pues, durante el ajuste de los algoritmos, han surgido modificaciones en el preprocesamiento de los datos, consecuencia de la naturaleza cíclica del proceso CRISP-DM, donde la evaluación de un modelo te lleva a realizar nuevas propuestas de configuración de datos de entrada o del algoritmo.

En cuanto a la integración en la plataforma empresarial, solo se inicia una vez finalizado cada modelo.

1.4. Organización de la memoria

La memoria se ha organizado en varios capítulos.
En este primer capítulo, se presenta la motivación, objetivos y planificación del proyecto.

En el capítulo 2, se incluyen antecedentes teóricos que se consideran necesarios para comprender el trabajo. En el capítulo 3, se describe el desarrollo de los diferentes modelos y su integración en la plataforma Fast Monitoring. Seguidamente, se aborda la generación de alertas, avisos y automatización de trabajos de mantenimiento. Finalmente, en el capítulo 6, se presentan las conclusiones.

2. Antecedentes

En este capítulo se presentan conceptos fundamentales sobre aprendizaje automático así como metodologías, tecnologías y herramientas para construir modelos predictivos con el fin de ayudar al lector a comprender el desarrollo realizado en el trabajo.

2.1. Aprendizaje automático

El aprendizaje automático o el *Machine Learning* (ML) se define como el campo de la informática que estudia cómo dotar a los ordenadores de la capacidad de aprender automáticamente a partir de la experiencia, mejorando su rendimiento en tareas específicas sin estar programados de forma explícita para ello [3].

2.1.1. Paradigmas de aprendizaje automático

Dependiendo de cómo cada algoritmo aprende a partir de los datos para realizar las predicciones, así como de la disponibilidad de los mismos, naturaleza de estos datos y el objetivo del problema, se pueden definir cuatro tipos de paradigmas de aprendizaje:

- Aprendizaje supervisado: El modelo de ML utiliza datos etiquetados, es decir, al modelo se le proporciona unas variables de entrada y el modelo conoce la correspondiente salida asociada. Este tipo de aprendizaje está comúnmente relacionado con tareas de regresión y clasificación, que se detallarán más adelante. En el caso de PdM, se aplica a problemas como la estimación del tiempo de vida útil restante. En este contexto es importante conocer el funcionamiento correcto del sistema y los eventos que provocan fallo [1] [4] .

  En función de la naturaleza del valor que se desea predecir, se distinguen dos casos principales:

  - Clasificación: busca agrupar elementos en una serie de categorías predefinidas basado en sus características. Dicho esto, la salida de un modelo basado en este tipo de problema, será discreta, por ejemplo, *normal* o *anómalo*, útil, por ejemplo, en la detección automática del estado de una máquina y decidir si es necesario actuar [5].
  - Regresión: busca modelar el comportamiento de una variable cuantitativa en función de otras variables predictoras, que pueden ser cuantitativas o cualitativas, con el objetivo habitual de realizar predicciones o estimaciones. Por ejemplo, la salida de un modelo basado en un problema de regresión, podrá tener como salida un número real 3.0 o 101.3, estimación numérica que podría permitir anticipar el tiempo de vida útil restante de una máquina o el consumo de energía [5].

- Aprendizaje no supervisado: Los datos con los que se entrena el modelo de ML no se encuentran etiquetados. Además, no se conoce el correcto funcionamiento del sistema o los eventos de fallo. Se espera que el modelo encuentre patrones o estructuras subyacentes en los datos. Entre las técnicas más comunes se encuentran el *clustering*, que agrupa instancias similares en función de sus características, y las reglas de asociación, que buscan dependencias o relaciones frecuentes entre variables. Este tipo de aprendizaje se aplica en problemas como la detección de anomalías [1] [4] .

- Aprendizaje semi-supervisado: Para el entrenamiento del modelo de ML se utiliza una combinación de datos etiquetados y no etiquetados. En el contexto de PdM, se aplica comúnmente a problemas con fallos raros o infrecuentes, donde los datos están etiquetados pero no hay ejemplos de fallos en el sistema [1] [4] .

- Aprendizaje por refuerzo: Este tipo de aprendizaje permite al modelo de ML aprender a través de reglas o acciones de intento y error con el fin de descubrir las mejores acciones. Aplicado en el contexto, por ejemplo, de la conducción autónoma, aunque su aplicación no es común en el ámbito de PdM [1] [4] .

2.1.2. Fundamentos del entrenamiento en modelos de aprendizaje automático

Antes de seleccionar un algoritmo o un conjunto de ellos para entrenar un modelo predictivo de ML, es necesario comprender una serie de fundamentos que tienen relación directa con la calidad y precisión de los resultados predichos.

A continuación se presentan los principales conceptos a considerar para generar un modelo fiable, eficiente y coherente con el objetivo de predicción a conseguir.

- Parametrización del modelo "Hiperparámetros": Conforman una configuración de un modelo cuyos valores no pueden ser estimados por los datos y tienen que ser establecidos antes del proceso de entrenamiento. Los hiperparámetros controlan aspectos como la complejidad del modelo, la velocidad de aprendizaje o la regularización [5].

- Concepto de desajuste y sobreajuste: Se dice que un modelo está sobreajustado o presenta ***overfitting*** cuando se ajusta demasiado a los datos de entrenamiento, capturando el ruido y las características específicas de los datos de entrenamiento en lugar de aprender patrones generales, dando lugar a rendimiento deficiente en datos nuevos o no conocidos. Al contrario, se dice que un modelo está desajustado o presenta ***underfitting*** cuando es demasiado simple para los datos, lo que resulta en un rendimiento deficiente en los datos de entrenamiento y prueba [5].

  En este sentido, el proceso de entrenamiento de un modelo de *machine learning* no puede entenderse de forma aislada, sino junto a la aplicación de métodos de evaluación que permitan detectar tanto el sobreajuste como el desajuste mencionados. Dichos métodos dependen del tipo de problema: en problemas de clasificación se emplean métricas como la precisión, el *recall* o el F1-score. Por otro lado, en problemas de regresión se utilizan medidas como el *Mean Absolute Error*, el *Root Mean Squared Error* o el R^2. Estas métricas permiten estimar la capacidad de generalización del modelo y comprobar si el ajuste logrado durante el entrenamiento se traslada a datos no vistos.

- Métricas de rendimiento: Dependiendo del tipo de problema a resolver, se evaluarán unas métricas u otras.

  Métricas empleadas en problemas de clasificación:
  Con el fin de medir el rendimiento de un modelo, se emplean diferentes métodos de evaluación, destacando estrategias de validación basadas en la división del conjunto de datos en subconjuntos de entrenamiento, validación y test, así como técnicas de búsqueda de hiperparámetros donde mediante la validación cruzada (*k-fold cross-validation*) permiten obtener una estimación más fiable del desempeño del mismo. Una vez definido el procedimiento de evaluación, las métricas que se presentan a continuación permiten cuantificar la calidad de las predicciones generadas por el modelo.

  Precision — Proporción de verdaderos positivos sobre el total de predicciones positivas realizadas. Indica qué tan preciso es el modelo al predecir la clase positiva [5].

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

  Donde TP es el número verdaderos positivos (predichos como positivos y realmente positivos), y FP el número de falsos positivos (predichos como positivos pero realmente negativos).

  Recall — Proporción de verdaderos positivos que el modelo fue capaz de identificar correctamente sobre el total real de positivos [5].

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

Donde FN es el número de falsos negativos (predichos como negativos pero realmente positivos). En problemas multiclase, la precisión se calcula para cada clase de manera independiente y, posteriormente, se agregan los resultados mediante promedios (*macro* o *weighted*), según se desee dar igual peso a cada clase o ponderar por su frecuencia en el conjunto de datos.

F1 Score — Media armónica entre precisión y *recall*. Es útil cuando existe un desbalance de clases y se quiere un equilibrio entre ambas métricas.

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

Métricas empleadas en problemas de regresión:

MAE — Error absoluto medio entre las predicciones y los valores reales [6].

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{4}$$

RMSE — Raíz del error cuadrático medio. Penaliza más los errores grandes [6].

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{5}$$

Donde:

- y_i: valor real
- $\hat{y}_i$: valor predicho
- n: número total de observaciones

RMSE se dice de la métrica de evaluación que mide la diferencia promedio al cuadrado entre los valores predichos y los reales, devolviendo la raíz cuadrada de dicho promedio.

R² — Mide la proporción de la varianza total de la variable dependiente que es explicada por el modelo.

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{6}$$

Donde $\bar{y}$ representa la media de los valores reales [6].

2.1.3. Algoritmos de aprendizaje automático

A continuación, se describen los algoritmos de Machine Learning utilizados en este trabajo, con el fin de justificar su selección y facilitar la comprensión de su funcionamiento. Los algoritmos empleados han sido seleccionados por dos motivos: (i) son usados frecuentemente en problemas de mantenimiento predictivo, presentando un buen equilibrio entre rendimiento e interpretación, (ii) fueron introducidos durante la formación previa realizada antes de iniciar el proyecto, lo que permitió adquirir una base teórica y práctica para su aplicación.

Algoritmo 1 Isolation Forest

Isolation Forest se define como un método de detección de anomalías u *outliers* que se basa en aislar aquellos valores que se consideran atípicos, esto es, valores que difieren notablemente de la norma. [7]

El algoritmo construye un conjunto de árboles de aislamiento binarios de carácter aleatorio (*iTrees*) como modelo base. Cada *iTree* se construye mediante la partición recursiva de una muestra de los datos. En cada nodo, se elige al azar una característica p y se selecciona al azar un valor de división q entre los valores máximo y mínimo de esa característica en el subconjunto actual. Esta división divide los datos en dos subconjuntos (nodos hijo izquierdos y derechos), como en un árbol de decisión. La partición continúa recursivamente hasta que se cumple una de las condiciones de parada: *(i)* el nodo alcanza un límite máximo de altura predefinido, *(ii)* el subconjunto solo tiene una instancia, es decir, el punto está aislado, o *(iii)* todas las instancias del subconjunto tienen valores idénticos, haciendo imposible realizar más divisiones. Por definición, cuando un *iTree* crece completamente sin restricciones para aislar todos los n puntos de una muestra, tendrá n nodos externos (hojas, representando una instancia aislada) y $n - 1$ nodos de división internos, convirtiéndolo en un árbol binario propiamente dicho. Esto significa que el tamaño del modelo crece linealmente con el número de instancias, haciéndolo ineficiente en términos de memoria. En la práctica, es común utilizar solo una pequeña submuestra aleatoria de los datos, de tamaño *psi* para construir cada árbol, creando un conjunto de t *iTrees*, mejorando así la eficiencia y mitigando el enmascaramiento en el que las anomalías podrían quedar ocultas entre grandes cantidades de datos normales. Dicho esto, el algoritmo Isolation Forest tendría dos parámetros principales (sin incluir límites de altura): el tamaño de submuestreo *psi* y el número de árboles t [7].

Una vez el algoritmo ha sido entrenado, se puede evaluar cualquier instancia nueva de datos pasándola por cada árbol para calcular la longitud del camino $h(x)$ en ese árbol, es decir, el número de aristas recorridas desde la raíz para llegar al nodo final. Una instancia que presenta un valor atípico, terminaría más cerca de la raíz ($h(x)$ corto) mientras que un dato normal suele estar mezclado con otros puntos similares, por lo que haría falta dividirlo muchas veces antes de aislarlo, es decir, para llegar a él, el recorrido desde el nodo raíz del árbol sería más largo ($h(x)$ mayor) [7] .

En la figura 2, se representa un diagrama ilustrativo del mecanismo de aislamiento en Isolation Forest, un conjunto de árboles que se combinan para formar un bosque de árboles, utilizados para detectar los valores atípicos (nodos rojos), mientras que los puntos de datos normales (nodos azules) se encontrarían mucho más profundos en el árbol al requerir más particiones hasta aislarse, por tanto, no se muestran:

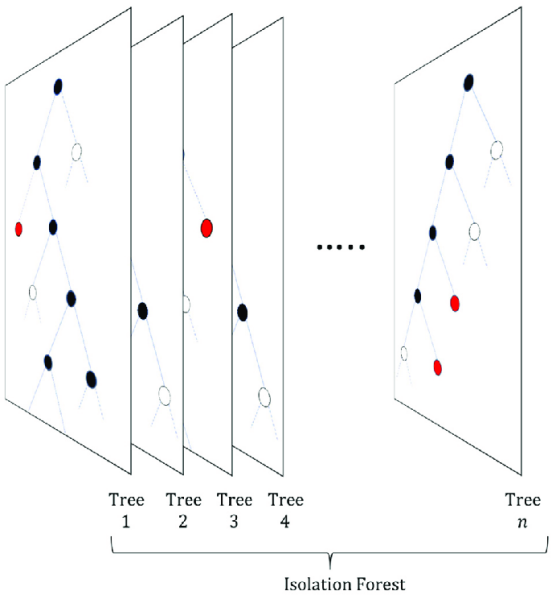

Figura 2: Ilustración esquemática del mecanismo de Isolation Forest. [8]

En base a este análisis, se podría obtener una puntuación de anomalía para un punto de datos (x) basada en la longitud de trayectoria esperada en todos los árboles del bosque, denominada (*score*). La longitud esperada del camino para una instancia x, que se denota $E(h(x))$, se compara con una constante de normalización $c(n)$, que representa la longitud media esperada de un árbol binario con n nodos. Basándose en esta relación, se define el siguiente *score*:

$$s(x) = 2^{-\frac{E(h(x))}{c(n)}} \tag{7}$$

Donde n es el número de instancias utilizadas para construir cada árbol (el tamaño de la submuestra *psi*, a efectos de evaluación).

Esta fórmula, produce $s(x)$ en el rango $(0, 1]$, con valores más altos que indican más puntos anómalos, donde los puntos con trayectorias muy cortas producirían $s(x)$ más cerca de 1 (muy probable que sean anomalías). En la práctica, se pueden clasificar todos los puntos por su *score* de anomalía y marcar aquellos con mayores resultados como valores atípicos, permitiendo a Isolation Forest funcionar de manera no supervisada, sin necesidad de etiquetas, pues $s(x)$ se calcula solamente a partir de propiedades de distribución de los datos en los árboles. [7]

Una de las principales ventajas que presenta este algoritmo para la detección no supervisada de anomalías, es su eficiencia computacional y su gran capacidad para trabajar con grandes conjuntos de datos multidimensionales.

La complejidad temporal que presenta el algoritmo es lineal con el número de puntos de datos y requisitos de memoria mínimos. Durante el entrenamiento, la construcción de t árboles en muestras de tamaño ψ tiene una complejidad del orden de $O(t \cdot \psi \log \psi)$ (cada construcción de iTree es $O(\psi \log \psi)$ de media). La puntuación de nuevas instancias es igual de eficiente: evaluar n puntos a través de t árboles cuesta $O(n \cdot t \log \psi)$ tiempo. Además, el algoritmo, dado su enfoque centrado en las anomalías, puede entrenarse solamente con datos normales, ya que no necesita etiquetas para identificar patrones anómalos durante la inferencia.

En resumen, Isolation Forest se trata de una herramienta robusta y computacionalmente eficiente para la detección no supervisada de anomalías, gracias a su independencia de etiquetas

o suposiciones sobre la distribución del *dataset*, principio de aislamiento aleatorio y bajo coste computacional [7].

Algoritmo 2 XGBoost

XGBoost es un sistema escalable de boosting basado en árboles de decisión diseñado con énfasis en la eficiencia y manejo de grandes volúmenes de datos [9].

El algoritmo se presenta como código abierto colaborativo y se ha convertido en una de las herramientas más utilizadas en la práctica por su capacidad de generar resultados competitivos en múltiples problemas de clasificación y regresión, encontrándose ampliamente adoptado tanto en entornos académicos como industriales.

El algoritmo XGBoost se fundamenta en el *gradient boosting*, un método en el que el modelo final se construye como un conjunto de árboles de decisión. En cada paso, se añade un nuevo árbol que corrige los errores de los anteriores, optimizando una función que combina la pérdida con una penalización de complejidad para evitar *overfitting*.

Más concretamente, el entrenamiento se realiza de manera aditiva. En cada iteración del algoritmo, se añade un nuevo árbol que trata de mejorar la predicción acumulada hasta ese momento. Para decidir cómo dividir los nodos de cada árbol, XGBoost evalúa las posibles divisiones mediante información del gradiente y la segunda derivada de la función de pérdida, permitiendo elegir las particiones que más reducen el error del modelo.

En la Figura 3 se muestra un ejemplo esquemático del funcionamiento de XGboost. Cada árbol aplica reglas de decisión sencillas y asigna una predicción parcial. En el ejemplo, el primer árbol considera la edad, y el segundo el uso del ordenador. La predicción final se obtiene como la suma de las salidas de ambos árboles. Este mecanismo de combinación de múltiples árboles débiles permite a XGboost capturar patrones complejos y lograr mayor precisión que un único árbol.

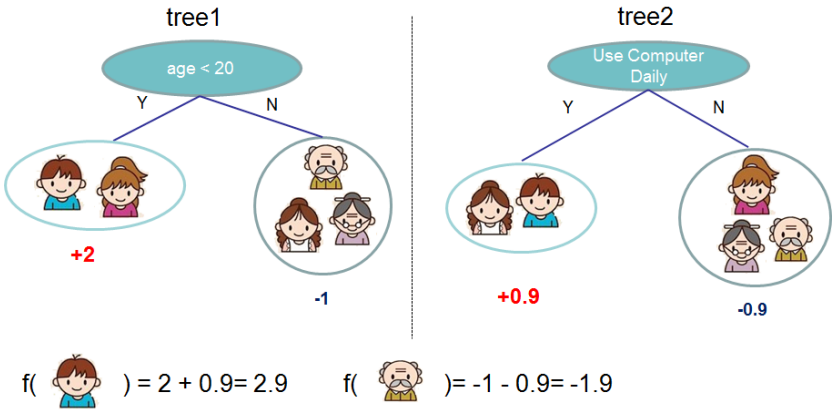

Figura 3: Modelo de ensamble en XGBoost. [9]

Relativo a la complejidad temporal, XGBoost es considerablemente eficiente pues se aprovecha de varias optimizaciones. En el algoritmo exacto con datos dispersos, la complejidad es proporcional a:

$$O(K \cdot d \cdot \|x\|_0 \log n)$$

donde K es el número de árboles, d la profundidad máxima, $\|x\|_0$ el número de entradas no nulas en los datos y n el número de instancias. [9]

Sin embargo, con la estructura de bloques optimizada propuesta en la literatura, esta complejidad se reduce a:

$$O(K \cdot d \cdot \|x\|_0 + \|x\|_0 \log n)$$

El segundo término, $\|x\|_0 \log n$, corresponde al coste de preprocesamiento necesario para ordenar los datos una única vez antes del entrenamiento. Este coste se amortiza en las sucesivas iteraciones, de modo que en la práctica el entrenamiento resulta casi lineal en el número de datos no nulos [9].

XGboost ha demostrado un gran impacto en diversos contextos, desde la predicción de clics en anuncios web hasta la clasificación de eventos en física de partículas. [9] En resumen, el algoritmo XGBoost se trata de una herramienta robusta y eficiente para problemas de clasificación y regresión a gran escala. Su capacidad de manejar datos dispersos, tratar valores faltantes, incorporar regularización y aprovechar optimizaciones de memoria lo han convertido en un estándar en múltiples aplicaciones prácticas [9], características que propician que el algoritmo sea adecuado para escenarios industriales como los que se estudiarán en este trabajo.

2.1.4. Aprendizaje automático aplicado al mantenimiento de sistemas industriales

En los últimos años, el mantenimiento de los sistemas industriales está evolucionando de estrategias reactivas o preventivas a un enfoque más eficiente, inteligente y basado en datos para respaldar las decisiones. En este contexto de cambio, el Machine Learning ejerce un papel importante, permitiendo el análisis de múltiples fuentes de datos (generalmente con series temporales) generadas por sensores, históricos de mantenimiento y características operacionales [10] [11].

Siendo altamente potenciado por teorías como la probabilidad, estadística, reconocimiento de patrones o el aprendizaje computacional en la inteligencia artificial, el ML es capaz de identificar patrones y realizar predicciones sobre datos, capacidad de gran utilidad en el ámbito del mantenimiento predictivo, donde se busca anticipar los fallos y realizar las actividades de mantenimiento en el momento oportuno, minimizando costes y evitando eventos inesperados no deseables. De este modo, el PdM supera a otras estrategias de mantenimiento como la correctiva —costosa por recambios de inventario— o preventiva —pudiendo provocar costes asociados a tareas correctivas innecesarias [10] [11].

Dicho esto, los algoritmos de Machine Learning, alimentados con datos recogidos en entornos industriales, permiten generar modelos predictivos. Estos modelos predictivos entrenados, son capaces detectar patrones anómalos, predecir la continuación de señales o incluso estimar el tiempo estimado de vida útil que le queda a un sistema o equipo industrial en un tiempo dado [10] [11].

Ejemplos en la literatura evidencian la aplicabilidad del aprendizaje automático en el campo del mantenimiento predictivo. Bekar et al. llevaron a cabo un estudio orientado al preprocesamiento y análisis de datos en entornos de fabricación, con el fin de mejorar la calidad de la información antes de alimentar los modelos predictivos y, de este modo, aumentar la fiabilidad de las predicciones de fallo generadas [12]. Otro ejemplo es el de Fathi et al., que aplicaron autoencoders para la detección de anomalías en robots delta industriales, mostrando que esta técnica es capaz de identificar desviaciones sutiles en el comportamiento de los actuadores y anticipar fallos en líneas de producción altamente automatizadas [12]. Estos casos ilustran cómo los modelos de ML permiten anticipar fallos, reducir costes y optimizar los planes de mantenimiento.

2.2. Tecnologías y herramientas utilizadas

A continuación, se describen las tecnologías empleadas en el proyecto, entre ellas: lenguajes de programación, librerías, IDEs y aplicaciones de uso general.

- C#: Lenguaje de alto nivel tipado orientado a objetos.
- Fast Monitoring: Plataforma diseñada por la empresa Soincon que gestiona, opera y explota datos de carácter industrial generados por plantas de tratamiento, dispositivos IoT, sistemas operativos y equipamiento de red, entre otros.
- ONNX (Open Neural Network Exchange): Estándar diseñado para representar modelos de *Machine Learning* permitiendo a los desarrolladores intercambiar modelos entre diferentes *frameworks*. De este modo, un modelo se puede entrenar en un entorno particular e implementarlo en otro sin realizar conversiones complejas de forma transparente.
- Python: Lenguaje de alto nivel no tipado característico por su eficiencia y uso en aplicaciones de inteligencia artificial.
- scikit-learn: Librería de código abierto con la que se pueden implementar y entrenar de forma sencilla diferentes algoritmos de *Machine Learning* en Python.
- TensorFlow: Biblioteca de código abierto para implementar algoritmos de *Machine Learning* desarrollada por Google.
- Visual Studio: IDE y editor de código compatible con C# y .NET.
- Visual Studio Code: Editor de código fuente general con soporte para operaciones de desarrollo.

2.3. Metodología CRISP-DM

En este trabajo, se ha optado por seguir la metodología CRISP-DM (*Cross Industry Standard Process for Data Mining*), un modelo de procesos estandarizado aplicable a desarrollos de minería de datos, proporcionando una guía para el desarrollo de los modelos predictivos [13]. Este modelo divide el proceso en seis fases, que se relacionan entre sí como gráficamente se muestra en la Figura 4:

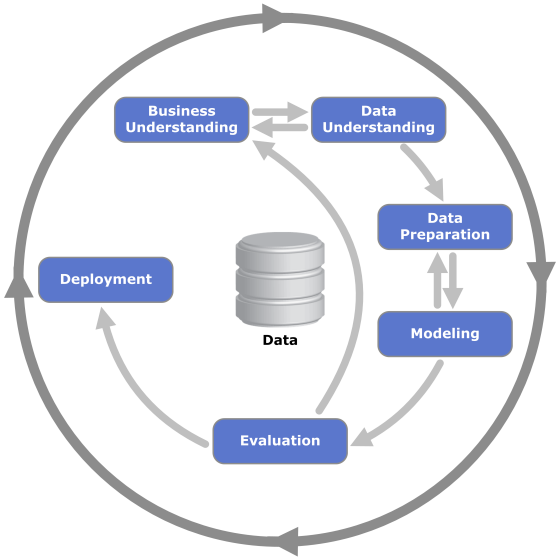

Figura 4: Representación de las fases del modelo CRISP-DM y cómo se relacionan [14].

A continuación, se desarrolla cada una de ellas y, en cada caso, se explica cómo ha sido aplicada dicha fase de forma general, describiendo sus objetivos y consideraciones principales:

1. **Comprensión del negocio**: Fase inicial durante la que se trata de extraer los objetivos y requisitos del proyecto, luego convertirlos en un problema de minería de datos y más tarde en un plan diseñado para alcanzar los objetivos propuestos. Para cada uno de los problemas propuestos, se han extraído los objetivos a conseguir mediante reuniones recurrentes con el interesado, objetivos que se detallarán en secciones posteriores, incluyendo desde la deteccion de valores anómalos hasta la reducción de paradas imprevistas de maquinaria crítica [13].

2. **Comprensión de los datos**: Durante esta fase se trata de analizar el conjunto inicial de datos con el fin de familiarizarse con el *dataset*, detectar problemas de calidad o extraer información oculta que puede ser de interés. Durante el proyecto, se han llevado a cabo diversos análisis exploratorios de los datos, que incluyen historiales de fallos, historiales de mantenimiento, condiciones de operación o señales de contadores, entre otros [13].

3. **Preparación de los datos**: Esta fase cubre todas las actividades relativas a construir el *dataset* final a partir de los datos iniciales. Estas tareas comúnmente se realizan múltiples veces y sin un orden prestablecido. Abarcan desde la creación de tablas, hasta la selección de atributos o limpieza de los datos, como la eliminación de registros nulos, entre otras. Para cada modelo probado en este proyecto, se han aplicado diferentes preprocesamientos de los datos convenientes para cada situación, como la limpieza de datos, relativo a la eliminación de valores nulos o eliminación de valores atipicos, la división de conjuntos de datos en función de su propósito, normalizacion de los datos, para que los valores tengan magnitudes comparables y aquellas de mayor valor no eclipsen al resto, o la generación de características, que ayudan al modelo a detectar patrones o a contextualizar los datos [13] [5].

4. **Modelado**: En esta fase, se seleccionan diferentes modelos y se ajustan sus parámetros a valores óptimos. Para un mismo problema, suele haber varios algoritmos que pueden servir como propuesta de solución, cada uno con sus requisitos específicos del *dataset*, forzando en ocasiones a retroceder a la fase anterior para poder aplicar las técnicas de modelado deseadas. En el proyecto, se generaron diferentes propuestas de modelos como solución para cada problema y se ajustaron sus parámetros con el fin de obtener los mejores resultados posibles [13].

5. **Evaluación**: Llegados a esta fase, se debe de haber construido un modelo de calidad desde una perspectiva de análisis de datos. Sin embargo, antes de proceder al despliegue, se ha de evaluar el modelo con el fin de garantizar que cumpla con los objetivos establecidos. Durante el desarrollo, se evaluaron diferentes métricas estandarizadas en la evaluación de modelos de *Machine Learning* y se compararon los resultados generados con los ya conocidos con el fin de determinar si los modelos eran de calidad [13].

6. **Despliegue**: Tras la generación de un modelo evaluado, éste se ha de presentar de modo que le pueda ser de utilidad a un cliente. Dependiendo de los requisitos iniciales, se puede cerrar esta fase con la simple generación de informes o se puede hasta generar una implementación de un proceso de minería de datos. Aquellos modelos seleccionados como solución para los problemas presentados en este proyecto, han sido exportados a un modelo compatible con el lenguaje de programación C#, seleccionado por el interesado y posteriormente desplegados en el software de la empresa (Fast Monitoring), donde son probados por los clientes finales [13].

3. Modelos de predicción

En esta sección, se explican los procesos que conllevan el entrenamiento de los modelos de *Machine Learning* propuestos como solución, desde la extracción de los datos, su preparación, el entrenamiento de los algoritmos y la validación de los modelos.

Los modelos desarrollados se han aplicado a tres casos de uso: la detección de anomalías en señales industriales, la estimación de la vida útil restante en componentes críticos y la predicción de la evolución futura de señales monitorizadas.

3.1. Conjuntos de datos

En este proyecto, donde se tratan problemas de mantenimiento predictivo, la calidad y fiabilidad de las fuentes de los datos a utilizar es fundamental pues, se requiere que los datos reflejen de manera precisa el comportamiento real de aquello que se está monitorizando.

Por ello, se ha optado para el desarrollo de los distintos problemas planteados, por utilizar tanto conjuntos de datos públicos, ampliamente validados en la literatura científica, como datos generados por máquinas reales en entornos industriales, asegurando así que los modelos obtenidos son fiables y representativos.

3.2. Modelo de detección de anomalías en señales

La detección de anomalías, también conocida como detección de valores atípicos, persigue identificar los objetos de datos o comportamientos que significativamente se desvían de la mayoría. Dada las amplias aplicaciones que tiene esto, durante varias décadas se han propuesto numerosos algoritmos para la detección de anomalías, aplicando aprendizajes tanto supervisados como no supervisados o semi-supervisados. Sin embargo, en el aprendizaje con etiquetas, el etiquetado debe ser anotado manualmente por expertos en la materia, lo cual tiende a ser caro y requerir de mucho tiempo. Además, marcar con precisión todos los tipos de muestras anómalas es a veces inviable en la práctica, especialmente en el campo industrial. Dicho esto, la detección de anomalías no supervisada (UAD, por sus siglas en inglés, *Unsupervised Anomaly Detection*) es la que termina por resultar más interesante en estos casos donde los datos provienen de entornos industriales y el etiquetado no es una opción.

3.2.1. Definición del problema y concepto de anomalía

A la hora de preparar un algoritmo UAD, no se tiene conocimiento previo sobre qué tipos de datos son normales o anómalos; la tarea que perseguiremos será la de encontrar todas las instancias que más se desvían de las demás instancias entre todo el conjunto de datos, con el fin de identificar comportamientos anormales que sugieren fallos o malfuncionamientos incipientes. En particular, el modelo de detección de anomalías no supervisado que se pretende desarrollar en esta sección, tratará con datos procedentes de un contador de agua industrial. Este instrumento metrológico es un contador acumulativo especificamente diseñado para medir el consumo de agua en intervalos regulares.

El contador de este estudio registra datos en forma de pares, guardando el valor acumulado en un tiempo dado y el momento del registro de la muestra. Dada la naturaleza del dispositivo, se esperan de él valores que presenten un crecimiento progresivo en condiciones normales. Dicho esto, todo aquello que se aleje del comportamiento habitual esperado podrá considerarse como anómalo.

A continuación, a modo orientativo, se listan algunos de los patrones que podrían presentar indicios de irregularidades en los registros:

- Incrementos inusuales de consumo: Lecturas superiores a las habituales, sugiriendo el inicio de eventos anómalos como fugas o errores de medición.
- Ausencia de variación: Períodos prolongados sin cambios en el valor acumulado de los registros del contador, lo que podría significar una detención inesperada o un problema en el flujo de lecturas.
- Registro de muestras decrecientes: Descensos en las muestras registradas del contador, incoherentes con su funcionamiento acumulativo.
- Inconsistencias estadísticas: Comportamientos anómalos alejados del patrón histórico.
- Valores atípicos: Lecturas muy alejadas de la media o lo esperado, pudiendo deberse a fallos ocasionales, ruido o eventos raros.

3.2.2. Preparación y preprocesamiento de datos

Cada lectura registrada por el contador consiste en el instante temporal de la toma de la muestra, en formato *yyyy-mm-dd hh:mm:ss.sss* y su valor correspondiente, que representa el valor acumulado en ese momento, expresado como un número en coma flotante de doble precisión.

El *dataset* inicial con el que se va a trabajar presenta tres columnas:

- *Moment*: Instancia temporal de la toma de la muestra.
- *3004-CONT-P23EA823640C-CONTADOR-VALUE*: Valor acumulado del dispositivo.
- *3004-CONT-P23EA823640CONTADOR-STATE*: Estado del contador, tomando valor *Normal* (cero variabilidad) a lo largo de todas las filas .Luego se considera irrelevante dada su nula aportación al contexto de la muestra y potencialmente poco interesante para el aprendizaje del modelo.

La frecuencia de muestreo es por cada hora transcurrida, presentándose datos desde 2025-02-05 04:00:00.000 hasta 2025-03-24 23:00:00.000, disponiéndose de un total de 1125 registros.

Aunque en los últimos años se ha extendido el uso de modelos capaces de aprender directamente de los datos crudos, como en el caso de redes neuronales profundas, en entornos industriales de este tipo, esta estrategia no es siempre viable o eficiente, al disponer de fuentes de datos que generan información muy estructurada y limitada.

Dicho esto, la mayoría de los sistemas de mantenimiento predictivo, requieren de una fase manual de preprocesamiento y generación de *features* adaptadas al contexto de la predicción. Esta fase alimenta a los modelos con información útil que propicia que estos aprendan patrones, tendencias o anomalías estadísticas en los datos.

Para el caso del contador de agua acumulativo, se ha aplicado una técnica de *feature engineering* basada en varias estrategias de *statistical summary*. Esta categoría agrupa diferentes estrategias que se basan en incrementar la información disponible de los datos calculando estadísticas clásicas como valores máximos, medias o desviación estándar. En la gran mayoría de los trabajos de mantenimiento predictivo es común aplicar también estrategias como *time-lagged features* dada la alta temporalidad de los datos. Esta última se basa en generar *features* a partir de los últimos n valores históricos de una serie como un nuevo vector de entrada.

A continuación, se muestra una tabla con las *features* generadas, su motivación y naturaleza (si sirven como entrada del modelo o para construir otras), seleccionadas para ayudar al modelo a detectar las peculiaridades antes presentadas:

Feature	**Descripción**	**Motivación**	**Tipo**
drop-zone	Indicador binario que indica si el cambio entre el valor de la muestra actual y la inmediatamente anterior es negativo	Detectar desviaciones decrecientes en la señal, incoherentes con lo esperado de un contador acumulativo	Final
flat-zone	Indicador binario de una zona plana durante una ventana n	Identificar períodos prolongados sin variación, pudiendo tratarse de fallos en el dispositivo	Final
signal-norm	Normalización del valor de una muestra respecto al máximo y mínimo de su ventana local n	Detectar muestras en la señal donde el valor es inferior al máximo local, indicativo de anomalía	Final
trend-slope	Pendiente de la recta de regresión lineal ajustada a una ventana de n muestras	Detectar cambios en el patrón de la señal, indicador de patrones anómalos	Final
diff-rel	Diferencia relativa entre el valor de una muestra y su inmediata anterior	Identificar cambios relativos, detectando incrementos o decrementos fuera de lo común	Final
std-threshold	Diferencia entre el valor de una muestra y la media móvil de su ventana	Detectar valores fuera del rango esperado, identificando outliers o valores fuera del umbral estadístico	Final
pct-diff-ma-w	Diferencia porcentual entre el valor de una muestra y la media móvil de ventana	Identificar variaciones respecto a la media local	Final
diff	Diferencia entre el valor de una muestra y su inmediata anterior	Utilizada en el cálculo de *diff-rel* y *drop-zone*	Auxiliar
ma-w	Media móvil de un valor respecto a su ventana local	Utilizada en el cálculo de *pct-diff-ma-w* y *std-threshold*	Auxiliar
rolling-std-w	Desviación estándar de una ventana n	Utilizada en el cálculo de *std-threshold*	Auxiliar
same-as-prev	Binario que indica si el valor actual es igual al anterior	Utilizada en el cálculo de *flat-zone*	Auxiliar

Tabla 1: Características derivadas generadas para la detección de anomalías.

A modo ilustrativo, para comprender mejor cómo evolucionan estas características a lo largo de la secuencia temporal y facilitar la interpretación visual de los patrones en los datos, se adjunta una representación gráfica que muestra la evolución de estas variables.

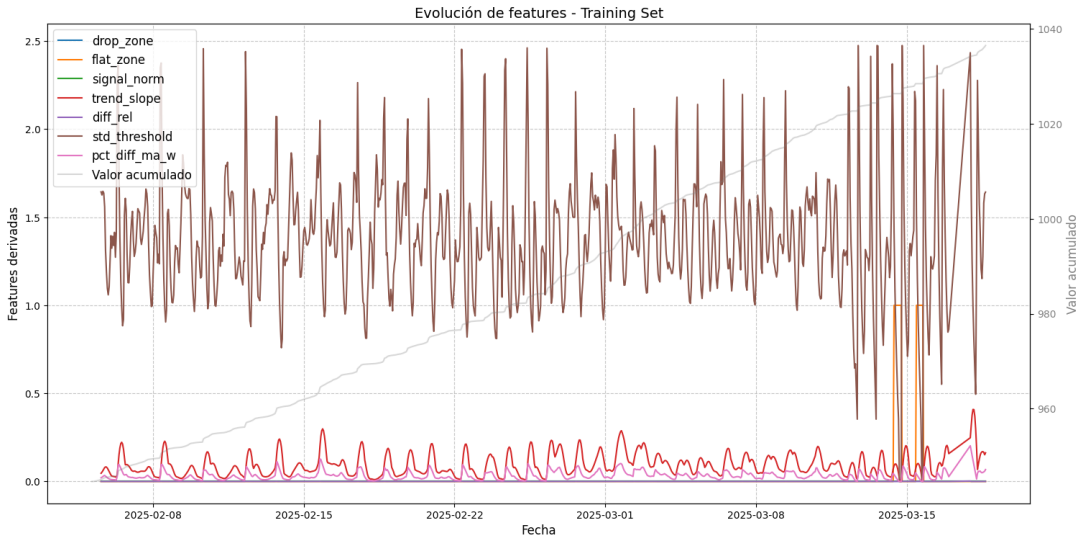

Figura 5: Evolución de las *features* a lo largo del *dataset* inicial

Adicional a las tareas de *feature engineering*, se han realizado una serie de acciones con el fin de preparar los datos adecuadamente:

- *Conversión de datos temporales*: Se ha transformado la columna *Moment* a formato datetime. De este modo se posibilitan las operaciones basadas en el tiempo.
- *Ordenación temporal*: Se ha ordenado los datos cronológicamente en función *Moment*. Esto permite reflejar el orden natural de la obtención de las muestras, que no reflejaba el *dataset* inicial.
- *Eliminación de valores nulos*: Se eliminan los valores nulos generados por cálculos, eliminando la posibilidad de obtener una *feature* nula.

Asimismo, con el fin de validar el modelo final, se ha subdividido el conjunto de datos original en dos subconjuntos con diferentes fines:

- Dataset de entrenamiento: Incluye las primeras 974 filas del conjunto de datos original, equivalente a un 86.67 % del total. A partir de este subconjunto, los modelos a probar en el entrenamiento aprenderán los patrones de la señal. Se asume que las muestran que conforman este *dataset* no contiene anomalías o contienen un número pequeño indeterminado de anomalías. Este posible porcentaje de anomalías presentes en el conjunto de datos de entrenamiento, es parametrizable en algunos modelos, como por ejemplo en Isolation Forest, a través del parámetro de *contaminación*, que indica la fracción estimada de datos anómalos en el *dataset*.
- Dataset de test: Incluye las últimas 152 filas del conjunto de datos original, equivalente a un 13.33 % del total. Este subconjunto de datos se utilizará para comprobar la capacidad de los modelos entrenados para detectar anomalías. Con el fin de evaluar la capacidad de predicción de los modelos de forma controlada, se han generado cuatro variantes de este conjunto de test, cada una conteniendo un segmento anómalo introducido manualmente. Esto permite evitar que la coexistencia de múltiples secciones anómalas en un *dataset* reducido distorsione los resultados o dificulte la detección de patrones anómalos más sutiles al verse rodeados por anomalías más evidentes. Además, de este modo, la realidad

del contexto del contador se ve mejor representada, donde las anomalías son escasas y esporádicas. A continuación, se describen las características para cada segmento anómalo simulado, incluyendo el tipo de anomalía, el número de registros afectados y el intervalo temporal correspondiente.

- Segmento 1: Afecta 13 registros, desde 2025-03-19 11:00:00.000 a 2025-03-19 23:00:00.000. La anomalia que se simula corresponde a un segmento de tiempo sin variación o estancamiento en el valor acumulado del dispositivo.
- Segmento 2: Afecta a 7 registros, desde 2025-03-20 15:00:00.000 a 2025-03-20 21:00:00.000. Se representa un segmento de tiempo con una tendencia creciente sensiblemente mayor a la esperada por la señal en ese momento.
- Segmento 3: Afecta 10 registros, desde 2025-03-22 17:00:00.000 a 2025-03-23 02:00:00.000. Se pretende simular un incremento inusual previo a un descenso anómalo del valor del contador.
- Segmento 4: Afecta a 3 registros, desde 2025-03-24 17:00:00.000 a 2025-03-24 19:00:00.000. Se simula un salto positivo atípico en el valor de las muestras.

A continuación, como se hizo con el *dataset* original, se muestra la evolución de las *features* a lo largo de los cuatro subconjuntos de validación, cada uno incluyendo una variante de segmento anómalo en diferentes puntos de los datos.

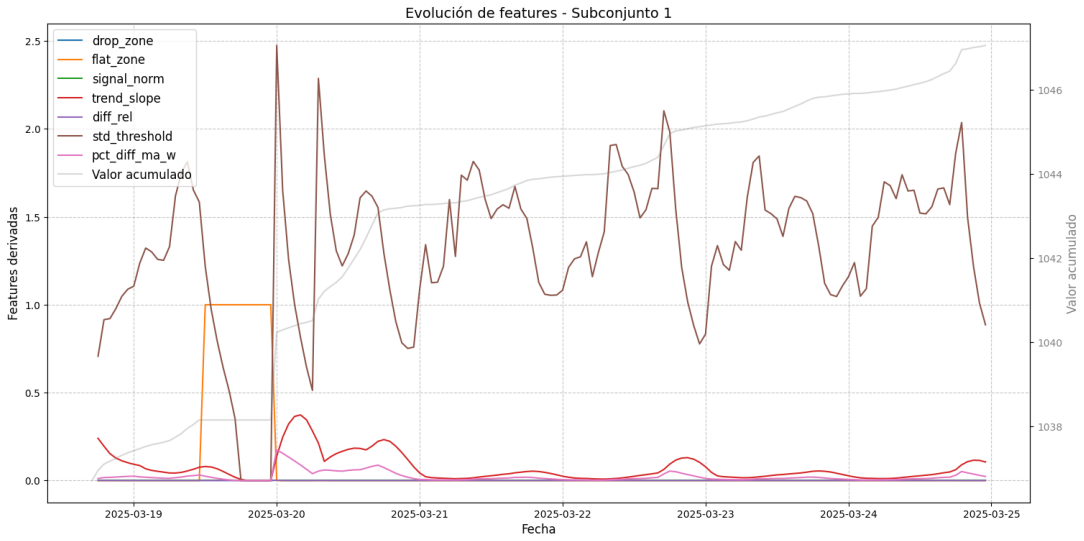

Figura 6: Evolución de las *features* a lo largo del subconjunto de validación '1'

Se aprecia una respuesta relevante de la característica *flat-zone* ante el segmento anómalo plano introducido al comienzo del conjunto.

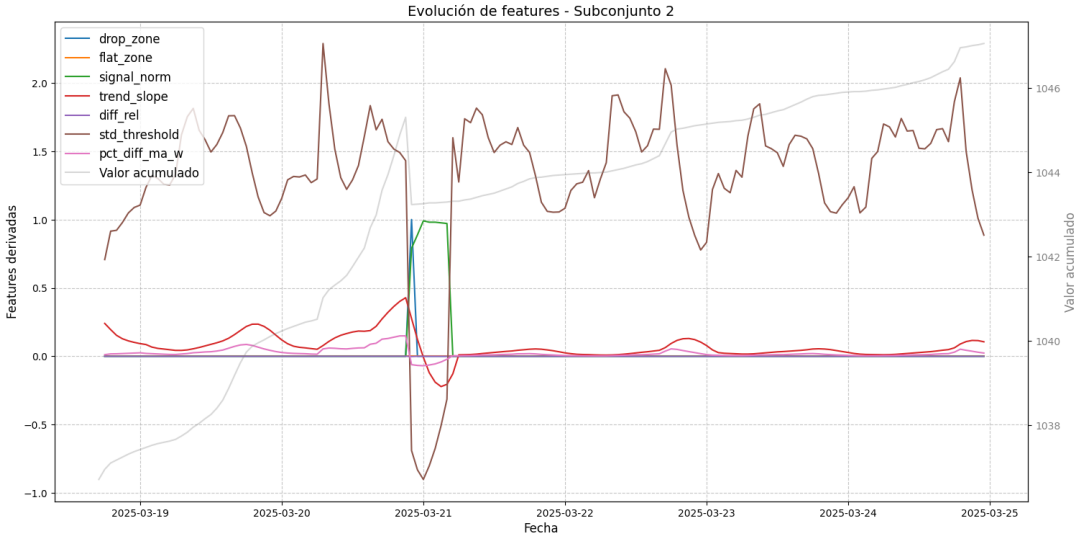

Figura 7: Evolución de las *features* a lo largo del subconjunto de validación '2'

La características *trend-slope* y *drop-zone* responden ante el incremento introducido en el segundo tramo de la señal. Asimismo, *signal-norm* detecta la anomalía aunque con cierto retardo, dada la naturaleza de su cálculo en ventana. Otras características como *flat-zone* se mantienen inactivas al no haber condiciones que la active, como un estancamiento.

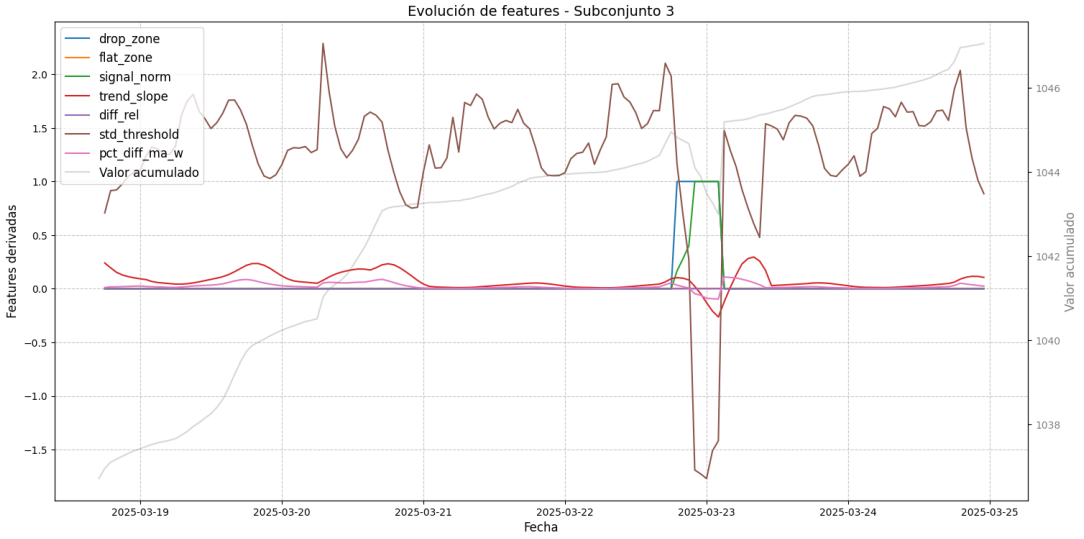

Figura 8: Evolución de las *features* a lo largo del subconjunto de validación '3'

La caída introducida en el tercer cuarto de la señal es detectada principalmente por *drop-zone*, *std-threshold* y *signal-norm*. La característica *pct-diff-ma-w* y *trend-slope* la detectan ligeramente con retardo, pudiéndose generar falsos positivos.

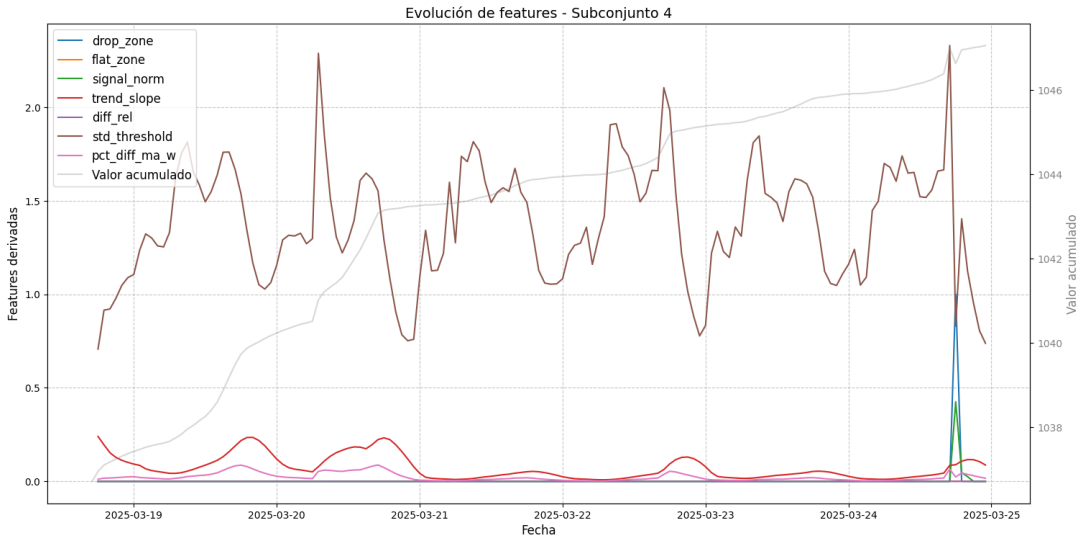

Figura 9: Evolución de las *features* a lo largo del subconjunto de validación '4'

El comportamiento anómalo del final de la señal, es marcado por *std-threshold*, *drop-zone* y *signal-norm*, sin embargo, como ocurre con otras figuras anteriores, no es el único segmento anómalo marcado con esa intensidad; otras características también reaccionan, pero en menor medida, como puede ser *std-threshold* en el primer tercio de la señal.

3.2.3. Selección y entrenamiento del modelo

El objetivo del modelo es identificar las anomalías introducidas de forma controlada en los subconjuntos de test.

Para que el modelo se considere aceptable, este ha de ser capaz de detectar al menos el 66 % de las anomalías presentes en cada uno de los segmentos anómalos introducidos, manteniendo la proporción de falsos positivos al mínimo posible.

Se establece que la presencia de falsos positivos solo será admitida si estos no se agrupan en exceso. En concreto, se permite un máximo de dos falsos positivos consecutivos por subconjunto. En caso contrario, el modelo se consideraría no válido puesto que, una mayor concentración de detecciones incorrectas indicaría una pérdida significativa de especificidad, comprometiendo la fiabilidad del modelo.

En lo que se refiere a la selección del modelo, se entrenará un algoritmo de Isolation Forest, tratándose de un modelo eficiente en cuanto a tiempo de respuesta, con altas capacidades en problemas UAD donde las anomalías a detectar son escasas y se diferencian fácilmente del conjunto de datos general.

En cuanto a la implementación del modelo, se ha realizado a través de la librería *scikit-learn versión 1.6.1*. La clase *IsolationForest* de esta misma proporciona una implementación del algoritmo original, con soporte para múltiples hiperparámetros configurables, cada uno de ellos con un impacto diferente en la capacidad de detección del modelo. Se definen aquellos que han sido configurados durante el entrenamiento[2]:

- *n_estimators*: Número de árboles estimadores base del conjunto. A mayor número estimadores, mayor será el tiempo de ejecución. (Por defecto *100*)

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html`

- *max_samples*: Número de muestras a extraer de X para construir cada árbol. Afecta a la variabilidad y eficiencia del modelo. Se le puede asignar un número absoluto o proporcional (por ejemplo, 0.5 indicando el 50 % del total de las muestras).
- *contamination*: Cantidad de contaminación presente en el *dataset*, por ejemplo, la proporción de anomaías presentes en los datos de entrenamiento. Se recomienda un rango de [0, 0.5]
- *max_features*: Número de *features* a extraer de X para entrenar a cada árbol estimador. Controla la cantidad de información considerada en cada división de los estimadores.
- *boostrap*: Si activado, durante la construcción de árboles, se utilizan muestras con reemplazo, pudiendo aumentar la diversidad de árboles estimadores. Por defecto, desactivado.
- *n_jobs*: Número de tareas a realizar en paralelo para el entrenamiento del modelo. Un valor de *-1* permitiría utilizar todos los núcleos de CPU disponibles. Por defecto *1*.
- *random_state*: Semilla que garantiza la reproducibilidad de los resultados.

Con el fin de asegurar que el modelo final ha sido entrenado con la mejor combinación de hiperparámetros posible, se ha aplicado la técnica Grid Search, técnica de optimización a partir de la cual se prueban múltiples combinaciones de valores y se evalúa el rendimiento del modelo para cada una. A continuación, se muestran los rangos de valores explorados, evaluando todas las combinaciones posibles:

Hiperparámetro	**Valores explorados**
n_estimators	50, 100, 200
max_samples	auto, 0.5, 0.8
contamination	0.03, 0.05, 0.07, 0.1, 0.15
max_features	1.0, 0.8, 0.6
bootstrap	True, False
random_state	42
n_jobs	-1

Tabla 2: Rangos de valores explorados en el Grid Search

Asimismo se han explorado diferentes combinaciones tanto de percentiles (umbrales de decisión aplicados sobre los *scores* del modelo para determinar si una observación es anómalo o no) como de tamaños de ventana, utilizadas para el cálculo de *features* derivadas, asegurando así cubrir todas las combinaciones posibles en el espacio de búsqueda. A continuación se muestran los rangos de valores explorados:

Parámetro de proceso	**Valores explorados**
percentil	3.5, 4, 4.5, 5, 5.5
window	4, 6, 8, 10

Tabla 3: Rangos de valores explorados en parámetros configurables adicionales en el proceso de detección

Cabe mencionar que, además, se ha experimentado con todas las agrupaciones posibles de características pues, la cantidad de *features* no presenta relación directa con la calidad de las

predicciones y se han de seleccionar solo aquellas notablemente informativas para el ejercicio de la detección.

Las combinaciones de configuración que han superado los criterios de aceptación del modelo son las siguientes:

```
# Configuracion 1
['bootstrap': False, 'contamination': 0.03, 'max_features': 1.0, 'max_samples': 'auto',
 'n_estimators': 50, 'n_jobs': -1, 'random_state': 42], Percentil: 5.5, Anomalias: 34.
```

```
# Configuracion 2
['bootstrap': False, 'contamination': 0.1, 'max_features': 1.0, 'max_samples': 'auto',
 'n_estimators': 50, 'n_jobs': -1, 'random_state': 42], Percentil: 5.5, Anomalias: 34.
```

```
# Configuracion 3
['bootstrap': False, 'contamination': 0.05, 'max_features': 1.0, 'max_samples': 'auto',
 'n_estimators': 50, 'n_jobs': -1, 'random_state': 42], Percentil: 5.5, Anomalias: 34.
```

```
# Configuracion 4
['bootstrap': False, 'contamination': 0.07, 'max_features': 1.0, 'max_samples': 'auto',
 'n_estimators': 50, 'n_jobs': -1, 'random_state': 42], Percentil: 5.5, Anomalias: 34.
```

```
# Configuracion 5
['bootstrap': False, 'contamination': 0.15, 'max_features': 1.0, 'max_samples': 'auto',
 'n_estimators': 50, 'n_jobs': -1, 'random_state': 42], Percentil: 5.5, Anomalias: 34.
```

Asimismo, las cinco configuraciones presentadas comparten el mismo subconjunto de *features* derivadas, estando formado por las que se listan a continuación:

- `flat_zone`
- `signal_norm`
- `trend_slope`
- `diff_rel`
- `std_threshold`

3.2.4. Evaluación del modelo

Con el objetivo de seleccionar la mejor configuración como solución al problema, se calcularon métricas de rendimiento para cada propuesta, de manera que puedan ser comparadas. La configuración que mostró mejor equilibrio se consideraró la más óptima.

Config.	TP	FP	FN	Precisión	Recall	F1 Score
1	22	12	11	0.647	0.667	0.657
2	22	12	11	0.647	0.667	0.657
3	22	12	11	0.647	0.667	0.657
4	22	12	11	0.647	0.667	0.657
5	22	12	11	0.647	0.667	0.657

Tabla 4: Matriz de resultados y métricas por configuración.

Comparando las métricas anteriores, se observa que todas las configuraciones han presentado un rendimiento idéntico. Sin embargo, se selecciona como propuesta de solución aquella con el menor índice de *contamination*, correspondiente al 3 %. Esta selección permite validar que el modelo no requiere un umbral elevado de anomalías asumidas en el conjunto de datos original para generar predicciones que cumplen los criterios establecidos.

Propuesta de solución: *Configuración 1*

3.2.5. Interpretación y visualización comparativa con anomalías reales

Con el objetivo de ilustrar la capacidad del modelo desarrollado para detectar patrones anómalos en series temporales, se presenta a continuación una comparación gráfica entre los segmentos anómalos insertados manualmente y las anomalías identificadas por el modelo. En cada figura se representa la señal original del contador acumulativo con los segmentos anómalos sombreados de color gris. Las predicciones del modelo se indican mediante puntos rojos superpuestos sobre la señal. Adicionalmente, se incluye la evolución del *score* de anomalía generado por el modelo que ayuda a entender las decisiones tomadas por el mismo en cada punto de la señal en lo referido a si un punto puede considerarse anómalo o no, así como también el umbral de *threshold*, que actúa como punto de corte ajustando la sensibilidad del modelo en el momento de decidir si un comportamiento es suficientemente anómalo como para ser clasificado como tal.

A continuación, se muestran las representaciones gráficas corresponsietes a cada subconjunto:

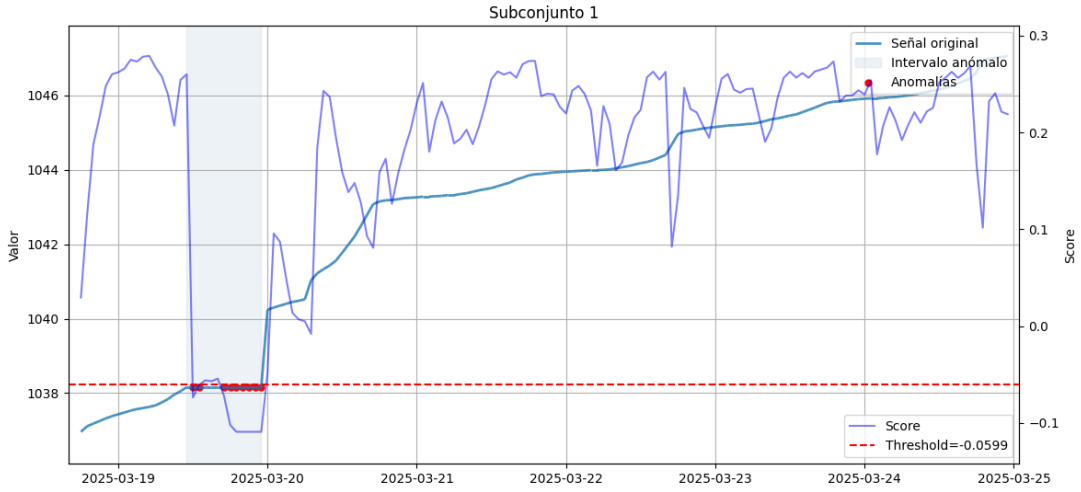

Figura 10: Detección del segmento anómalo en el subconjunto '1'.

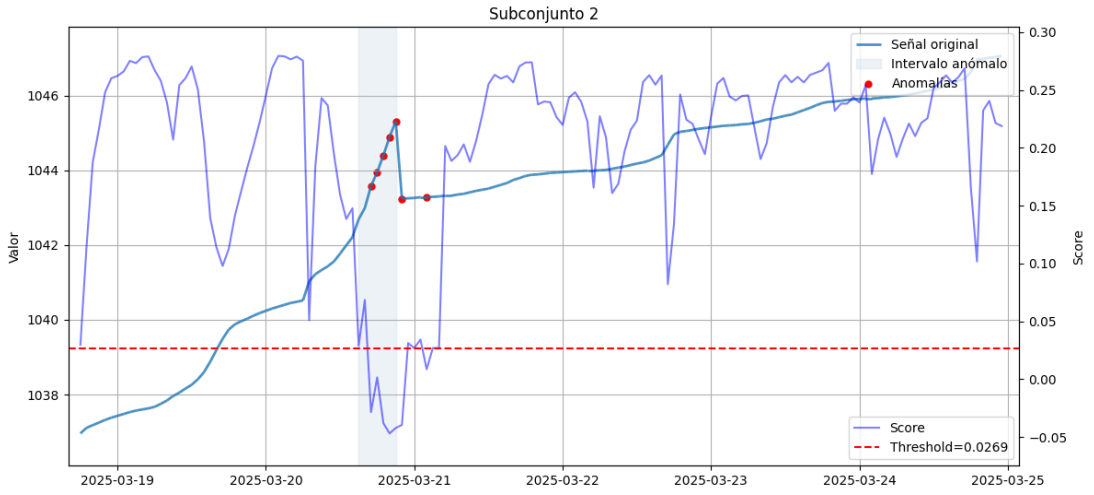

Figura 11: Detección del segmento anómalo en el subconjunto '2'.

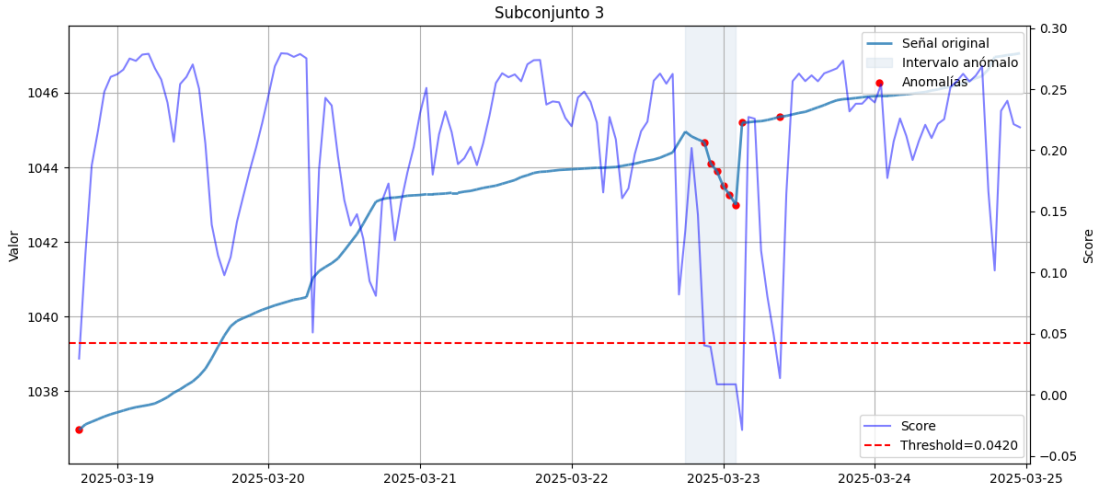

Figura 12: Detección del segmento anómalo en el subconjunto '3'.

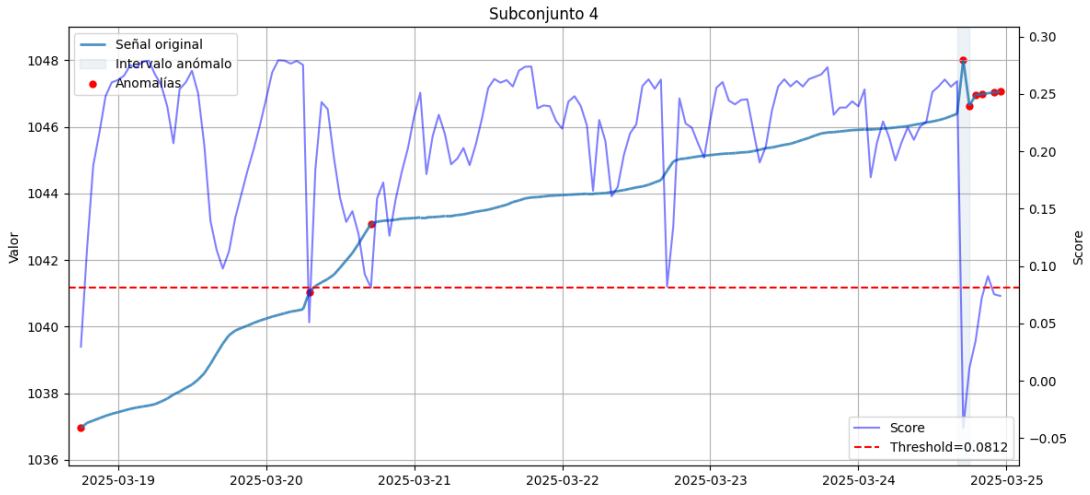

Figura 13: Detección del segmento anómalo en el subconjunto '4'.

En conclusión, se puede observar que el modelo consigue identificar adecuadamente los segmentos anómalos insertados en cada subconjunto, detectando tanto zonas planas como incrementos repentinos o comportamientos incoherentes con el comportamiento acumulativo esperado por la señal, como puede ser un descenso en el valor de la muestra.

Asimismo, las predicciones generadas tienden a concentrarse dentro de los intervalo simulados, lo que permite distinguir visualmente los patrones anómalos en un entorno industrial como el analizado.

3.2.6. Análisis sobre sensibilidad, especificidad y rendimiento del modelo

Uno de los principales retos a la hora de desarrollar modelos para la detección de anomalías, se encuentra en conseguir un equilibrio entre detectar la mayor cantidad de anomalías reales (para ello el modelo tendría que ser altamente sensible) pero sin comprometer la especificidad del mismo, es decir, la capacidad del modelo para identificar correctamente las observaciones normales.

En un entorno industrial, los falsos positivos reiterados podrían suponer costes operativos o interrupciones innecesarias, de ahí la importancia de minimizar este tipo de errores sin comprometer la detección de verdaderas anomalías.

En el modelo desarrollado, se ha ajustado el valor del índice de *contamination* junto con el umbral de (*threshold*) con el objetivo de reducir los falsos positivos, incluso si ello implica que algunas anomalías marginales menos evidentes no sean detectadas.

En base a los resultados expuestos, se concluye que el modelo propuesto presenta un rendimiento aceptable, siendo capaz de identificar correctamente todos los segmentos anómalos simulados sin mostrar una tasa significativa de falsos positivos. Esto justifica su selección como propuesta de solución para el problema de detección automática de anomalías en señales acumulativas como las generadas por contadores de agua industriales de carácter acumulativo.

3.2.7. Propuestas de mejora

Aunque el modelo desarrollado ha generado resultados competitivos, existen varias líneas de mejora a explorar en futuros trabajos.

El algoritmo seleccionado (Isolation Forest) ha cumplido lo esperado y es ampliamente utilizado en trabajos de este tipo, sin embargo, podrían explorarse otros algoritmos más complejos basados en redes neuronales que podrían ser más adecuados para la detección de tendencias temporales complejas o anomalías sutiles. Incluso se podría explorar el uso de algoritmos híbridos (combinación de varios modelos) con el fin de mejorar la calidad de las predicciones o filtrar las anomalías residuales.

Asimismo, gran parte del peso del entrenamiento del algoritmo Isolation Forest se ha basado en la generación de *features* para la identificación específica de los eventos anómalos considerados. Podría plantearse una mejora en este proceso con otras características, ya que tienen un impacto directo en el rendimiento del modelo.

3.3. Modelo predictivo de RUL basado en señales históricas

El seguimiento del tiempo de vida útil restante en máquinas críticas, en el contexto del mantenimiento predictivo (PdM), es esencial para reducir costes operativos, reemplazos innecesarios y minimizar tiempos de inactividad. Esto tiene especial importancia en aquellas máquinas o sistemas en las que la inspección visual del ojo humano, experto o no, es insuficiente o resulta poco fiable, como suele suceder en motores y equipos industriales que tienen múltiples sensores integrados [15].

3.3.1. Definición del problema

El objetivo del modelo a desarrollar es estimar, en ventanas de tiempo, el tiempo de vida útil restante de un motor turbofán. Los datos a emplear provienen de sensores integrados en un motor de aviación, recogidos a lo largo del tiempo.

3.3.2. Preparación y preprocesamiento de datos

Dada la complejidad de simular o generar un conjunto de datos suficiente para abordar el problema , se ha optado por utilizar el *dataset* C-MAPSS [3] (Simulación de un Sistema Modular de Propulsión Aérea Comercial, por sus siglas en inglés), un conjunto de datos público generado por la NASA a través de una herramienta que da nombre al *dataset*, diseñado para simular la degradación nominal y por fallos de un motor turbofán a lo largo de una serie de vuelos. Este es el usado en el campo científico como *benchmark* estándar para estimación de RUL.

El *dataset* mencionado, contiene cuatro subconjuntos de datos, desde FD001..FD004, que representan varias combinaciones de condiciones operacionales y modos de fallo.

Para el problema a desarrollar, se ha decidido proponer una solución para el *dataset* FD001, un subconjunto de datos en el que el motor presenta un único tipo de operación y un mismo entorno de condición ambiental. Otros subconjuntos presentan complejidades superiores tratando múltiples tipos de operaciones y condiciones ambientales.

Se ha optado por el *dataset* FD001 pues se considera propicio como punto de partida para la predicción de RUL, el preprocesamiento a realizar está alineado con el alcance del proyecto y está ampliamente documentado, luego los resultados que se obtengan podrán ser fácilmente validables.

El subconjunto de datos seleccionado, contiene 26 columnas de datos, que se desarrollan en la Tabla 5:

[3] https://data.nasa.gov/dataset/c-mapss-aircraft-engine-simulator-data

Columna	Nombre	Descripción
1	engine-id	identificador del motor
2	time-cycle	tiempo transcurrido medido en ciclos de operación
3	operational-setting-1	configuración operacional
4	operational-setting-2	configuración operacional
5	operational-setting-3	configuración operacional
[6, 26]	sensor1...21	sensores consultados

Tabla 5: Características derivadas generadas para la detección de anomalías.

El repositorio utilizado para este caso de uso contiene múltiples subconjuntos de datos para distintos fines, de los cuales se utilizarán dos archivos principales, aquel dedicado para el entrenamiento del modelo (*train-FD001*), que contiene 20.631 registros, y el dedicado para la fase de validación del modelo (*test-FD001*), con 13.096 registros.

Cada muestra representa el estado de un motor (marcado por un identificador numérico único) en un ciclo específico (determinado por un contador de ciclos), acompañados de tres configuraciones operacionales y veintiuna lecturas de sensores.

Relativo a la calidad de los datos, ninguno de los subconjuntos presenta registros nulos, facilitando así el preprocesamiento de los mismos. Mencionar que se encontraron dos columnas residuales completamente vacías, que fueron eliminadas dada su nula aportación contextual.

Con motivo de abordar el problema de estimación de RUL, se generará una columna adicional en el subconjunto de entrenamiento como cálculo de la diferencia entre el ciclo final en el que el motor falla y su ciclo actual. Se le asignará a esta nueva columna el nombre de *RUL*.

En cuanto al subconjunto de validación se refiere, los motores no fueron operados hasta el momento de fallo. Debido a eso, el repositorio facilita un archivo adicional (*RUL-FD001*) con los valores de RUL en el último ciclo registrado para cada motor. Este valor representa el tiempo de vida útil restante para el motor en ese instante de tiempo, pudiendo corresponder a un momento de fallo o no.

A continuación, en la Figura 14, a modo ilustrativo, se muestra la evolución de la degradación de los motores a lo largo del subconjunto de entrenamiento, dada en función del valor de RUL calculado previamente.

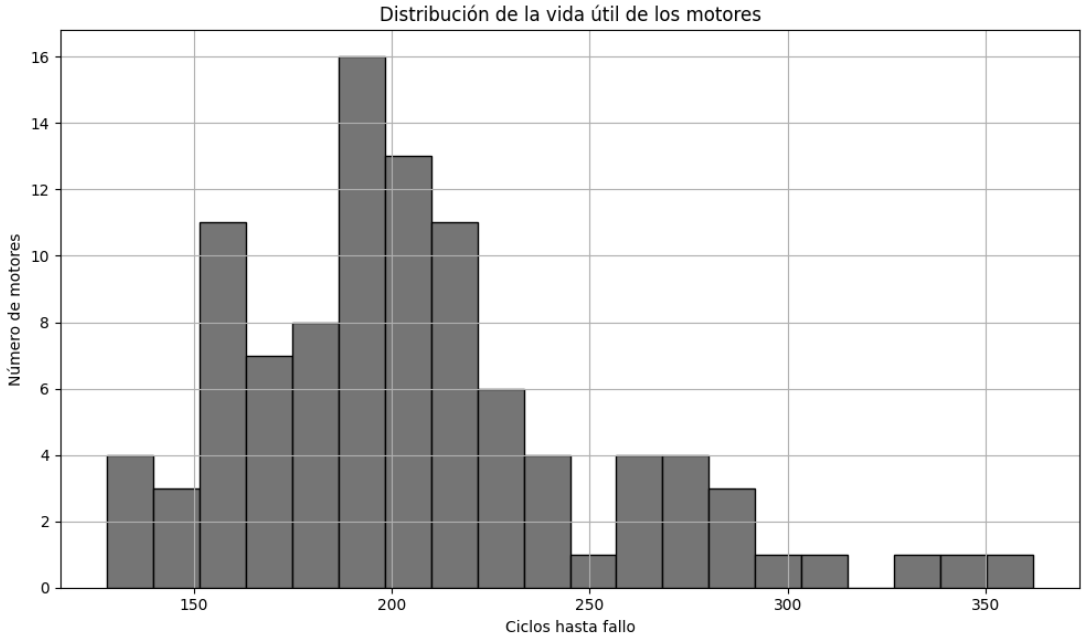

Figura 14: Distribución de la vida útil de los motores en el *dataset* FD001.

Asimismo, con objeto de facilitar la posterior selección de variables informativas en la fase de preprocesamiento, se generá una matriz de correlación entre los valores de RUL y el resto de variables, con énfasis en el estudio de su relación con las lecturas de los sensores. De este modo, se podrán filtrar aquellos sensores que no muestren una relación signficativa con el deterioro del motor, reduciendo así la dimensionalidad del conjunto de datos sin perder información relevante para el ejercicio de la predicción. A continuación se muestra la gráfica de la matriz resultante:

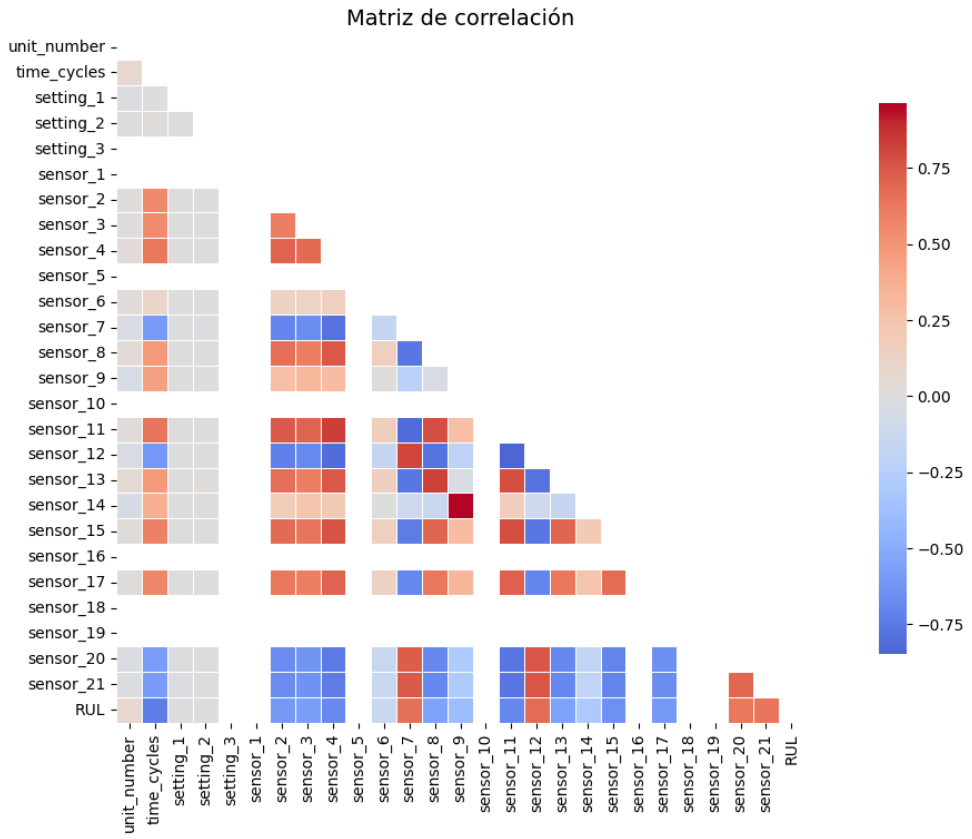

Figura 15: Matriz de correlación de las variables en el *dataset* FD001.

Para el proposito de mejorar la calidad de las predicciones del modelo a desarrollar, se eliminaran aquellos sensores que no sean informativos, es decir, cuya correlación absoluta con la variable de RUL sea nula o despreciable, considerando correlaciones tanto positivas como negativas potencialmente informativas.

Dicho esto, se eliminan los siguientes sensores del proceso de inferencia:

sensor_1, sensor_5, sensor_6, sensor_10, sensor_16, sensor_18, sensor_19.

Con el fin de mejorar la capacidad del modelo para capturar patrones temporales, se ha aplicado la técnica de *feature engineering* sobre las variables iniciales. En la Tabla 6, se muestran las *features* propuestas y su motivación:

Feature	**Descripción**	**Motivación**
rolling-mean	Valor medio de una medida sensorial calculado sobre una ventana de n ciclos de operacion	Captura tendencias de degradación a corto plazo y reduce el ruido.

Tabla 6: Características derivadas generadas para la predicción de RUL.

3.3.3. Selección y entrenamiento del modelo XGBoost

El objetivo del modelo a desarrollar será predecir el tiempo restante de vida útil de un motor de turbina con el fin de, a partir de los sensores integrados en su sistema, cuantificar el nivel de

degradación del componente tras cada ciclo de operación para el propósito de anticipar posibles fallos de funcionamiento y optimizar la toma de decisiones relativa a tareas de mantenimiento.

Para garantizar la adecuación del modelo, este deberá de satisfacer los siguientes criterios:

1. El resultado de la métrica relativa al error cuadrático medio (RMSE) no deberá de superar el umbral de 20 ciclos de operación. De hacerlo, se asume que el modelo no presenta la precisión suficiente para la toma de decisiones fiables, pudiendo causar la adopción de tareas de mantenimiento prematuras o desfasadas al no capturar correctamente la evolución de degradación del motor.

2. De forma paralela, se evaluará también el error absoluto medio (MAE), el cual no deberá de superar el umbral de los 15 ciclos de operación. Esta variable, permite medir el error promedio sin dar importancia en exceso a posibles *outliers*. De superar el umbral, indicaría que el modelo presenta una imprecisión generalizada.

3. Por último, el modelo deberá ser capaz de poder explicar al menos el 75 % de la varianza de la variable de RUL. De no cumplirse este último criterio, el modelo no presentaría capacidad predictiva suficiente, comprometiendo la confianza necesaria para poder ser desplegado en un entorno productivo.

La elección de los valores deseables para cada métrica se ha establecido como criterio de aceptación en este trabajo, tomando como referencia los resultados promedios reportados en estudios previos para FD001 [16], donde los modelos convencionales suelen obtener valores de RMSE en torno a 18–20 ciclos.

Para generar el modelo deseado, se ha optado por seleccionar XGBoost, un algoritmo de *machine learning* ampliamente utilizado en problemas de regresión, basado en técnicas de boosting sobre árboles de decisión.

Para el desarrollo, se ha utilizado la implementación oficial del algoritmo, disponible como repositorio de código abierto. En la librería *xgboost* se presenta la clase *XGBRegressor* que presenta soporte para la configuración de múltiples hiperparámetros que influyen en el aprendizaje y capacidad de generalización del modelo.

A continuación, se indican los hiperparámetros que han sido configurados durante el entrenamiento acompanados de una breve explicación basada en la documentación oficial[4]:

- *n_estimators*: Determina el número de iteraciones (o árboles) que se entrenarán. Un mayor número incrementa el tiempo de ejecución y capacidad del modelo, pero también existe el riesgo de que se produzca *overfitting* si no se regula con una tasa de aprendizaje adecuada (por defecto *100*).

- *learning_rate (eta)*: Controla la tasa de aprendizaje del modelo. Tras cada iteración de boosting, reduce los pesos de las nuevas características aprendidas por los árboles, haciendo el proceso de aprendizaje más conservador y ayudando a prevenir el *overfitting* (por defecto *0.3*).

- *max_depth*: Determina la profundidad máxima de los árboles. A mayor profundidad, mayor complejidad presentará el modelo. Sin embargo, esto también incrementa el riesgo de *overfitting* y consumo de memoria (por defecto *6*).

- *subsample*: Ratio de muestreo sobre las instancias del conjunto de entrenamiento. Por ejemplo, un valor de 0.5 indica que el modelo seleccionará aleatoriamente el 50 % de las muestras para construir cada árbol, reduciendo el *overfitting* al introducir aleatoriedad. El submuestreo se realiza una vez por iteración (por defecto *1*).

[4] `https://xgboost.readthedocs.io/en/stable/parameter.html`

- *random_state* : Semilla aleatoria que garantiza la reproducibilidad de los resultados.

Con el fin de alcanzar la mejor configuración del modelo, se ha realizado un proceso de optimización de hiperparámetros del algoritmo XGboost mediante de la técnica de Grid Search, técnica que consiste en, a partir de un conjunto discreto de valores definidos para cada hiperparámetro, evaluar todas las combinaciones posibles.

Hiperparámetro	**Valores explorados**
n_estimators	500, 1000, 2000
learning_rate	auto, 0.01, 0.1
max_depth	3, 5, 7
subsample	0.8, 1.0
random_state	42

Tabla 7: Rangos de valores explorados en el Grid Search

Asimismo, también se han probado múltiples combinaciones para el tamaño de la ventana de la *feature* generada, representando el número de ciclos de funcionamiento considerados en el cálculo. A continuación se muestran los rangos explorados:

Parámetro de proceso	**Valores explorados**
window	4, 5, 6, 8, 10

Tabla 8: Rango de valores explorados en el parámetro relativo a la ventana deslizante de la característica generada.

Asimismo, distintos estudios han demostrado que, durante el inicio de funcionamiento de un motor, este opera en condiciones de degradación imperceptibles [17] luego, un valor de RUL excesivamente alto podría no proporcionar necesariamente datos informativos al modelo. Por este motivo, se ha optado por aplicar una técnica etiquetada como *piecewise linear*, que consiste en limitar los valores RUL a un umbral máximo predefinido. Este valor limitante se entiende como el punto a partir del cuál los efectos de la degradación comienzan a ser evidentes y perceptibles por los sensores que registran el fenómeno (la literatura científica recomienda un valor alrededor de 125 para este caso de estudio). La aplicación de este enfoque sobre la degradación del motor, ha evitado que el modelo tuviera sesgos hacia valores elevados y presentar así un mejor rendimiento. En la Tabla 9 se indican los rangos de valores explorados para este nuevo índice limitante:

Parámetro de proceso	**Valores explorados**
piecewise linear	100, 125, 150, 175

Tabla 9: Rango de valores explorados en el parámetro limitante de la variable RUL.

Una vez definido el espacio de búsqueda relativo a los hiperparametros del algoritmo y ventana deslizante, para cada configuración, se entrenó un modelo y se calcularon las respectivas métricas de evaluación requeridas por los criterios desarrollados anteriormente, utilizando la biblioteca *scikit-learn*.

3.3.4. Evaluación del modelo

La combinación óptima de hiperparametros y tamaño de ventana es seleccionada de forma automatica en función del rendimiento global de las métricas en el conjunto de validación,

permitiendo así una comparación objetiva y reproducible entre distintas configuraciones.

A modo de ilustrar las diferencias en los resultados según los hiperparámetros empleados, en la Tabla 10 se muestran las mejores configuraciones de XGBoost.

Config.	**RMSE**	**R^2**	**MAE**
1: win=5, piece=125, n_estim=500, lr=0.1, max_depth=5, subs=1.0	18.22	0.808	13.21
2: win=5, piece=125, n_estim=500, lr=0.01, max_depth=5, subs=0.8	18.36	0.805	13.09
3: win=5, piece=125, n_estim=500, lr=0.01, max_depth=5, subs=1.0	18.38	0.804	13.13
4: win=5, piece=125, n_estim=1000, lr=0.01, max_depth=5, subs=1.0	18.47	0.802	13.21
5: win=5, piece=125, n_estim=2000, lr=0.01, max_depth=5, subs=1.0	18.49	0.802	13.41
6: win=5, piece=125, n_estim=1000, lr=0.1, max_depth=5, subs=1.0	18.50	0.802	13.71
7: win=5, piece=125, n_estim=500, lr=0.1, max_depth=3, subs=0.8	18.51	0.802	13.62
8: win=5, piece=125, n_estim=500, lr=0.01, max_depth=7, subs=0.8	18.55	0.801	13.25
9: win=5, piece=125, n_estim=1000, lr=0.01, max_depth=5, subs=0.8	18.56	0.801	13.35
10: win=5, piece=125, n_estim=1000, lr=0.01, max_depth=3, subs=0.8	18.59	0.800	13.45

Tabla 10: Comparación de las 10 mejores configuraciones de XGBoost en el dataset FD001 (ordenadas por RMSE).

La configuración que de entre aquellas que superaban los criterios de evaluación establecidos presento mejor rendimiento fue la siguiente:

```
['n\_estimators': 500, 'learning\_rate': 0.1, 'max\_depth': 5, 'subsample': '1.0',
 'random\_state': 42], Window: 5, Piecewise_linear: 125.
```

Los resultados de dicha configuración para las métricas evaluadas fueron los siguientes:

$$\text{RMSE} = 18{,}2234$$
$$\text{MAE} = 13{,}2089$$
$$\text{R2} = 0{,}8076$$

Para contextualizar los resultados obtenidos, se compara la métrica de error del modelo generado con las reportadas en la literatura para el mismo subconjunto del *dataset*. La Tabla 11 recoge una selección de modelos representativos.

Modelo	**FD001 RMSE**	**Año**
XGBoost (este trabajo)	18.22	–
CNN	18.44	2016
SVR	20.96	2017
MLP	37.56	2017
LSTM	16.14	2017
BiLSTM	13.65	2018
DAG-CNN+LSTM	11.96	2019
CNN+LSTM	16.16	2019
Random Forest	12.01	2020
Multi-head CNN+LSTM	12.19	2020
CNN+LSTM+BiLSTM	10.41	2020

Tabla 11: Comparación del rendimiento del modelo propuesto (XGBoost) con distintos enfoques de la literatura [16].

Como se puede observar, el modelo XGboost desarrollada en este trabajo presenta un rendimiento comparable al de enfoques clásicos como CNN e incluso superior a otros modelos más tradicionales como SVR. Sin embargo, otros modelos de aprendizaje profundo, como BiLSTM u otras variantes híbridas, logran errores menores. Dicho esto, aunque el modelo propuesto como solución es competitivo frente a técnicas convencionales, otras arquitecturas de *deep learning* ofrecen mejor desempeño para la predicción de RUL en este subconjunto.

3.3.5. Interpretación y visualización de resultados

En la Figura 16 que se muestra a continuación se representa una comparativa entre los valores de vida útil restante (RUL) y las predicciones generadas por el modelo entrenado para los 100 motores del subconjunto evaluado. Cada punto en la gráfica muestra el valor real (en azul) y el valor predicho (en naranja) para cada motor en específico identificado por su número de únidad.

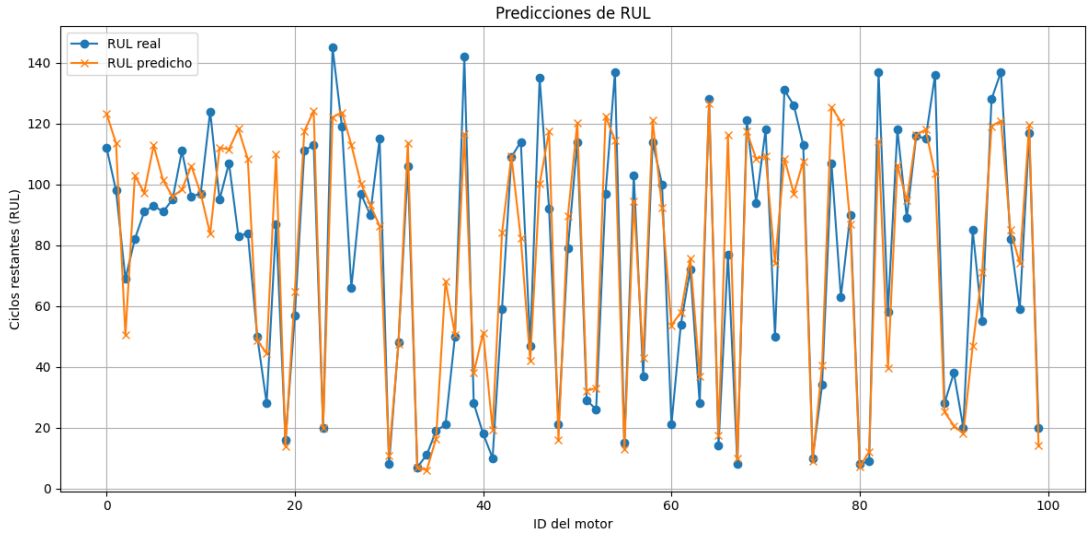

Figura 16: Comparación entre el RUL real y predicho por el modelo para los 100 motores presentes en el *dataset*.

Se puede observar que el modelo propuesto como solución es capaz de capturar la tendencia general de los valores RUL en la mayoría de los motores, prediciendo con mayor precisión aquellos puntos cuyo RUL se encuentra en un umbral intermedio o bajo. Por el contrario, se pueden identificar algunas variaciones en las predicciones o puntos donde el modelo sobreestima o subestima, pero estos se encuentran en un rango razonable. Estas desviaciones se deben tratar con especial atención sobre todo en el contexto del mantenimiento predictivo, donde una ligera anticipación suele ser más aceptable que una predicción retrasada.

En conclusión, el modelo entrenado cumple con los criterios de evaluación establecidos, demostrando una capacidad robusta para estimar el tiempo de vida útil restante de los motores examinados. Manteniendo al margen algunas desviaciones puntuales, la precisión conseguida es suficiente para su posible aplicación en un entorno industrial real, permitiendo realizar las intervenciones de mantenimiento en el momento óptimo, reduciendo así costes operativos asociados tanto a mantenimientos prematuros como a fallos procedentes de actuaciones tardías.

3.3.6. Propuestas de mejora

Como líneas de mejora a explorar en futuros trabajos, se podría considerar el entrenamiento del modelo con algoritmos específicamente diseñados para series temporales, como ARIMA, LSTM o GRU, o algoritmos de aprendizaje profundo basados en *Transformers*, que han demostrado una mejor capacidad predictiva en diversos documentos científicos que han estudiado este mismo problema. Los resultados competitivos generados por el modelo entrenado con XGBoost dependen en gran medida de la *feature engineering* realizada o el enfoque *piecewise linear* elegido, mientras que otros modelos más complejos basados en aprendizaje profundo, podrían generar mejores resultados con un simple ajuste de hiperpárametros.

Por otra parte, tendría interés desarrollar RUL en otros *datasets* públicos disponibles en el repositorio de CMAPSS.

3.4. Modelo predictivo para la continuación iterativa de señales temporales

La predicción de la continuación de señales, en un contexto industrial, se centra en proyectar la evolución de una señal sensorizada a partir de su historial más reciente. En la literatura, se destaca que la predicción a corto plazo puede desempeñar un papel sustancial en la monitorización en tiempo real y la anticipación de desviaciones tempranas, permitiendo reforzar la capacidad preventiva de un sistema [15].

Asimismo, en el marco de la Industria 4.0, la incorporación de estas funcionalidades predictivas dentro de las arquitecturas ciberfísicas se señala como un elemento clave para enriquecer la supervisión y facilitar la toma de decisiones operativas [18].

Dicho esto, la continuación de señales se puede presentar como una funcionalidad auxiliar que complementa los modelos principales (como los presentados en este trabajo), aportando información predictiva sobre la evolución futura de la señal monitorizada.

3.4.1. Definición del problema

El objetivo del modelo a desarrollar es predecir la continuación de una señal temporal a partir de una ventana histórica de muestras, estimando un horizonte $H > 1$ valores futuros. Es decir, dado un vector de entradas

$$(x_{t-m+1}, \ldots, x_t),$$

que corresponde a los m últimos instantes de la señal, el modelo debe generar como salida el conjunto de predicciones

$$(\hat{x}_{t+1}, \hat{x}_{t+2}, \ldots, \hat{x}_{t+H}).$$

Generalizando, se deberá de predecir una cantidad de muestras futuras variable a partir un vector de registros correspondiente a la situación de la señal en ese momento.

Este problema se etiqueta como *multi-step-ahead forecasting* y, el factor de predecir un horizonte variable, plantea dificultades adicionales frente a la predicción de un único paso, como la acumulación de error y la creciente incertidumbre conforme aumenta el horizonte de predicción, es decir, a medida que el valor a predecir se aleja más de la señal original, dada la naturaleza iterativa del modelo a desarrollar (a partir de cierto punto, el propio *input* del modelo serán sus propias predicciones), la fiabilidad de la predicción se verá reducida [19]. Para abordarlo, se han propuesto múltiples estrategias. A continuación se señalan algunas de las más relevantes:

- la **iterada o recursiva**, que entrena un único modelo de un paso y lo aplica de forma sucesiva,
- la **directa**, que entrena modelos específicos para cada horizonte,

- o la **multi-output (MIMO)**, que entrena un único modelo multisalida para predecir simultáneamente el vector de futuros.

En este trabajo se ha optado por la estrategia **iterada**, debido a su simplicidad y a que permite aprovechar al máximo la información contenida en los datos disponibles. No obstante, es importante considerar que, como se ha comentado anteriormente, esta aproximación es especialmente sensible a la propagación de errores entre pasos sucesivos, lo que implica que la fiabilidad de las predicciones tiende a decrecer a medida que se amplía el horizonte temporal.

3.4.2. Preparación y preprocesamiento de datos

El *dataset* a emplear en este problema serán las muestras pertenecientes a una señal univariante acumulativa con frecuencia horaria perteneciente al entorno de Fast Monitoring, que representa al tipo de variables monitorizadas en el entorno.

Cada registro de la señal está compuesto por una marca temporal *Moment* y el valor muestreado en dicho instante temporal.

La Figura 17 presenta la muestra de la señal exportada con la que se va a trabajar.

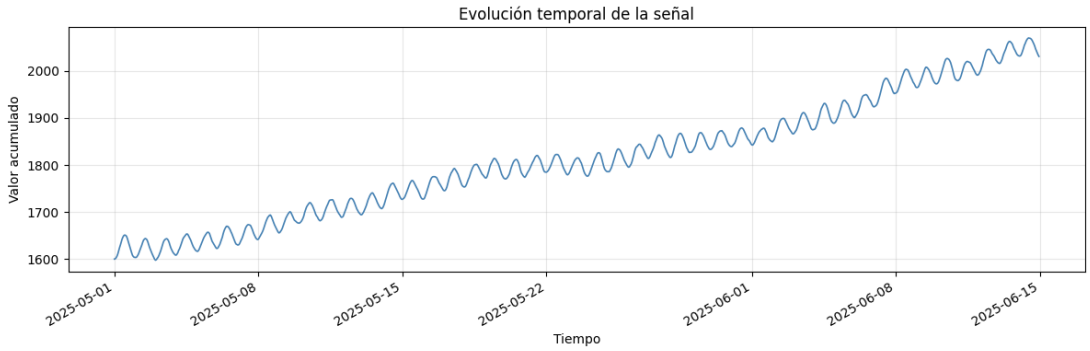

Figura 17: Evolución temporal de la señal.

Como se puede observar, la señal presenta una tendencia ascendente, correspondiente a un valor acumulado que crece a medida que avanza en el tiempo. Por consecuencia, se deduce que cada nueva medida representa la suma de la anterior más el incremento producido en ese intervalo, comportamiento similar al contador de agua acumulativo.

Dicho esto, con el fin de que el algoritmo no aprenda solamente esta tendencia acumulada, se transforma la serie en *diferencias*. Esta técnica se utiliza comúnmente en series temporales para hacerlas más estacionarias. [20]

De esta manera, se representa cuánto sube o baja la señal en cada instante, ayudando al algoritmo a entender la naturaleza de la serie sin verse sesgado por el crecimiento continuado. La operación de *diferencias* se obtiene de restar cada valor de su inmediato anterior, es decir:

$$d_t = x_t - x_{t-1}.$$

Asimismo, se aplica al problema un enfoque de ventana deslizante, para cada instante i, se seleccionan los últimos W valores ya observados de la serie

$$\{d_{i-W+1}, \ldots, d_i\},$$

y el objetivo de predicción es el siguiente valor d_{i+1}. De este modo, cada ejemplo de entrenamiento se compone de una entrada (los últimos W pasos) y una salida (el siguiente paso). Por

consecuencia, si se deseara realizar una predicción sobre el futuro de la señal, el vector de muestras deberá de tener al menos un tamaño de W registros.Además, se generan distintas *features* sobre dichas ventanas con el fin de mejorar la calidad de las predicciones.

En la Tabla 12 se presentan las características generadas, su motivación y naturaleza.

Feature	**Descripción**	**Motivación**	**Tipo**
diff	Valor de la señal en el instante $t-k$ (últimos W pasos)	Capturar el comportamiento reciente de la señal.	Final
mean-short	Media de los últimos n	Reducir ruido y resumir comportamiento reciente.	Final
std-short	Desviación típica de los últimos $t-k$ pasos	Medir la variación a reciente y detectar cambios rápidos.	Final
mean-long	Media de los últimos i pasos	Capturar la tendencia general en una ventana más amplia.	Final
std-long	Desviación típica de los últimos i pasos	Detectar cambios de comportamiento a medio plazo.	Final

Tabla 12: Características derivadas para la predicción iterativa sobre señales temporales.

Asimismo, la señal muestra patrones cíclicos que se repiten diariamente por lo que podría ser interesante ayudar al algoritmo a identificar este fenómeno. Sin embargo, si se tratara de utilizar la hora muestreada como una variable numérica (0,1,2, ..., 23) el modelo interpretaría que las horas 23 y 0 son muy diferentes (aunque en realidad sean dos momentos consecutivos en el ciclo diario).

Para evitarlo, se realizará una codificación temporal utilizando funciones trigonométricas [21]:

$$\text{sin-hour} = \sin\left(\frac{2\pi h}{24}\right), \qquad \text{cos-hour} = \cos\left(\frac{2\pi h}{24}\right)$$

donde h es la hora local expresada en formato $0 \leq h \leq 23$.

Con esta transformación, cada hora se representa como un punto en un círculo donde la 0 y 24 ocupan el mismo lugar y las horas intermedias son puntos adyacentes.

3.4.3. Selección y entrenamiento del modelo XGBoost Iterativo

El objetivo del modelo es predecir la evolución futura de una señal utilizando un enfoque iterativo. Consecuentemente, cada nueva predicción se incorporará como entrada al propio modelo para generar la siguiente. De este modo, a medida que avanza el horizonte de predicción, la ventana de entrada va perdiendo datos reales y se va componiendo sucesivamente de valores predichos. En el caso de horizontes suficientemente largos ($h > W$), la serie de entrada estaría formada únicamente por predicciones.

Para que el modelo se considere válido, se deberán cumplir los siguientes criterios de aceptación:

- Mantener un error cuadrático medio (RMSE) bajo en las primeras iteraciones del horizonte y poder observar el fenómeno del error acumulado a medida que avanza la continuación.
- Capturar adecuadamente la periodicidad de la señal, evitando perder el patrón general de la misma.

Relativo a la selección del algoritmo, se ha optado por entrenar XGBoost, para el que se han probado múltiples combinaciones de hiperparámetros con el fin de obtener predicciones óptimas.

En cuanto a las combinaciones probadas, dado que cada prueba es computacionalmente costosa para este caso, se optó por utilizar *Random Search* [22].

Este método consiste en muestrear de forma aleatoria combinaciones de hiperparámetros a partir de rangos previamente definidos, en lugar de probar de forma exhaustiva todas las posibilidades. En la literatura, se señala que *Random Search* en ciertos casos es más eficiente que un simple *Grid Search* al permitir cubrir mejor el espacio de búsqueda con un menor número de evaluaciones y encontrar las mejores configuraciones de combinaciones con un coste computacional reducido [22].

A continuación se muestran los rangos de valores explorados para cada variable:

Hiperparámetro	**Valores explorados**
max_depth	Número entero entre [3, 9]
eta (learning rate)	Número real entre [0.001, 0.12]
subsample	Número real entre [0.7, 1.0]
colsample_bytree	Número real entre [0.7, 1.0]
num_boost_round	2000
early_stopping_rounds	100
random_state	42

Tabla 13: Rangos de valores explorados en el entrenamiento del modelo XGBoost para continuación de señales.

Asimismo, se han explorado diferentes combinaciones de ventana (W) y longitudes de estadísticas de corto y medio plazo. Los rangos se presentan en la Tabla 14:

Parámetro de proceso	**Valores explorados**
W	12, 18, 24, 30
n	4, 6, 8
i	10, 12, 14

Tabla 14: Rangos de valores explorados en características.

3.4.4. Evaluación del modelo

Tras la evaluación de todas las combinaciones, en la Tabla 15 se muestran las diez configuraciones con menor RMSE.

W	n	i	max_depth	eta	subsample	colsample	RMSE(val)	RMSE(test)
30	8	14	3	0.0924	0.8177	0.9772	0.7138	36.72
30	4	10	4	0.0563	0.7991	0.9058	0.7359	47.32
30	6	10	3	0.0169	0.9042	0.8181	0.7422	42.95
30	6	14	4	0.0884	0.7494	0.7135	0.7438	38.11
30	8	12	4	0.0201	0.7979	0.8925	0.7445	42.63
30	6	12	4	0.0148	0.9267	0.7772	0.7451	40.95
30	4	12	3	0.0106	0.7990	0.7434	0.7467	45.87
30	8	10	5	0.0238	0.8671	0.9352	0.7501	42.66
30	6	12	3	0.0090	0.7387	0.7457	0.7520	39.97
30	4	10	5	0.0738	0.9288	0.7812	0.7545	53.92

Tabla 15: Top 10 combinaciones con menor RMSE en validación

Una vez comparados los resultados anteriores, la configuración que presenta menor RMSE y por lo tanto se considera la mejor candidata, ha sido la siguiente:

```
['n_estimators': 2000, 'learning_rate': 0.0924, 'max_depth': 3,
 'subsample': 0.8177, 'colsample_bytree': 0.9772, 'early_stopping_rounds': 100,
 'random_state': 42], Window: 30, n: 8, i: 14
```

3.4.5. Interpretación y visualización comparativa de resultados

En la Figura 18 se representa la comparación entre la continuación de la señal real y la predicha por el modelo generado. En la figura, se observan diferentes zonas sombreadas:

- Zona verde: Representa las muestras pertenecientes a la señal real que sirven como entrada al modelo.
- Zona roja: Son los valores predichos por el modelo de manera iterativa.
- Zona azul: Son los valores predichos por el modelo si la entrada del mismo estuviera formada por muestras reales y no valores predichos.

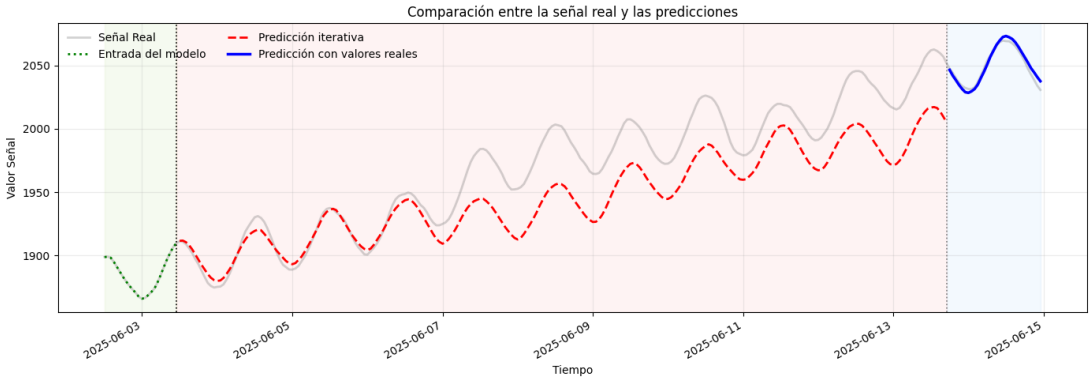

Figura 18: Comparación entre la señal real y la señal predicha por el modelo de predicción iterativa.

Durante el tramo iterativo, los valores generados por el modelo mantienen la periodicidad pero a medida que avanza en el horizonte, se refleja un crecimiento progresivo del error acumulado. Este comportamiento queda representado en la Figura 19, donde se observa cómo el RMSE acumulado se mantiene bajo en las primeras iteraciones de la predicción y aumenta conforme avanza el horizonte.

En contraste, cuando la predicción se realiza mediante valores reales de la señal y se elimina el factor del error acumulado, el modelo mantiene un mayor ajuste a la señal, lo que confirma que la degradación observada proviene principalmente de la naturaleza iterativa y no de la capacidad predictiva del algoritmo en sí.

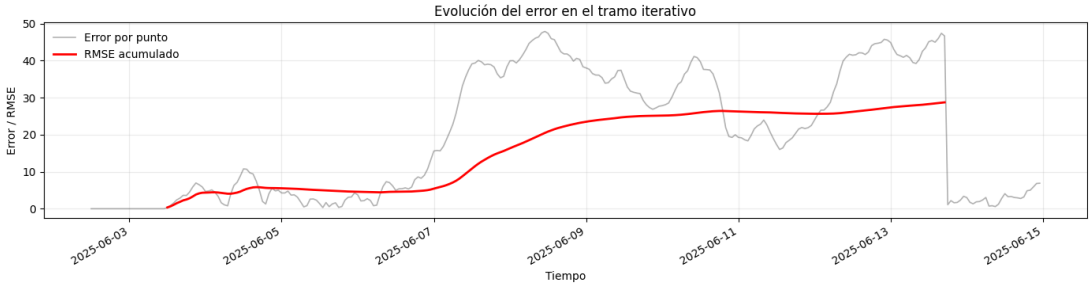

Figura 19: Evolución del RMSE acumulado durante la predicción iterativa a lo largo de la señal.

3.4.6. Resultados respecto a los criterios de aceptación establecidos

Una vez analizados los resultados, la propuesta de solución presentada se considera válida, cumpliendo el modelo con los criterios establecidos anteriormete:

- En las primeras iteraciones del horizonte, el RMSE se mantiene reducido.
- Se puede observar el fenómeno de la curva de error acumulado en tramo iterativo.
- Se conserva la periodicidad de la señal.

3.4.7. Limitaciones y posibles mejoras del modelo iterativo

Si se decidiera implementar el modelo en un entorno industrial real, la principal limitación sería la propagación del error, reduciendo de fiabilidad de las predicciones conforme avanza el horizonte.

Como posibles mejoras, se propone la generación de más y más informativas variables contextuales, que ayuden al modelo a predecir mejor el comportamiento de la señal, y la prueba y estudio de otras estrategias para abordar el problema de la propagación del error. Además, sería conveniente estudiar el entrenamiento de algoritmos más específicos para series temporales y la aplicación de técnicas de normalización, si estos fueran sensibles a la escala de las variables.

4. Integración de los modelos en Fast Monitoring

En esta sección, se describe el entorno final en el que los modelos generados son desplegados, así como también el proceso que se ha seguido para su implementación en esta plataforma industrial.

4.1. Fast Monitoring

Fast Monitoring [5] es una solución industrial IoT (*Industrial Internet of Things*, por sus siglas en inglés) orientada a la digitalización de entornos industriales. Su principal propósito es conectar y estandarizar los diferentes orígenes de información, como sensores, máquinas y otros dispositivos de planta, para crear una fuente única y segura de acceso a los datos.

Asimismo, Fast Monitoring, permite integrar cualquier dispositivo, máquina o PLC, independientemente de su fabricante o protocolo de comunicación, mediante conectividad multiprotocolo.

[5] `https://emisuite.es/fast-monitoring/`

Uno de los pilares fundamentales de esta plataforma se basa en la visualización de datos en tiempo real, permitiendo consultar, desde cualquier dispositivo, variables como la pérdida de rendimiento, cantidad de producción o variables relativas al proceso de las máquinas.

Además, la plataforma incluye un sistema de alarmas y notificaciones en tiempo real, permitiendo a operarios y supervisores de planta recibir información relativa a eventos críticos de forma inmediata. Esta supervisión, se puede realizar desde una gran variedad de dispositivos, incluyendo dispositivos móviles y *smartwatches*.

Fast Monitoring también ofrece múltiples herramientas adicionales orientadas a la gestión industrial y trazabilidad, entre las que destacan:

- Control de usuarios
- Estadísticas e informes
- Acceso multidispositivo
- Redundancia de datos
- Desarrollo de módulos a medida
- Integración con sistemas ERP
- Limpieza de datos

4.2. Exportación de los modelos a formato compatible

Todos los modelos generados durante la realización de este trabajo, han sido programados, entrenados y validados en Python. Sin embargo, para poder integrarlos en la plataforma industrial de Fast Monitoring, desarrollada en C#, es necesario convertir los modelos a un formato que permita su ejecución fuera del entorno de Python.

Para ello, se ha optado por emplear ONNX (*Open Neural Network Exchange*), un estándar abierto que define un modelo de grafo de computación extensible con el fin de representar algoritmos de inteligencia artificial, tanto de *deep learning* como *machine learning* tradicional. ONNX posibilita que un modelo entrenado en un lenguaje como pudiera ser Python, pueda ser posteriormente desplegado en otro distinto, sin necesidad de reentrenar el modelo o realizar conversiones específicas[6]. En cuanto a sus principales ventajas, se destacan:

- ONNX es compatible con multitud de bibliotecas como scikit-learn, XGBoost o TensorFlow, utilizadas en este proyecto, y ejecutarlos en una gran variedad de lenguajes, incluyendo C#.
- El estándar proporciona un formato común compatible con la industria que da pie a integrar los modelos exportados en entornos de producción variados.
- ONNX Runtime requiere de pocos recursos para su ejecución, ofreciendo latencias bajas y modelos fácilmente escalables, factor importante para ser compatible con el entorno de Fast Monitoring.

El procedimiento que se ha seguido para exportar los modelos se puede resumir en:

1. Tras entrenar el modelo, se renombrar sus variables para cumplir con el formato requerido por ONNX (f0, f1, f2, ..., fn).

[6] `https://onnxruntime.ai/docs/`

2. Empleando la librería *onnxmltools* el modelo se convierte a formato ONNX, especificando durante el proceso los tipos de entrada esperados por el modelo exportado, mediante *FloatTensorType*.
3. Una vez exportado el modelo, se valida en el propio entorno de Python y se comparan los resultados de inferencia con los generados por el modelo original.
4. El componente generado, en formato *.onnx* se exporta al servidor de Fast Monitoring para su posterior uso y despliegue.

Aunque se han mencionado múltiples ventajas que avalan la selección de este estándar, también presenta ciertas limitaciones:

- ONNX no ofrece soporte para todos los algoritmos de *Machine Learning* presentes en la literatura, lo que limita la selección de los mismos.
- Aunque los modelos exportados son reutilizables en múltiples lenguajes, el preprocesamiento realizado en los datos que sirven como entrada al modelo se ha de realizar en el lenguaje destino. Para este proyecto, se ha tenido que traducir el preprocesamiento de Python a C#, pudiéndose generar discrepancias durante el mismo si no se realiza correctamente.

En conclusión, la exportación de los modelos a ONNX ha permitido garantizar su correcta integración en Fast Monitoring participando en múltiples flujos industriales como componentes del BPM.

4.3. Integración como componentes BPM

Una vez entrenados y exportados, los modelos desarrollados relativos a la detección de anomalías (UAD) y la predicción del tiempo de vida útil restante (RUL), se integran en la plataforma Fast Monitoring como componentes BPM (*Business Process Module*, por sus siglas en inglés), programados en C#.

El objetivo es encapsular cada uno de los modelos en módulos independientes que permitan su ejecución autónoma en tiempo real como parte de un posible flujo industrial.

Cada componente se diseñó para comportarse como un bloque funcional dentro del lienzo de Fast Monitoring, contando con puertas de entrada y salida que permitieran su conexión con otros módulos ya existentes que formen parte del sistema, como operadores aritméticos, convertidores, comparadores o temporizadores, entre otros.

Como norma general, los componentes se ejecutan de forma periódica en función de un temporizador, reciben señales procedentes del entorno, procesan los datos recibidos mediante el modelo cargado y generan una salida.

En los apartados siguientes se detallan los componentes desarrollados para los modelos mencionados, describiendo su estructura interna y la lógica de funcionamiento implementada.

4.3.1. Componente UAD

El componente *UAD* se encarga de implementar el modelo de detección de anomalías. Como se ha explicado en apartados anteriores, su función principal es la de identificar comportamientos anómalos en una señal industrial generada por un contador de agua de tipo acumulativo, en un entorno de ejecución en tiempo real.

De forma más específica, el componente fue diseñado para cumplir las siguientes funciones:

- Generar una salida *booleana* a través de una puerta de salida en función de si la muestra correspondiente a un *moment* se detecta como anómala. Se interpretará *False* como comportamiento normal y *True* como anómalo.

- Registrar en una base de datos aquellos períodos anómalos identificados, indicando su momento de inicio y fin, el *score* asociado a la inferencia y un identificador de componente configurable. Este identificador permitirá distinguir los registros cuando existan múltiples instancias del componente *UAD* operando de forma paralela.
- Almacenar en un fichero local (cuya ruta podrá configurarse) todas aquellas muestras detectadas como anómalas, indicando la marca temporal el *score* asociado a cada una de ellas.

Cabe destacar que tanto el registro en base de datos como en fichero serán configurables por el usuario y podrán desactivarse si así se desea. Sin embargo, de existir puerta de salida, el componente siempre generará un resultado.

Para representar de forma esquemática la estructura del componente *UAD* dentro del entorno de Fast Monitoring, se ha elaborado la Figura 20. En ella se visualizan las entradas funcionales necesarias para su correcta ejecución y la salida generada tras la inferencia:

Figura 20: Esquema lógico del componente UAD en Fast Monitoring.

Como se observa, el componente cuenta con dos entradas: una puerta por la que recibirá la señal en tiempo real y otra conectada a un temporizador. Esta última será la encargada de marcar la frecuencia de ejecución del componente, controlando cuando se realiza una nueva lectura.

Tal como está diseñado el entorno de Fast Monitoring, el *temporizador* se implementa como un componente configurable cuyo *tick rate* puede definirse mediante una expresión temporal (en milisegundos). De forma periódica, este componente emite un valor booleano *True* cada vez que se cumple el intervalo indicado.

Dicho esto, cuando el componente *UAD* reciba un valor *True* desde la entrada del temporaizador, leerá el valor actual de la señal recibida por la otra puerta de entrada y procederá con el procesamiento del dato.

Una vez recibido el dato de la señal, el componente realiza una serie de operaciones internas con el fin de evaluar si la muestra puede considerarse como un comportamiento anómalo. Estas operaciones se ejecutan de forma secuencial en cada activación del temporizador y se resumen en los pseudocódigos que se muestran a continuación.

Cabe destacar que, dado que el modelo entrenado procesa features calculadas sobre una ventana temporal, las primeras n muestras leídas no generarán resultado hasta que el buffer esté completamente lleno y las características que sirven de entrada al modelo puedan ser calculadas.

Setup Inicializa el componente.
El *setup()* se trata de un método que se ejecuta una vez al inicio de la vida del componente y su objetivo es el de iniciar todos los recursos necesarios para su correcto funcionamiento.

A continuación, se muestra un esquema general de las operaciones que se realizan en el método:

```
1 def setup(conf):
2     log.info("Inicio del setup")
```

```
    try:
        LeerParamsConfig(conf)
        CargarModeloONNX()
        PrintMetadataModelo()
    except error:
        log.error("Fallo en la inicializacin del componente")
```

Listing 1: método setup()

En primer lugar, se leen los parámetros de configuración proporcionados desde la interfaz de Fast Monitoring, que incluyen:

- Nombre del modelo *.onnx* a utilizar.
- Modo de salida (*Fichero*, *Base de Datos* o *Ambos*).
- Identificador alfanumérico único para las anomalías.
- Ruta del fichero de salida.
- Cadena de conexión a la base de datos.

Se añade un valor por defecto a todos los parámetros dado que existe la posibilidad de que el usuario no introduzca algunos o ninguno de ellos antes de pretender realizar la inferencia.

```
def LeerParamsConfig(conf):
    for parametro in configuracion:
        if parametro.valor is valido:
            asignar al atributo correspondiente
            log.info("Configurado correctamente")
        else:
            mantener valor por defecto
            log.warning("Usando valor por defecto")
```

Listing 2: lectura de parámetros de configuración

A continuación, se procede con la carga modelo ONNX desde la ruta previamente especificada y se crea una sesión de inferencia. Si la carga es satisfactoria, el componente está listo para comenzar a recibir datos.

```
def CargarModeloONNX():
    if la ruta del modelo es absoluta:
        usar como path final
    else:
        construir ruta relativa al directorio actual

    log.info("Mostrando ruta y existencia del fichero")

    try:
        crear sesion de inferencia con el modelo ONNX
        log.info("Modelo cargado correctamente")
    except error:
        log.error("Fallo al cargar modelo ONNX")
```

Listing 3: carga del modelo ONNX

Por último, se imprime en el sistema de logs la información relativa a los metadatos de entrada y salida del modelo, con el fin de validar la ejecución y facilitar la depuración si se realizara.

```
def PrintMetadataModelo():
    for entrada in metadatos de entrada:
        log.info("INPUT -> nombre, dimensiones y tipo")

    for salida in metadatos de salida:
        log.info("OUTPUT -> nombre, dimensiones y tipo")
```

Listing 4: impresión de metadatos del modelo ONNX

Execute Funcionamiento periódico del componente
El *execute()* se trata de un método que se activa cada vez que el componente recibe datos por alguna de sus puertas. En general, su objetivo es procesar las muestras de la señal, verificar si hay suficientes datos en el *buffer*, calcular las *features* que sirven como *input* al modelo y realizar la inferencia.

A continuación se muestra un esquema de su funcionamiento:

```
def Execute():
    if la sesion ONNX no esta cargada correctamente:
        log.error("Sesin ONNX no inicializada")
        return

    if el numero de entradas no es igual a 2:
        log.error("Nmero incorrecto de entradas")
        return

    (timerPulse, moment) = LeerEntradasGates()

    if timerPulse es True y el buffer tiene suficientes datos:
        inputVector = EjecutarPreprocesamientoEnCSharp()

        if inputVector no es nulo:
            RealizarInferencia(inputVector, moment)
        else:
            log.warning("Fallo en el preprocesamiento, vector nulo")
```

Listing 5: método execute()

En primer lugar, se comprueba tanto que la sesión de inferencia haya sido inicializada como que el usuario haya añadido las dos puertas de entrada necesarias para la correcta ejecución del componente.

Una vez realizadas las comprobaciones iniciales, se leen los valores de las puertas de entrada. En el caso de la señal *signal*, si el valor recibido es distinto al anterior, la muestra se añade al *buffer*. Respecto al temporizador, el método devuelve si se ha activado (*True*;*False*) y la marca temporal de la última muestra procesada.

```
def LeerEntradasGates():
    timerPulse = False
```

```
3
4       for cada puerta de entrada:
5           tag = puerta.GetValue()
6           if tag es nulo:
7               continuar con la siguiente
8
9           if puerta es "Signal":
10              extraer valor y momento de la signal
11
12              if el valor es diferente al anterior:
13                  actualizar buffer con el nuevo valor
14                  actualizar momento de la signal
15                  log.debug("Valor de seal actualizado")
16              else:
17                  log.debug("Valor identico, no se actualiza el buffer")
18
19          if puerta es "Timer":
20              timerPulse = valor booleano del tag
21              log.debug("Pulso de timer recibido")
22
23      return (timerPulse, momento de la signal)
```

Listing 6: método leerEntradasGates()

A continuación, si el temporizador se ha activado y el *buffer* tiene suficientes datos, se procede con el cálculo de las *features* que se usan como entrada del modelo.

```
1   def EjecutarPreprocesamientoEnCSharp():
2       if el buffer tiene menos de N valores:
3           log.warning("Buffer insuficiente para preprocesamiento")
4           return null
5
6       flat_zone = 1 si todos los ultimos N valores son iguales, 0 en otro caso
7       signal_norm = normalizacion respecto a max y min de la ventana
8       trend_slope = pendiente de la regresion lineal sobre la ventana
9       diff_rel = variacion relativa entre las dos ultimas muestras
10      std_threshold = desviacion estandar de los ultimos n valores
11
12      return vector con las 5 features
```

Listing 7: método ejecutarPreprocesamientoEnCSharp()

Por último, se realiza la inferencia. Las *features* generadas se pasan al modelo y este genera una predicción, compuesta por un indicador de anomalía (-1, para el caso de anomalía detectada) y el *score* del resultado.

En caso de anomalía, se realizan diferentes tareas para la salida:

- Se envía una señal *True* a través de la puerta de salida, propagando la información al resto del BPM.
- Si el modo de salida incluye la opción de *fichero*, se almacena la muestra detectada en el *path* indicado junto con la marca temporal y *score*.
- Si no se encontraba activo un período anómalo en curso, se considera que se ha iniciado

uno nuevo. Para este caso, si el modo de salida incluye la opción de *base de datos*, registra el inicio de un período en la tabla correspondiente.

Por el contrario, en el caso de que no se haya detectado una anomalía y anteriormente se hubiera detectado un caso, se considera que el período anómalo ha finalizado, y se registra mediante una inserción SQL en la tabla correspondiente si procede.

```
def RealizarInferencia(inputVector, moment):
    crear tensor con forma [1, 5] a partir de inputVector
    preparar entrada nombrada para la sesin ONNX
    ejecutar la inferencia con el modelo cargado
    obtener la prediccion y el score de los resultados

    if los resultados son invalidos:
        log.error("Resultados de inferencia no validos")
        return

    guardar la prediccion y el score
    log.info("Prediccion y score obtenidos")

    if la prediccion es anomala (-1):
        enviar True por la puerta de salida (si existe)

        if el modo de salida incluye fichero:
            escribir resultado en el fichero

        if no habia una anomalia activa:
            marcar inicio del periodo anomalo
            if el modo de salida incluye base de datos:
                registrar inicio en BBDD
    else if habia una anomalia activa:
        marcar fin del periodo anmalo
        if el modo de salida incluye base de datos:
            registrar fin en BBDD
```

Listing 8: Pseudocódigo del método `RealizarInferencia`

De este modo, el componente *UAD* queda completamente integrado en el ecosistema BPM de Fast Monitoring, operando de forma autónoma y configurable, permitiendo automatizar la supervisión de señales industriales en tiempo real, mejorando la capacidad del sistema para detectar comportamientos anómalos sin necesidad de supervisión constante.

4.3.2. Componente RUL

El componente *RUL* implementa el modelo dedicado a la estimación de tiempo de vida útil restante de un sistema a partir de los datos sensoriales disponibles en tiempo real.

El objetivo principal del componente es predecir el tiempo restante de vida útil de un motor en distintas marcas temporales con el fin de estudiar su patrón de degradación y anticipar posibles fallos de funcionamiento a la vez que se optimizan las tareas de mantenimiento. El motor a supervisar está monitorizado por los mismos sensores utilizados para el entrenamiento del modelo y que sirven como entrada al componente.

Más específicamente, el componente cumple las siguientes funciones:

- Generar una salida a través de una puerta con las predicciones generadas, de modo que otros bloques del BPM puedan reaccionar a las mismas.
- Si el valor *RUL* predicho es inferior a cierto umbral crítico configurable, se considera que ha comenzado un período de riesgo operacional. Este evento se registra en una base de datos.
- Actualizar los registros correspondientes para reflejar el cierre de un período crítico en caso de detectar su finalización.
- Almacenar los resultados generados por el modelo en un fichero local junto con la marca temporal correspondiente.

A continuación, de forma esquemática, se ha elaborado la Figura 21 con el fin de representar la estructura del componente RUL dentro del entorno Fast Monitoring, representándose las entradas funcionales necesarias y la salida tras la inferencia.

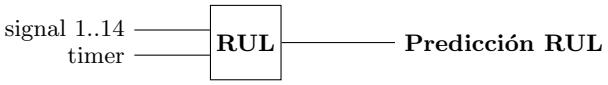

Figura 21: Esquema lógico del componente RUL en Fast Monitoring.

Como se observa, el componente cuenta con dos fuentes de entrada, un *input* correspondiente al temporizador, marcando la frecuencia de ejecución, y un conjunto de 14 señales procedentes de sensores seleccionadas durante la fase de preprocesamiento del modelo por su valor informativo. Dicho esto, cada vez que el componente reciba un pulso de activación, recogerá las últimas muestras de los múltiples sensores conectados y procesará los datos con el fin de conformar una entrada al modelo que, a su vez, generará una predicción mediante la ejecución de la inferencia.

A continuación, se presenta el pseudocódigo relativo a la lógica de ejecución del componente, con el fin de detallar las operaciones internas realizadas en cada activación del temporizador.

Setup Inicializa el componente.
Al igual que el componente *UAD*, se realiza la inicialización de los recursos mediante el método *setup()*.

De la misma manera, el componente *RUL* comenzará por la lectura de los parámetros de configuración definidos por el usuario en el entorno de ejecución, incluyendo:

- Nombre del modelo *.onnx* a utilizar.
- Identificador del motor al que se le aplicará la predicción RUL.
- Umbral a partir del cual se considera que se inicia un período crítico operacional.
- Modo de salida (*Fichero*, *Base de Datos* o *Ambos*).
- Ruta del fichero de salida.
- Cadena de conexión a la base de datos.

Seguidamente, se carga el modelo ONNX desde la ruta especificada y, en caso de haberse creado una sesión de inferencia exitosa, se imprimen los metadatos del modelo mediante el sistema de *logs* poniendo fin a la inicialización del componente.

```
def setup(config):
    log.info("Inicio del setup del componente RUL")

    try:
        LeerParametros(config)
        CargarModeloONNX()
        MostrarMetadatosModelo()
    except error:
        log.error("Error durante la fase de setup")
```

Listing 9: método setup()

Cabe mencionar que los métodos llamados en esta fase han sido descritos en la sección correspondiente al componente UAD.

Execute Funcionamiento periódico del componente
El *execute()* es el método que se activa cada vez que el componente recibe un dato nuevo por alguna de sus puertas.

Siendo la lógica de ejecución general análoga a la implementada en el componente *UAD*, en primer lugar, se leen los valores de las puertas de entrada. De la misma manera, se detecta si el temporizador ha emitido un pulso de activación. Asimismo, se recogen los últimos datos disponibles de las 14 señales procedentes de sensores y, en caso de ser diferentes a la anterior inmediata, se almacenan en un *buffer* temporal que mantiene el histórico para el cálculo de la *feature*.

Una vez se reciben suficientes datos para rellenar el *buffer*, las muestras se transforman para alimentar al modelo. A través de un método auxiliar se generan las *features* requeridas para alimentar al modelo.

Con el vector de características ya construido, se procede a ejecutar el modelo previamente cargado. Para ello, se transforma el vector en un objeto compatible con ONNX y se ejecuta la inferencia. La salida será un valor numérico que representa el número estimado de ciclos restantes antes del fallo del sistema supervisado.

```
def RealizarInferencia(inputVector, moment):
    log.info("Inicio de inferencia con input: " + inputVector)

    crear objeto de entrada con forma (1, 14)
    asignar los valores del vector al objeto

    preparar la entrada ONNX con el objeto
    ejecutar el modelo y obtener resultados

    rul_predicho = extraer valor de salida

    ProcesarPrediccion(rul_predicho, moment)
```

Listing 10: método realizarInferencia()

Por último, ya generada la predicción, es necesario gestionar su almacenamiento y evaluar si el sistema entra o sale de un estado crítico. Además de registrar el valor predicho en un fichero, se evalúa si el valor cae por debajo de un umbral crítico y, en ese caso, se marcan los inicios y finales de períodos de riesgo en base de datos, permitiendo la trazabilidad en un futuro.

```
def ProcesarPrediccion(rul_predicho, moment):
    log.info("RUL predicho: " + rul_predicho)

    if rul_predicho < umbral crtico:
        if no estamos en perodo crtico:
            marcar inicio de perodo crtico
            if el modo de salida incluye base de datos:
                registrar inicio en BBDD
            log.info("Inicio de perodo crtico registrado")
    else:
        if estamos en perodo crtico:
            marcar fin de perodo crtico
            if el modo de salida incluye base de datos:
                registrar fin en BBDD
            log.info("Fin de perodo crtico registrado")

    if el modo de salida incluye fichero:
        escribir rul_predicho en el fichero

    if hay puerta de salida:
        enviar rul_predicho por la puerta
```

Listing 11: método procesarPrediccion()

De esta manera, el componente *RUL* se integra de forma completa en el ecosistema de Fast Monitoring, proporcionando las herramientas necesarias para la anticipación de escenarios de riesgo y la toma de decisiones preventivas en el momento adecuado.

4.4. Integración como funcionalidad auxiliar de visualización

Además de la integración de los modelos (RUL y UAD) como componentes BPM en Fast Monitoring, se planteó integrar el modelo de continuación iterativa de señales temporales como una funcionalidad auxiliar en la interfaz de generación de gráficas de la plataforma.

La idea de esta integración era ofrecer al usuario la posibilidad de extender la visualización de una señal monitorizada más allá de los valores históricos registrados. Para ello, se planteó la incorporación de un botón en el menú de monitorización de señales que permitiese generar su prolongación temporal dado un horizonte de predicción especificado por el usuario. De esta manera, el sistema representaría la serie real y la continuación predicha conjuntamente.

Esta funcionalidad aportaría al sistema un valor significativo. Se podrían anticipar tendencias o incluso facilitar valores futuros de una señal que podrían ser aprovechados por otros componentes en un flujo industrial, no limitándose a la visualización solamente.

Sin embargo, dada la complejidad técnica de la pintar la gráfica y alcance del proyecto, la integración de esta funcionalidad no fue posible en el plazo disponible. Aunque el modelo de continuación se ha desarrollado y validado satisfactoriamente, su integración en Fast Monitoring requeriría modificaciones en la lógica de la plataforma que exceden el alcance de este trabajo.

Dicho esto, la integración de este modelo en Fast Monitoring se presenta como una propuesta de mejora futura, potencialmente interesante en un entorno industrial en tiempo real.

5. Generación de alertas, avisos y automatización de trabajos de mantenimiento

Una de las propuestas iniciales para mejorar la plataforma de Fast Monitoring se basaba en la automatización del ciclo de gestión de incidencias a partir de las predicciones de los modelos UAD y RUL.

La idea era que estos modelos, además de generar las inferencias como componentes de un flujo industrial en tiempo real, alimentaran un sistema de alertas y notificaciones que desencadenasen otras tareas adicionales. Para ello, se contemplaba un flujo basado en dos fases:

1. **Registro de incidencias en ficheros locales y base de datos.**
   Los modelos fueron preparados para escribir en ficheros y en la base de datos del servidor de la plataforma los resultados obtenidos en cada ejecución así como intervalos anómalos o críticos.

2. **Conexión con el sistema de notificaciones.**
   A partir de los registros anteriores, la arquitectura prevista invocaría a una API de notificación con el identificador de la máquina afectada y el de la incidencia detectada. De esta forma, el sistema podría elevar la información a servicios externos encargados de la gestión de avisos y alarmas en el sistema o incluso la planificación de otras tareas como propuestas de trabajo de mantenimiento.

La primera fase del flujo se ha completado en el marco de este trabajo. Sin embargo, la segunda fase no ha podido ser implementada, ya que requería integrar APIs externas y lógica de negocio que excedía el alcance del proyecto pues, la propuesta, implicaba extender el ámbito del trabajo hacia entornos complementarios como la gestión de notificaciones industriales y gestión de procesos de mantenimiento.

Dicho esto, la funcionalidad se presenta como una propuesta de mejora futura que, de ser implementada, añadiría gran valor a la plataforma al poder automatizar la respuesta frente a anomalías o el registro de valores que indiquen el estado crítico de componentes industriales.

6. Conclusiones

Este trabajo de fin de grado ha cumplido mayormente con los objetivos propuestos inicialmente, desarrollando e integrando diferentes modelos de aprendizaje automático en la plataforma industrial de la empresa Soincon.

A pesar de no haber abordado algunos aspectos como el flujo completo de generación de alertas o la automatización de tareas de mantenimiento, el trabajo principal sobre cómo abordar el desarrollo e integración de modelos en Fast Monitoring se ha realizado con éxito y sienta las bases para futuras líneas de mejora.

En lo personal, este proyecto me ha supuesto un reto significativo. Antes de comenzar el período de prácticas externas, no contaba con experiencia previa en el campo del aprendizaje automático, obligándome a estudiar conceptos nuevos y adquirir competencias que no estaban alineadas con la especialización que estaba siguiendo en la universidad. Este proceso me ha permitido tanto comprender y aplicar técnicas de *machine learning* como también integrarlas en un entorno industrial.

En definitiva, este proyecto no solo ha contribuido a validar la viabilidad de aplicar técnicas de aprendizaje automático en un entorno industrial, sino que también ha supuesto un punto de inflexión en mi formación como graduado en ingeniería informática, ampliando mi perfil más allá de la ingeniería del software.

Bibliografía

[1] A. Esteban, A. Zafra, and S. Ventura, "Data mining in predictive maintenance systems: A taxonomy and systematic review," *WIREs Data Mining and Knowledge Discovery*, vol. 12, no. 5, e1471, 2022. doi: 10.1002/widm.1471.

[2] A. T. Keleko, B. Kamsu-Foguem, R. H. Ngouna, and others, "Artificial intelligence and real-time predictive maintenance in industry 4.0: a bibliometric analysis," *AI and Ethics*, vol. 2, pp. 553–577, 2022. doi: 10.1007/s43681-021-00132-6.

[3] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.

[4] F. Arena, M. Collotta, L. Luca, M. Ruggieri, and F. G. Termine, "Predictive maintenance in the automotive sector: A literature review," *Mathematical and Computational Applications*, vol. 27, no. 1, p. 2, 2022. doi: 10.3390/mca27010002.

[5] A. Burkov, *The Hundred-Page Machine Learning Book*. Québec City, QC, Canada: Andriy Burkov, 2019.

[6] D. Chicco, M. J. Warrens, and G. Jurman, "The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation," *PeerJ Computer Science*, vol. 7, e623, Jul. 2021. doi: 10.7717/peerj-cs.623.

[7] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. 2008 Eighth IEEE International Conference on Data Mining (ICDM)*, Pisa, Italy, 2008, pp. 413–422. doi: 10.1109/ICDM.2008.17.

[8] D. Ferguson, A. Henderson, E. F. McInnes, and P. Gardner, "Weakly supervised anomaly detection coupled with Fourier transform infrared (FT-IR) spectroscopy for the identification of non-normal tissue," *The Analyst*, vol. 148, no. 16, pp. 3817–3826, 2023. doi: 10.1039/D3AN00618B.

[9] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD)*, San Francisco, CA, USA, 2016, pp. 785–794. doi: 10.1145/2939672.2939785.

[10] Z. Ge, Z. Song, S. X. Ding, and B. Huang, "Data mining and analytics in the process industry: The role of machine learning," *IEEE Access*, vol. 5, pp. 20590–20616, Oct. 2017. doi: 10.1109/ACCESS.2017.2756872.

[11] A. Alam, "What is machine learning?," Zenodo, Aug. 2023. doi: 10.5281/zenodo.8231580.

[12] P. G. Esteban, F. Zamora-Martínez, Á. Cuenca, and N. Aliane, "Data mining in predictive maintenance systems: A taxonomy and systematic review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 12, no. 3, e1458, 2022. doi: 10.1002/widm.1458.

[13] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth, "CRISP-DM 1.0: Step-by-step data mining guide," CRISP-DM Consortium, 2000. [Online]. Available: https://www.kde.cs.uni-kassel.de/wp-content/uploads/lehre/ws2012-13/kdd/files/CRISPWP-0800.pdf

[14] M. J. Nodeh, M. H. Calp, and I. Şahin, "Analyzing and processing of supplier database based on the cross-industry standard process for data mining (CRISP-DM) algorithm," in *Computer Science and Engineering: Proceedings of ICCSEEA 2019*, Cham, Switzerland: Springer, 2020, pp. 544–558. doi: 10.1007/978-3-030-36178-5_44.

[15] M. Paolanti, L. Romeo, A. Felicetti, A. Mancini, E. Frontoni, and P. Zingaretti, "Machine learning approach for predictive maintenance in Industry 4.0," in *Proc. 2018 14th IEEE/ASME Int. Conf. on Mechatronic and Embedded Systems and Applications (MESA)*, Oulu, Finland, 2018, pp. 1–6. doi: 10.1109/MESA.2018.8449150.

[16] R. Dintén and M. Zorrilla, "Using time series foundation models for few-shot remaining useful life prediction of aircraft engines," *Computer Modeling in Engineering & Sciences*, vol. 144, no. 1, pp. 239–265, 2025. doi: 10.32604/cmes.2025.065461.

[17] O. Asif, S. A. Haider, S. R. Naqvi, J. F. W. Zaki, K.-S. Kwak, and S. M. R. Islam, "A Deep Learning Model for Remaining Useful Life Prediction of Aircraft Turbofan Engine on C-MAPSS Dataset," *IEEE Access*, vol. 10, pp. 94064–94075, Sep. 2022. doi: 10.1109/ACCESS.2022.3203406.

[18] J. Lee, B. Bagheri, and H. A. Kao, "A cyber-physical systems architecture for Industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, Jan. 2015. doi: 10.1016/j.mfglet.2014.12.001.

[19] J. De Stefani, "Thèse présentée en vue de l'obtention du grade académique de Docteur en Sciences Informatiques," Ph.D. dissertation, Université Libre de Bruxelles, Brussels, Belgium, 2022.

[20] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. Hoboken, NJ, USA: Wiley, 2015.

[21] A. Bansal, K. Balaji, and Z. Lalani, "Temporal encoding strategies for energy time series prediction," *arXiv preprint arXiv:2503.15456*, 2025. doi: 10.48550/arXiv.2503.15456.

[22] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, Feb. 2012.