

Aplicación de técnicas de machine learning para el mantenimiento predictivo



Junjie Zhu

Máster en Ciencia de Datos
Data Science en el ámbito
industrial

Tutor/a de TF

Lorena Polo Navarro

**Profesor/a responsable de la
asignatura**

Antonio Lozano Bagén

Fecha Entrega

01/2022

Universitat Oberta
de Catalunya



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada [3.0 España de Creative Commons](#)

Ficha del Trabajo Final

Título del trabajo:	Aplicación de técnicas de machine learning para el mantenimiento predictivo
Nombre del autor/a:	Junjie Zhu
Nombre del Tutor/a de TF:	Lorena Polo Navarro
Nombre del/de la PRA:	Antonio Lozano Bagén
Fecha de entrega:	01/2022
Titulación o programa:	Máster Universitario de Ciencia de Datos
Área del Trabajo Final:	Data Science en el ámbito industrial
Idioma del trabajo:	Castellano
Palabras clave	Mantenimiento predictivo, machine learning.
Resumen del Trabajo	
Uno de los grandes problemas en los procesos de fabricación es la rotura de suministro, una de las causas es el fallo de las máquinas de fabricación. Actualmente, la solución más común es la realización de mantenimientos periódicos, cuyo método conlleva a un coste constante y en mayoría de los casos es innecesario. En este proyecto se estudia cómo pueden contribuir las técnicas de análisis de datos avanzadas en la prevención de estas roturas mientras se reducen los costes de los mantenimientos redundantes. En concreto, se aplicarán estas técnicas para encontrar una solución analítica, con el fin de predecir cuándo una máquina fallará y así poder actuar antes de que ocurra dicho fallo, y así prevenir cortes en el proceso de fabricación.	
Abstract	
The breakage of supply is one of the big problems in the manufacturing processes; one of these causes is the failure of the manufacturing machinery. Currently, the most common solution is to perform periodic maintenance, This method leads to a constant cost where in most cases it is unnecessary. This project studies how advanced data analysis techniques can contribute to prevent these breakages while reducing the cost of constant and redundant maintenance. Specifically, these techniques will be applied to find an analytical solution, in order to predict when a machine will fail and thus be able to act before said failure occurs, to prevent outages in the manufacturing process.	

Índice

1.	Introducción	1
1.1.	Contexto y justificación del Trabajo	1
1.2.	Objetivos del Trabajo	2
1.3.	Enfoque y método seguido	3
1.4.	Planificación del trabajo	4
2.	Estado del arte	6
2.1.	Contexto y actualidad	6
2.2.	Revisión bibliográfica	7
3.	Análisis inicial y procesamiento previo	10
3.1.	Análisis de las fuentes de datos	10
3.1.1.	PdM_errors.csv	10
3.1.2.	PdM_failures.csv	10
3.1.3.	PdM_machines.csv	11
3.1.4.	PdM_maint.csv	11
3.1.5.	PdM_telemetry.csv	12
3.2.	Análisis exploratorio de los datos(EDA)	13
3.2.1.	Telemetry	14
3.2.2.	Machines	16
3.2.3.	Errores	17
3.2.4.	Fallos	19
3.2.5.	Mantenimiento	21
3.3.	Preparación de los datos	22
3.3.1.	Limpieza de datos y valores nulos	22
3.3.2.	Valores atípicos u <i>outliers</i>	22
3.3.3.	Codificación de las variables	22
3.3.4.	Unificación de los datos	23
3.3.5.	Creación de las variables objetivo	23
3.3.6.	Dimensionalidad de los datos	25
3.3.6.1.	Dimensionalidad (segunda iteración)	26
3.3.7.	Normalización de los datos	28
4.	Análisis y modelado	29

4.1.	Creación del conjunto entrenamiento y pruebas	32
4.2.	Modelos	33
4.2.1.	Linear Regression	33
4.2.2.	Logistic Regression Binomial	33
4.2.3.	Support-vector machiens (SVM)	34
4.2.3.1.	SVM Regressor	35
4.2.3.2.	SVM Classifier	35
4.2.4.	Random Forest	35
4.2.4.1.	Random Forest Regressor	36
4.2.4.2.	Random Forest Classifier	36
4.2.5.	Gradient Boosting	37
4.2.5.1.	Gradient Boosting Regressor	37
4.2.5.2.	Gradient Boosting Classifier	37
4.2.6.	Extreme Gradient Boosting	38
4.2.6.1.	Extreme Gradient Boosting Regressor	38
4.2.6.2.	Extreme Gradient Boosting Classifier	38
4.2.7.	Long Short Term Memory (LSTM)	39
4.2.7.1.	LSTM Regressor	41
4.2.7.2.	LSTM Classifier	42
5.	Evaluación de los resultados	43
5.1.	Métricas de regresión	43
5.2.	Resultados de regresión	45
5.3.	Métricas de clasificación	47
5.4.	Resultados de la clasificación	51
5.5.	Desarrollo de los resultados de la regresión	53
6.	Conclusiones y trabajos futuros	55
6.1.	Conocimientos adquiridos	56
6.2.	Objetivos adquiridos	55
6.3.	Seguimiento y planificación	57
6.4.	Líneas de trabajo futuro	57
7.	Bibliografía	58
8.	Anexos	60

Índice de Figuras

Ilustración 1. Diagrama del método seguido.....	3
Ilustración 2. Diagrama de Gantt.....	4
Ilustración 3. Fechas y medidas de los conjuntos de datos	13
Ilustración 4. Exploración de valores nulos.....	13
Ilustración 5. Exploración de registros duplicados	14
Ilustración 6. Volt, rotate, pressure y vibration de la máquina 1	14
Ilustración 7. Boxplot del voltaje de la máquina 1	15
Ilustración 8. Distribución y rangos de valores	15
Ilustración 9. Tipos de modelos y cantidad por modelo	16
Ilustración 10. Distribución de edad de los modelos del conjunto de datos	16
Ilustración 11. Total de errores generados	17
Ilustración 12. Distribución de errores por tipo de error	17
Ilustración 13. Errores por máquina.....	17
Ilustración 14. Evolución del total de errores a lo largo de un año, por cada mes y día	18
Ilustración 15. Total de fallos/roturas de componentes.....	19
Ilustración 16. Distribución de fallos entre tipos y máquinas	19
Ilustración 17. Evolución del total de fallos a lo largo de un año, por cada mes y día.....	20
Ilustración 18. Codificación de la tabla errors	23
Ilustración 19. Dataset unificado	23
Ilustración 20. Gráfico de las RULs	24
Ilustración 21. Conjunto de datos con su RUL por cada ciclo de trabajo	24
Ilustración 22. Correlaciones de los campos	25
Ilustración 23. Ejemplo visual de la selección de ciclos	26
Ilustración 24. Conjunto de datos, segunda iteración.	27
Ilustración 25. Matriz de correlaciones, segunda iteración.	27
Ilustración 26. Conjunto de datos normalizados con MinMaxScaler()	28
Ilustración 27 Ejemplo ilustrado de la convolución. Fuente. Bengio, Goodfellow y Couville (2016)	30
Ilustración 28. Ejemplo de una celda de la red recurrente Fuente: Bosch, Casas y Lozano (2019) [16].....	31
Ilustración 29. Ejemplo visual de los datos de entrada	32
Ilustración 30. Modelo regresión lineal	33
Ilustración 31. Búsqueda de los mejores hiperparámetros de la regresión logística	34
Ilustración 32. Modelo de regresión logístico.....	34
Ilustración 33. Modelo de regresión SVM.....	35
Ilustración 34. Búsqueda de los mejores hiperparámetros de la SVM	35
Ilustración 35. Modelo de clasificación SVM.....	35
Ilustración 36. Modelo de regresión Random Forest	36
Ilustración 37. Búsqueda de los mejores hiperparámetros del Random Forest	36
Ilustración 38. Modelo de clasificación Random Forest.....	37
Ilustración 39. Modelo de regresión Gradient Boosting	37

Ilustración 40. Búsqueda de los mejores hiperparámetros del Gradient Boosting	37
Ilustración 41. Modelo de clasificación Gradient Boosting	38
Ilustración 42. Modelo de regresión XGBoosting	38
Ilustración 43. Búsqueda de los mejores hiperparámetros del Extreme Gradient Boosting	38
Ilustración 44. Modelo de clasificación Extreme Gradient Boosting	39
Ilustración 45. Imagen de una Celda LSTM extraído de [15]	39
Ilustración 46. Prueba de modelos LSTM.....	40
Ilustración 47. Selección de los hiperparámetros del modelo con mejores resultados	41
Ilustración 48. Modelo de regresión LSTM	41
Ilustración 49. Selección de los hiperparámetros del modelo con mejores resultados	42
Ilustración 50. Modelo classificador LSTM	42
Ilustración 51.Predicción de 200 muestras con el modelo Random Forest.....	45
Ilustración 52. Aplicación del modelo Random Forest al conjunto con los ciclos de trabajos normalizados.....	46
Ilustración 53. Imagen ejemplo de la matriz de confusión. Fuente: https://www.datasource.ai/es/data-science-articles/comprehension-de-la-matriz-de-confusion-y-como-implementarla-en-python	47
Ilustración 54. Cálculo de la precisión	48
Ilustración 55. Cálculo del recall.....	48
Ilustración 56. Cálculo del F1 score.....	48
Ilustración 57. Cálculo del accuracy	49
Ilustración 58. Curvas ROC Fuente: Minería de datos, modelos y algoritmos [14].....	50
Ilustración 59. Comparativa de las predicciones del Random Forest y LSTM	51
Ilustración 60. Conversión de RUL (igual o menor de 72) a State	53
Ilustración 61. Matriz de confusión y métricas a partir del modelo de regresión Random Forest.....	54
Ilustración 62. Comparación del modelo de clasificación (grafico superior) con el modelo de regresión convertido en clases (grafico inferior)	54

Índice de Tablas

Tabla 1. Artículo, problema y técnicas usadas	9
Tabla 2. Campos de PdM_errors.csv	10
Tabla 3. Campos de PdM_failres.csv	10
Tabla 4. Campos de PdM_machines.csv	11
Tabla 5. Campos de PdM_maint.csv.....	11
Tabla 6. Campos de PdM_telemetry.csv	12
Tabla 7. Comparación de los resultados de regresión.....	45
Tabla 8. Comparación de resultados de clasificación	51

1. Introducción

1.1. Contexto y justificación del Trabajo

En esta época actual donde la normalidad vuelve poco a poco después de la pandemia del COVID-19. Hemos presenciado a una mayor escala el efecto de la rotura de suministros y fabricación, la cual ha desatado un efecto en cadena donde los usuarios finales se han visto afectados.

Para las empresas este problema es aún mayor, ya que una rotura de suministro o fabricación crean consecuencias directas, como la disminución de ventas y pedidos que no se sirven, que se traducen como una disminución de ingresos y beneficios; o consecuencias indirectas, como la pérdida de confianza de los clientes y proveedores, o la acumulación de materias primas.

Desde el punto de vista industrial, ocurre un efecto similar cuando aparece un fallo en las máquinas de las líneas de producción, causando un gran problema debido que ocasionan paros de emergencia y tiempos muertos, que se traducen a una reducción de ingresos e impacto negativo para la empresa.

En la actualidad con el auge de los dispositivos conectados y el internet de las cosas (IoT, Internet of Things), y junto a la creciente industria 4.0, se nos pone en disposición un conjunto de herramientas y técnicas de análisis y procesamiento de datos para no solo encontrar la causa de los fallos sino que también enfocar a las variables que han causado el fallo, de modo que hay un cambio de paradigma donde en vez de centrarse en la “causalidad” se centra en la “correlación” de estas variables que han causado el fallo.

El presente trabajo está enfocado para cubrir una necesidad existente en las empresas y fábricas que están totalmente o parcialmente automatizadas, utilizando las herramientas disponibles para analizar y procesar estos datos como algoritmos de machine learning, con el fin de obtener un modelo capaz de aprender los posibles patrones que puedan preceder al fallo de una máquina y de este modo realizar el mantenimiento necesario antes de que ocurra dicho fallo.

1.2. Objetivos del Trabajo

Objetivo general:

- Obtención de un modelo de predicción de posibles fallos y defectos de máquinas antes de que ocurra dicho fallo.

Objetivos específicos:

- Conocer y estudiar el estado del arte del machine learning aplicado al ámbito industrial, en concreto, al mantenimiento predictivo.
- Identificación de indicadores relevantes que apuntan a una posible anomalía.
- Estudio y comparación de diferentes modelos para la obtención del mejor modelo.
- Utilización transversal del conocimiento adquirido a lo largo del master para un posible caso real

1.3. Enfoque y método seguido

Para la realización se seguirá el método CRISP-DM del inglés *Cross Industry Standard Process for Data Mining*, uno de los modelos más utilizados y de estándar abierto que describe enfoques comunes que se utiliza para la minería de datos.

Estos modelos consisten en seis fases principales:

- Comprensión del negocio, en este caso se enfoca a la comprensión de la problemática y necesidad existente en el mercado, como los estudios y soluciones ya existentes actualmente.
- Comprensión de los datos, recopilación, descripción y exploración de los datos que se usarán para trabajar. Se realizará un estudio de la relevancia, calidad y variedad de los mismos datos para determinar y seleccionar aquellos valores que son verdaderamente útiles.
- Preparación de los datos, como indica el nombre, se trata y prepara los datos para su consumo, en este caso para la creación de modelos
- Fase de modelado, donde se constituye en el uso de los datos preparados y se aplicara los algoritmos y técnicas de análisis para obtener diversos modelos para una posterior comparación de la eficacia y rendimiento de cada modelo obtenido.
- Evaluación e implementación, aunque normalmente son consideradas dos fases independientes, en el caso de este proyecto lo simplificamos en única fase, donde se realizará una evaluación de los modelos obtenidos basándose en los resultados y la calidad de las predicciones.

El proceso seguido se representa en el siguiente diagrama

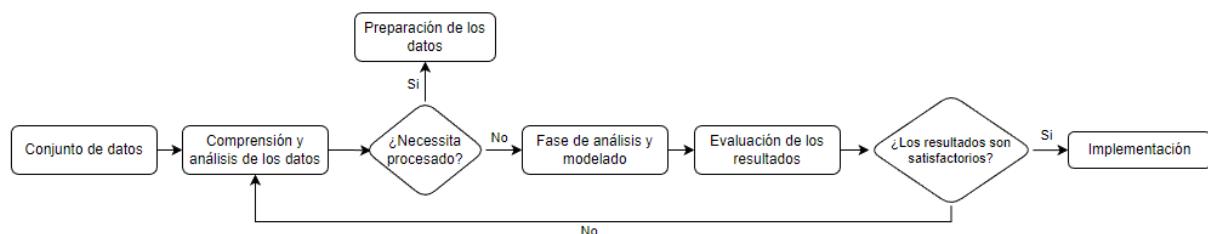


Ilustración 1. Diagrama del método seguido

1.4. Planificación del trabajo

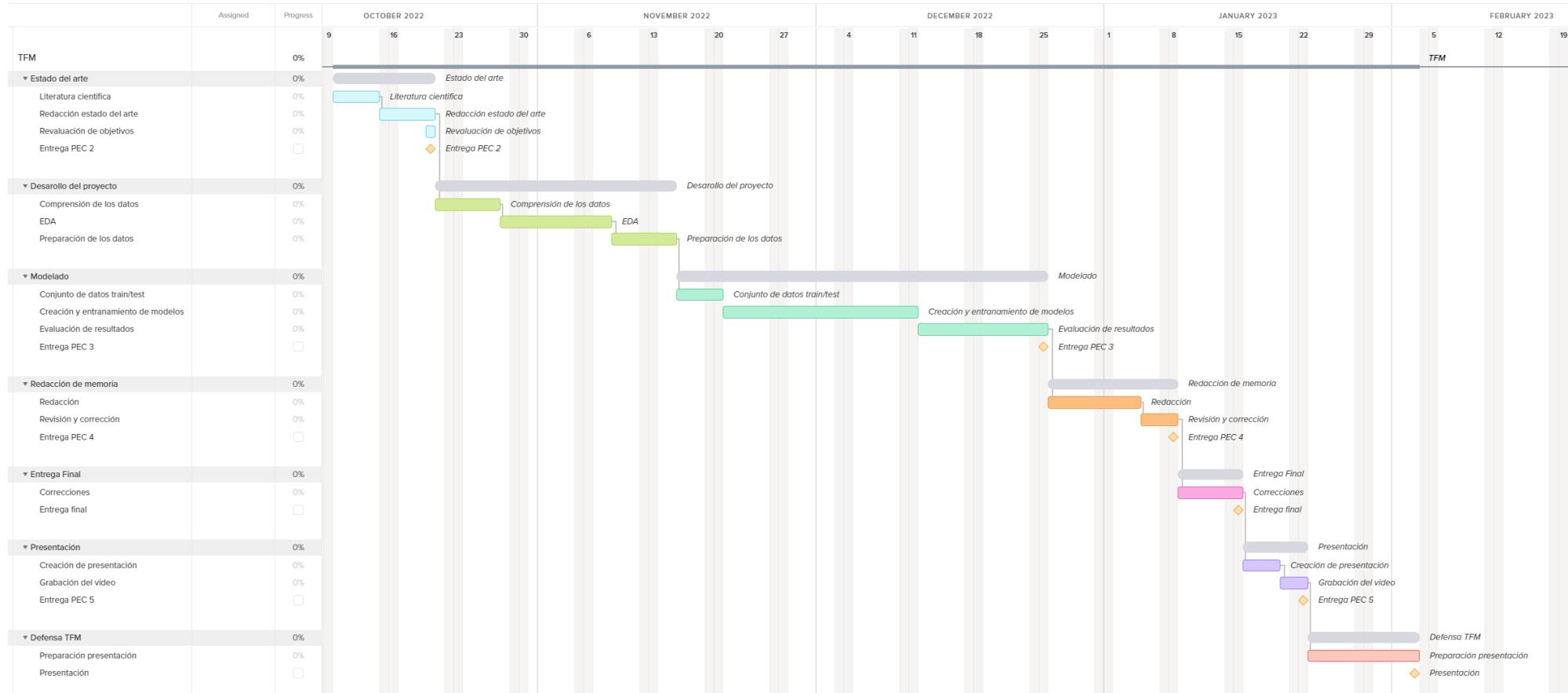


Ilustración 2. Diagrama de Gantt

Como se observa en el diagrama de Gantt, la planificación se divide en 7 grupos de tareas.

En primer lugar, se realizará el “Estudio del estado del arte” con el fin de conocer las técnicas existentes y como indica el nombre el “estado del arte” en cuanto a la predicción de fallos.

Una vez estudiado el campo se procederá, si es necesario, a una reevaluación de objetivos.

En caso contrario, se procederá al desarrollo del proyecto empezando por el estudio y comprensión de los datos a tratar, seguido de un análisis exploratorio de los mismos y seguido de la preparación de los datos para la siguiente fase, el modelado.

En la fase del modelado se aplicará diferentes técnicas de machine learning para obtener diferentes modelos las cuales evaluaremos y compararemos entre ellos para elegir cual o cuales de ellas cumplen mejor con el objetivo establecido.

Finalmente, una vez validado los resultados se procederá a la redacción de la memoria, aplicación de correcciones y la creación de una presentación para exponer el proyecto realizado.

2. Estado del arte

2.1. Contexto y actualidad

En la actualidad podemos encontrar varios tipos de mantenimiento, las principales y más conocidas son el mantenimiento preventivo, el correctivo y el predictivo, donde cada uno de estos tipos se han aparecido para la cubrir una necesidad existente en la industria.

Con el auge del interés por la Inteligencia artificial i los dispositivos IoT (*Internet of Things*), debido a estos dos factores que han facilitado la recolección de datos y a una disponibilidad inmediata de estos, han contribuido en gran medida al estudio y aplicación del mantenimiento predictivo como una solución viable para los negocios. Ejemplos como el artículo de forbes donde la CIO de United Airlines menciona que estudian los datos para mejorar el mantenimiento predictivo y experiencia de los clientes [1] o empresas como Preditec del Grupo Álava que ofrecen servicios de mantenimiento [2].

Empresas de automatización como Siemens disponen de aplicaciones para la utilización de Machine Learning e inteligencia artificial en sus sistemas de mantenimiento predictivo a través de los sistemas Scada. Este sistema se basa en los 3 conceptos de Assessment, Connectivity i Analytics, del Siemens Predictive systems [10].

Otro proyecto en este ámbito es el AI-PROFIDENT (*Artificial Intelligence for improved Production efficiency, quality and maintenance*) [11]. Este proyecto subvencionado por la Unión Europea que tiene como objetivo la aplicación de la inteligencia artificial para mejorar la eficiencia productiva, la calidad y el mantenimiento en las industrias. El proyecto es liderado por la *Universite de Lorrain*, con la colaboración de 4 centros de investigación, 2 grandes empresas y 2 pymes.

2.2. Revisión bibliográfica

Existen diferentes técnicas de aprendizaje de las cuales podemos dividirlas en dos clases diferentes según los datos que se dispone:

- Métodos supervisados, son aquellas técnicas requieren un conjunto de datos de entrada y salida etiquetados durante la fase de entrenamiento del modelo. En el ámbito industrial estos datos a menudo son conjuntos de datos obtenido de sensores después de una falla en una máquina.
- Métodos no supervisados, es el entrenamiento de modelos de datos sin procesar y sin etiquetar. Como el mismo nombre indica, estas técnicas no requieren de tanta intervención humana en comparación a los métodos supervisados.

Dentro de los métodos supervisados encontramos modelos de clasificación que pretenden discretizar las maquinas saludables de las maquinas con fallos, o determinar si es posible que se produzca un fallo de la máquina dentro de un cierto tiempo o ciclo de trabajo. Podemos encontrar trabajos como el estudio de Ramy et al [3] con la aplicación de SVM (Support Vector Machines) en la clasificación de obleas (Wafer). Casos como Li et al [4] que aplican tanto modelos SVM como arboles de decisión para predecir posibles alarmas y vagones con más probabilidades de tener fallos respectivamente.

Por otra parte, encontramos estudios, Baptista et al [5], que trata sobre la predicción de fallos de componentes de una aeronave y cuando se debe tomar medidas para sustitución de componentes o mantenimiento mediante la combinación de técnicas autoregresivas como ARMA (*Autoregressive Moving Average*) i técnicas de clasificación como KNN (*K-Nearest Neighbors*), SVM, GLM (Generalized Linear Model), NN (Neuronal Networks) y RF (Random Forest).

No únicamente se ha propuesto la utilización de ARMA sino que otros artículos proponen la utilización de modelos como el ARIMA (*Autoregressive integrated moving average*) para tener una mayor precisión en el modelo prediciendo estados futuros de los sistemas. Artículos como Eduardo et al [12] o Zeyang et al [13] han utilizado este modelo para predecir los ciclos de vida y fallas y utilizarlos como datos de entrada para un modelo supervisado y con el fin de realizar una estimación de los estados futuros del sistema.

Artículos como De Vita et al [6] o Dong et al [7] proponen el uso de aprendizaje profundo mediante uso de LSTM (*Long short-term memory*) para estudiar series de tiempo de sensores para estimar la vida útil restante (RUL) de motores.

Estudios como Han et al [9] proponen el uso de una combinación entre una red neuronal convolucional con máquinas de soporte de vectores (CNN-SVM), donde utilizan la CNN para la extracción de características y clasificarlas posteriormente con la SVM para el diagnóstico de fallas en rodamientos.

Revisando y buscando en las diversas fuentes académicas disponibles como la misma Biblioteca de la UOC, Web of Science, IEEE Xplore, Reaseach Gate, ScienceDirect i Web of Science, se ha resumido en la tabla 1 los trabajos con objetivos que se alinean al objetivo principal de este trabajo:

Artículo	Problema	Modelos/Técnica
Wafer classification using support vector machines	Aprendizaje supervisado	Máquinas de soporte vectorial (SVM)
Improving rail network velocity: A machine learning approach to predictive maintenance	Aprendizaje supervisado, clasificación. Predicción de fallo y eventos de alarmas	Máquinas de soporte vectorial (SVM) y árboles de decisión
Forecasting Fault Events for Predictive Maintenance using Data-driven Techniques and ARMA Modeling	Aprendizaje supervisado, regresión. Tiempo para el próximo fallo	Modelo ARMA, redes neuronales, K-Nearest Neighbors (K-NN), Random Forest, regresión lineal generalizada (GLM) y máquinas de soporte vectorial (SVM)
On the use of LSTM networks for Predictive Maintenance in Smart Industries	Aprendizaje no supervisado. Predicción del RUL del sistema	<i>Long short-term memory</i> (LSTM)
Life prediction of jet engines based on LSTM-recurrent neural networks	Aprendizaje no supervisado. Predicción del RUL del sistema	<i>Long short-term memory</i> (LSTM), Red neuronal recurrente (RNN),
Machine learning for predictive maintenance of industrial machines using IoT sensor data	Aprendizaje supervisado, clasificación.	Modelo ARIMA, Naive Bayes, máquinas de soporte vectorial (SVM), árboles de decisión y redes neuronales
Rolling bearing fault diagnosis with combined convolutional neural networks and support vector machine	Aprendizaje supervisado Diagnóstico y detección de fallas	Red neuronal convolucional CNN i máquinas de soporte vectorial (SVM)

Macro-level accident fatality prediction using a combined model based on arima and multivariable linear regression	Aprendizaje supervisado Identificación de elementos y predicción de accidentes.	Modelo ARIMA i modelo linear regresivos multivariante (MLR)
--	---	---

Tabla 1. Artículo, problema y técnicas usadas

3. Análisis inicial y procesamiento previo

3.1. Análisis de las fuentes de datos

El dataset utilizado para este TFM es el dataset “Microsoft Azure Predictive Maintenance proporcionado por Kaggle (https://www.kaggle.com/datasets/arnabbiswas1/microsoft-azure-predictive-maintenance?select=PdM_failures.csv).

Todos los datos disponibles son registros de cada hora durante el año 2015 hasta el 2016.

Este conjunto dispone de los siguientes archivos:

3.1.1. PdM_errors.csv

Contiene la información relativa a los errores encontrados en las máquinas durante su fase operativa, debido a que estos errores no apagan la máquina no se les considera los mismo como un fallo o rotura de máquina.

Se dispone de instancias de errores con su fecha redondeado a la hora más cercana de su detección.

Nombre del campo	Tipo	Ejemplo
datetime	Fecha	2015-01-03 07:00:00
machineID	Numérico	1
errorID	Texto	error1

Tabla 2. Campos de PdM_errors.csv

Observaciones

El campo <>datetime<> tiene el formato: YYYY-MM-DD hh:mm:ss

Total: 3919 registros.

3.1.2. PdM_failures.csv

Cada registro de este archivo representa una sustitución de componente debido a una rotura de la máquina.

Se dispone de instancias de roturas con su fecha redondeado a la hora más cercana de su detección.

Nombre del campo	Tipo	Ejemplo
datetime	Fecha	2015-01-05 06:00:00
machineID	Numérico	1
failure	Texto	comp4

Tabla 3. Campos de PdM_failures.csv

Observaciones

El campo <>datetime<> tiene el formato: YYYY-MM-DD hh:mm:ss

Total: 761 registros.

3.1.3. PdM_machines.csv

Conjunto de datos que contiene los metadatos de las máquinas, en concreto, el modelo y la edad.

Nombre del campo	Tipo	Ejemplo
machineID	Numérico	1
model	Texto	model3
age	Numérico	18

Tabla 4. Campos de PdM_machines.csv

Observaciones

Total: 100 registros.

3.1.4. PdM_maint.csv

Este archivo contiene los registros de los mantenimientos y sustituciones de componentes. Estos componentes pueden ser sustituidos bajo dos condiciones:

1. Se ha remplazado un componente durante un mantenimiento programado (Mantenimiento proactivo)
2. Se ha remplazado un componente debido a la rotura (Mantenimiento reactivo).

A diferencia de PdM_failures.csv este registro no hace diferencia si la sustitución es debido a una rotura o simplemente a un mantenimiento programado.

Nombre del campo	Tipo	Ejemplo
datetime	Fecha	2015-06-01 06:00:00
machineID	Numérico	1
comp	Texto	comp2

Tabla 5. Campos de PdM_maint.csv

Observaciones

El campo <>datetime<> tiene el formato: YYYY-MM-DD hh:mm:ss

Total: 3286 registros.

3.1.5. PdM_telemetry.csv

Conjunto de datos que contiene la media de voltage, rotación, presión y vibración de cada hora de las 100 máquinas durante el año 2015-2016

Se dispone de registros con su fecha redondeado a la hora más cercana de su detección.

Nombre del campo	Tipo	Ejemplo
datetime	Fecha	2015-01-01 06:00:00
machineID	Numérico	1
comp	Texto	comp2
volt	Numérico	176.217853015625
rotate	Numérico	418.504078221616
pressure	Numérico	113.077935462083
vibration	Numérico	45.0876857639276

Tabla 6. Campos de PdM_telemetry.csv

Observaciones

El campo <>datetime>> tiene el formato: YYYY-MM-DD hh:mm:ss

Total: 876100 registros.

3.2. Análisis exploratorio de los datos(EDA)

El análisis exploratorio de los datos o estadística descriptiva es un paso fundamental con el objetivo de comprender los datos más allá de los metadatos analizados en el análisis de las fuentes.

Este paso permitirá comprender qué tipo de datos disponemos, cual es la distribución de los mismos, la existencia de valores extremos u *outliers*, la existencia de valores nulos y finalmente las correlaciones que tienen entre estos los diferentes conjuntos. También facilitará el trabajo a la hora de realizar la selección de componentes, visualización de los mismos datos y orientar a la correcta metodología de análisis.

Para empezar, observamos si los datos cargados son los mismos descrito en el apartado anterior:

¿Cuál es el rango de fechas de los datos que disponemos?

```
print(f"Primera fecha: {telemetry_df.datetime.min()} \n")
print(f"Última fecha: {telemetry_df.datetime.max()} \n")
```

Primera fecha: 2015-01-01 06:00:00
Última fecha: 2016-01-01 06:00:00

¿Qué tamaños tienen las tablas?

```
print(f"Medida de la tabla Telemetry:\t {telemetry_df.shape} \n")
print(f"Medida de la tabla Errors:\t {errors_df.shape} \n")
print(f"Medida de la tabla Maintenance:\t {maint_df.shape} \n")
print(f"Medida de la tabla Failures:\t {failures_df.shape} \n")
print(f"Medida de la tabla Machines:\t {machines_df.shape} \n")
```

Medida de la tabla Telemetry: (876100, 6)
Medida de la tabla Errors: (3919, 3)
Medida de la tabla Maintenance: (3286, 3)
Medida de la tabla Failures: (761, 3)
Medida de la tabla Machines: (100, 3)

Ilustración 3. Fechas y medidas de los conjuntos de datos

Una vez conocido la envergadura y el alcance de los datos procedemos a revisar si existe algún valor nulo o duplicado que pueda afectar al análisis posterior:

¿Hay algún valor nulo?

```
: print(f"\"Falta algún valor en Telemetry?\t\t {telemetry_df.isnull().values.any()}\"")
print(f"\"Falta algún valor en Errors?\t\t {errors_df.isnull().values.any()}\"")
print(f"\"Falta algún valor en Maintenance?\t\t {maint_df.isnull().values.any()}\"")
print(f"\"Falta algún valor en Failures?\t\t {failures_df.isnull().values.any()}\"")
print(f"\"Falta algún valor en Machines?\t\t {machines_df.isnull().values.any()}\"")
```

Falta algún valor en Telemetry?	False
Falta algún valor en Errors?	False
Falta algún valor en Maintenance?	False
Falta algún valor en Failures?	False
Falta algún valor en Machines?	False

Ilustración 4. Exploración de valores nulos

¿Hay algún dato duplicado?

```
: print(f"Hay un total de {check_duplicate(telemetry_df, ['datetime', 'machineID','volt','rotate','pressure','vibration'])} telemetria duplicadas")
print(f"Hay un total de {check_duplicate(errors_df, ['datetime','machineID','errorID'])} errores duplicados")
print(f"Hay un total de {check_duplicate(machines_df, ['machineID','model','age'])} maquinas duplicadas")
print(f"Hay un total de {check_duplicate(failures_df, ['datetime','machineID','failure'])} fallos duplicados")
print(f"Hay un total de {check_duplicate(maint_df, [ 'datetime' , 'machineID','comp'])} mantenimientos duplicados")

Hay un total de 0 telemetria duplicadas
Hay un total de 0 errores duplicados
Hay un total de 0 maquinas duplicadas
Hay un total de 0 fallos duplicados
Hay un total de 0 mantenimientos duplicados
```

Ilustración 5. Exploración de registros duplicados

Una vez revisado los campos comunes se procede a revisar cada uno de los datasets disponibles:

3.2.1. Telemetry

Los campos de este conjunto son:

- datetime: Fecha y hora de adquisición de los datos
- machineID: Identificador de la máquina
- volt: Voltaje de la máquina detectado por el sensor
- rotate: Rotación de la máquina detectado por el sensor
- pressure: Presión de la máquina detectado por el sensor
- vibration: Vibración de la máquina detectado por el sensor

Seleccionamos la máquina 1 y vemos la evolución de las variables volt, rotate, pressure y vibration a lo largo del tiempo:

Variación de las variables en el tiempo

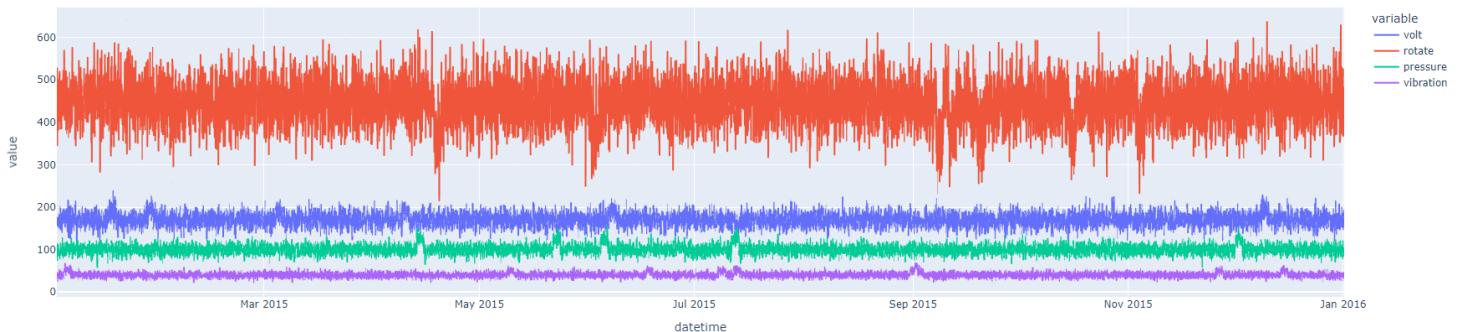


Ilustración 6. Volt, rotate, pressure y vibration de la máquina 1

De entrada, observamos en la ilustración 6 picos de valores irregulares en diferentes fechas lo que puede ser un indicio de actividad irregular de la máquina.

Por ejemplo, en la fecha del 19 de abril, vemos que hay un pico negativo en la rotación donde es considerablemente inferior a lo normal junto a la presión donde unos días anteriores tenía un valor más alto en comparación a otros días.

Por otro lado, también nos interesa saber si se considera valores extremos en conjunto ya que estos pueden ser útiles como signo indicativo de posibles roturas:

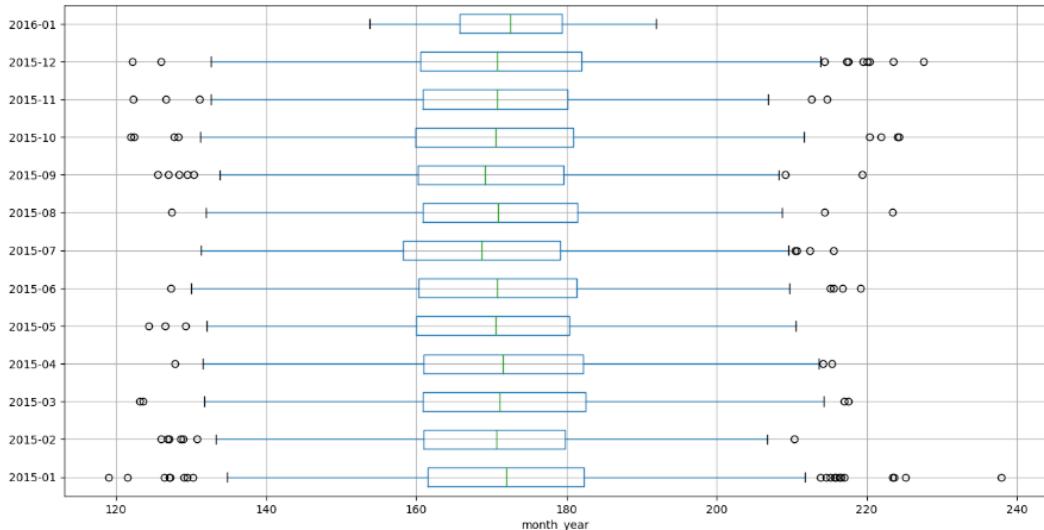


Ilustración 7. Boxplot del voltaje de la máquina 1

Como podemos observar en la ilustración 6, encontramos que en los meses existen varios registros con valores extremos, en este estudio se conservarán estos valores extremos como indicativos de roturas o posibles roturas de componentes.

Seguidamente comprobamos que distribución i rango de valores siguen las variables:

Distribución máquina 1

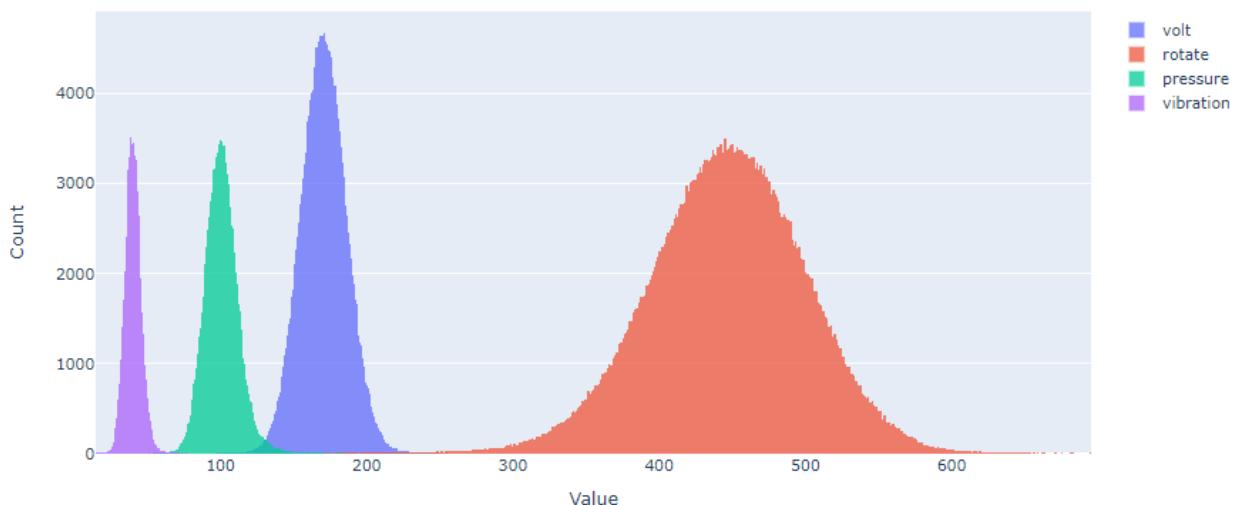


Ilustración 8. Distribución y rangos de valores

Como era de esperar cumplen con el teorema de límite central, debido a la gran cantidad de muestras que disponemos (867.100) las distribuciones de las variables tienden a una distribución normal.

Esta información puede resultar útil para realizar testeos de hipótesis o el uso de modelos de regresión.

3.2.2. Machines

Los principales campos son:

- machineID: Identificador de la máquina
- model: Modelo de la máquina
- age: Edad/años en servicio

En este conjunto de datos encontramos un total de 4 modelos diferentes en diferentes cantidades [Ilustración 8]:

```
print(machines_df.model.unique(),'\n')
print(machines_df.groupby('model').size())

['model3' 'model4' 'model2' 'model1']

model
model1    16
model2    17
model3    35
model4    32
dtype: int64
```

Ilustración 9. Tipos de modelos y cantidad por modelo

Además, la distribución de la edad de los modelos es el siguiente:

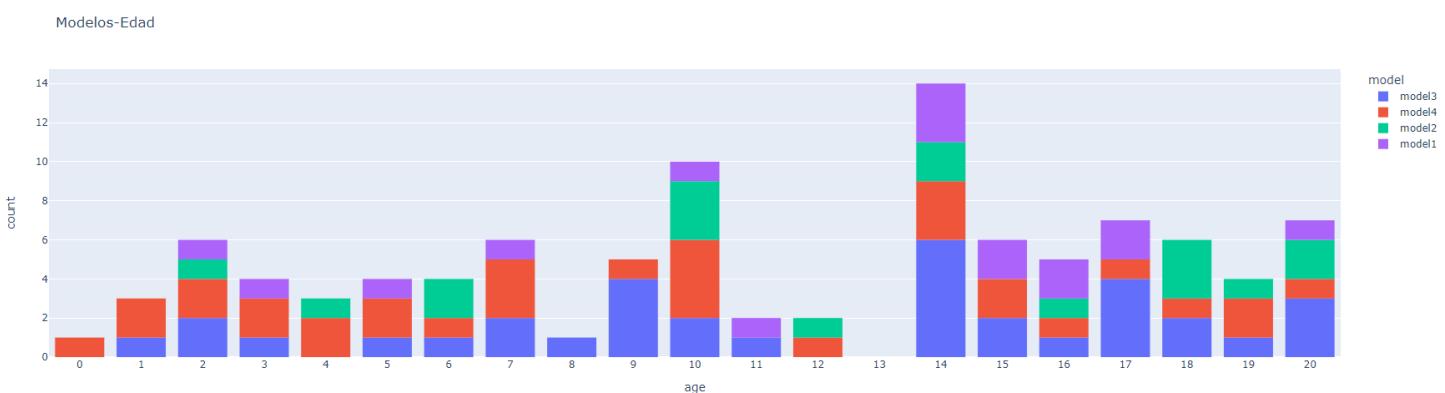


Ilustración 10. Distribución de edad de los modelos del conjunto de datos

Tenemos que la mayoría de las máquinas tienen una edad más avanzado con un máximo de 14 máquinas con 14 años de edad, y en caso contrario vamos que no se dispone de ni una máquina con una edad de 13 años.

3.2.3. Errores

Los principales campos son:

- datetime: Fecha y hora de adquisición de los datos
- machineID: Identificador de la máquina
- errorID: Identificador del error

En el conjunto de errores se ha analizado el total de errores que existen y cuál es su distribución por cada uno de los errores existentes.

```
print(f"Total de {errors_df.shape[0]} errores\n")
print(errors_df.errorID.unique(), '\n')
Total de 3919 errores
```

Ilustración 11. Total de errores generados

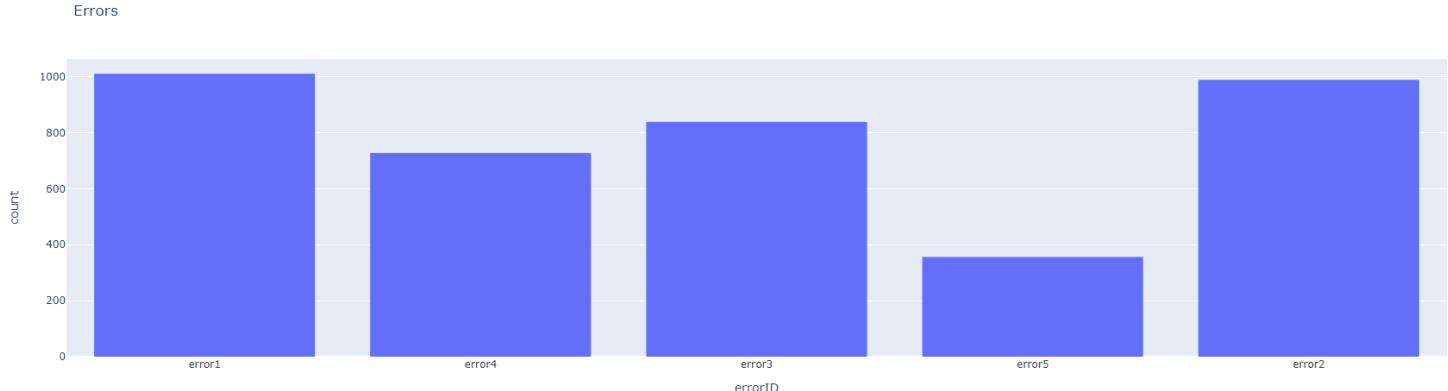


Ilustración 12. Distribución de errores por tipo de error

También se ha estudiado el número de errores generado por máquina y la evolución de estos errores a lo largo del tiempo, de este modo podemos analizar si el tiempo es un factor que pueda afectar a los errores generados por la máquina

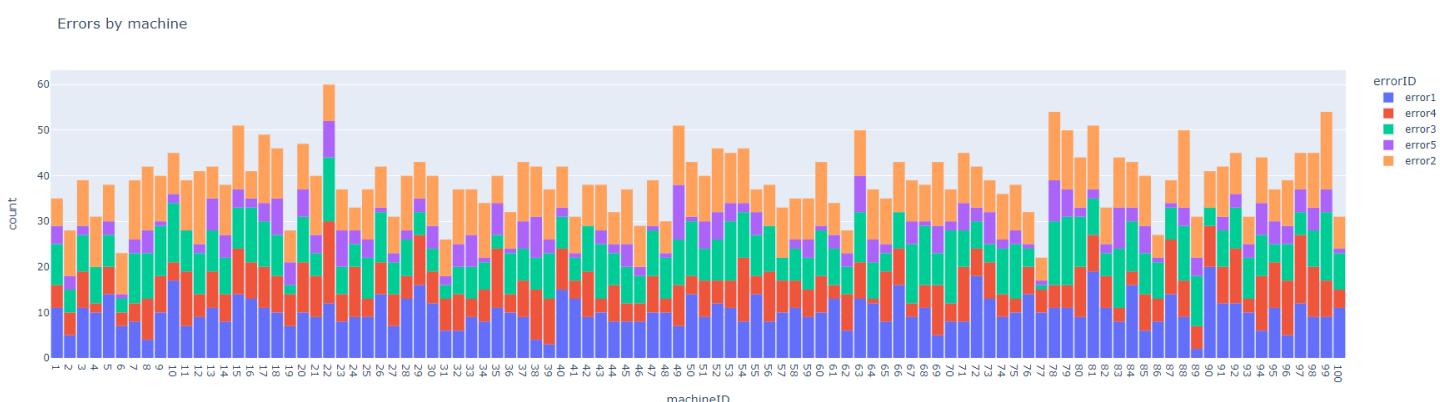
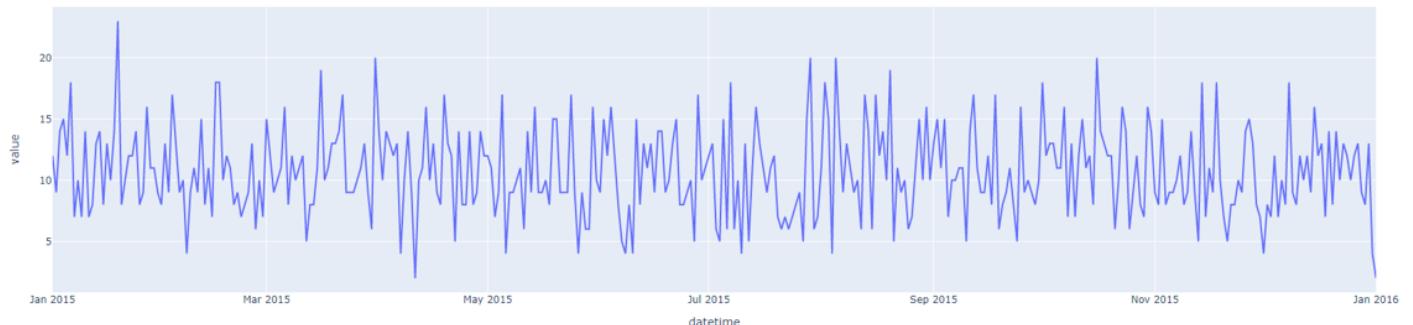


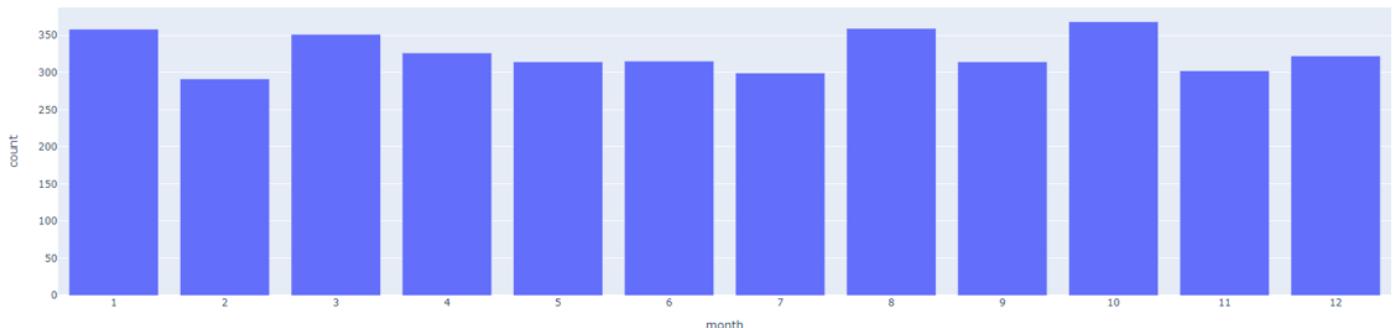
Ilustración 13. Errores por máquina

Observamos que la distribución de errores no parece tener ninguna característica que destaque, máquinas con menor edad tienden a tener una menor cantidad de errores acumulados mientras aquellas con más años tienen un total un mayor número de errores acumulados.

Variación de las variables en el tiempo



Errores por mes



Errores por día del mes

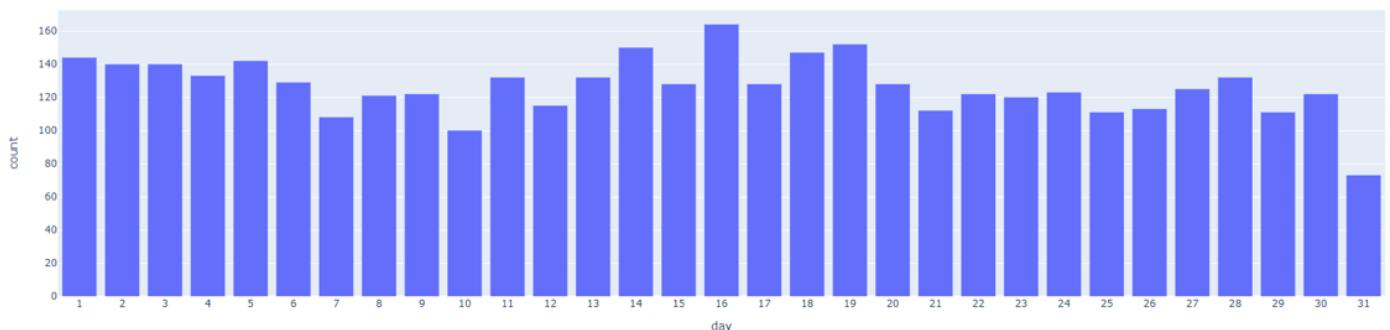


Ilustración 14. Evolución del total de errores a lo largo de un año, por cada mes y día

Analizando la evolución no parece haber signos de dependencia ni estacionalidad en los errores generados por las máquinas.

3.2.4. Fallos

Los principales campos son:

- datetime: Fecha y hora de adquisición de los datos
- machineID: Identificador de la máquina
- failure: Componente/causa del fallo

Finalmente analizamos el número de fallos registrados, los componentes afectados, su distribución sobre los tipos de componentes y las máquinas

```
print(f"Total de {failures_df.shape[0]} fallos\n")
print(failures_df.failure.unique(), '\n')

Total de 761 fallos
```

Ilustración 15. Total de fallos/roturas de componentes

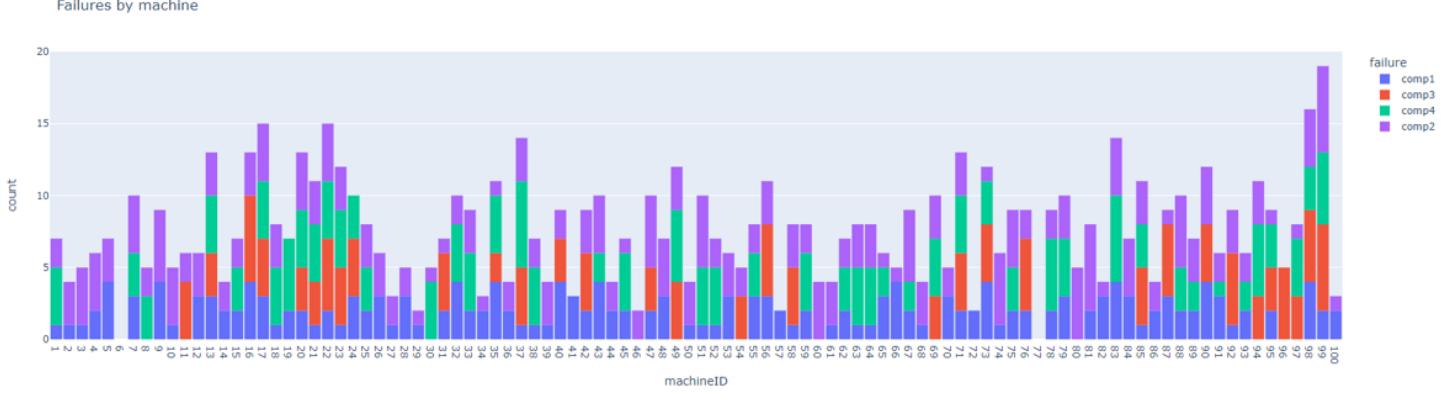
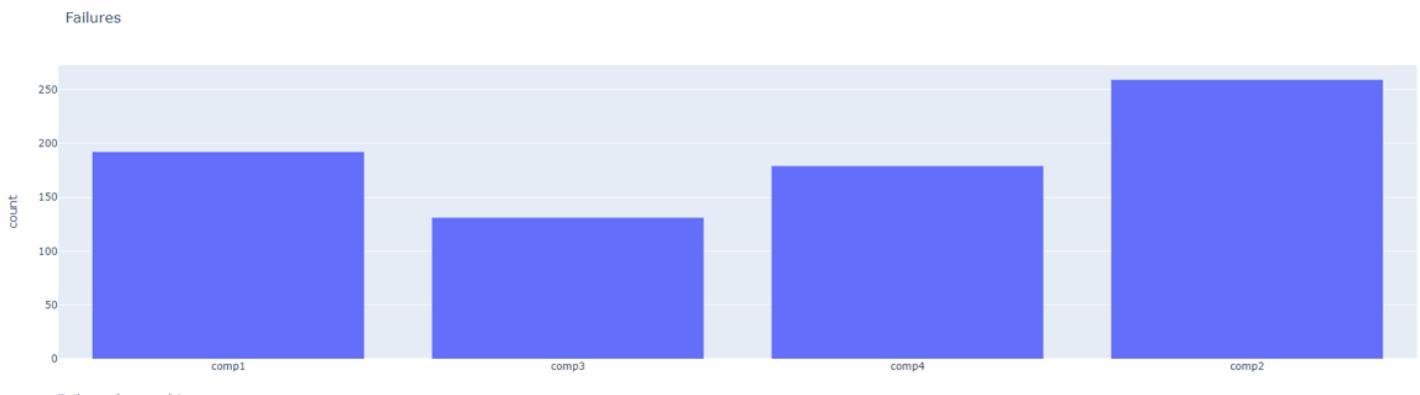


Ilustración 16. Distribución de fallos entre tipos y máquinas

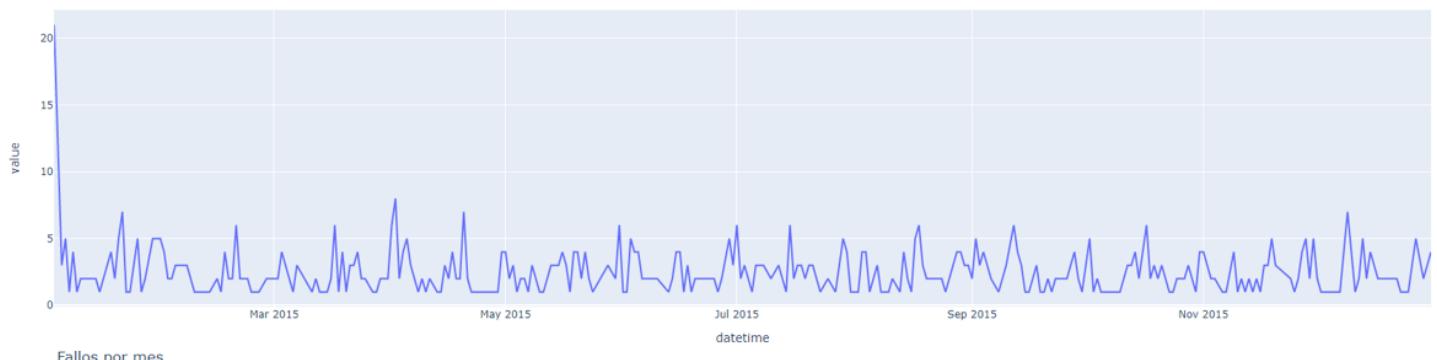
Podemos ver que en la ilustración 16 ha habido una mayor cantidad de fallos/rotura del componente 2 en comparación a los otros. Esto ya nos permite saber que, en caso de necesitar existencias de componente para poder hacer el recambio rápido, es preferible priorizar el componente 2 respecto al componente 3.

Por otro lado, vamos que la máquina 98 y 99 tienden a tener más roturas en comparación a los demás, por lo que puede ser de interés monitorizarlos y estudiar cuales son las causas que provoca que tengan un mayor número de fallos respecto a las demás máquinas.

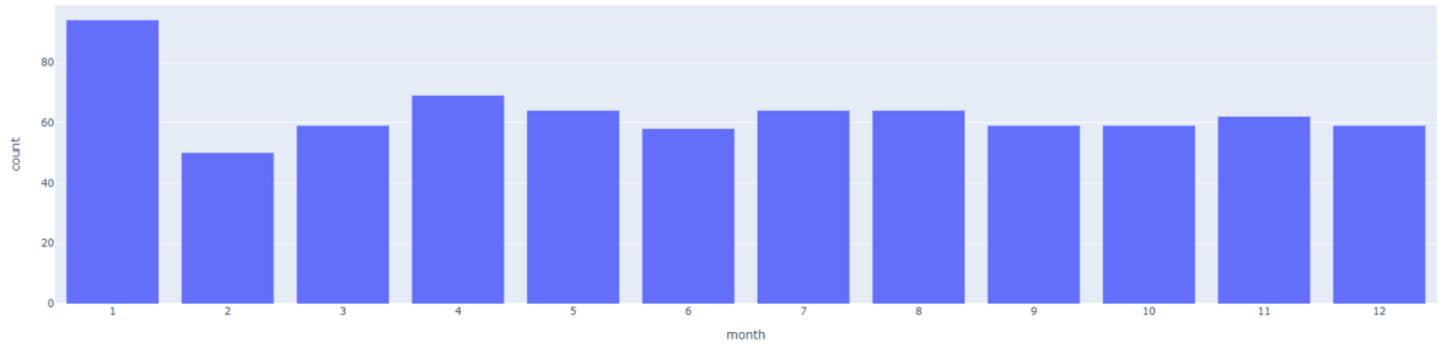
Analizando la evolución de los fallos a lo largo del año, los meses y los días. Observamos que a principios de año se ha producido muchos fallos, este caso lo eliminaremos debido a que se ha registrado todos los errores anteriores al empezar la monitorización de la máquina.

En cuanto a la distribución por día vemos que parece haber más fallos los primeros días, a mediados y últimos días del mes.

Variación de las variables en el tiempo



Fallos por mes



Fallos por dia

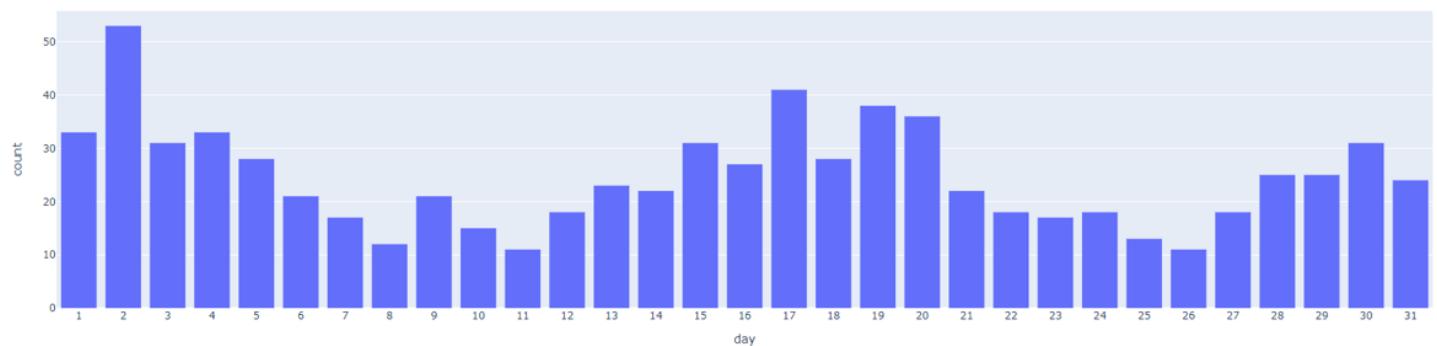


Ilustración 17. Evolución del total de fallos a lo largo de un año, por cada mes y día

3.2.5. Mantenimiento

En cuanto al mantenimiento, aunque se ha realizado el análisis exploratorio del conjunto, se ha decidido no añadirlo en esta sección.

La razón por su omisión es debido que la información del dataset se compone de reemplazos de componentes tanto cuando la máquina está en el estado de funcionamiento normal como cuando ha tenido un fallo. Por lo que se ha considerado que esta información no aporta valor al conjunto total de los datos ni al objetivo de este trabajo.

3.3. Preparación de los datos

3.3.1. Limpieza de datos y valores nulos

La etapa de la limpieza de datos y valores nulos es esencial para corregir la información del conjunto que pueden afectar al análisis de los mismos. Por lo que en general se busca estos valores nulos se aplican técnicas de imputación de valores nulos o valores ausentes, como por ejemplo la imputación de la media, la imputación de la moda o técnicas más avanzadas como la imputación mediante K-Nearest Neighbour (KNN). En el caso de este trabajo no se ha encontrado ningún valor nulo o ausente, por lo que no es necesario la aplicación de estas técnicas.

Pero como se ha comentado en el apartado 3.2.5, de todos los conjuntos de datos disponibles haremos caso omiso del conjunto de mantenimiento PdM_main.csv, debido que se ha considerado que estos datos no aportan información útil en el objetivo de este estudio.

3.3.2. Valores atípicos u outliers

En esta etapa, como indica el nombre, se centra en la corrección de aquellos valores atípicos o considerados que son extremos. En la exploración hemos encontrado valores atípicos en el conjunto telemtry y en el conjunto de fallos.

En el caso de telemetry, se ha observado que el voltaje de la máquina 1 presenta varios casos de valores atípicos a lo largo de los meses [Ilustración 7]. Aunque normalmente se procedería a normalizar o eliminar estos registros en este caso los mantenemos debido a que estos valores son precisamente el objetivo del trabajo.

En el contexto del conjunto de los datos, estos valores son indicativos de cuando una máquina ha sufrido una rotura o está a punto de sufrir una rotura mientras trabaja en un estado fuera de las condiciones usuales.

3.3.3. Codificación de las variables

Durante la exploración de los datos, conjuntos como PdM_errors.csv y PdM_failure.csv tienen únicamente un campo categórico, los errores y fallos registrados. Para facilitar el uso de estos campos se ha optado por una codificación de estos campos, con el objetivo de convertir estos valores categóricos en valores binarios para indicar la presencia del error o del fallo.

En el caso de los errores, se dispone de un conjunto donde el campo errorID indica el tipo de error registrado, mediante el uso de la función OneHotEncoder(), se ha trasformado en una tabla codificada de los distintos errores [Ilustración 18].



	datetime	machineID	errorID	error1	error2	error3	error4	error5
0	2015-01-01 06:00:00	24	error1	0	1	0	0	0
1	2015-01-01 06:00:00	73	error4	0	0	0	1	0
2	2015-01-01 06:00:00	81	error1	1	0	0	0	0
3	2015-01-01 07:00:00	43	error3	0	0	1	0	0
4	2015-01-01 08:00:00	14	error4	0	0	0	1	0

Ilustración 18. Codificación de la tabla errors

Se ha aplicado el mismo proceso a la tabla de fallos.

3.3.4. Unificación de los datos

En este proceso se ha unificado los distintos conjuntos de datos en un único conjunto. El objetivo de esta unificación es disponer y relacionar todos los registros de cada una de las máquinas en un solo conjunto de registros, de este modo, se dispone de un único dataset para utilizar posteriormente en el entrenamiento de los modelos.

Para unificar estos datos, principalmente se ha utilizado el identificador de la máquina (machineID) y la fecha de los registros (datetime), dando como resultado:

	datetime	machineID	volt	rotate	pressure	vibration	error1	error2	error3	error4	error5	comp1	comp2	comp3	comp4	failure	model	age
0	2015-01-01 06:00:00	1	176.217853	418.504078	113.077935	45.087686	0	0	0	0	0	0	0	0	0	0	model3	18
1	2015-01-01 06:00:00	2	176.558913	424.624162	76.005332	43.767049	0	0	0	0	0	0	0	0	0	0	model4	7
2	2015-01-01 06:00:00	3	185.482043	461.211137	87.453199	28.216864	0	0	0	0	0	0	0	0	0	0	model3	8
3	2015-01-01 06:00:00	4	169.710847	463.646727	95.929877	38.400372	0	0	0	0	0	0	0	0	0	0	model3	7
4	2015-01-01 06:00:00	5	165.082899	452.283576	84.591722	40.298803	0	0	0	0	0	0	0	0	0	0	model3	2

Ilustración 19. Dataset unificado

3.3.5. Creación de las variables objetivo

Finalmente, partiendo del dataset unificado es necesario crear una variable objetivo que predecir.

Aunque se dispone de los campos comp1, comp2, comp3, comp4 y failure, que indican que cuando hay una rotura, el objetivo de este estudio es predecir cualquier tipo de fallo, independientemente del componente. Para ello, se ha eliminado los campos de componentes dejando únicamente el failure, ya que este campo muestra cuando ha habido un fallo en la máquina.

Posteriormente se ha creado una nueva variable llamado RUL (*Remaining Useful Lifetime*) utilizado en trabajos como De Vita et al [6] o Dong et al [7]. El objetivo de esta variable será indicar el tiempo restante de la máquina hasta su próximo fallo.

Si observamos la variación del voltaje de la máquina 79 y se marca los momentos que ha tenido una rotura se obtiene el siguiente gráfico:

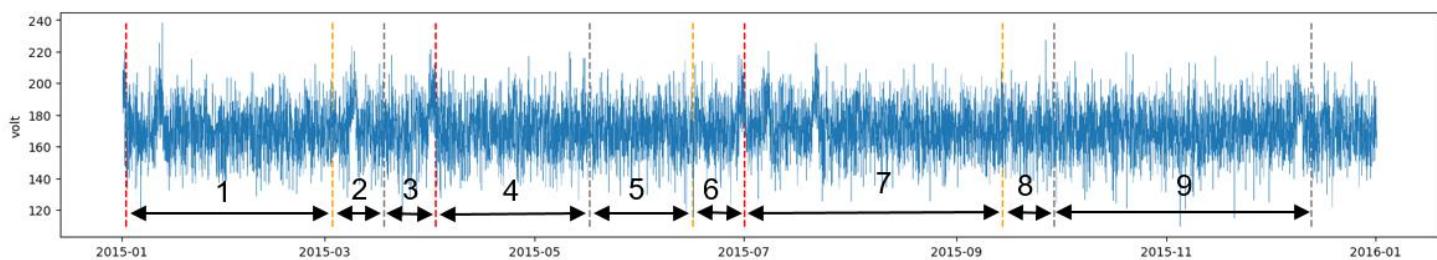


Ilustración 20. Gráfico de las RULs

En la ilustración 20, se puede observar distintos intervalos entre roturas, este intervalo de tiempo es lo que determina como ciclo de trabajo de la máquina, cada ciclo de trabajo tiene su contraparte conocido como RUL de la máquina.

Como ejemplo, si se considera el intervalo 1, la máquina está funcionando estado normal desde enero hasta principios de marzo, si consideramos que cada día del mes es una unidad del RUL, obtenemos un total de 59 días hasta marzo. A medida que los días pasan el valor de RUL va disminuyendo del 59, primer día de enero, hasta llegar a 0 lo cual nos indica que ese día ha habido una rotura de la máquina.

Por lo tanto, esta variable servirá como indicador para predecir el tiempo que le falta una máquina hasta su próxima rotura, por lo que el técnico podrá planificar mantenimientos a medida que el valor de RUL se acerca a 0.

Basado en los registros de los conjuntos de datos, se ha considerado una unidad de RUL cada hora del ciclo de trabajo, usando la referencia el ejemplo anterior, los 59 días se convierten en 1416 horas, es decir, un RUL decreciente basado en horas restantes.

	datetime	machineID	volt	rotate	pressure	vibration	error1	error2	error3	error4	error5	model	age	RUL
0	2015-01-01 06:00:00	79	194.167651	535.302327	84.071434	35.654640	0	0	0	0	0	model3	14	21
1	2015-01-01 07:00:00	79	180.989438	429.031142	96.245928	37.163490	0	0	0	0	0	model3	14	20
2	2015-01-01 08:00:00	79	208.452257	374.095525	106.340279	37.617635	0	0	0	0	0	model3	14	19
3	2015-01-01 09:00:00	79	192.161379	438.784836	94.309427	47.383279	0	0	0	0	0	model3	14	18
4	2015-01-01 10:00:00	79	207.822427	391.115326	103.079753	39.579032	0	0	0	0	0	model3	14	17
5	2015-01-01 11:00:00	79	193.647095	484.526874	95.397495	32.340949	0	0	0	0	0	model3	14	16
6	2015-01-01 12:00:00	79	201.603444	318.811141	110.102601	38.413517	0	0	0	0	0	model3	14	15
7	2015-01-01 13:00:00	79	213.373747	489.746066	105.192482	31.554710	0	0	0	0	0	model3	14	14

Ilustración 21. Conjunto de datos con su RUL por cada ciclo de trabajo

3.3.6. Dimensionalidad de los datos

Durante el análisis exploratorio disponemos de más de 876.100 registros, registros de todas las máquinas cada hora del año, para reducir el nombre de registro con el que trabajar seleccionaremos aquellas máquinas que consideremos parecidas, para esta selección se ha basado en el modelo y la edad de la máquina. En el apartado 3.2.2, concretamente la ilustración 10, hemos visto que existen varias máquinas del modelo 3 y con una edad de 14 años.

Debido la falta de información si una máquina es idéntica a otra o tienen la misma función, se ha decidido utilizar las máquinas con las especificaciones del modelo 3 y edad de 14 años con el fin de considerarlos como si fuesen máquinas que trabajan en un entorno y procesos similares entre ellos.

Seguido de la reducción de registros, se ha realizado una reducción de campos, ya que información como la edad y el modelo de la máquina ya no son necesarios después de la selección de registros, realizado anteriormente.

Una vez obtenida este conjunto se ha realizado analizado las correlaciones:



Ilustración 22. Correlaciones de los campos

Como se observa parece que las dos únicas correlaciones que podríamos considerar fuertes con el campo RUL son el rotate y pressure. Por lo que, si se optase por una

reducción de dimensionalidad dejando a las únicas variables correlacionadas fuertemente con RUL, podría ser variables insuficientes para el entrenamiento.

Como último paso, de las máquinas seleccionadas, se ha hecho un análisis y normalización de los ciclos de trabajo. Se ha seleccionado cada una de los ciclos de trabajos de las máquinas y se ha seleccionado aquellos ciclos de trabajo que cumpliesen un mínimo de 30 días trabajados antes de su rotura.

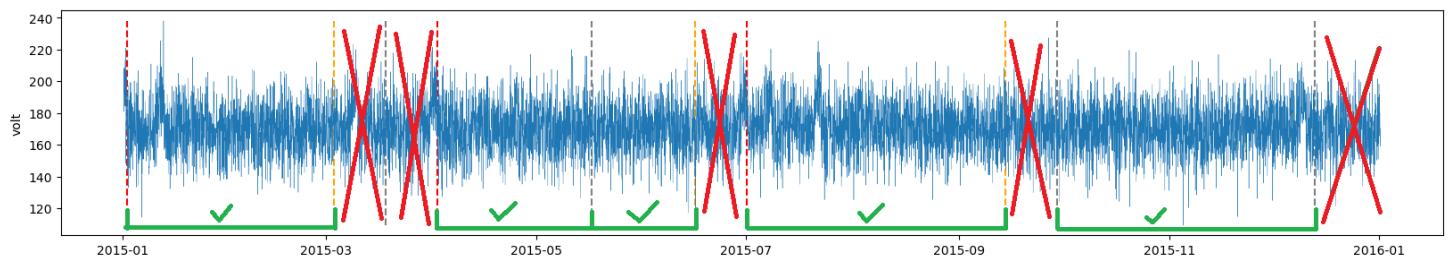


Ilustración 23. Ejemplo visual de la selección de ciclos

De este modo se dispone de dos datasets finales, una con todos los ciclos de trabajo completo de la máquina y otra solamente con ciclos de trabajos normalizados.

3.3.6.1. Dimensionalidad (segunda iteración)

Debido a los resultados de la primera con los datos de la primera iteración se ha realizado una nueva iteración para mejorar los datos de entrada de los modelos.

En esta una nueva iteración se ha optado a un aumento de la dimensionalidad basado en las variables existentes, como por ejemplo la creación de la media de voltaje, rotación, presión y vibración de las 3 y 24 horas anteriores. También se ha añadido una acumulación de los errores, que se restablece cada nuevo ciclo de trabajo, con el fin de llevar un mejor control de problemas detectado en la máquina previamente hasta su rotura.

Finalmente, también se ha creado una nueva variable binaria basada en el campo RUL, llamado State. Esta nueva variable servirá como variable objetivo para los modelos de clasificación. El valor de este estado es 0 cuando la máquina trabaja con normalidad y 1 cuando este empieza a trabajar en condiciones anormales. La creación de esta variable ha sido basada en la variable objetivo RUL, de modo que si queremos considerar que trabaja en estado de peligro (estado 1) cuando le queda pocas horas hasta la rotura, se le asignará un 1 según el rango de horas que se considere cerca de la rotura.

El conjunto de datos final es el siguiente:

	datetime	machineID	volt	rotate	pressure	vibration	error1	error2	error3	error4	error5	rotate_24h_mean	pressure_24h_mean	vibration_24h_mean	error1_count	error2_count	error3_count	error4_count	error5_count	RUL	State
745	2015-02-01 07:00:00	79	171.585268	480.714388	123.036718	36.549834	0	0	0	0	...	461.654747	101.428542	39.133370	0	0	1	0	0	719	0
746	2015-02-01 08:00:00	79	180.671569	401.104200	98.973284	35.288739	0	0	0	0	...	454.048837	101.542633	38.909825	0	0	1	0	0	718	0
747	2015-02-01 09:00:00	79	182.152843	462.262032	89.172008	43.614592	0	0	0	0	...	450.610435	101.558032	39.238626	0	0	1	0	0	717	0
748	2015-02-01 10:00:00	79	187.582362	421.302176	107.973283	42.696285	0	0	0	0	...	449.933323	101.651752	39.576371	0	0	1	0	0	716	0
749	2015-02-01 11:00:00	79	183.456793	354.912587	113.297576	38.556803	0	0	0	0	...	443.197615	101.948484	39.457171	0	0	1	0	0	715	0

Ilustración 24. Conjunto de datos, segunda iteración.

Con este nuevo conjunto de datos obtenemos una matriz de correlaciones más variables con correlaciones fuertes con las variables objetivo:

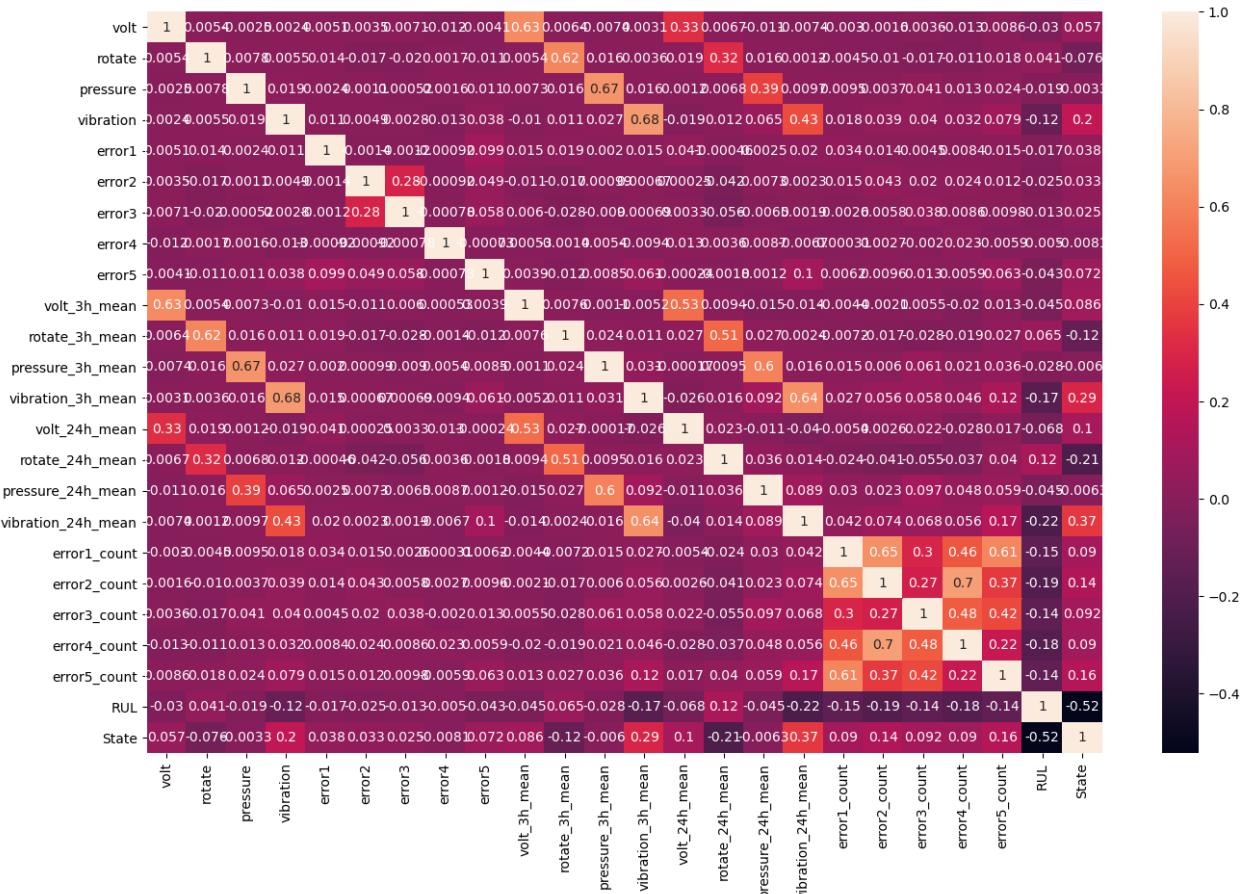


Ilustración 25. Matriz de correlaciones, segunda iteración.

3.3.7. Normalización de los datos

Como es conocido, la mayoría de los modelos de aprendizaje automático aprenden mapeando los puntos de los datos de entrada hasta la salida del mismo. Las distribuciones de los puntos pueden ser muy grandes entre cada uno de las variables como hemos podido observar en la distribución de las variables de Telemetry [Ilustración 8] o el conjunto de datos [Ilustración 24]. Como mayor son las diferencias entre dos puntos de datos mayor es la incertidumbre de los resultados obtenidos del modelo. Por otro lado, otros modelos otorgan pesos a las variables de acuerdo con los datos de entradas e inferencias para la salida. Si la diferencia entre los datos es demasiada alta el modelo deberá proporcionar un mayor peso a los y en resultados finales, el modelo suele ser inestable produciendo resultados poco precisos.

Para evitar estos problemas en el conjunto de datos se le ha aplicado una normalización escalando sus valores entre 0 y 1. Dando como resultado el conjunto de la siguiente ilustración:

	volt	rotate	pressure	vibration	error1	error2	error3	error4	error5	volt_3h_mean	...	volt_24h_mean	rotate_24h_mean	pressure_24h_mean	vibration_24h_mean	error1_count	error2_count	error3_count	error4_count	error5_count	RUL
0	0.384246	0.635590	0.287496	0.550316	0.0	0.0	0.0	0.0	0.0	0.319571	...	0.370829	0.428252	0.398337	0.219217	0.571429	0.500000	0.0	0.333333	0.333333	1.000000
1	0.196252	0.595322	0.298321	0.255010	0.0	0.0	0.0	0.0	0.0	0.214811	...	0.341286	0.432249	0.388561	0.202982	0.571429	0.500000	0.0	0.333333	0.333333	0.998609
2	0.395865	0.468667	0.396966	0.388182	0.0	0.0	0.0	0.0	0.0	0.188577	...	0.332091	0.419061	0.387892	0.197786	0.571429	0.500000	0.0	0.333333	0.333333	0.997218
3	0.487156	0.700909	0.545746	0.547169	0.0	0.0	0.0	0.0	0.0	0.235855	...	0.322535	0.439945	0.394677	0.210147	0.571429	0.500000	0.0	0.333333	0.333333	0.995828
4	0.479313	0.592491	0.348040	0.421442	0.0	0.0	0.0	0.0	0.0	0.365896	...	0.326367	0.442053	0.381300	0.209606	0.571429	0.500000	0.0	0.333333	0.333333	0.994437

Ilustración 26. Conjunto de datos normalizados con MinMaxScaler()

4. Análisis y modelado

La naturaleza del problema puede ser compleja debido a la necesidad de una estimación del RUL, esta estimación puede verse determinada por varios sensores o variables. En este caso, el uso de técnicas de aprendizaje automático puede ser muy útil para una comprensión de correlaciones entre los diversos datos de entrada.

El punto de partida usado para resolver el problema ha sido el planteamiento de dos preguntas:

¿Cuál es RUL restante de la máquina hasta su próxima rotura?

Con el fin de responder esta pregunta el objetivo pasa a ser encontrar el valor de la RUL en cada momento de la máquina, por lo tanto, se está buscando un valor numérico.

Métodos utilizados para obtener una predicción numérica son el uso de **modelos de regresión** como Linear Regression, SVM Regressors, Random Forest Regressor, Gradient boosting Regressor, XGBoosting Regressor y LSTM. Estos modelos han sido seleccionados debido a la posibilidad de utilizar los mismos con el fin de utilizarlos como modelos de clasificación.

¿Fallará la máquina en los próximos 3 días?

A diferencia de la pregunta anterior, en este caso, no existe la necesidad de encontrar la RUL de la máquina, sino que se busca el estado de la máquina, como se ha descrito anteriormente en el apartado 3.3.5 Creación de la variable objetivo, se creó una variable llamada State. Esta variable binaria, servirá como resultado para definir si la máquina fallará en los próximos días, para ello, solamente es necesario que la variable tenga un valor de 0 (estado “Normal”) cuando la RUL sea mayor de 72, por otro lado, se le asignara el valor 1 (estado “Warning”) cuando este tenga un RUL igual o menor a 72.

Partiendo de este planteamiento podemos aplicar **modelos de clasificación** para obtener el estado de la máquina según los datos de entrada. Para poder hacer una comparativa posterior utilizaremos modelos muy similares a los de regresión, pero en su versión de clasificador.

La razón que se ha seleccionado los modelos mencionados ha sido en base a la revisión bibliográfica realizada, estos modelos son los que mejores resultados han obtenido estudios similares a este. Entre las técnicas avanzadas de machine learning encontramos 3 modelos populares, *Recurrent Neuronal Network (RNN)*, *Convolutional Neuronal Network (CNN)* y una variante de la RNN conocido como *Long Short Term Memory (LSTM)*.

Los modelos CNN son un tipo de redes neuronales para procesar datos con una tipología cuadriculada, como las imágenes. Estos son muy útiles para el procesado de imagen como la visión por computación, reconocimiento facial o clasificación de imágenes. Esto es debido a la aplicación de la operación de convolución que aplica esta red, aunque el nombre es usado por convención, ya que realmente lo que se aplica es una operación de *cross-correlation*. Esta convolución es la aplicación de una función de filtro o kernel (función g) aplicado sobre la entrada de los datos (la variable f de la ecuación) dando como resultado un mapa de características o *feature map* como salida (s).

$$s(t) = (f * g)(t) \sum_{\tau=-\infty}^{\infty} f(\tau)g(t - \tau)$$

Como se ha mencionado anteriormente la tipología de los datos de entrada es un vector multidimensional por lo que el *kernel* también será un vector multidimensional de parámetros. Debido por la naturaleza de los datos redes convolucionales se aplica la convolución sobre más de un eje.

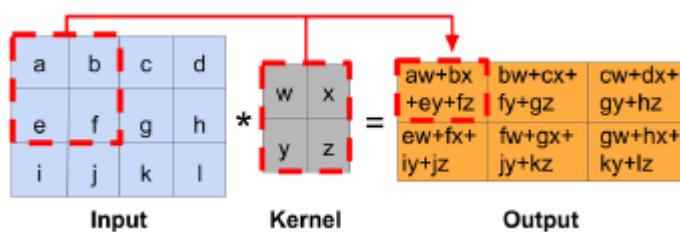


Ilustración 27 Ejemplo ilustrado de la convolución. Fuente. Bengio, Goodfellow y Courville (2016)

Esta operación resulta muy útil para el cálculo entre los datos de entrada con el *kernel* permitiendo obtener una capa de datos simplificada con las características principales de los mismos datos de entrada, siendo ideal para la extracción de características sobre las imágenes.

Los modelos RNN usan el concepto de recurrencia para generar una salida también llamado activación, la red usa la entrada actual junto la salida o activación de la iteración anterior para generar una nueva activación (conocido como *feed-forward*), en pocas palabras, los modelos RNN usan este principio como una “memoria” para generar la salida. Este tipo de conexiones pueden ser muy útiles para tratar secuencias de datos, como el caso de las series temporales o tratamiento de textos.

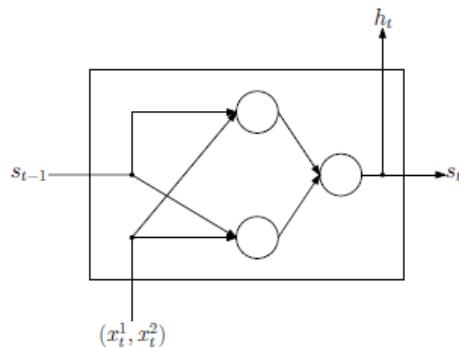


Ilustración 28. Ejemplo de una celda de la red recurrente Fuente: Bosch, Casas y Lozano (2019) [16]

Finalmente tenemos la LSTM, una variación de la RNN. Este modelo cubre una limitación que tiene las RNNs, cuando se dispone varias celdas RNN estas parten del estado anterior de la celda inmediata por lo que se aplica una función de activación para cada una de las celdas acumulando el efecto de las funciones de activaciones hasta obtener un resultado final. Este efecto se entiende que la RNN tiene una memoria, pero a corto plazo debido a que solo tiene en cuenta el estado anterior.

Para suplir esta necesidad entran la red LSTM que no solo tiene en cuenta el estado anterior inmediato sino también otros estados anteriores relevantes para la predicción.

Se puede encontrar una explicación más profundizada sobre una celda LSTM en el apartado 4.2.7 Long Short Term Memory (LSTM), pero en pocas palabras, esta variante permite no sólo tener una memoria a corto plazo como las RNNs sino que también se dispone de una memoria a largo plazo. Siendo ideal para series temporales como el problema planteado en este trabajo.

Por esta razón y estudios anteriores, se ha seleccionado el modelo LSTM como modelo óptima para la detección de roturas y obtención del RUL de una máquina.

4.1. Creación del conjunto entrenamiento y pruebas

Una vez determinado los modelos que se utilizarán es la creación de los conjuntos de datos que utilizaran estos modelos, estos son los dataset de *train* y de *test*. Con el fin de reducir el tiempo de entrenamiento se ha utilizado el conjunto de datos con los ciclos de trabajos normalizados a 30 días, es decir, RUL máximo de 720 horas antes de la rotura.

Partiendo del dataset creado en el apartado 3.3 Preparación de los datos, realizaremos una partición aleatoria del 25% usando la función *train_test_split* de la librería scikit-learn. Esta función devolverá 4 conjuntos basados en el conjunto de entrada que le hemos dado, de estos 4 conjuntos podemos los podemos separar en los dos grupos *train* y *test* mencionados anteriormente.

Los dos conjuntos de *train* componen el 75% de los datos iniciales, las cuales serán utilizados para el entrenamiento del modelo.

En el caso del modelo LSTM este requiere de un input de datos diferente a los otros modelos, este requiere que los datos de entrada tengan una forma tridimensional compuesta por las muestras conjunto total de 20.808 muestras (restando los primeros 72 pasos de tiempo); pasos de tiempo de 72 horas (equivalente a 3 días); y finalmente, las características, que son los 22 campos del conjunto.

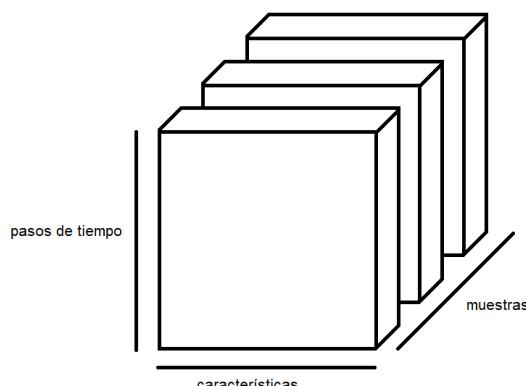


Ilustración 29. Ejemplo visual de los datos de entrada

En cuanto al resto de conjuntos, los conjuntos de test componen el 25% de los registros y será utilizados para las pruebas de validación de los modelos obtenidos mediante el conjunto de entrenamiento

4.2. Modelos

La regresión es una de las tareas de aprendizaje inductivo que ha sido ampliamente estudiada y utilizada, el objetivo principal consiste en asignar diferentes valores numéricos en instancias de un dominio, estos pueden ser descritos por un conjunto de atributos tanto discretos o de valor continuo.

Para ello hemos utilizado los siguientes modelos:

4.2.1. Linear Regression

El modelo de regresión lineal permite construir un modelo para explicar la relación entre variables, el objetivo es explicar el comportamiento de una variable, Y, que se le denomina variable explicada o variable dependiente, a partir de otra variable X, que se denomina variada explicativa o variable independiente.

Este algoritmo muestra una relación lineal entre las variables descritas anteriormente sobre un diagrama de dispersión, la representación en puntos en el espacio de las variables. Posteriormente se busca una relación lineal entre estos variables con el fin de encontrar la recta de regresión que define esta relación.

```
model_lr = LinearRegression().fit(X_train, y_train)
pred_lr = model_lr.predict(X_test)
```

Ilustración 30. Modelo regresión lineal

4.2.2. Logistic Regression Binomial

Los modelos de regresión logísticos binomial son modelos de regresión que permiten estudiar si una variable binomial depende o no de otra u otras variables, no necesariamente binomiales.

Este modelo se caracteriza por la predicción según la probabilidad de éxito representado por p , y una probabilidad de fracaso que se representa con q . La suma de estas dos probabilidades debe dar como resultado 1.

Este caso similar a la función *softmax*, donde se obtiene una probabilidad de éxito y una de fracaso, y según qué probabilidad sea mayor se le es asignado el resultado.

En nuestro caso de estudio, se usa este modelo para poder clasificar si el estado de la máquina es “Normal” (estado 0) o “Warning” (estado 1).

En la experimentación se le ha aplicado la función de GridsearchCV() de la librería scikit-learn para encontrar los mejores hiperparámetros con el fin de obtener la mayor precisión posible. Cabe decir que no se le hecho un estudio con todos y cada uno de los parámetros ya que el tiempo necesario se incrementa exponencialmente debido a la gran cantidad de combinaciones que se debe realizar.

Logistic Regression

```
: parameters = {'solver':['newton-cg', 'lbfgs', 'liblinear'],'C':[0.001, 0.01, 0.1, 1, 10]}
grid_search = GridSearchCV(LogisticRegression(), parameters, verbose=0)
grid_search.fit(X_train, y_train)
grid_df = pd.DataFrame(grid_search.cv_results_['params'])
grid_df['mean_test_score'] = grid_search.cv_results_['mean_test_score']
grid_df['rank'] = grid_search.cv_results_['rank_test_score']
grid_df.sort_values(by='rank').head()

:   C      solver  mean_test_score  rank
3  0.010  newton-cg        0.938560    1
4  0.010      lbfgs        0.938560    1
5  0.010    liblinear       0.938312    3
0  0.001  newton-cg        0.938015    4
1  0.001      lbfgs        0.938015    4
```

Ilustración 31. Búsqueda de los mejores hiperparámetros de la regresión logística

Como observamos los hyperparámetros con mejores resultados con empate:

C = 0.01

solver = newton-cg / lbfgs

```
model_lr = LogisticRegression(solver='newton-cg', C=0.01).fit(X_train, y_train)
pred_lr = model_lr.predict(X_test)
```

Ilustración 32. Modelo de regresión logístico

4.2.3. Support-vector machiens (SVM)

Las máquinas de soporte vectorial (support-vector machines, SVM en adelante) es un algoritmo de aprendizaje supervisado capaz de resolver problemas tanto lineales como no lineales.

El enfoque de las SVM se basa en la minimización del riesgo estructural, de modo que se busca construir modelos que estructuralmente tengan el mínimo riesgo posible de cometer errores. Su principal modo de trabajo es la construcción de un hiperplano (separador lineal) que sea capaz de dividir los puntos, de los datos de entrada, con la máxima distancia posible entre el hiperplano y los puntos generando los diferentes grupos.

4.2.3.1. SVM Regressor

En el caso del modelo de regresión de las SVM se ha utilizado el modelo base para su entrenamiento:

```
model_svm = svm.SVR().fit(X_train, y_train.to_numpy().ravel())
pred_svm = model_svm.predict(X_test)
```

Ilustración 33. Modelo de regresión SVM

4.2.3.2. SVM Classifier

Con la versión de clasificación se ha realizado la búsqueda de los hiperparámetros dando el mejor resultado los siguientes valores:

SVM

```
parameters = {'kernel': ['poly', 'rbf', 'sigmoid'], 'C':[0.001, 0.01, 0.1, 1, 10]}

grid_search = GridSearchCV(svm.SVC(), parameters, verbose=0)
grid_search.fit(X_train, y_train)
grid_df = pd.DataFrame(grid_search.cv_results_['params'])
grid_df['mean_test_score'] = grid_search.cv_results_['mean_test_score']
grid_df['rank'] = grid_search.cv_results_['rank_test_score']
grid_df.head().sort_values(by='rank').head()
```

C	kernel	mean_test_score	rank
3	0.010	0.904398	1
0	0.001	0.899826	2
1	0.001	0.898611	3
2	0.001	0.898611	3
4	0.010	0.898611	3

Ilustración 34. Búsqueda de los mejores hiperparámetros de la SVM

Como observamos los hiperparámetros con mejores resultados con empate:

C = 0.01

solver = poly

```
model_svm = svm.SVC(kernel='poly', C=0.01, probability=True).fit(X_train, y_train)
pred_svm = model_svm.predict(X_test)
```

Ilustración 35. Modelo de clasificación SVM

4.2.4. Random Forest

Basado en los clasificadores de árboles de decisión junto la utilización del muestreo de los elementos del conjunto original de entrenamiento junto la de sus variables, obtenemos este clasificador combinado.

Este algoritmo en la práctica habitual consiste en la generación de versiones diferentes del conjunto de entrenamiento usado con un muestreo con reemplazo, conocido como *bagging*. En el proceso de construcción de un árbol de decisión selecciona aleatoriamente un subconjunto de los datos totales, esto permite medir la importancia relativa de cada variable mientras se estima el error cometido por la clasificación cuando se altera la variable, de modo que permuta aleatoriamente los valores en el conjunto de test. En el caso que al permutar la variable hace aumentar el error, eso significa que la variable tiene mayor peso en el problema a resolver.

4.2.4.1. Random Forest Regressor

Similar al caso anterior con las SVM, se ha utilizado la función base represora del random forest, en este caso se considera como clases individuales cada uno de los valores de la RUL generando un enorme conjunto de árboles que cada hoja de los arboles equivalen a un valor de RUL

```
model_rf = RandomForestRegressor().fit(X_train, y_train.to_numpy().ravel())
pred_rf = model_rf.predict(X_test)
```

Ilustración 36. Modelo de regresión Random Forest

4.2.4.2. Random Forest Classifier

Para la clasificación se ha realizado de nuevo la búsqueda de los mejores hiperparámetros con tal de obtener un modelo lo máximo de preciso para clasificar el estado de la máquina y obteniendo los siguientes resultados:

Random Forest Classifier

```
parameters = {'max_depth': [6,7,8,9,10,11,12], 'n_estimators': [10,50,100,200]}
grid_search = GridSearchCV(RandomForestClassifier(), parameters, verbose=0)
grid_search.fit(X_train, y_train)
grid_df = pd.DataFrame(grid_search.cv_results_['params'])
grid_df['mean_test_score'] = grid_search.cv_results_['mean_test_score']
grid_df['rank'] = grid_search.cv_results_['rank_test_score']
grid_df.sort_values(by='rank').head()
```

	max_depth	n_estimators	mean_test_score	rank
0	6	10	0.883275	1
2	6	100	0.882986	2
7	7	200	0.881539	3
3	6	200	0.881366	4
6	7	100	0.879919	5

Ilustración 37. Búsqueda de los mejores hiperparámetros del Random Forest

Como observamos los hiperparámetros con mejores resultados con empate:

Max_depth = 6

N_estimators = 10

```
model_rf = RandomForestClassifier(max_depth=6, n_estimators=10).fit(X_train, y_train)
pred_rf = model_rf.predict(X_test)
```

Ilustración 38. Modelo de clasificación Random Forest

4.2.5. Gradient Boosting

A diferencia de los anteriores el Gradient Boosting es un modelo que empieza con la construcción de un primer clasificador base sencillo. A partir de este primer clasificador y de sus errores cometidos construye el siguiente modelo, usando aquellos elementos mal clasificado anteriormente, realiza una ponderación para que estos obtengan un mayor peso en el nuevo modelo.

Utilizando este método el nuevo clasificador es una combinación de la predicción del primer modelo más el segunda, ponderadas de acuerdo al esquema de pesos obtenidos. Posteriormente, se repite el procedimiento con el nuevo modelo combinado para crear un nuevo modelo hasta obtener el modelo final.

4.2.5.1. Gradient Boosting Regressor

Creamos el modelo de regresión del Gradient Boosting para obtener el RUL:

```
model_gb = GradientBoostingRegressor().fit(X_train, y_train.to_numpy().ravel())
pred_gb = model_gb.predict(X_test)
```

Ilustración 39. Modelo de regresión Gradient Boosting

4.2.5.2. Gradient Boosting Classifier

Realizamos la búsqueda de hiperparámetros utilizando la función GridSearchCV() y utilizamos los mejores resultados para el modelo final:

Gradient boosting

```
parameters = {'learning_rate': [0.001, 0.01, 0.1, 1, 2], 'n_estimators': [10, 50, 100, 200]}
grid_search = GridSearchCV(GradientBoostingClassifier(), parameters, verbose=0)
grid_search.fit(X_train, y_train)
grid_df = pd.DataFrame(grid_search.cv_results_[ 'params' ])
grid_df[ 'mean_test_score' ] = grid_search.cv_results_[ 'mean_test_score' ]
grid_df[ 'rank' ] = grid_search.cv_results_[ 'rank_test_score' ]
grid_df.sort_values(by='rank').head()
```

	learning_rate	n_estimators	mean_test_score	rank
6	0.010	100	0.908275	1
8	0.100	10	0.906192	2
7	0.010	200	0.903009	3
0	0.001	10	0.898611	4
1	0.001	50	0.898611	4

Ilustración 40. Búsqueda de los mejores hiperparámetros del Gradient Boosting

Como observamos los hiperparámetros con mejores resultados con empate:

Learning_rate = 0.01

N_estimators = 100

```
model_gb = GradientBoostingClassifier(learning_rate=0.01, n_estimators=100).fit(X_train, y_train)
pred_gb = model_gb.predict(X_test)
```

Ilustración 41. Modelo de clasificación Gradient Boosting

4.2.6. Extreme Gradient Boosting

Con un procedimiento similar al Gradient Boosting, el Extreme Gradient Boosting (XGB) es una versión más regularizada, este utiliza regularización L1 y L2 con el fin de mejorar sus capacidades de generalización del problema.

4.2.6.1. Extreme Gradient Boosting Regressor

Creamos el modelo de regresión siguiendo los mismos pasos que el Gradient Boosting

```
model_xgb = XGBRegressor().fit(X_train, y_train.to_numpy().ravel())
pred_xgb = model_xgb.predict(X_test)
```

Ilustración 42. Modelo de regresión XGBoosting

4.2.6.2. Extreme Gradient Boosting Classifier

Realizamos de nuevo la búsqueda de los mejores hiperparámetros:

XGBoosting

booster	eval_metric	reg_alpha	reg_lambda	mean_test_score	rank
22	dart	mlogloss	0.5	1.0	0.874306
6	gbtree	mlogloss	0.5	1.0	0.874306
27	dart	mlogloss	1.0	5.0	0.868229
11	gbtree	mlogloss	1.0	5.0	0.868229
28	dart	mlogloss	5.0	0.0	0.867650

Ilustración 43. Búsqueda de los mejores hiperparámetros del Extreme Gradient Boosting

Como observamos los hiperparámetros con mejores resultados con empate:

booster = dart / gbtree

reg_alpha = 0.5

reg_lambda = 1

```
model_xgb = XGBClassifier(booster='dart', reg_alpha=0.5, reg_lambda=1, eval_metric='mlogloss').fit(X_train, y_train.astype(int))
pred_xgb = model_xgb.predict(X_test)
```

Ilustración 44. Modelo de clasificación Extreme Gradient Boosting

4.2.7. Long Short Term Memory (LSTM)

La Long Short Term Memory (LSTM en adelante), es la evolución de las redes neuronales recurrentes debido a la necesidad de una memoria a largo plazo que pueda tener en cuenta secuencias anteriores, esta característica permite que la red neuronal aprenda de experiencias pasada.

La característica más importante de las LSTM es la posibilidad de recordar datos considerados relevantes en la secuencia y preservarlos durante largos tiempos. Por lo que puede tener una memoria de corto plazo (como las Redes Neuronales Recurrentes) como también una memoria a largo plazo.

A diferencia de una celda de red recurrente básica, una celda LSTM tiene una entrada y una salida adicional, la cual se le conoce como celda de estado.

Esta celda de estado permite añadir y eliminar datos con el fin de seleccionar que queremos y no queremos que recuerde el modelo. Para ello, se utiliza lo que se conocen como compuertas, estas pueden ser:

- Forget gate: Compuertas para eliminar elementos de la memoria
- Update gate: Compuertas que permite añadir nuevos elementos a la memoria
- Output gate: Que permite crea un nuevo estado oculto actualizado.

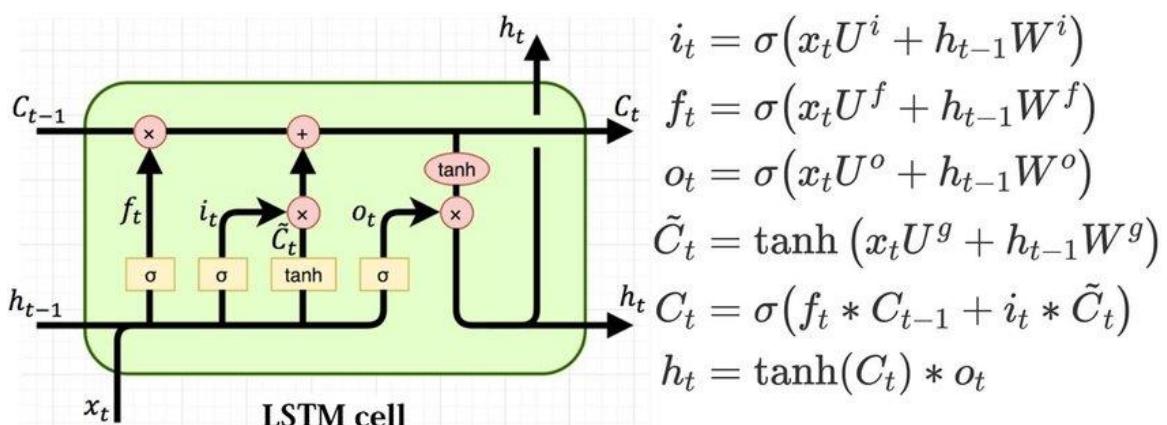


Ilustración 45. Imagen de una Celda LSTM extraído de [15]

Cada una de estas compuertas están compuesta por una red neuronal, una función sigmoide y un elemento multiplicador.

La compuerta de olvido (Forget gate), permite la eliminación de datos o la información que no le llegará a la celda de estado, para ello tiene de entrada el estado oculto anterior h_{t-1} y la entrada actual x_t [Ilustración 45], transformándola con una red neuronal con una función de activación sigmoide f_t . En el caso que uno de los resultados del vector f_t es 0 o cercano 0, entonces la celda LSTM eliminará esa información.

Seguidamente para la compuerta Update gate, permite actualizar los datos de la celda LSTM, usando el estado oculto anterior h_{t-1} y la entrada actual x_t se les aplica una nueva red con la función sigmoide de la función i_t permitiendo entrar todos aquellos datos con un resultado igual a 1 o cercano a 1.

Finalmente la compuerta Output gate con ecuación o_t , que solamente permite el paso de la información útil para el actual estado de la celda. Mientras se usa la función tangente hiperbólica tanto en la función i_t y \tilde{C}_t , como en el estado C_{t-1} para escalar y garantizar que el valor esté en el rango de -1 a 1.

Tras haber realizado pruebas con diferentes modelos con una misma configuración de hiperparámetros [Ilustración 43], se ha seleccionado el modelo que ha dado mejores resultados.

```
def modelo_lstm3(input_shape, optimizer):
    model = Sequential()
    #   model.add(Bidirectional(LSTM(1024, input_shape=input_shape, return_sequences=True)))
    #   model.add(Bidirectional(LSTM(512, return_sequences=True)))
    model.add(Bidirectional(LSTM(256, input_shape=input_shape, return_sequences=True)))
    model.add(Bidirectional(LSTM(128, return_sequences=True)))
    model.add(Bidirectional(LSTM(64, return_sequences=False)))
    model.add(Dense(1, activation='tanh'))

    model.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['mae'])

    return model

def modelo_lstm4(input_shape, optimizer):
    model = Sequential()
    #   model.add(Bidirectional(LSTM(1024, input_shape=input_shape, return_sequences=True)))
    #   model.add(Bidirectional(LSTM(512, return_sequences=True)))
    model.add(LSTM(128, input_shape=input_shape, return_sequences=True))
    #   model.add(Bidirectional(LSTM(128, return_sequences=True)))
    model.add(LSTM(64, return_sequences=False))
    model.add(Dense(1, activation='relu'))

    model.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['mae'])

    return model

def modelo_lstm5(input_shape, optimizer):
    model = Sequential()
    model.add(LSTM(100, input_shape=input_shape, units=30, return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=15, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='relu'))

    model.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['mae'])

    return model
```

Ilustración 46. Prueba de modelos LSTM

La misma arquitectura del modelo LSTM ha sido utilizado en ambos modelos, compuesto por una capa de 100 unidades LSTM que espera la entrada datos en un formato tridimensional comentado en el apartado 4.1 Creación del conjunto de entrenamiento y

pruebas. Seguido por una capa de *Dropout* con el fin de controlar el sobreajuste (*overfitting*) del modelo.

Después de la capa de *Dropout* se ha añadido otra capa LSTM de 30 unidades, otra capa *Dropout* y finalmente una capa sencilla de una unidad con una función de activación.

4.2.7.1. LSTM Regressor

Una vez disponible el modelo se ha realizado un estudio de hiperparámetros para obtener el mejor resultado:

```
: %time
epochs = [100]
batches = [8, 16]
optimizers = ['adam', 'rmsprop']
lrs = [0.01, 0.001, 0.0001]
input_shape=(X_train_3d.shape[1], X_train_3d.shape[2])

for epoch in epochs:
    for batch in batches:
        for opt in optimizers:
            for lr in lrs:
                if opt == 'adam':
                    optimizer = Adam(learning_rate=lr)
                elif opt == 'rmsprop':
                    optimizer = RMSprop(learning_rate=lr)

                model = modelo_lstm(input_shape, optimizer)

                print('=====Training model=====')

                print('Hiperparámetros:')
                print('Optimizer:', opt)
                print('Learning Rate:', lr)
                print('Epochs:', epoch)
                print('Batch:', batch)

                res_m = model.fit(X_train_3d, y_train_3d, validation_data=(X_test_3d, y_test_3d), epochs=epoch, batch_size=batch,
pred = model.predict(X_test_3d)
pred
pred_list = [x[0] for x in pred]

print('\nEvaluation:')
print('R2 score:', r2_score(y_test_3d, pred_list))
print('MSE score:', mean_squared_error(y_test_3d, pred_list))
print('MAE score:', mean_absolute_error(y_test_3d, pred_list))

print('=====')
```

Ilustración 47. Selección de los hiperparámetros del modelo con mejores resultados

Los hiperparámetros del mejor modelo son:

Parámetro	Valor
épocas	100
batch	8
activación	relu
optimizador	adam
learning rate	0.001

```
epochs = 100
optimizer = Adam(learning_rate=0.001)
input_shape=(X_train_3d.shape[1], X_train_3d.shape[2])

model = modelo_lstm(input_shape, optimizer)

res_m = model.fit(X_train_3d, y_train_3d, validation_data=(X_test_3d, y_test_3d), epochs=epochs, batch_size=8, verbose = 1, callbacks=[early_stopping])
```

Ilustración 48. Modelo de regresión LSTM

4.2.7.2. LSTM Classifier

Se ha realizado la misma búsqueda para el modelo clasificador LSTM:

```
%time
epochs = [100]
batches = [8, 16]
optimizers = ['adam', 'rmsprop']
lrs = [0.01, 0.001, 0.0001]
input_shape=(X_train_3d.shape[1], X_train_3d.shape[2])

for epoch in epochs:
    for batch in batches:
        for opt in optimizers:
            for lr in lrs:
                if opt == 'adam':
                    optimizer = Adam(learning_rate=lr)
                elif opt == 'rmsprop':
                    optimizer = RMSprop(learning_rate=lr)

                model = modelo_lstm(input_shape, optimizer)

                print('=====Training model=====')  

                print('Hiperparámetros:')
                print('Optimizer:', opt)
                print('Learning Rate:', lr)
                print('Epochs:', epoch)
                print('Batch:', batch)

                res_m = model.fit(X_train_3d, y_train_3d, validation_data=(X_test_3d, y_test_3d), epochs=epoch, batch_size=batch)

                pred = model.predict(X_test_3d)
                y_test_3d_inv = label_scaler.inverse_transform(y_test_3d)
                pred_inv = label_scaler.inverse_transform(pred)

                y_test_list = []
                pred_list = []
                for i in range(pred_inv.shape[0]):
                    y_test_list.append(y_test_3d_inv[i][0])
                    pred_list.append(round(pred_inv[i][0]))

                print('Accuracy:', accuracy_score(y_test_list, pred_list))
                print('Precision:', precision_score(y_test_list, pred_list))
                print('Recall:', recall_score(y_test_list, pred_list))
                print('F1 score:', f1_score(y_test_list, pred_list))
                print('ROC AUC score:', roc_auc_score(y_test_list, pred_list))

                print('=====\\n')
```

Ilustración 49. Selección de los hiperparámetros del modelo con mejores resultados

Los hiperparámetros del mejor modelo son:

Parámetro	Valor
épocas	100
batch	16
activación	sigmoid
optimizador	rmsprop
learning rate	0.01

```
epochs = 100
optimizer = RMSprop(learning_rate=0.01)
input_shape=(X_train_3d.shape[1], X_train_3d.shape[2])

model = modelo_lstm(input_shape, optimizer)

res_m = model.fit(X_train_3d, y_train_3d, validation_data=(X_test_3d, y_test_3d), epochs=epochs, batch_size=16, verbose = 1, callbacks=[callback])
```

Ilustración 50. Modelo clasificador LSTM

5. Evaluación de los resultados

5.1. Métricas de regresión

Para evaluar la fiabilidad y precisión de los modelos creados se ha usado las siguientes métricas:

- **Coeficiente de determinación R^2**

Usamos el coeficiente de determinación para evaluar cuál es el grado de ajuste de la recta de regresión obtenida a partir de las muestras, y también define la proporción de varianza explicada por la recta de regresión.

El cálculo del coeficiente se puede obtener de dos modos; el primero (1) es realizando la división entre la varianza explicada por el modelo de regresión lineal (suma de cuadrados de la regresión, SCR) entre la varianza total observada (suma de cuadrados totales, SCT); el segundo (2), es a partir de la resta de 1 menos la división de la varianza de los residuos (suma de cuadrados de los errores, SCE) entre la varianza total observada (suma de cuadrados totales, SCT).

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

$$R^2 = 1 - \frac{SCE}{SCT} = 1 - \frac{\sum_{i=1}^n e_i^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (2)$$

Esto significa que el coeficiente puede tener un valor dentro del rango infinito negativo y 1, con lo que podemos interpretar de la siguiente forma:

En el caso que $R^2 = 1$, denota que el ajuste es perfecto, es decir, todos los puntos se encuentran en la recta de regresión. En este caso los residuos son cero y la suma de los cuadrados también, por tanto, la suma cuadrados de la regresión es igual a la suma de los cuadrados totales.

En el caso que $R^2 \leq 0$, indica que no hay relaciones entre las variables dependientes e independientes, en este caso, la suma de residuos es máxima.

- **RMSE (Root mean squared error)**

El estimador RMSE evalúa la diferencia de la raíz cuadrática entre los valores observados y los valores predichos, es decir, el estimador es la media de todas las distancias entre el valor observado y el valor predicho.

El estimador permite saber la regresión contiene más o menos errores dependiendo si el valor es alto, indicando que los puntos observados están muy alejados de los predichos. O si es menor, indicando que los puntos son más cercanos entre ellos.

La ecuación para el cálculo de la RMSE es la siguiente:

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

Donde \hat{y}_i es el valor predicho y y_i es el valor observado/real.

- **MAE (Mean absolute error)**

El estimador MAE es la diferencia absoluta entre los valores observados y los valores predichos, es decir, a diferencia del MSE no se le aplica una suma cuadrática por lo que es más robusto para los valores atípicos y no es tan penalizado por errores extremos.

Aunque muy útil en muchas ocasiones, esta métrica no es ideal para aquellas aplicaciones donde el objetivo es obtener valores atípicos o extremos como la detección de roturas.

La ecuación para el cálculo del MAE es:

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

Similar al MSE, la \hat{y}_i es el valor predicho y y_i es el valor observado/real. Para la interpretar este estimador, se basa en lo mayor que sea el valor del mismo estimador, un valor mayor sugiere que existe una media de error elevado, mientras que un menor valor del estimador sugiere más menos error entre el valor predicho y el observado.

5.2. Resultados de regresión

Una vez entrenado y probado los modelos con los conjuntos de *test* y *train*, se ha obtenido los resultados usando las métricas descritas anteriormente:

Modelo	R ²	RMSE	MAE
Linear regression	0.25872	196.82874	168.90561
SVM	0.57397	152.89543	122.32863
Random forest	0.78527	104.23695	70.92382
Gradient boosting	0.48566	159.85565	132.81475
Extreme gradient boosting	0.63734	133.41533	103.31960
LSTM	0.46977	149.05080	123.02281

Tabla 7. Comparación de los resultados de regresión

Basándonos en los resultados obtenidos se puede observar que el modelo con mejores resultados es el modelo Random forest, con un coeficiente de determinación del 0,78; seguido del modelo Extreme gradient boosting con un coeficiente de 0,64. Observamos que tanto el RMSE como el MAE del Random forest es considerablemente menor que los demás, eso significa que el error promedio es de ±104.2 horas, aproximadamente 4 días.

Es interesante mencionar que el modelo LSTM que se esperaba obtener el mejor resultado de los modelos, este solamente tiene un coeficiente de 0.46977, casi la mitad del mejor modelo.

Visualizando la predicción de la “RUL” del mejor modelo de regresión, Random Forest:

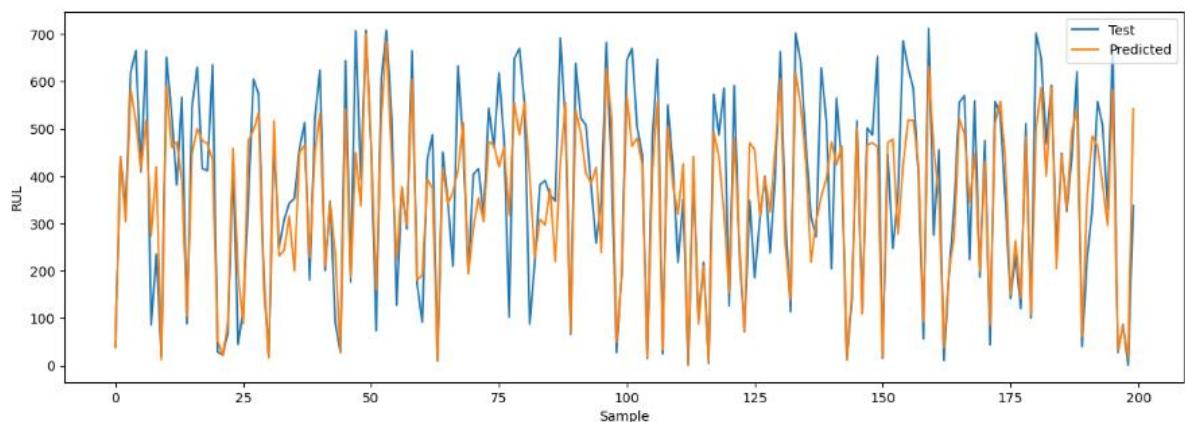


Ilustración 51. Predicción de 200 muestras con el modelo Random Forest

Si aplicamos el modelo sobre la secuencia de ciclos de trabajos ordenados, obtenemos un resultado que se ajusta en gran medida al RUL:

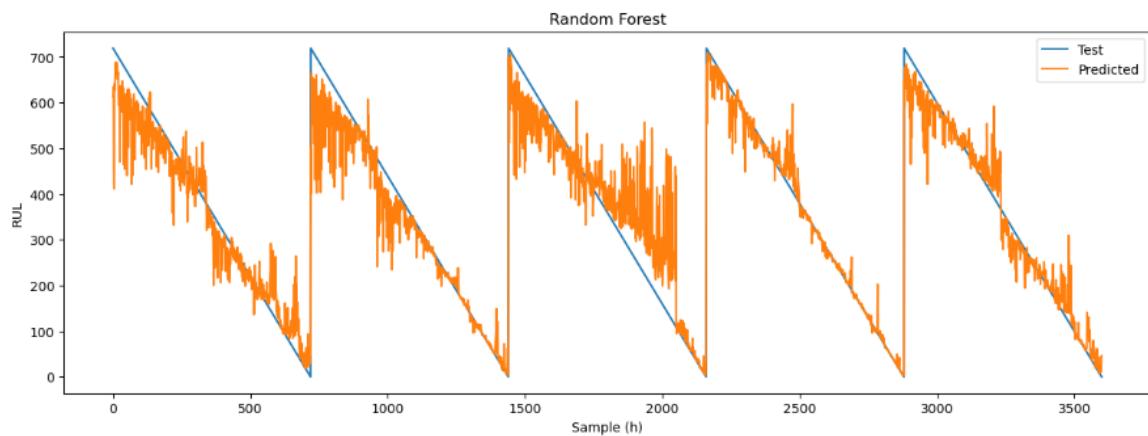


Ilustración 52. Aplicación del modelo Random Forest al conjunto con los ciclos de trabajos normalizados

Observamos que muy probablemente el error de la predicción de RUL afecte en los estados normales intermedios de la máquina y no tanto a las últimas horas antes de la rotura.

5.3. Métricas de clasificación

• Matriz de confusión

La matriz de confusión es una tabla que describe el rendimiento del modelo, esta tabla está compuesta por filas y columnas que agrupan e indican los valores reales y las predichas. Partiendo de esta composición, cada uno de las celdas o posiciones de la matriz indica los diferentes tipos de clases. Utilizando la ilustración 48 como referencia, estas clases son:

- *True Positive*: Son aquellos valores donde la clase observada y predicha son “Yes” y coincidentes.
- *True Negative*: Similar a la clase *True Positive*, son aquellos valores donde la clase observada y predicha son coincidentes, pero con valor “No”.
- *False Positive*: Son los valores que la clase observada es “No” y la clase predicha es “Yes”
- *False Negative*: Son los valores que la clase observada es “Yes” y la clase predicha es “No”

		Actual Values	
		Yes	No
Predicted Values	Yes	True Positive	False Positive
	No	False Negative	True Negative

Ilustración 53. Imagen ejemplo de la matriz de confusión.

Fuente: <https://www.datasource.ai/es/data-science-articles/comprendiendo-la-matriz-de-confusion-y-como-implementarla-en-python>

A partir de esta matriz de confusión podemos obtener otras métricas sobre el rendimiento del modelo.

- **Precision**

La precisión es el total de valores identificados como *True Positive* dividido entre la suma de los elementos predichos como esa clase, es decir, la suma de *True Positive* y *False Positive* de la misma clase:

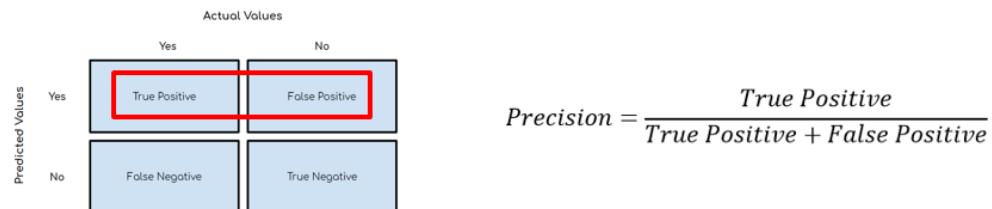


Ilustración 54. Cálculo de la precisión

- **Recall**

El recall se obtiene mediante el cálculo de los elementos *True Positive* dividido entre la suma de los *True Positive* más *False Negative*.

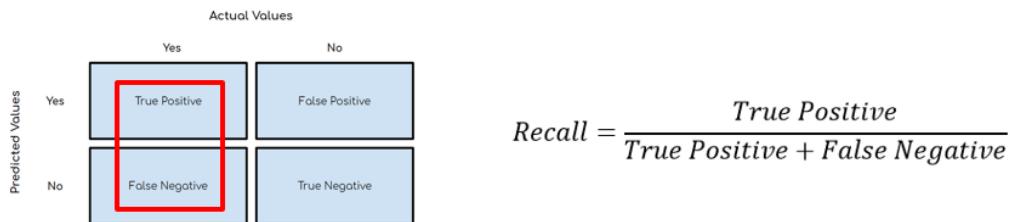


Ilustración 55. Cálculo del recall

En este caso, es de interés esta métrica debido a que un trabajo en estado de “Warning” clasificado como “Normal” puede provocar costes muy elevado en la productividad.

- **F1 score**

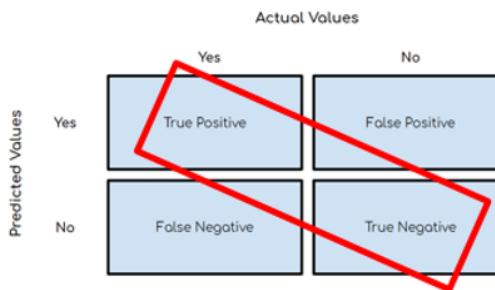
El F1 score es obtenido con la combinación de las medidas de Precision y Recall. Este valor facilita la comparación entre diferentes modelos. Esta métrica es calculada a través de la siguiente ecuación:

$$F1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Ilustración 56. Cálculo del F1 score

- **Accuracy**

La Accuracy o exactitud es el porcentaje de casos que el valor predicho y el valor observado son coincidentes.



$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}}$$

Ilustración 57. Cálculo del accuracy

- **Curva ROC AUC**

La curva AUC - ROC es una medida de rendimiento para los problemas de clasificación en varias configuraciones. ROC es una curva de probabilidad y AUC representa el grado o medida de separabilidad. Indica cuánto es capaz el modelo de distinguir entre clases. Cuanto mayor sea el AUC, mejor será el modelo para predecir 0 clases como 0 y 1 clases como 1.

Su curva está representada por los parámetros:

- *True Positive Rate (TPR)* o *Recall*

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

- *False Positive Rate (FPR)*

$$\text{Recall} = \frac{\text{False Positive}}{\text{True Negative} + \text{False Positive}}$$

Como indica el nombre de la métrica, esta se representa mediante una curva:

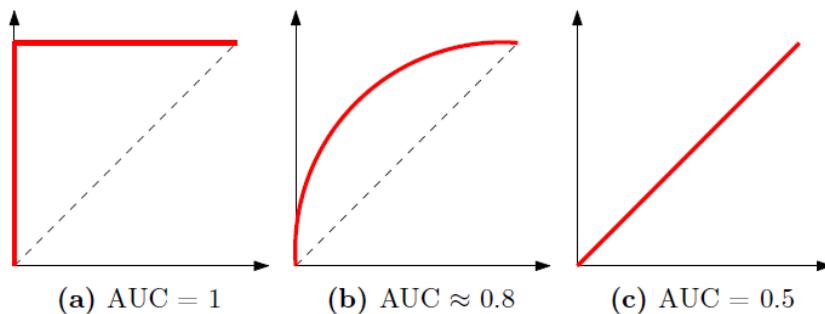


Ilustración 58. Curvas ROC
Fuente: Minería de datos, modelos y algoritmos [14]

Cuando la curva ROC es totalmente recta se interpreta como un modelo generado aleatoriamente, por otro lado, cuando AUC se acerca a 1 se interpreta que el modelo es capaz de distinguir entre clases sin errores.

5.4. Resultados de la clasificación

Una vez entrenado y probado los modelos con los conjuntos de *test* y *train*, se ha obtenido los resultados usando las métricas descritas anteriormente:

Modelo	Accuracy	Precision	Recall	F1 score	AUC score
Logistic regression	0.89861	0	0	0	0.79900
SVM	0.90583	0.6625	0.14520	0.23820	0.79007
Random forest	0.90194	0.53157	0.27671	0.36396	0.85219
Gradient boosting	0.89833	0.49635	0.18630	0.27091	0.82352
Extreme gradient boosting	0.89305	0.45652	0.28767	0.35294	0.79021
LSTM	0.92065	0.76729	0.33424	0.46564	0.66127

Tabla 8. Comparación de resultados de clasificación

De los resultados obtenidos, la regresión logística tiene una precisión, recall y F1 score de 0, pero aun así tiene una alta accuracy. Esto es debido a que el modelo ha predicho todas las clases como una de sola “Normal”, y por mayoría la clase normal el modelo obtienen una Accuracy alta.

Por otro lado, el modelo LSTM es el modelo con mejor Accuracy pero con una AUC score menor en comparación al Random Forest, si comparamos las dos predicciones:

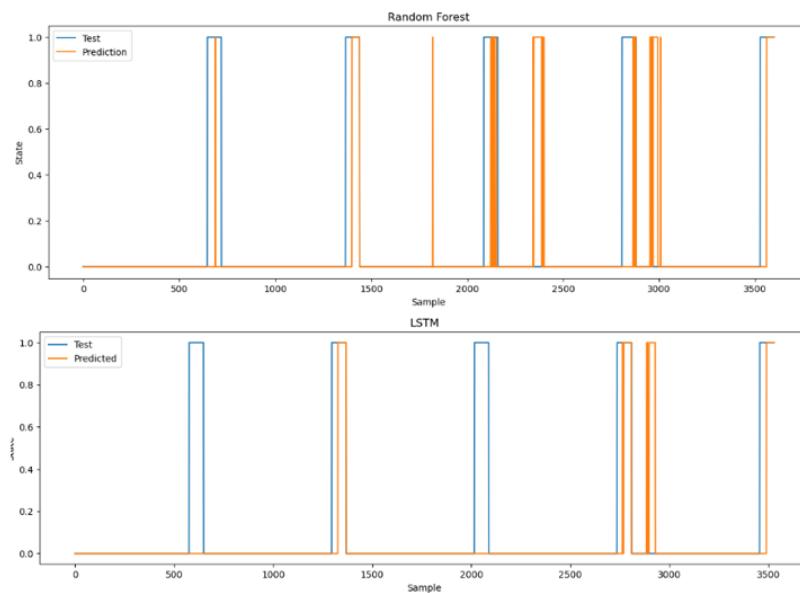


Ilustración 59. Comparativa de las predicciones del Random Forest y LSTM

En la comparativa vemos que efectivamente, el modelo LSTM tiene más aciertos correctas en comparación al Random Forest. Pero si observamos con detalle, el modelo Random Forest realiza distingue mejor de las clases “Normal” (valor 0) y “Warning” (valor 1).

Por lo que si evaluemos cuál de los dos modelos es más idóneo para el problema a resolver, se escogería el modelo que prediga más casos detectado como estado “Warning” en comparación al que prediga menos casos, debido a que un corte trabajo por rotura en medio de la producción tiene un coste mucho mayor que un corte planeado fuera de las horas de producción.

5.5. Desarrollo de los resultados de la regresión

En este apartado se quiere hacer mención del desarrollo adicional sobre los resultados obtenidos en los modelos de regresión. Una vez obtenido los resultados de clasificación menos satisfactorios en comparación a los resultados de los modelos de regresión, se ha planteado una hipótesis:

- ¿Daria mejor resultados para saber si la máquina fallará en 3 días basáandonos en el RUL resultante del mejor modelo de regresión?

Para responder a esta pregunta y partiendo de los resultados del modelo Random Forest, se ha realizado una conversión de valores “RUL” menor igual a 72 al “State” = 1, convertimos esta predicción de regresión a una predicción de clasificación, la cual este mismo modelo de regresión nos permite responder la pregunta si una máquina fallará en los próximos días.

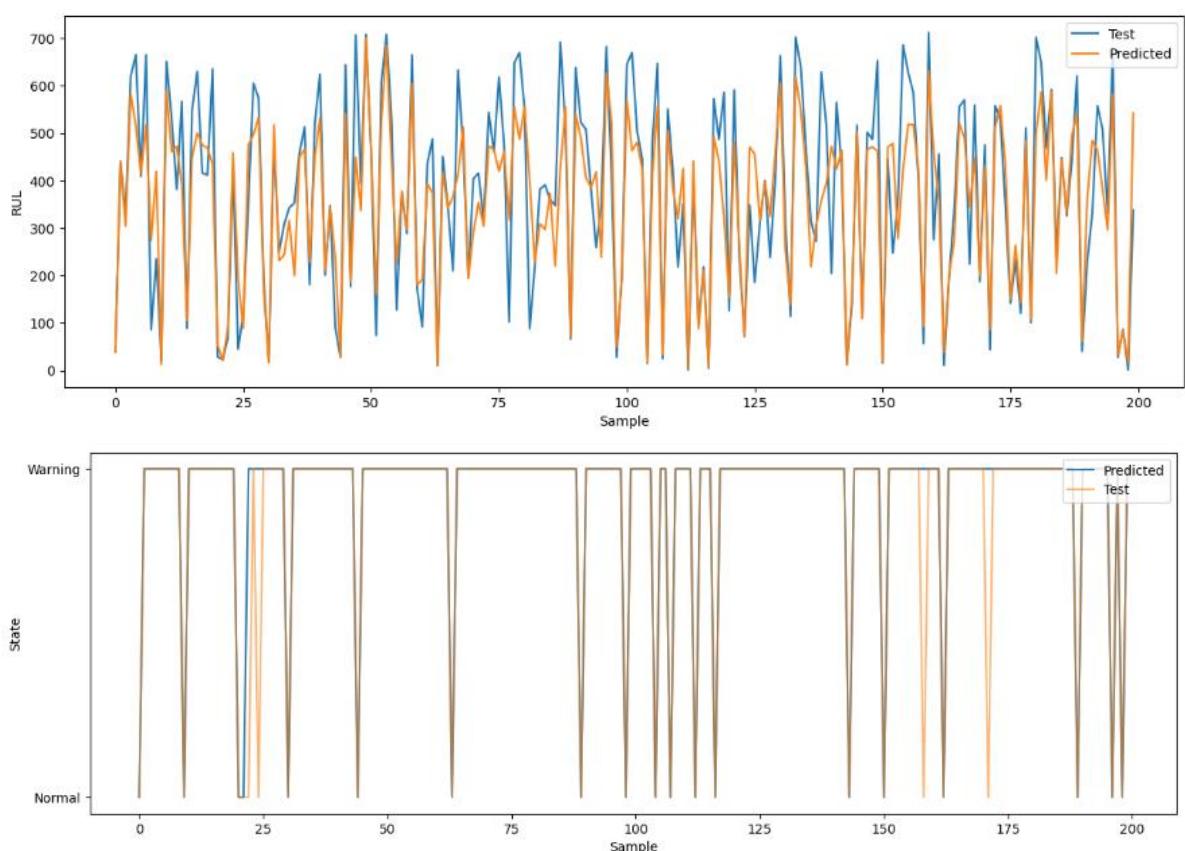


Ilustración 60. Conversión de RUL (igual o menor de 72) a State

Como observamos en la segunda gráfica de la ilustración 57, se muestra 2 líneas, una naranja de las etiquetas originales; y la azul, que son estados creados a partir de la RUL predicha. También se puede distinguir una línea marrón que es la combinación de las dos anteriores, debido a que la línea de Test tiene una opacidad podemos ver en qué puntos los valores reales y predichos son coincidentes.

Si se crea una matriz de confusión a partir de la conversión de todo los RULs predichos:

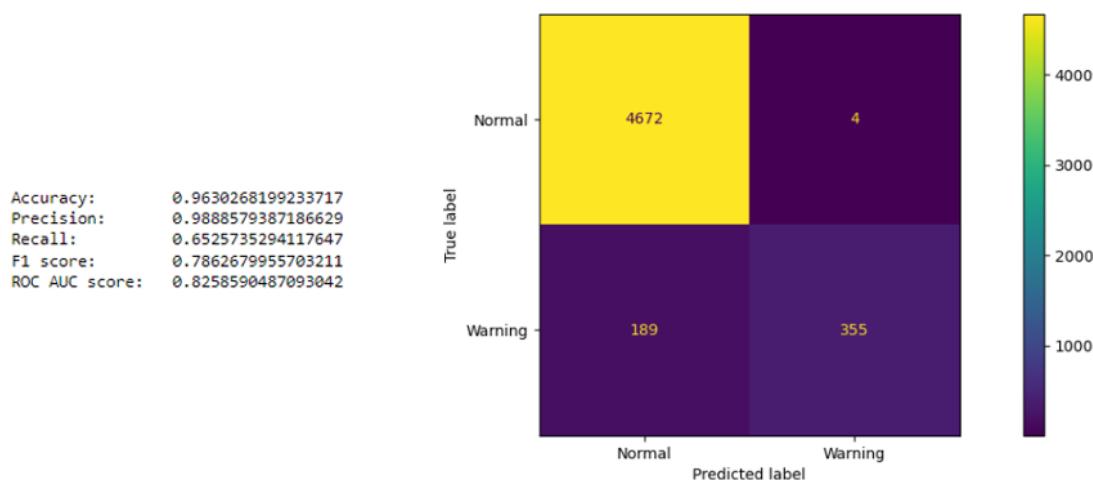


Ilustración 61. Matriz de confusión y métricas a partir del modelo de regresión Random Forest

Finalmente, si se aplica la misma conversión sobre el conjunto con los ciclos de trabajos normalizados y lo comparamos con el modelo de la clasificación:

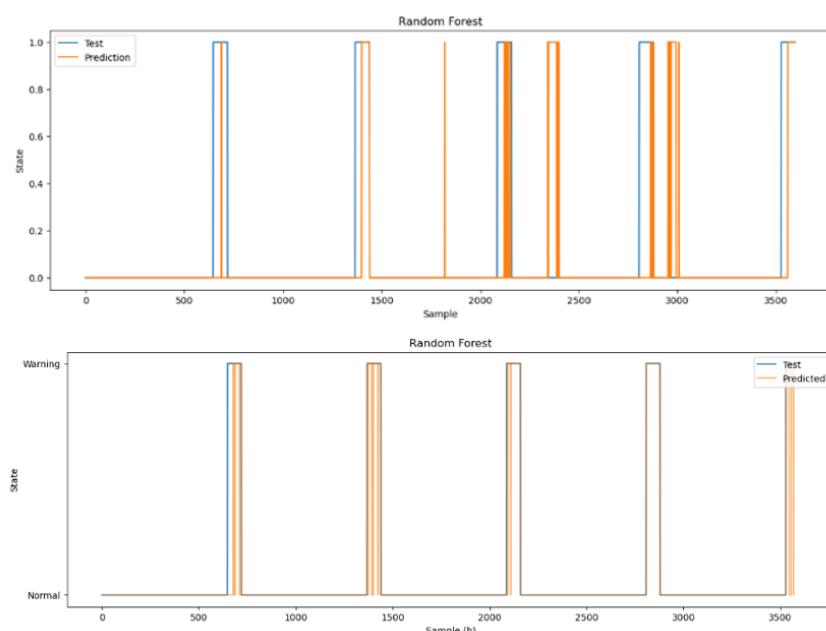


Ilustración 62. Comparación del modelo de clasificación (grafico superior) con el modelo de regresión convertido en clases (grafico inferior)

Se observa claramente una mejora de precisión en comparación al modelo de clasificación original. Esto puede ser debido a que usar los mismos datos del modelo de regresión en los modelos de clasificación no sean los idóneos para este tipo de algoritmo y se deba realizar un estudio y selección más profundo de estos datos según el tipo de modelo que se desea utilizar.

6. Conclusiones y trabajos futuros

6.1. Objetivos adquiridos

Objetivo general:

- **Obtención de un modelo de predicción de posibles fallos y defectos de máquinas antes de que ocurra dicho fallo.**

Objetivo cumplido. En el apartado 4. Análisis y modelado, se ha concretado este objetivo en dos preguntas:

- ¿Cuál es RUL que le queda a la máquina hasta su próxima rotura?
- ¿Fallará la máquina en los próximos 3 días?

Según los resultados obtenidos se ha podido predecir los errores registrados con el modelo entrenado, por lo que se puede considerar este objetivo como cumplido ya que a partir del modelo construido se puede crear una herramienta para predecir y programar mantenimientos días antes de que ocurra la rotura o fallo de la máquina.

Objetivos secundarios:

- **Conocer y estudiar el estado del arte del machine learning aplicado al ámbito industrial, en concreto, al mantenimiento predictivo.**

Objetivo cumplido. Como se ha comentado en el apartado anterior, se ha adquirido conocimiento de estudios anteriores aplicados en el mismo ámbito, de las cuales se ha podido sacar un beneficio y aplicar parcialmente en este proyecto.

- **Identificación de indicadores relevantes que apuntan a una posible anomalía.**

Objetivo cumplido parcialmente. Aunque se haya cumplido el objetivo general, se ha considerado que este objetivo solo se ha cumplido parcialmente. Esto es debido a que aún se considera que se podría realizar una nueva iteración del problema para mejorar los datos de entrada para aumentar más la precisión del modelo.

- **Estudio y comparación de diferentes modelos para la obtención del mejor modelo.**

Objetivo cumplido. Se ha seleccionado un conjunto de modelos en base a la revisión bibliográfica y posteriormente se ha comparado cual tiene mejor rendimiento en la comparación de resultados

- **Utilización transversal del conocimiento adquirido a lo largo del master para un posible caso real**

Objetivo cumplido. Aunque no se ha aplicado todo el conocimiento adquirido durante en el master, se puede afirmar la aplicación del conocimiento para complementar la información de estudios anteriores del mismo ámbito.

6.2. Conocimientos adquiridos

En el presente trabajo se ha podido estudiar un ámbito en que se ha podido poner en práctica los conocimientos adquirido a lo largo del máster y complementándolo con estudios previos sobre el mantenimiento predictivo. Este conocimiento junto al problema planteado, se ha podido enfocar este trabajo no únicamente como un problema académico, sino que también como un problema que puede surgir en un entorno profesional.

Gracias a la revisión bibliográfica sobre otros trabajos con un objetivo similar, se ha podido definir con mayor facilidad pequeñas metas a alcanzar durante las diferentes fases del desarrollo.

Un ejemplo de esta pequeña meta es la obtención de una variable objetivo, en este caso el RUL, estudios anteriores enfatizan el uso de esta métrica para medir el tiempo de vida una máquina la cuál ha sido muy útil para tener un punto de partida con el fin llegar al objetivo general planteado. Estos estudios también han influido en la selección de modelos, ya que han agilizado y reducido las opciones óptimas para un problema de esta naturaleza.

En cuanto los primeros resultados obtenidos eran malos, se ha planteado la hipótesis que la tabla resumen de los conjuntos de datos no eran suficientemente buenos para el estudio, por lo que se ha realizado una segunda iteración aumentando su dimensionalidad añadiendo campos compuestos. A partir de esta segunda iteración se ha podido obtener unos mejores resultados, por lo que verifica la hipótesis planteada.

Aun así, el modelo LSTM que parecía el modelo más idóneo para este tipo de problema, ha dado unos resultados considerablemente malos. Este hecho plantea la posibilidad en algún fallo durante la preparación de los datos, ya que precisamente este modelo requiere de un trabajo muy específico de los datos para poder trabajar correctamente. Otra alternativa, es la existencia de la posibilidad que un aumento de la dimensionalidad no sea beneficioso para ciertos modelos, uno de ellos el LSTM, sino que podría beneficiarse más con una reducción.

Finalmente, de los resultados obtenidos en la segunda iteración se ha obtenido que el mejor modelo de regresión es el Random Forest, mientras que el mejor modelo de clasificación es el LSTM. Pero en un desarrollo posterior sobre los resultados del modelo de regresión, realizando una conversión del RUL al Estado de la máquina se ha verificado que el Modelo de regresión de Random Forest convertido en una clasificación tiene unas métricas de clasificación muy superiores al LSTM.

Podemos concluir que el modelo de regresión Random Forest y su conversión a un clasificador son los más indicados para el problema planteado.

6.3. Seguimiento y planificación

La planificación del trabajo ha sido adecuada, se ha seguido todas las etapas definidas en la planificación del trabajo la cual han permitido una visión de los pasos a seguir para desarrollar dentro de los plazos establecidos. Por otro lado, hay que mencionar que en un principio no se había tenido en cuenta los plazos para realizar iteraciones en el desarrollo, sino que solamente se creó el diagrama de Gantt considerando el tiempo total estimado para cada fase.

6.4. Líneas de trabajo futuro

Aunque se ha cumplido el objetivo principal de este trabajo, aún quedan tareas a realizar que pueden mejorar los resultados obtenidos, estas tareas son las siguientes:

- Estudio de la razón por qué el modelo LSTM, que se esperaba los mejores resultados no ha dado unos resultados satisfactorios.
- Realización de más iteraciones con una reducción de dimensionalidad del conjunto de datos, existe la posibilidad que esta reducción puede llegar a ser igual o más beneficioso en comparación a un aumento de la dimensionalidad.
- Estudio de predicciones con unos rangos mayores, en este trabajo se ha planteado detectar una rotura en los próximos 3 días, se podría incrementar este rango a 7 días o 15 días. Un mayor rango permite más flexibilidad a la hora de encargar posibles componentes de recambio.
- Realización de la predicción del RUL de la máquina mediante el modelo ARIMA, debido al problema estudiado es de naturaleza similar sino prácticamente igual a una serie temporal, se considera que el modelo ARIMA podría dar unos resultados interesantes.
- En este trabajo simplemente se ha hecho un estudio para predecir las futuras roturas de las máquinas, para completar este estudio una opción es crear una solución integral completa para que usuarios puedan interactuar con una aplicación web o aplicación para dispositivos móviles.

7. Bibliografía

- [1] Forbes: <https://forbes.es/forbes-w/157536/como-linda-joho-ha-convertido-united-airlines-en-un-lider-tecnologico/> 10-2022
- [2] Preditec: <http://www.preditec.com/servicios/> 10-2022
- [3] Ramy Baly and Hazem Hajj, Wafer classification using support vector machines. IEEE Transactions on semiconductor manufacturing, 373–383 Vol.25 No.3, 2012
- [4] Hongfei Li, Dhaivat Parikh, Qing He, Buyue Qian, Zhiguo Li, Dongping Fang, and Arun Hampapur. Improving rail network velocity: A machine learning approach to predictive maintenance. Transportation Research Part C: Emerging Technologies, 17–26 Vol.45, 2014.
<https://doi.org/10.1016/j.trc.2014.04.013>
- [5] Baptista, M., Sankararaman, S., de Medeiros, o.P., Nascimento Jr., C., Prendinger, H., Heniquesa, E.M.P., Forecasting Fault Events for Predictive Maintenance using Data-driven Techniques and ARMA Modeling, Computers & Industrial Engineering (2017), doi: <https://doi.org/10.1016/j.cie.2017.10.033>
- [6] Bruneo D. y De Vita F. (2019). On the Use of LSTM Networks for Predictive Maintenance in Smart Industries. IEEE International Conference on Smart Computing (SMARTCOMP), 2019.
- [7] Dong, D., Li, X. y Sun, F. (2017). Life prediction of jet engines based on LSTM-recurrent neural networks, Prognostics and System Health Management Conference (PHMHarbin). DOI: 10.1109/PHM.2017.8079264
- [8] Ameeth Kanawaday and Aditya Sane. Machine learning for predictive maintenance of industrial machines using iot sensor data. In 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), p.87–90. IEEE, 2017.
- [9] Han, Tian; Zhang, Longwen. Rolling bearing fault diagnosis with combined convolutional neural networks and support vector machine, Measurement, Vol.117, 2021.
- [10] Siemesns: <https://www.siemens.com/global/en/products/services/digital-enterprise-services/analytics-artificial-intelligence-services/predictive-services.html> 10-2022
- [11] CORDIS: <https://cordis.europa.eu/project/id/957391/es> 10-2022
- [12] Eduardo HM Pena, Marcos VO de Assis, and Mario Lemes Proen,ca. Anomaly detection using forecasting methods arima and hwds. In 2013 32nd international conference of the chilean computer science society (sccc), p.63–66. IEEE, 2013.
- [13] Zeyang Chen, Zhen Gao, Rongjie Yu, Min Wang, and Ping Sun. Macro-level accident fatality prediction using a combined model based on arima and multivariable linear regression. In 2016 International Conference on Progress in Informatics and Computing (PIC), p.133–137. IEEE, 2016

- [14] Gironés Roig, J., Casas Roma, J., Minguillón Alfonso, J. & Caihuelas Quiles, R. (2017). Minería de datos : Modelos y algoritmos (Primera edición digital) [Pdf]. UOC. ISBN: 978-84-9116-904-8
- [15] Varsamopoulos, Savvas & Bertels, Koen & Almudever, Carmen. (2018). Designing neural network based decoders for surface codes.
- [16] Bosch Rué, A., Casas Roma, J. & Lozano Bagén, T (2020). Deep Learning principios y fundamentos (Primera edición digital) [Pdf]. UOC. ISBN: 978-84-9180-657-8

8. Anexos

Debido el peso y el formato interactivo de los gráficos generados durante el EDA, este no se encuentra disponible en los anexos para ello se debe consultar al repositorio github junto a los demás archivos.

<https://github.com/NetRunner5/MachineLearningOnPredictiveMaintenance>

A continuación, se adjuntan los anexos de los diferentes modelos de predicción.

- Modelos de regresión
- Modelo de clasificación
- Modelos LSTM de regresión y clasificación

Carga Librerias

```
In [1]: # Librerias
import warnings
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import plotly.express as px

import seaborn as sns

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.decomposition import PCA

from sklearn import svm
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor

from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error, confusion_matrix, ConfusionMatrix
```

```
In [2]: def plot_metrics(model, x, y, pred_y):
    print('R^2 score:\t', r2_score(y, pred_y))
    print('MSE:\t\t', mean_squared_error(y, pred_y))
    print('RMSE:\t\t', mean_squared_error(y, pred_y, squared=False))
    print('MAE:\t\t', mean_absolute_error(y, pred_y))

    fig, ax = plt.subplots(1)
    matplotlib.rc('figure', figsize=(15, 5))
    ax.plot(y)
    ax.plot(pred_y)
```

Carga Datos

```
In [3]: DATA_DIR = "C:/Users/NetRunner/OneDrive/UOC/Semestre 6/TFM/MultipleDatasets"

train_data_cut = pd.read_csv(f"{DATA_DIR}/train_data.csv")
test_data_cut = pd.read_csv(f"{DATA_DIR}/test_data.csv")

train_data_uncut = pd.read_csv(f"{DATA_DIR}/train_data_uncut.csv")
test_data_uncut = pd.read_csv(f"{DATA_DIR}/test_data_uncut.csv")

# X_train = pd.read_csv(f"{DATA_DIR}/X_train.csv")
# y_train = pd.read_csv(f"{DATA_DIR}/y_train.csv")
# X_test = pd.read_csv(f"{DATA_DIR}/X_test.csv")
# y_test = pd.read_csv(f"{DATA_DIR}/y_test.csv")
```

```
In [4]: data_cut = pd.concat([train_data_cut, test_data_cut])
data_uncut = pd.concat([train_data_uncut, test_data_uncut])

features = ['volt', 'rotate', 'pressure', 'vibration', 'error1', 'error2', 'error3',
            'error4', 'error5', 'volt_3h_mean', 'rotate_3h_mean',
            'pressure_3h_mean', 'vibration_3h_mean', 'volt_24h_mean',
            'rotate_24h_mean', 'pressure_24h_mean', 'vibration_24h_mean',
            'error1_count', 'error2_count', 'error3_count', 'error4_count',
            'error5_count']
label = ['RUL']

data_cut = data_cut[features+label]
data_uncut = data_uncut[features+label]
```

Normalización de datos MinMax

```
In [5]: feature_scaler_cut = MinMaxScaler(feature_range=(0,1))
label_scaler_cut = MinMaxScaler(feature_range=(0,1))

feature_scaler_cut.fit(data_cut[features])
label_scaler_cut.fit(data_cut[label].values.reshape(-1,1))

Out[5]: MinMaxScaler()
```

```
In [6]: feature_scaler_uncut = MinMaxScaler(feature_range=(0,1))
label_scaler_uncut = MinMaxScaler(feature_range=(0,1))
```

```
feature_scaler_uncut.fit(data_uncut[features])
label_scaler_uncut.fit(data_uncut[label].values.reshape(-1,1))
```

Out[6]: MinMaxScaler()

```
In [7]: data_norm_cut = data_cut.copy()
data_norm_cut[features] = feature_scaler_cut.transform(data_cut[features])
data_norm_cut[label] = label_scaler_cut.transform(data_cut[label].values.reshape(-1,1))
```

```
data_norm_uncut = data_uncut.copy()
data_norm_uncut[features] = feature_scaler_uncut.transform(data_uncut[features])
data_norm_uncut[label] = label_scaler_uncut.transform(data_uncut[label].values.reshape(-1,1))
```

Dataset Train/Test

```
In [8]: X_train_cut, X_test_cut, y_train_cut, y_test_cut = train_test_split(data_norm_cut[features], data_norm_cut[label], test_size=0.2, random_state=42)
```

```
In [9]: X_train_uncut, X_test_uncut, y_train_uncut, y_test_uncut = train_test_split(data_norm_uncut[features], data_norm_uncut[label], test_size=0.2, random_state=42)
```

Control/Swap de variables

Esta celda sirve para pasar de los datos con ciclos homogenizados al conjunto entero con todos los ciclos sin homogenizar

```
In [10]: # Cut cycles
X_train = X_train_cut
y_train = y_train_cut
X_test = X_test_cut
y_test = y_test_cut

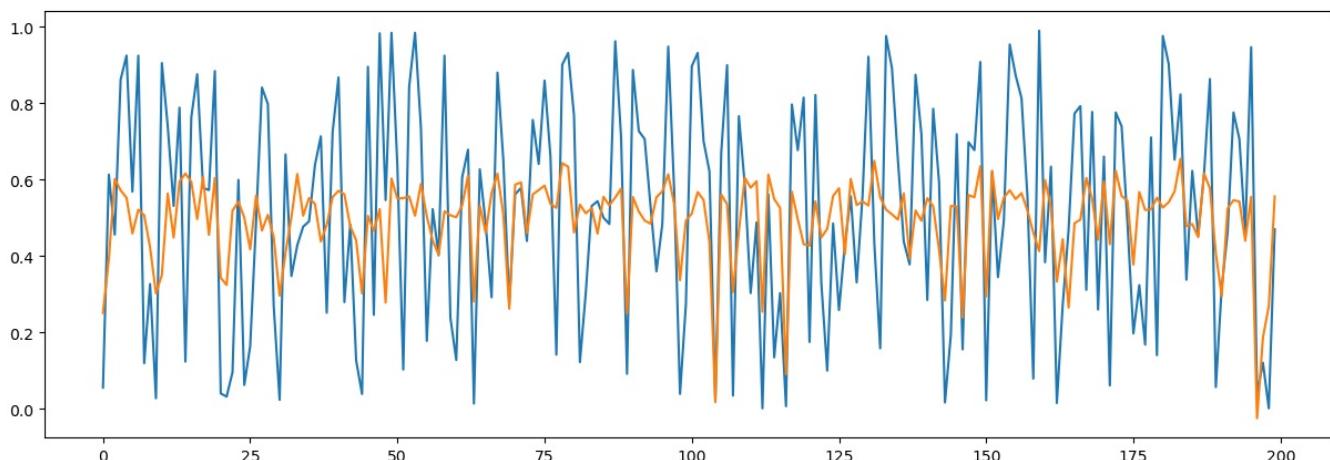
# Uncut cycles
# X_train = X_train_uncut
# y_train = y_train_uncut
# X_test = X_test_uncut
# y_test = y_test_uncut
```

Linear Regression

```
In [11]: model_lr = LinearRegression().fit(X_train, y_train)
pred_lr = model_lr.predict(X_test)
```

```
In [31]: plot_metrics(model_lr, X_test[:200], y_test[:200].values, pred_lr[:200])
```

```
R^2 score: 0.2588718134999083
MSE: 0.06381095593418949
RMSE: 0.25260830535473194
MAE: 0.21520199012181682
```

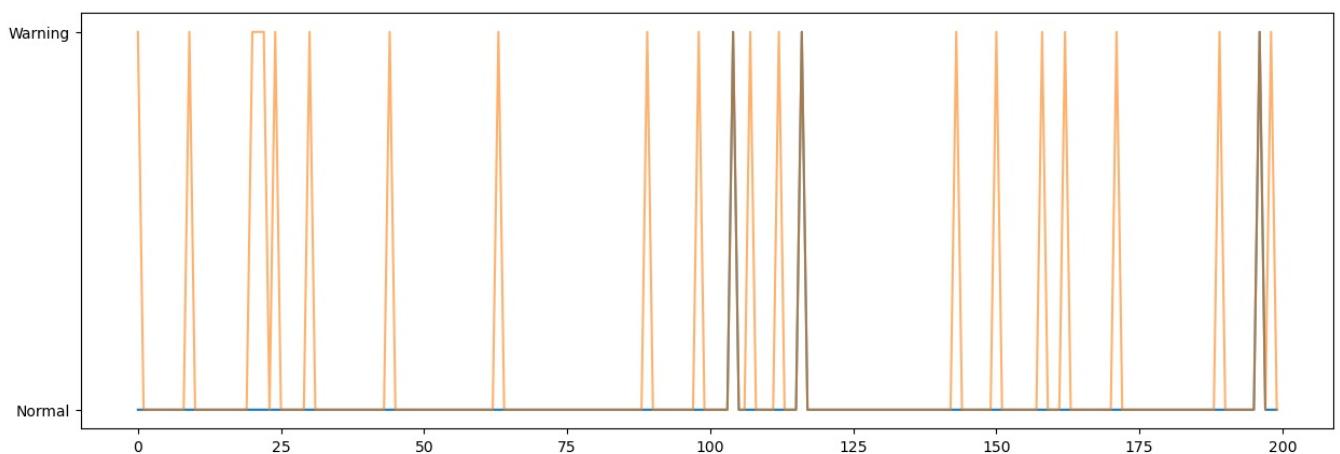


```
In [13]: pred_lr_inv = label_scaler_cut.inverse_transform(pred_lr.reshape(-1, 1))
y_test_cut_inv = label_scaler_cut.inverse_transform(y_test_cut.values.reshape(-1, 1))

y_test_cut_inv_class = ['Normal' if x >= 72 else 'Warning' for x in y_test_cut_inv]
pred_lr_inv_class = ['Normal' if x >= 72 else 'Warning' for x in pred_lr_inv]

plt.plot(pred_lr_inv_class[:200])
plt.plot(y_test_cut_inv_class[:200], alpha = 0.6)
```

```
Out[13]: [<matplotlib.lines.Line2D at 0x1f1575c26a0>]
```

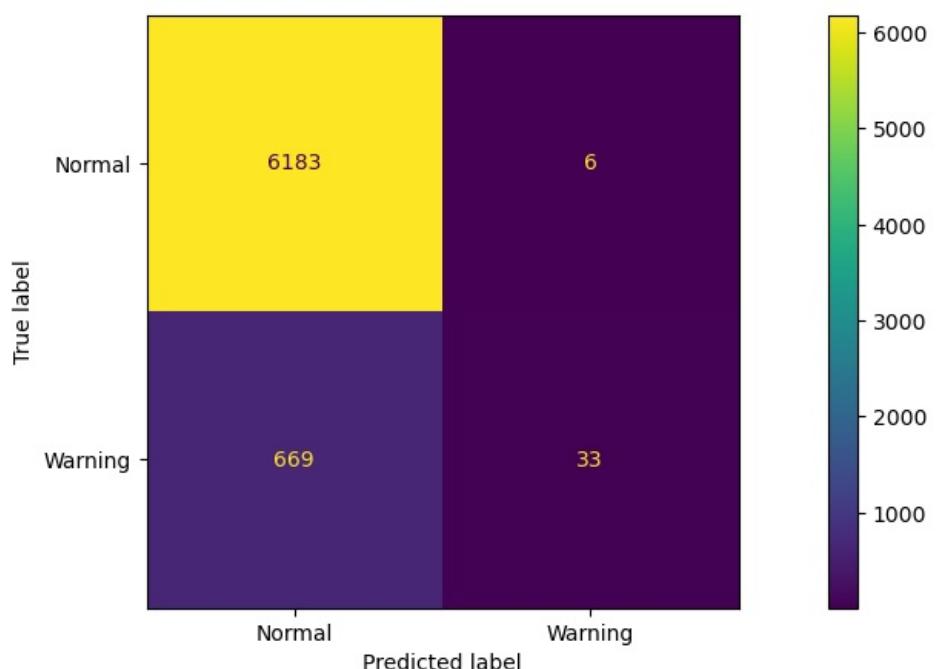


```
In [14]: lr_cm = confusion_matrix(y_test_cut_inv_class, pred_lr_inv_class)
```

```
print('Accuracy: ', accuracy_score(y_test_cut_inv_class, pred_lr_inv_class))
ConfusionMatrixDisplay(lr_cm, display_labels=['Normal', 'Warning']).plot()
```

```
Accuracy: 0.9020461471484545
```

```
Out[14]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f155a27640>
```

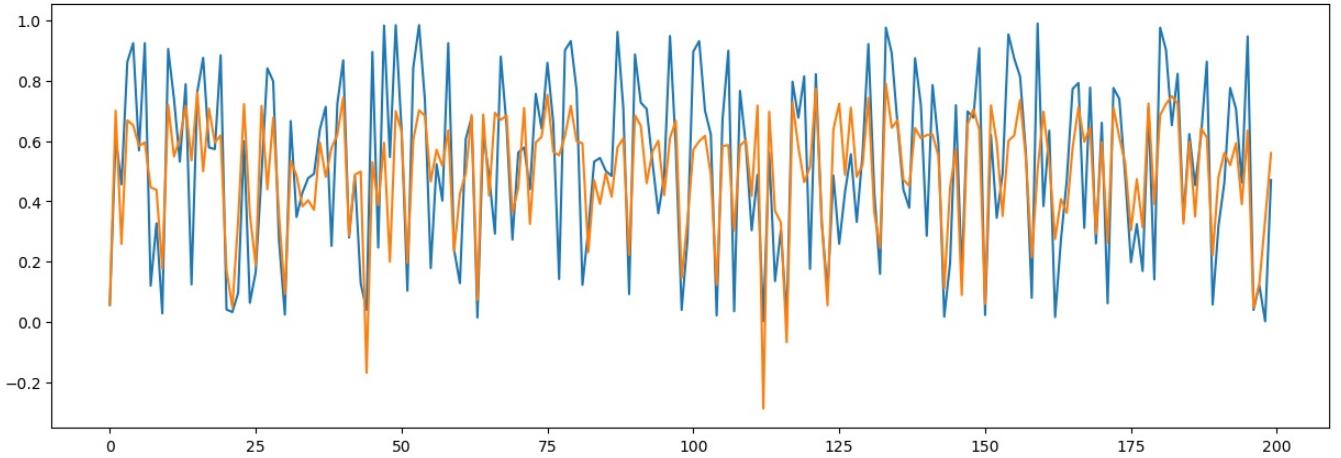


SVM Regressor

```
In [15]: model_svm = svm.SVR().fit(X_train, y_train.to_numpy().ravel())
pred_svm = model_svm.predict(X_test)
```

```
In [16]: plot_metrics(model_svm, X_test[:200], y_test[:200].values, pred_svm[:200])
```

```
R^2 score: 0.5657667864012093
MSE: 0.03738737368627998
RMSE: 0.1933581487454821
MAE: 0.15527680953807102
```

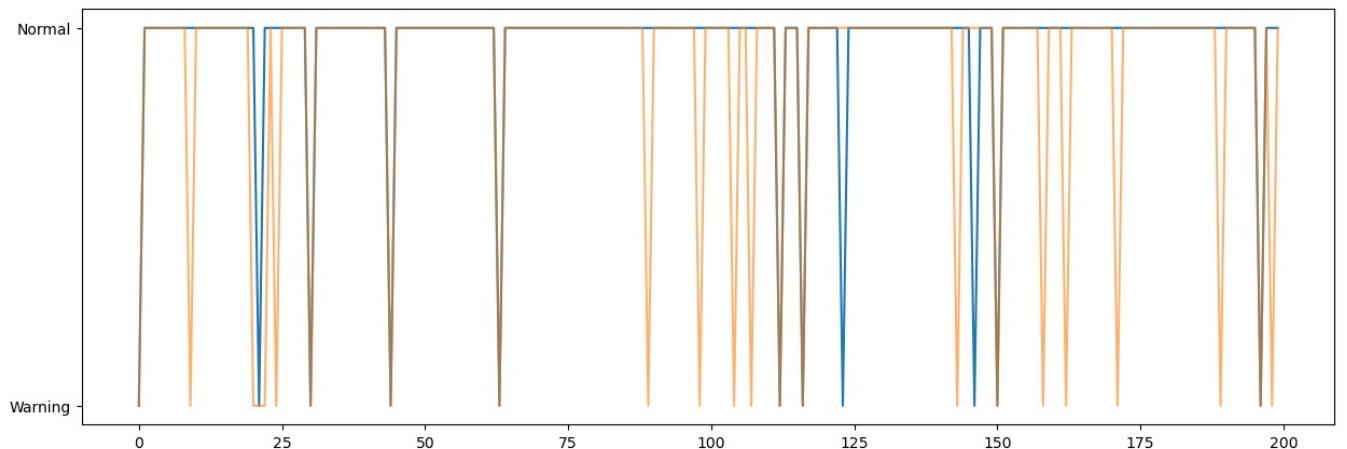


```
In [17]: pred_svm_inv = label_scaler_cut.inverse_transform(pred_svm.reshape(-1, 1))
y_test_cut_inv = label_scaler_cut.inverse_transform(y_test_cut.values.reshape(-1, 1))

y_test_cut_inv_class = ['Normal' if x >= 72 else 'Warning' for x in y_test_cut_inv]
pred_svm_inv_class = ['Normal' if x >= 72 else 'Warning' for x in pred_svm_inv]

plt.plot(pred_svm_inv_class[:200])
plt.plot(y_test_cut_inv_class[:200], alpha = 0.6)
```

Out[17]: <matplotlib.lines.Line2D at 0x1f1582bbca0>

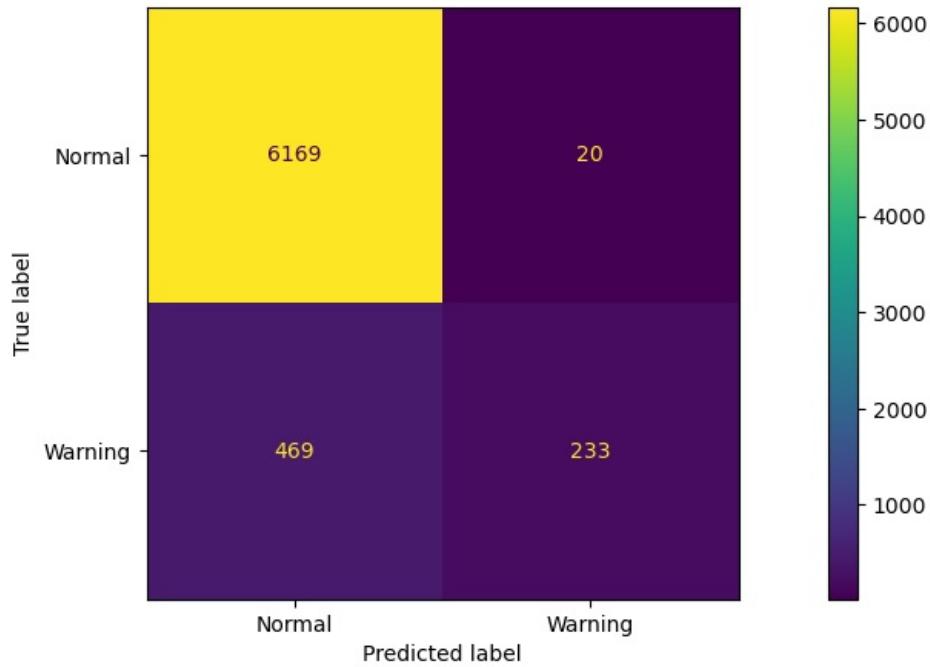


```
In [18]: svm_cm = confusion_matrix(y_test_cut_inv_class, pred_svm_inv_class)

print('Accuracy: ', accuracy_score(y_test_cut_inv_class, pred_svm_inv_class))
ConfusionMatrixDisplay(svm_cm, display_labels=['Normal', 'Warning']).plot()
```

Accuracy: 0.9290378754897692

Out[18]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f157a3a580>

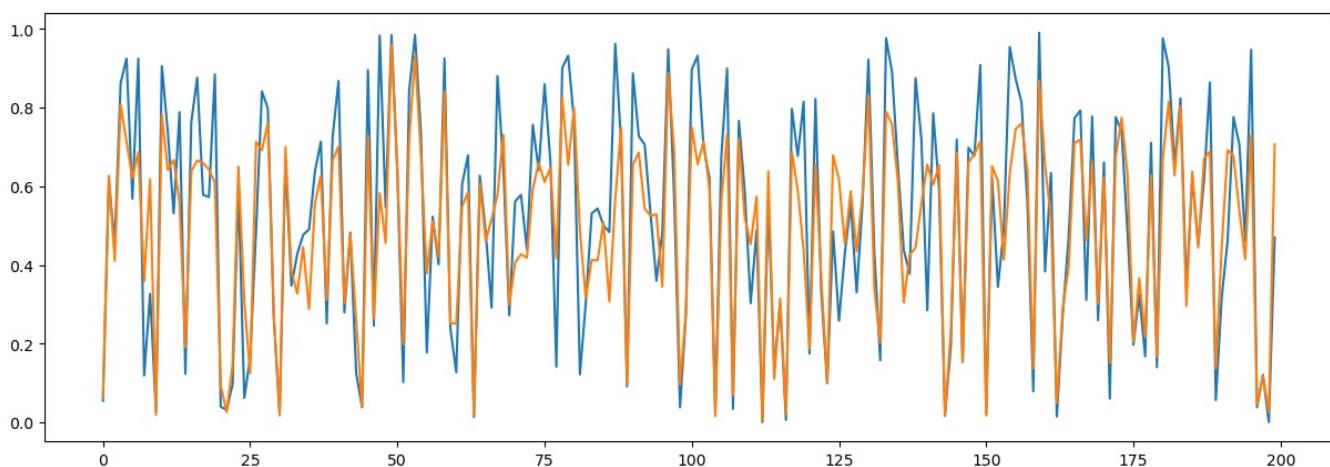


Random Forest Regressor

```
In [19]: model_rf = RandomForestRegressor().fit(X_train, y_train.to_numpy().ravel())
pred_rf = model_rf.predict(X_test)
```

```
In [20]: plot_metrics(model_rf, X_test[:200], y_test[:200].values, pred_rf[:200])
```

R^2 score: 0.7752798899252249
MSE: 0.0193483466189906
RMSE: 0.1390983343501661
MAE: 0.101028859527121

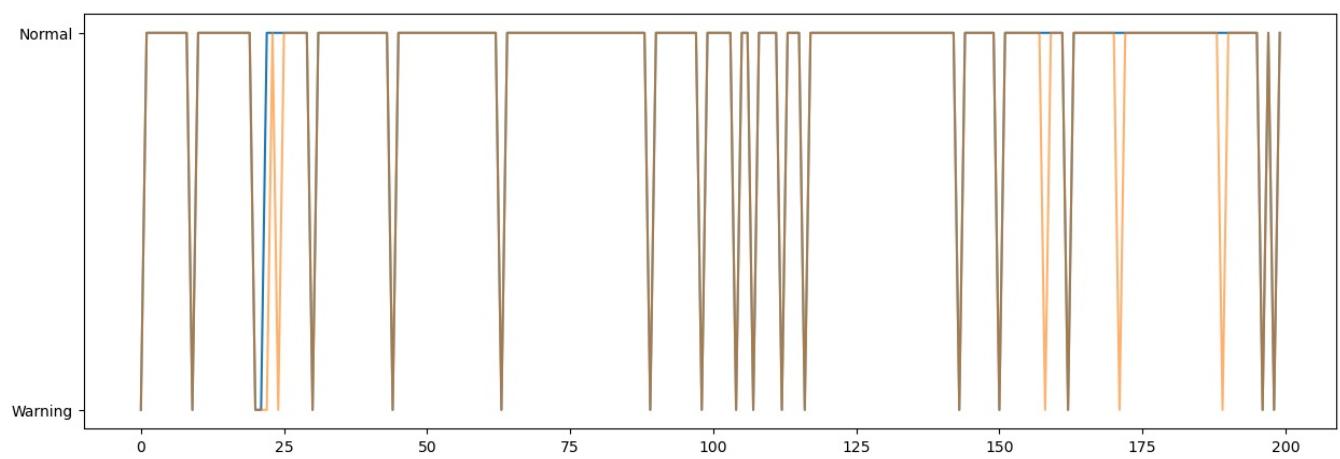


```
In [21]: pred_rf_inv = label_scaler_cut.inverse_transform(pred_rf.reshape(-1, 1))
y_test_cut_inv = label_scaler_cut.inverse_transform(y_test_cut.values.reshape(-1, 1))

y_test_cut_inv_class = ['Normal' if x >= 72 else 'Warning' for x in y_test_cut_inv]
pred_rf_inv_class = ['Normal' if x >= 72 else 'Warning' for x in pred_rf_inv]

plt.plot(pred_rf_inv_class[:200])
plt.plot(y_test_cut_inv_class[:200], alpha = 0.6)
```

```
Out[21]: <matplotlib.lines.Line2D at 0x1f158107400>
```

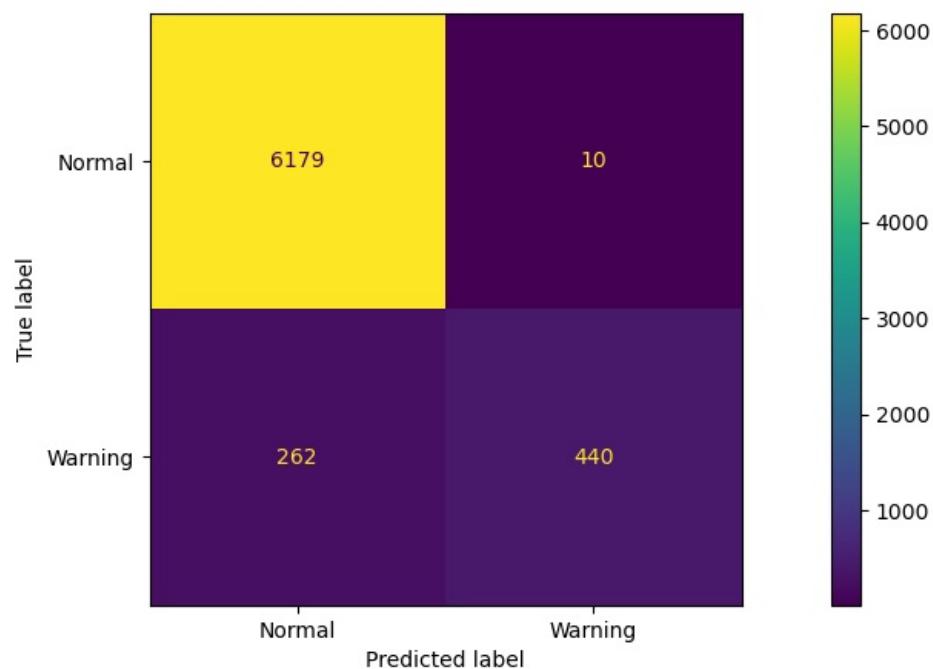


```
In [22]: rf_cm = confusion_matrix(y_test_cut_inv_class, pred_rf_inv_class)
```

```
print('Accuracy: ', accuracy_score(y_test_cut_inv_class, pred_rf_inv_class))
ConfusionMatrixDisplay(rf_cm, display_labels=['Normal', 'Warning']).plot()
```

```
Accuracy: 0.9605282252213031
```

```
Out[22]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f157e9c6a0>
```

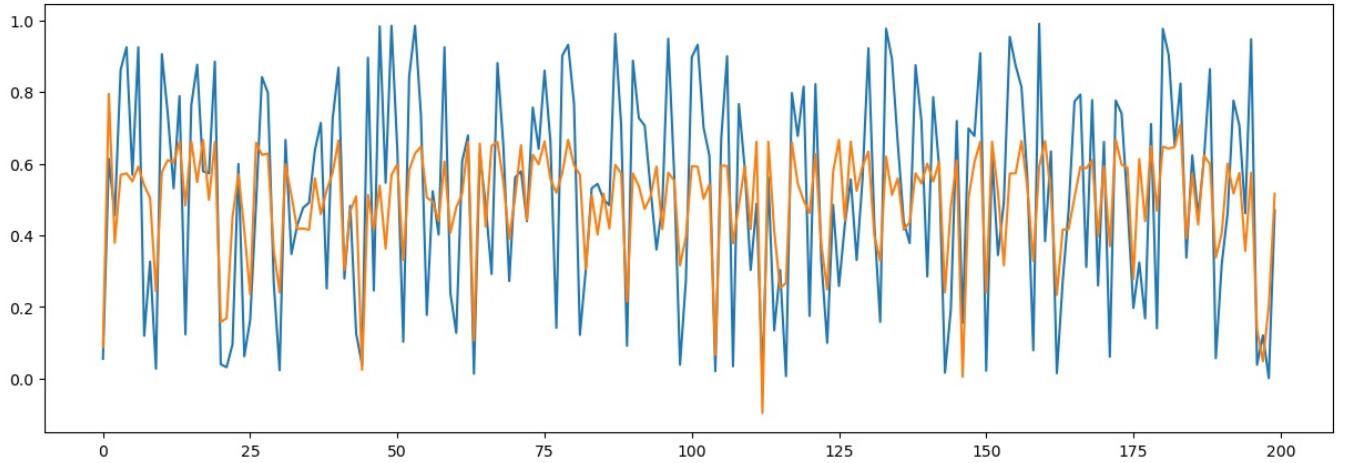


Gradient boosting Regressor

```
In [23]: model_gb = GradientBoostingRegressor().fit(X_train, y_train.to_numpy().ravel())
pred_gb = model_gb.predict(X_test)
```

```
In [24]: plot_metrics(model_gb, X_test[:200], y_test[:200].values, pred_gb[:200])
```

```
R^2 score: 0.481953454305322
MSE: 0.04460368111930944
RMSE: 0.21119583594216398
MAE: 0.17482294880465282
```

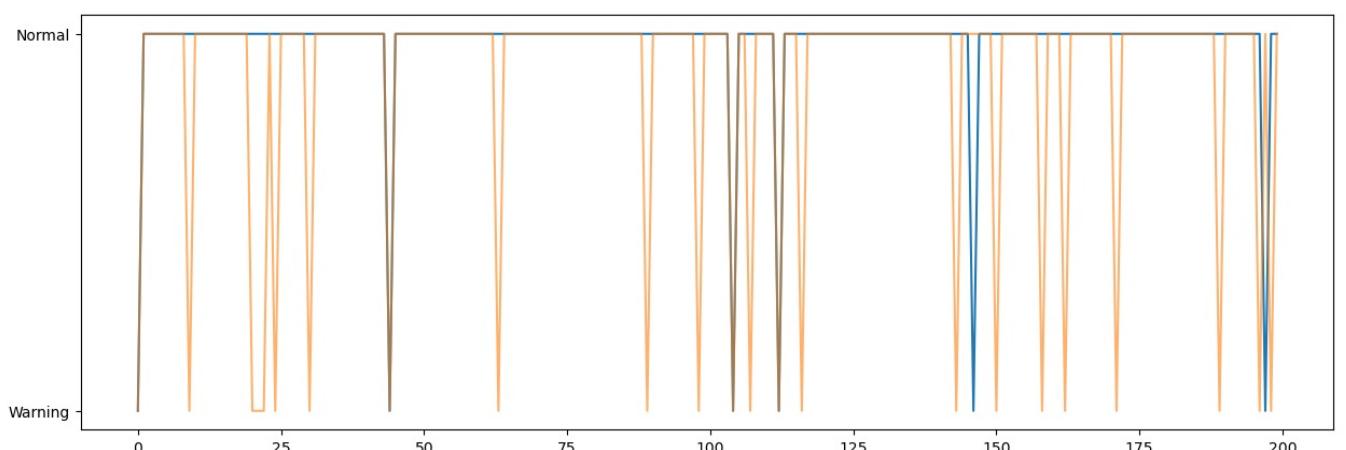


```
In [25]: pred_gb_inv = label_scaler_cut.inverse_transform(pred_gb.reshape(-1, 1))
y_test_cut_inv = label_scaler_cut.inverse_transform(y_test_cut.values.reshape(-1, 1))

y_test_cut_inv_class = ['Normal' if x >= 72 else 'Warning' for x in y_test_cut_inv]
pred_gb_inv_class = ['Normal' if x >= 72 else 'Warning' for x in pred_gb_inv]

plt.plot(pred_gb_inv_class[:200])
plt.plot(y_test_cut_inv_class[:200], alpha = 0.6)
```

Out[25]: <matplotlib.lines.Line2D at 0x1f1589c2fa0>

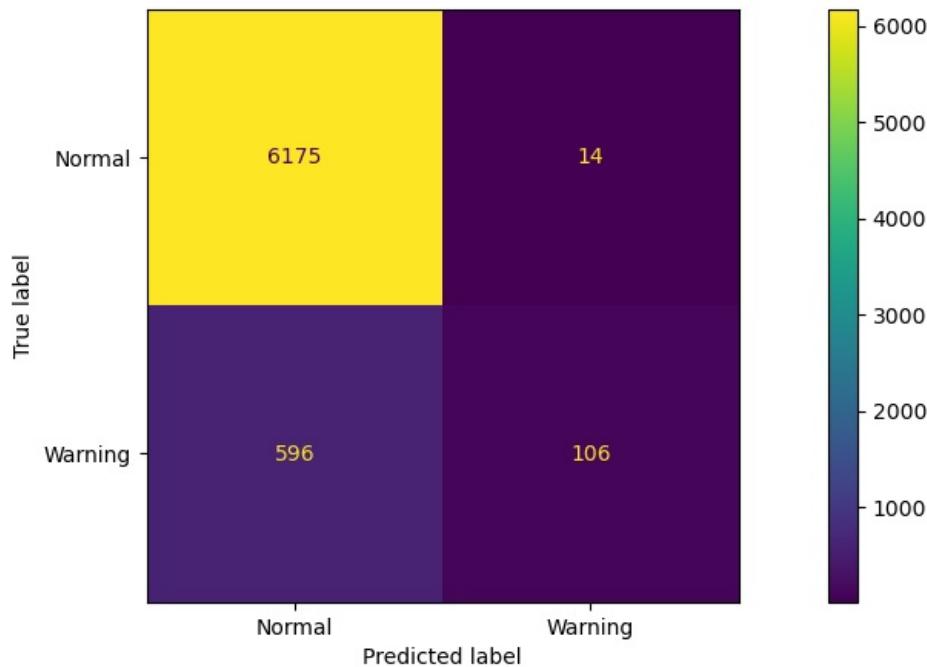


```
In [26]: gb_cm = confusion_matrix(y_test_cut_inv_class, pred_gb_inv_class)

print('Accuracy: ', accuracy_score(y_test_cut_inv_class, pred_gb_inv_class))
ConfusionMatrixDisplay(gb_cm, display_labels=['Normal', 'Warning']).plot()
```

Accuracy: 0.9114787403860107

Out[26]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f15831a2e0>

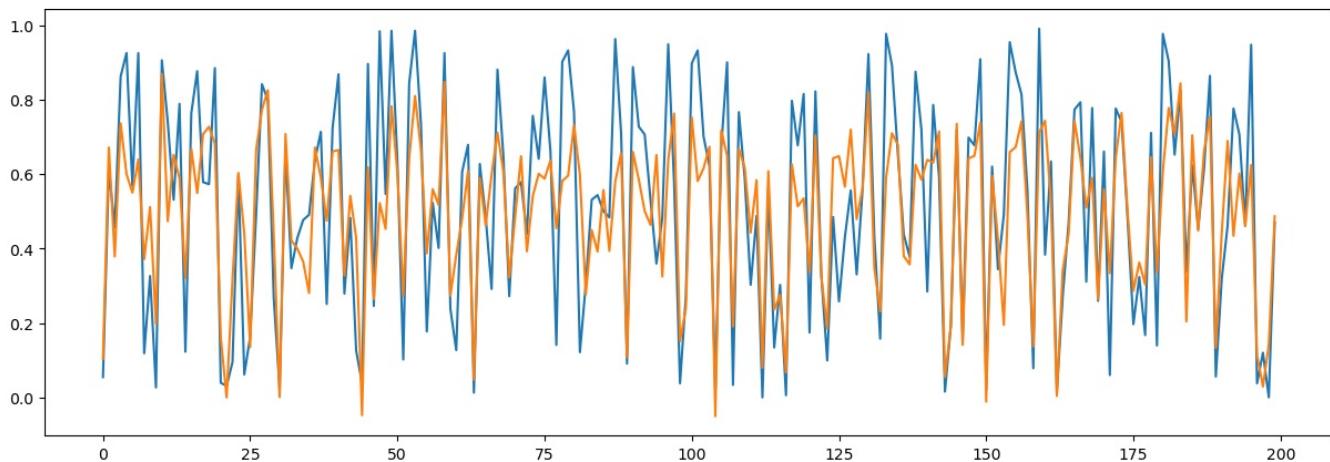


XGBoosting Regressor

```
In [27]: model_xgb = XGBRegressor().fit(X_train, y_train.to_numpy().ravel())
pred_xgb = model_xgb.predict(X_test)
```

```
In [28]: plot_metrics(model_xgb, X_test[:200], y_test.to_numpy().ravel()[:200], pred_xgb[:200])
```

R^2 score: 0.6682029106631473
 MSE: 0.028567648393930875
 RMSE: 0.1690196686599843
 MAE: 0.1320614130136721



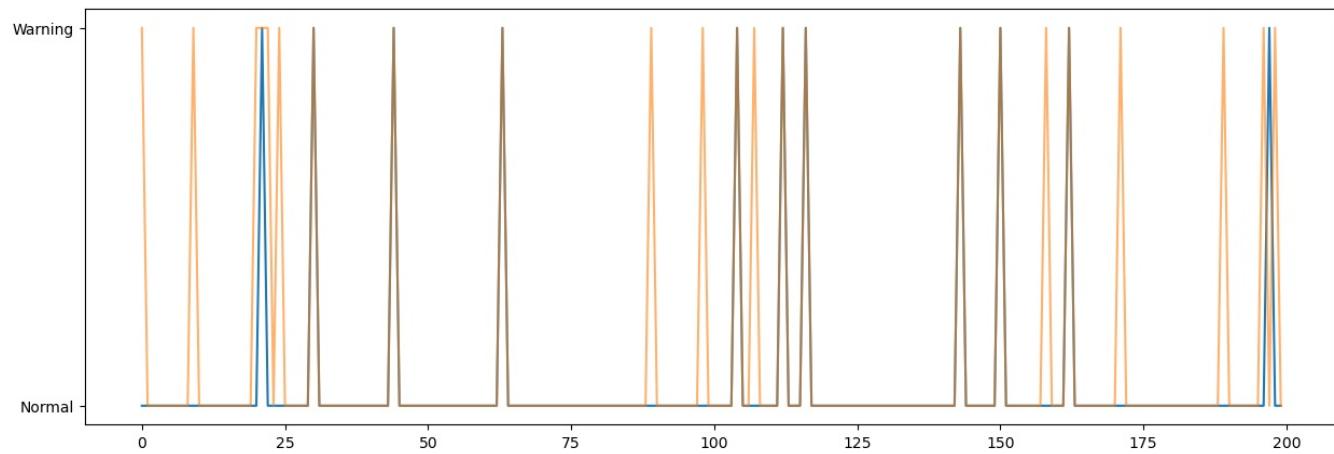
```
In [29]: pred_xgb_inv = label_scaler_cut.inverse_transform(pred_xgb.reshape(-1, 1))
y_test_cut_inv = label_scaler_cut.inverse_transform(y_test_cut.values.reshape(-1, 1))

y_test_cut_inv_class = ['Normal' if x >= 72 else 'Warning' for x in y_test_cut_inv]
pred_xgb_inv_class = ['Normal' if x >= 72 else 'Warning' for x in pred_xgb_inv]

plt.plot(pred_xgb_inv_class[:200])
```

```
plt.plot(y_test_cut_inv_class[:200], alpha = 0.6)
```

Out[29]: <matplotlib.lines.Line2D at 0x1f15bea7af0>

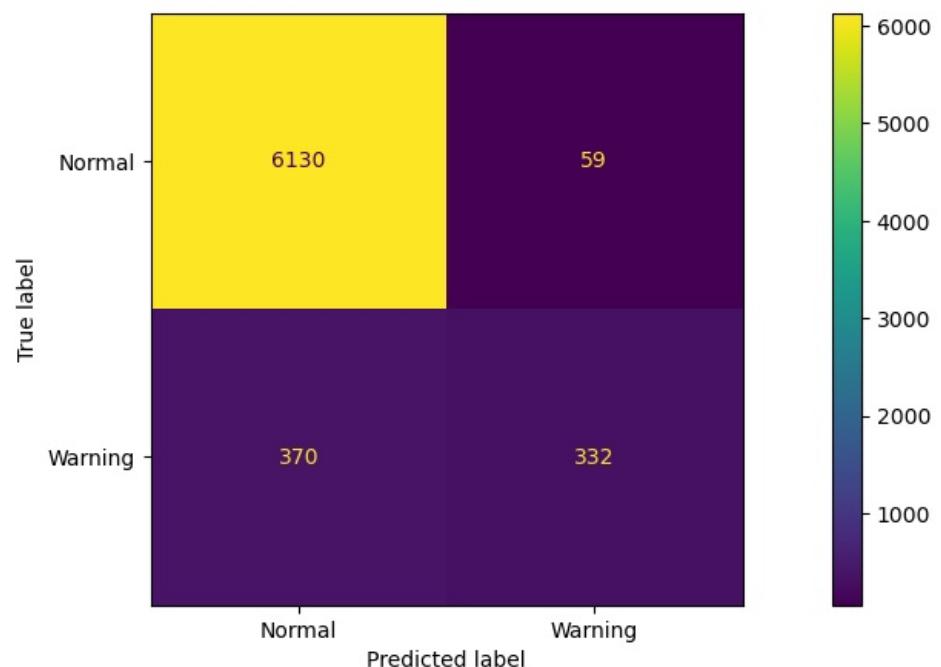


In [30]: `xgb_cm = confusion_matrix(y_test_cut_inv_class, pred_xgb_inv_class)`

```
print('Accuracy: ', accuracy_score(y_test_cut_inv_class, pred_xgb_inv_class))
ConfusionMatrixDisplay(xgb_cm, display_labels=['Normal', 'Warning']).plot()
```

Accuracy: 0.9377448846321289

Out[30]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f15beb040>



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Carga Librerias

```
In [1]: # Librerias
import warnings
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import plotly.express as px

import seaborn as sns

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.decomposition import PCA

from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score, precision_score, recall_score
warnings.filterwarnings('ignore')
```

```
In [2]: def plot_metrics(model, x, y, pred_y):
    print('Accuracy:\t', accuracy_score(y, pred_y))
    print('Precision:\t', precision_score(y, pred_y))
    print('Recall:\t', recall_score(y, pred_y))
    print('F1 score:\t', f1_score(y, pred_y))
    print('ROC AUC score:\t', roc_auc_score(y, model.predict_proba(x)[:, 1]))

    fig, ax = plt.subplots(1)
    matplotlib.rc('figure', figsize=(15, 5))
    ax.plot(y)
    ax.plot(pred_y)
    plot_roc_curve(model, x, y)
```

Carga Datos

```
In [3]: DATA_DIR = "C:/Users/NetRunner/OneDrive/UOC/Semestre 6/TFM/MultipleDatasets"

train_data_cut = pd.read_csv(f"{DATA_DIR}/train_data.csv")
test_data_cut = pd.read_csv(f"{DATA_DIR}/test_data.csv")

train_data_uncut = pd.read_csv(f"{DATA_DIR}/train_data_uncut.csv")
test_data_uncut = pd.read_csv(f"{DATA_DIR}/test_data_uncut.csv")

# X_train = pd.read_csv(f"{DATA_DIR}/X_train.csv")
# y_train = pd.read_csv(f"{DATA_DIR}/y_train.csv")
# X_test = pd.read_csv(f"{DATA_DIR}/X_test.csv")
# y_test = pd.read_csv(f"{DATA_DIR}/y_test.csv")
```

```
In [4]: data_cut = pd.concat([train_data_cut, test_data_cut])
data_uncut = pd.concat([train_data_uncut, test_data_uncut])

features = ['volt', 'rotate', 'pressure', 'vibration', 'error1', 'error2', 'error3',
            'error4', 'error5', 'volt_3h_mean', 'rotate_3h_mean',
            'pressure_3h_mean', 'vibration_3h_mean', 'volt_24h_mean',
            'rotate_24h_mean', 'pressure_24h_mean', 'vibration_24h_mean',
            'error1_count', 'error2_count', 'error3_count', 'error4_count',
            'error5_count']
label = ['State']

data_cut = data_cut[features+label]
data_uncut = data_uncut[features+label]
```

Normalización de datos MinMax

```
In [5]: feature_scaler_cut = MinMaxScaler(feature_range=(0,1))
label_scaler_cut = MinMaxScaler(feature_range=(0,1))

feature_scaler_cut.fit(data_cut[features])
label_scaler_cut.fit(data_cut[label].values.reshape(-1,1))

Out[5]: MinMaxScaler()
```

```
In [6]: feature_scaler_uncut = MinMaxScaler(feature_range=(0,1))
label_scaler_uncut = MinMaxScaler(feature_range=(0,1))

feature_scaler_uncut.fit(data_uncut[features])
label_scaler_uncut.fit(data_uncut[label].values.reshape(-1,1))

Out[6]: MinMaxScaler()
```

```
In [7]: data_norm_cut = data_cut.copy()
data_norm_cut[features] = feature_scaler_cut.transform(data_cut[features])
data_norm_cut[label] = label_scaler_cut.transform(data_cut[label].values.reshape(-1,1))

data_norm_uncut = data_uncut.copy()
data_norm_uncut[features] = feature_scaler_uncut.transform(data_uncut[features])
data_norm_uncut[label] = label_scaler_uncut.transform(data_uncut[label].values.reshape(-1,1))
```

Dataset Train/Test

```
In [8]: train_norm_cut = data_norm_cut[:len(train_data_cut)]
test_norm_cut = data_norm_cut[len(train_data_cut):(len(train_data_cut)+len(test_data_cut))]

X_train_cut = train_norm_cut.loc[:, train_norm_cut.columns != 'State']
y_train_cut = train_norm_cut['State'].values
X_test_cut = test_norm_cut.loc[:, test_norm_cut.columns != 'State']
y_test_cut = test_norm_cut['State'].astype(int)

print('X_train:\t', X_train_cut.shape)
print('y_train:\t', y_train_cut.shape)
print('X_test:\t\t', X_test_cut.shape)
print('y_test:\t\t', y_test_cut.shape)

X_train:      (17280, 22)
y_train:      (17280,)
X_test:       (3600, 22)
y_test:       (3600,)
```

```
In [9]: train_norm_uncut = data_norm_uncut[:len(train_data_uncut)]
test_norm_uncut = data_norm_uncut[len(train_data_uncut):(len(train_data_uncut)+len(test_data_uncut))]

X_train_uncut = train_norm_uncut.loc[:, train_norm_uncut.columns != 'State']
y_train_uncut = train_norm_uncut['State'].astype(int)
X_test_uncut = test_norm_uncut.loc[:, test_norm_uncut.columns != 'State']
y_test_uncut = test_norm_uncut['State'].astype(int)

print('X_train_uncut:\t', X_train_uncut.shape)
print('y_train_uncut:\t', y_train_uncut.shape)
print('X_test_uncut:\t', X_test_uncut.shape)
print('y_test_uncut:\t', y_test_uncut.shape)

X_train_uncut:   (40397, 22)
y_train_uncut:   (40397,)
X_test_uncut:    (8305, 22)
y_test_uncut:    (8305,)
```

Train/Test variable control

```
In [10]: # Cut cycles
X_train = X_train_cut
y_train = y_train_cut
X_test = X_test_cut
y_test = y_test_cut

# Uncut cycles
# X_train = X_train_uncut
# y_train = y_train_uncut
# X_test = X_test_uncut
# y_test = y_test_uncut
```

Logistic Regression

```
In [11]: parameters = {'solver':['newton-cg', 'lbfgs', 'liblinear'],'C':[0.001, 0.01, 0.1, 1, 10]}
grid_search = GridSearchCV(LogisticRegression(), parameters, verbose=0)
grid_search.fit(X_train, y_train)
grid_df = pd.DataFrame(grid_search.cv_results_['params'])
grid_df['mean_test_score'] = grid_search.cv_results_['mean_test_score']
grid_df['rank'] = grid_search.cv_results_['rank_test_score']
grid_df.sort_values(by='rank').head()
```

Out[11]:

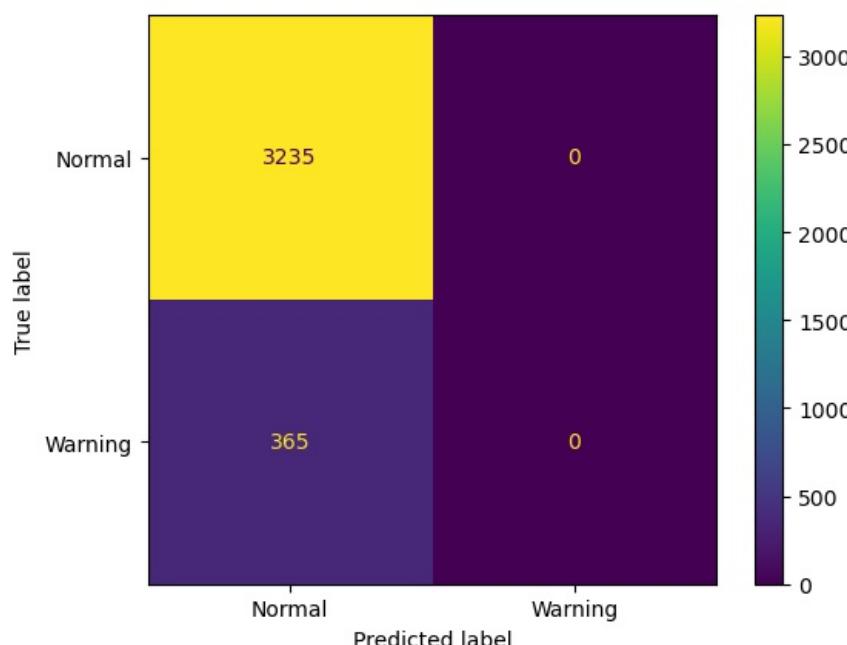
	C	solver	mean_test_score	rank
3	0.010	newton-cg	0.899884	1
4	0.010	lbfgs	0.899884	1
5	0.010	liblinear	0.898958	3
0	0.001	newton-cg	0.898611	4
1	0.001	lbfgs	0.898611	4

In [12]:

```
model_lr = LogisticRegression(solver='newton-cg', C=0.01).fit(X_train, y_train)
pred_lr = model_lr.predict(X_test)
lr_cm = confusion_matrix(y_test, pred_lr)
ConfusionMatrixDisplay(lr_cm, display_labels=['Normal', 'Warning']).plot()
```

Out[12]:

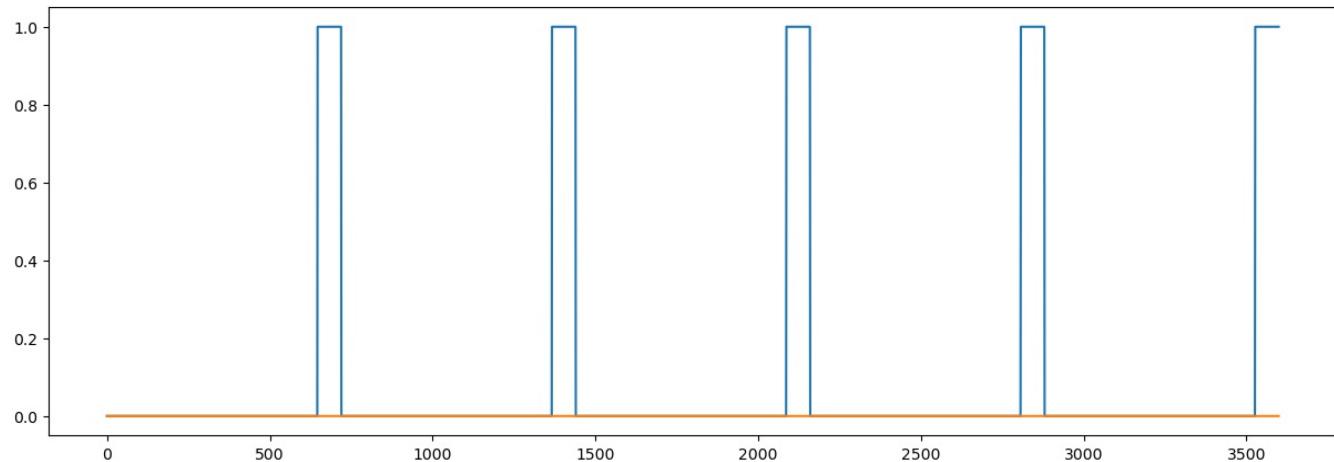
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1e09a866190>
```

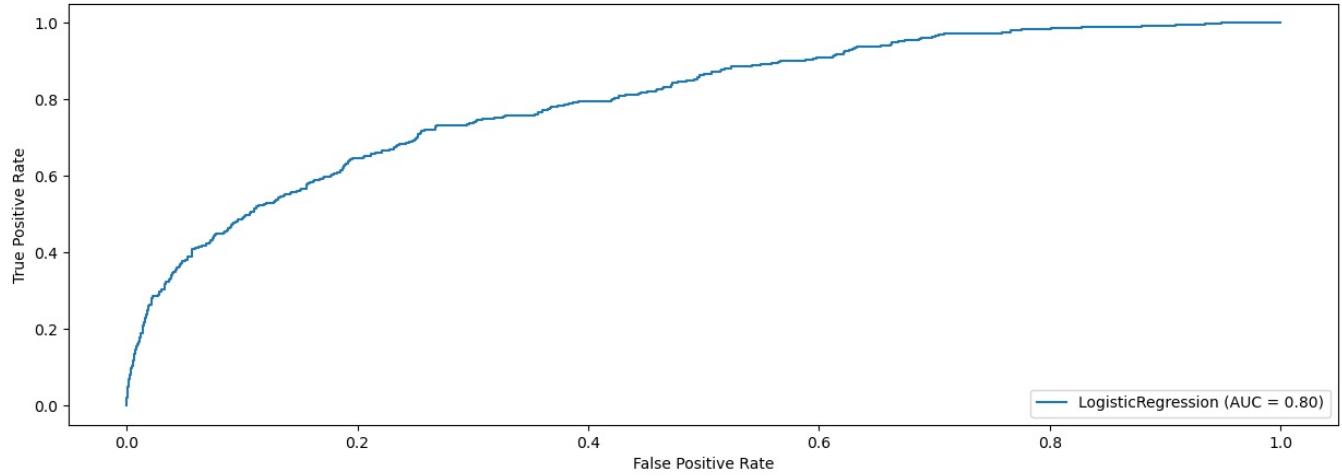


In [14]:

```
plot_metrics(model_lr, X_test, y_test, pred_lr)
```

Accuracy: 0.8986111111111111
Precision: 0.0
Recall: 0.0
F1 score: 0.0
ROC AUC score: 0.7990040439541828





SVM

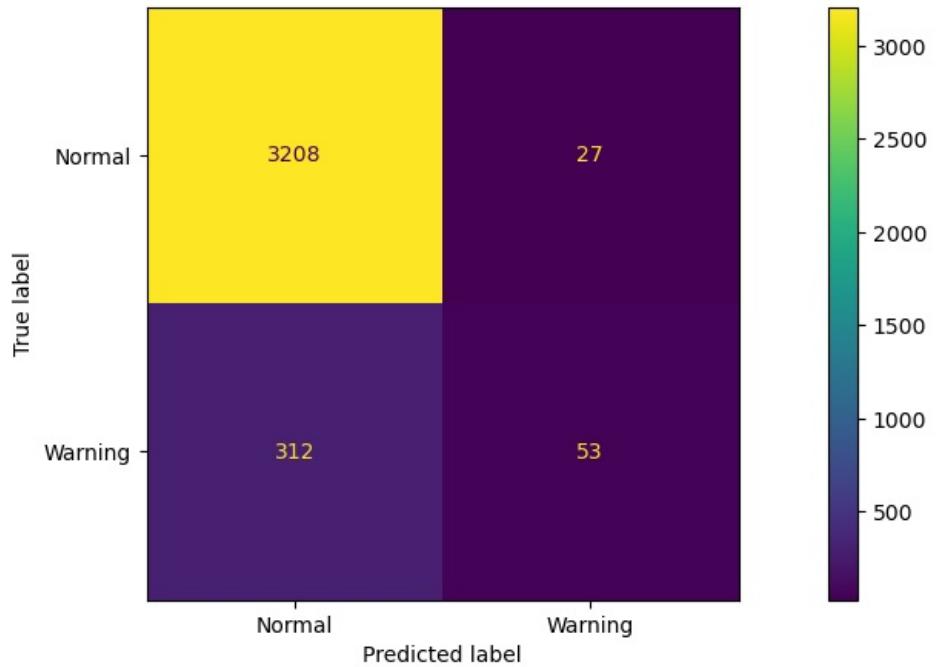
```
In [14]: parameters = {'kernel': ['poly', 'rbf', 'sigmoid'], 'C':[0.001, 0.01, 0.1, 1, 10]}

grid_search = GridSearchCV(svm.SVC(), parameters, verbose=0)
grid_search.fit(X_train, y_train)
grid_df = pd.DataFrame(grid_search.cv_results_['params'])
grid_df['mean_test_score'] = grid_search.cv_results_['mean_test_score']
grid_df['rank'] = grid_search.cv_results_['rank_test_score']
grid_df.head().sort_values(by='rank').head()
```

```
Out[14]:      C   kernel  mean_test_score  rank
  3  0.010    poly      0.904398     1
  0  0.001    poly      0.899826     2
  1  0.001     rbf      0.898611     3
  2  0.001   sigmoid     0.898611     3
  4  0.010     rbf      0.898611     3
```

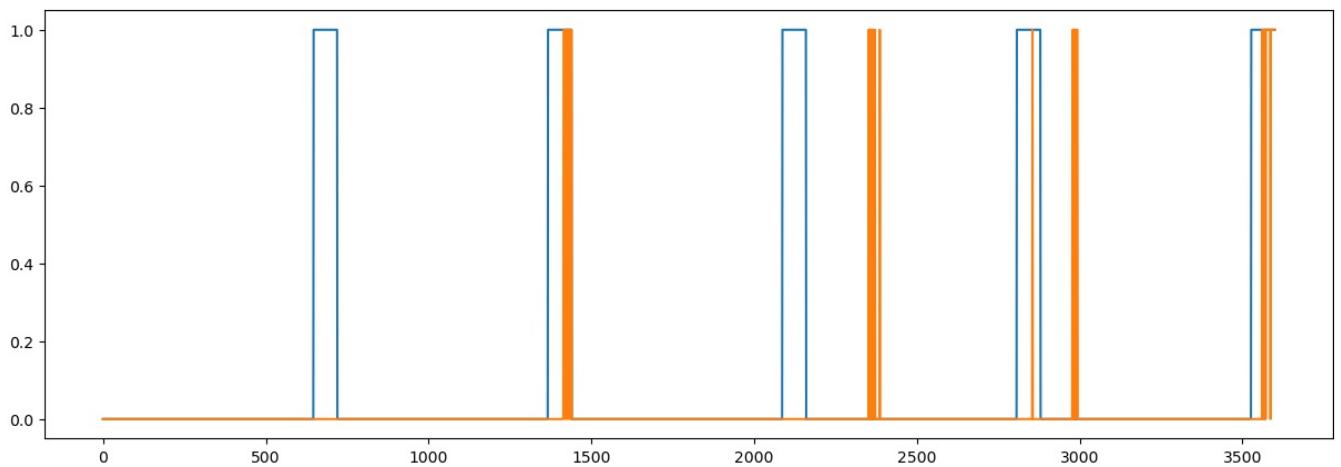
```
In [15]: model_svm = svm.SVC(kernel='poly', C=0.01, probability=True).fit(X_train, y_train)
pred_svm = model_svm.predict(X_test)
svm_cm = confusion_matrix(y_test, pred_svm)
ConfusionMatrixDisplay(svm_cm, display_labels=['Normal', 'Warning']).plot()
```

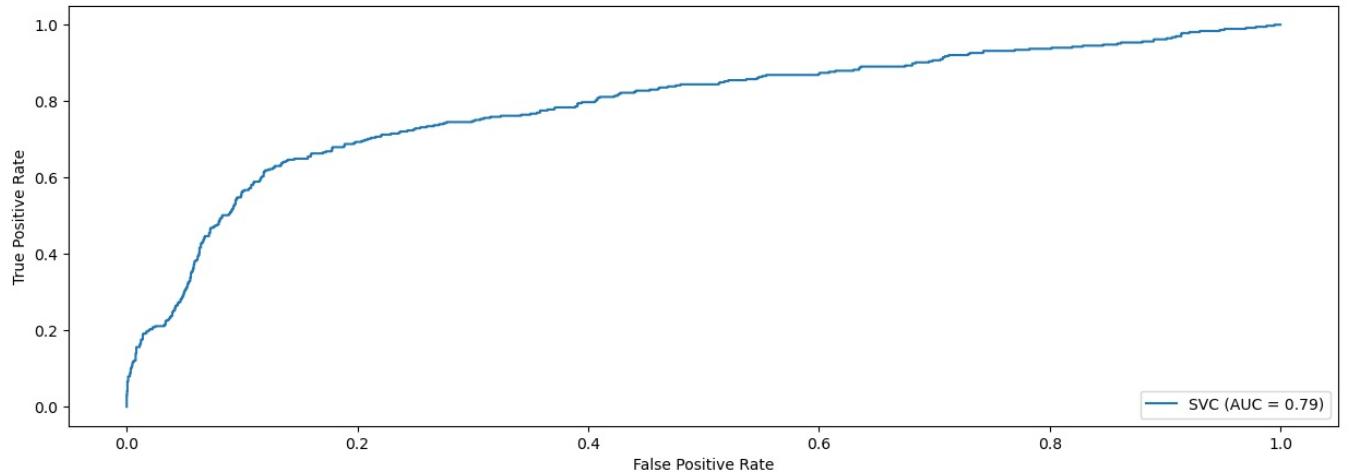
```
Out[15]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1e09d30faf0>
```



```
In [16]: plot_metrics(model_svm, X_test, y_test, pred_svm)
```

```
Accuracy:      0.9058333333333334
Precision:     0.6625
Recall:        0.14520547945205478
F1 score:      0.23820224719101124
ROC AUC score: 0.7900726217950075
```





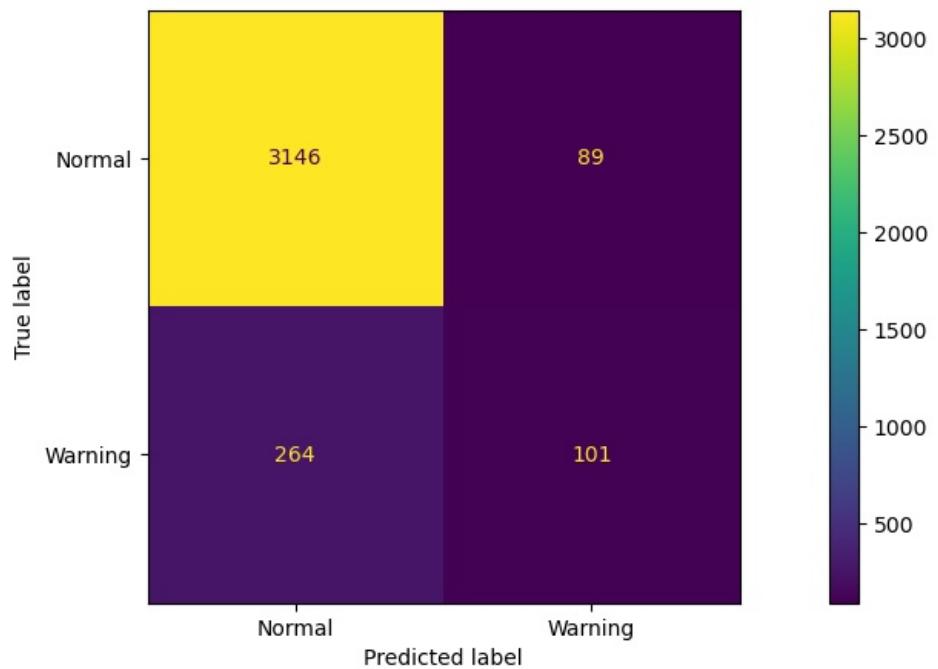
Random Forest Classifier

```
In [17]: parameters = {'max_depth': [6,7,8,9,10,11,12], 'n_estimators': [10,50,100,200]}
grid_search = GridSearchCV(RandomForestClassifier(), parameters, verbose=0)
grid_search.fit(X_train, y_train)
grid_df = pd.DataFrame(grid_search.cv_results_['params'])
grid_df['mean_test_score'] = grid_search.cv_results_['mean_test_score']
grid_df['rank'] = grid_search.cv_results_['rank_test_score']
grid_df.sort_values(by='rank').head()
```

```
Out[17]:   max_depth  n_estimators  mean_test_score  rank
0            6           10      0.883275      1
2            6          100      0.882986      2
7            7          200      0.881539      3
3            6          200      0.881366      4
6            7          100      0.879919      5
```

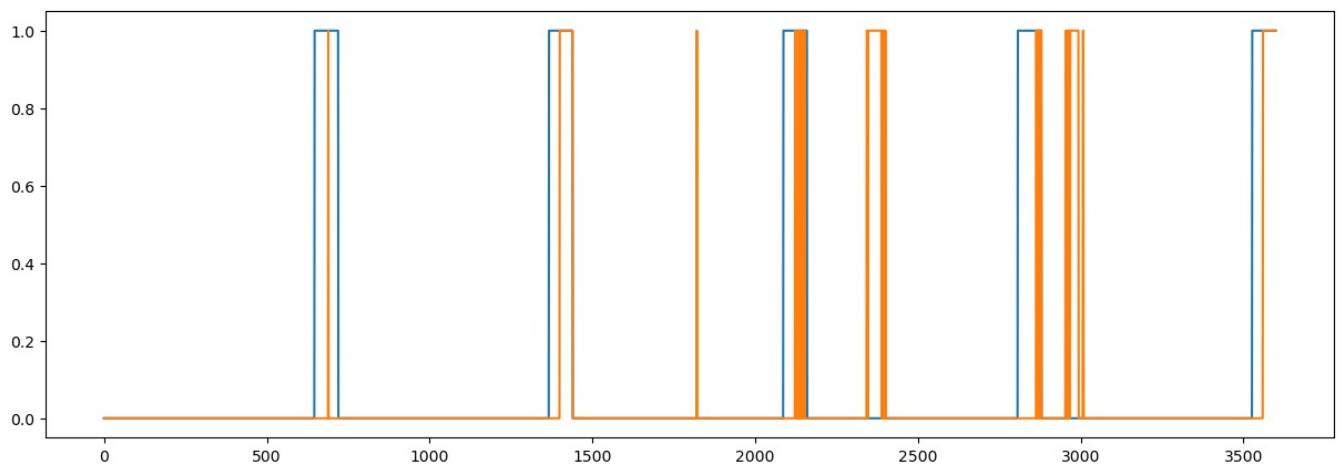
```
In [17]: model_rf = RandomForestClassifier(max_depth=6, n_estimators=10).fit(X_train, y_train)
pred_rf = model_rf.predict(X_test)
rf_cm = confusion_matrix(y_test, pred_rf)
ConfusionMatrixDisplay(rf_cm, display_labels=['Normal', 'Warning']).plot()
```

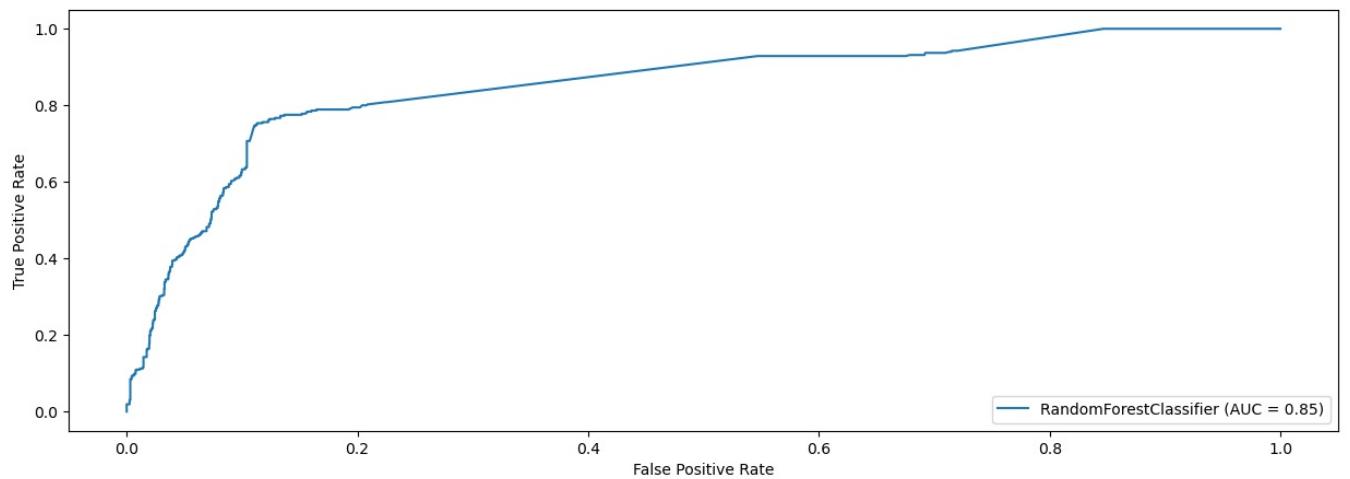
```
Out[17]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1e09d406b50>
```



```
In [18]: plot_metrics(model_rf, X_test, y_test, pred_rf)
```

```
Accuracy:      0.9019444444444444
Precision:     0.531578947368421
Recall:        0.27671232876712326
F1 score:      0.3639639639639639
ROC AUC score: 0.8521936863500666
```





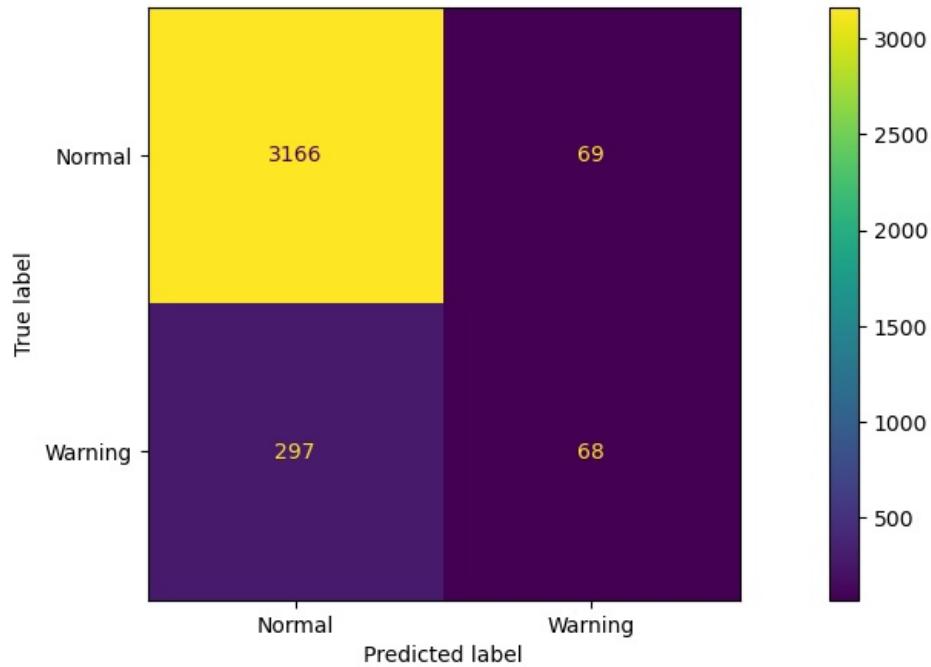
Gradient boosting

```
In [20]: parameters = {'learning_rate': [0.001, 0.01, 0.1, 1, 2], 'n_estimators': [10, 50, 100, 200]}
grid_search = GridSearchCV(GradientBoostingClassifier(), parameters, verbose=0)
grid_search.fit(X_train, y_train)
grid_df = pd.DataFrame(grid_search.cv_results_['params'])
grid_df['mean_test_score'] = grid_search.cv_results_['mean_test_score']
grid_df['rank'] = grid_search.cv_results_['rank_test_score']
grid_df.sort_values(by='rank').head()
```

```
Out[20]:   learning_rate  n_estimators  mean_test_score  rank
6           0.010         100        0.908275      1
8           0.100          10        0.906192      2
7           0.010         200        0.903009      3
0           0.001          10        0.898611      4
1           0.001          50        0.898611      4
```

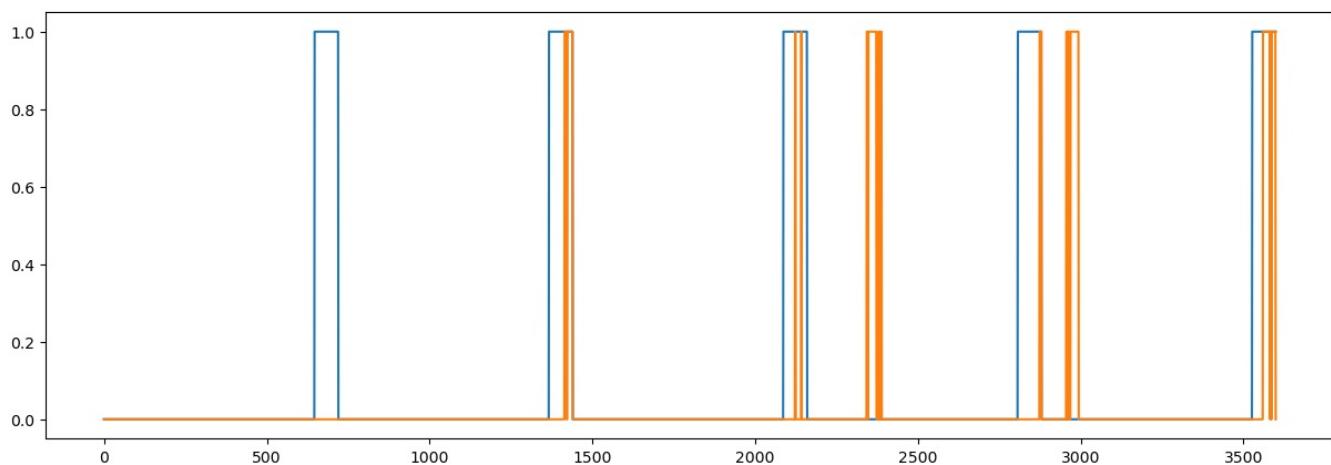
```
In [21]: model_gb = GradientBoostingClassifier(learning_rate=0.010, n_estimators=100).fit(X_train, y_train)
pred_gb = model_gb.predict(X_test)
gb_cm = confusion_matrix(y_test, pred_gb)
ConfusionMatrixDisplay(gb_cm, display_labels=['Normal', 'Warning']).plot()
```

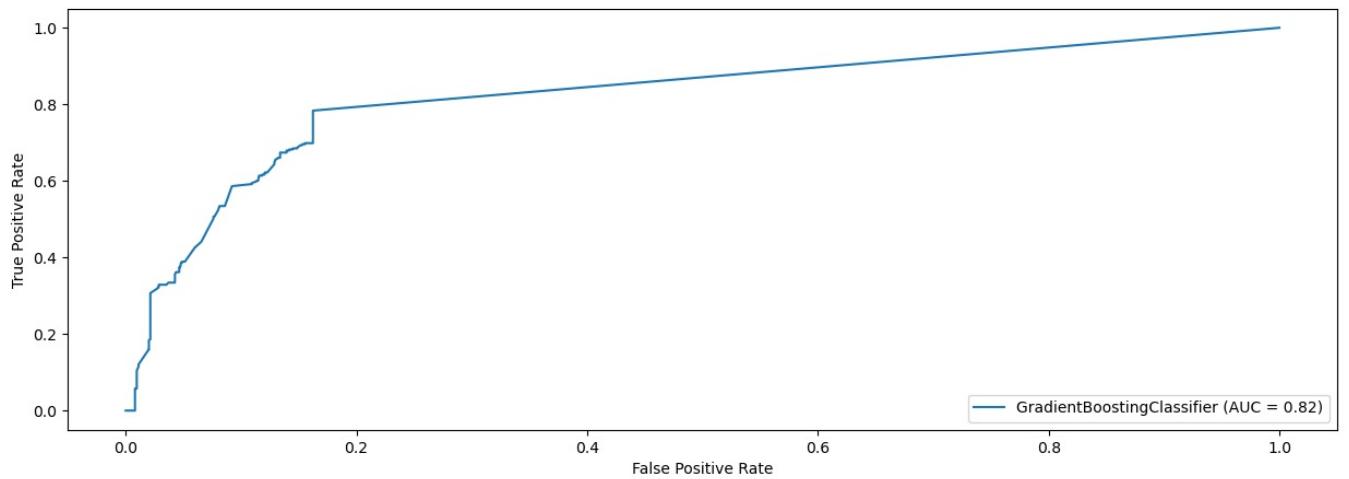
```
Out[21]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1e09e1c4dc0>
```



```
In [22]: plot_metrics(model_gb, X_test, y_test, pred_gb)
```

```
Accuracy: 0.8983333333333333
Precision: 0.49635036496350365
Recall: 0.1863013698630137
F1 score: 0.2709163346613546
ROC AUC score: 0.8235298850331352
```





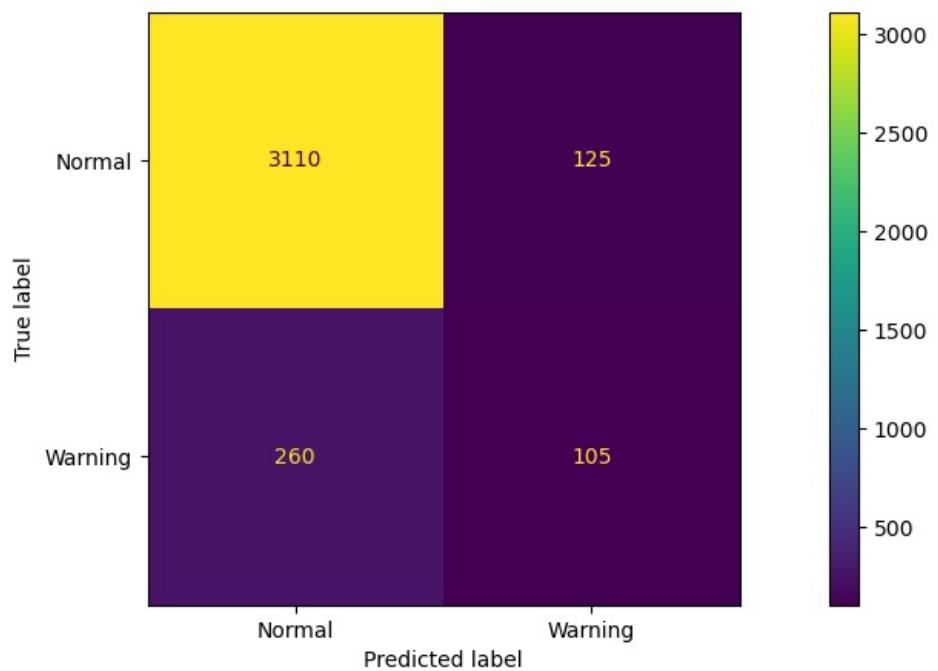
XGBoosting

```
In [23]: parameters = {'booster': ['gbtree', 'dart'], 'reg_alpha': [0, 0.5, 1, 5], 'reg_lambda': [0, 0.5, 1, 5], 'eval_metric': 'mlogloss'}
grid_search = GridSearchCV(XGBClassifier(use_label_encoder=False), parameters, verbose=0)
grid_search.fit(X_train, y_train.astype(int))
grid_df = pd.DataFrame(grid_search.cv_results_['params'])
grid_df['mean_test_score'] = grid_search.cv_results_['mean_test_score']
grid_df['rank'] = grid_search.cv_results_['rank_test_score']
grid_df.sort_values(by='rank').head()
```

```
Out[23]:   booster eval_metric reg_alpha reg_lambda mean_test_score rank
 22      dart    mlogloss     0.5         1.0       0.874306    1
  6     gbtree    mlogloss     0.5         1.0       0.874306    1
 27      dart    mlogloss     1.0         5.0       0.868229    3
 11     gbtree    mlogloss     1.0         5.0       0.868229    3
 28      dart    mlogloss     5.0         0.0       0.867650    5
```

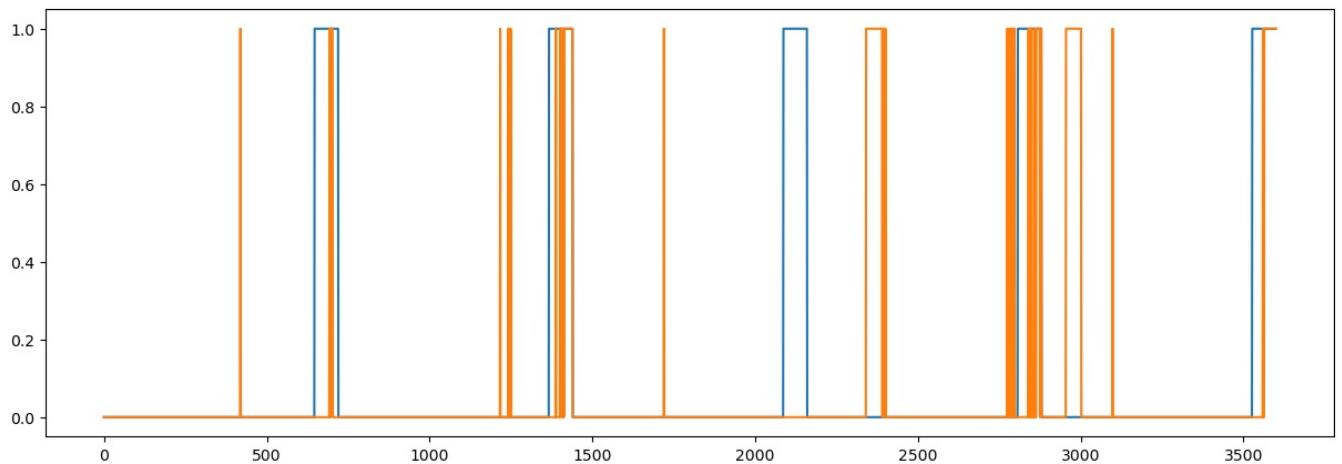
```
In [23]: model_xgb = XGBClassifier(booster='dart', reg_alpha=0.5, reg_lambda=1, eval_metric='mlogloss').fit(X_train, y_train)
pred_xgb = model_xgb.predict(X_test)
xgb_cm = confusion_matrix(y_test, pred_xgb)
ConfusionMatrixDisplay(xgb_cm, display_labels=['Normal', 'Warning']).plot()
```

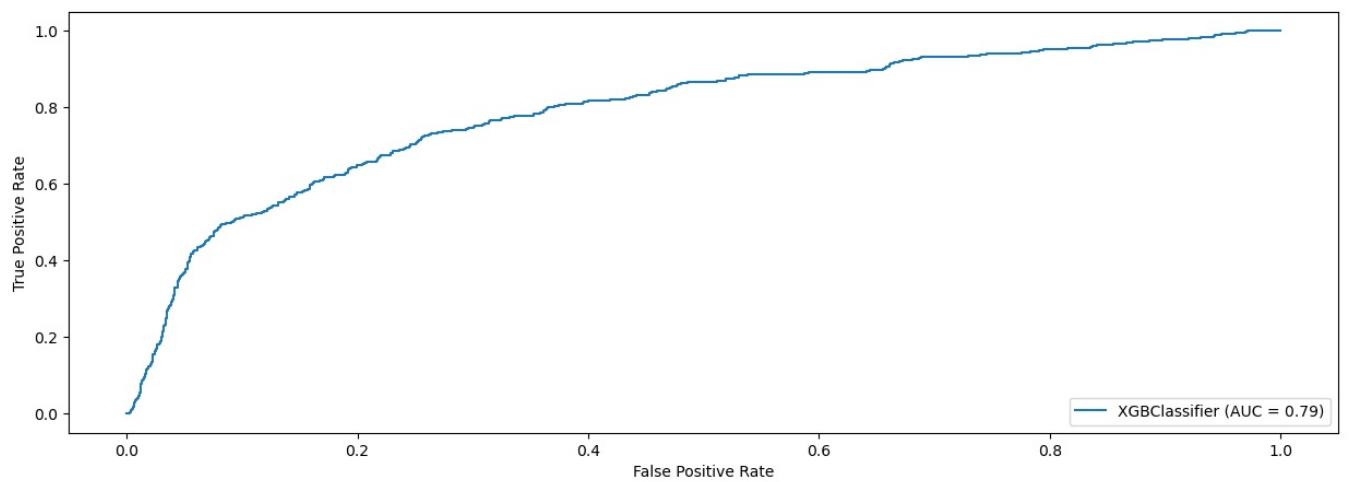
```
Out[23]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1e09e3d7d60>
```



```
In [24]: plot_metrics(model_xgb, X_test, y_test, pred_xgb)
```

```
Accuracy:      0.8930555555555556
Precision:     0.45652173913043476
Recall:        0.2876712328767123
F1 score:      0.3529411764705882
ROC AUC score: 0.7902123605259256
```





Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Carga de librerias

```
In [1]: # Librerias
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import plotly.express as px

import seaborn as sns
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, confusion_matrix, plot_confusion_matrix

from keras.layers import Input, Dense, LSTM, TimeDistributed, RepeatVector
from keras.models import Model
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Bidirectional, Lambda
from keras.losses import MeanSquaredError
from keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam, RMSprop
```

```
In [2]: # Plot del training loss i l'accuracy
def plot_prediction(n_epochs, mfit):
    # TODO
    #Plots
    fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(15,15))

    # plot accuracy during training
    ax[0].set_title('Mse')
    ax[0].plot(mfit.history['mse'], label='train')
    ax[0].plot(mfit.history['val_mse'], label='test')
    ax[0].legend()

    # plot loss during training
    ax[1].set_title('Loss')
    ax[1].plot(mfit.history['loss'], label='train')
    ax[1].plot(mfit.history['val_loss'], label='test')
    ax[1].legend()

def cut_cycles(df, lookback, future, column_features, column_label):
    df_feature = df[column_features]
    df_rul = df[column_label][lookback-1:]

    # Convertimos el dataframe en un numpy array
    numpy_features = df_feature.to_numpy()
    labels = df_rul.to_numpy()

    # Creación de listas vacías auxiliares
    features_set = []

    for i in range(lookback, df.shape[0] - future + 1):
        features_set.append(numpy_features[i - lookback:i])

    # Redimensionamiento numpy arrays
    features = np.array(features_set)
    features = np.reshape(features, (features.shape[0], features.shape[1], len(column_features) ))

    return features, labels

def modelo_lstm(input_shape, optimizer):
    model = Sequential()
    model.add(LSTM(units=100, input_shape=input_shape, return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=30, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='relu'))
    model.add(Activation('relu'))
    model.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['mse'])

    return model
```

Carga de datos

```
In [3]: DATA_DIR = "C:/Users/NetRunner/OneDrive/UOC/Semestre 6/TFM/MultipleDatasets"

train_data = pd.read_csv(f"{DATA_DIR}/train_data.csv")
test_data = pd.read_csv(f"{DATA_DIR}/test_data.csv")
```

```
# X_train = pd.read_csv(f"{DATA_DIR}/X_train.csv")
# y_train = pd.read_csv(f"{DATA_DIR}/y_train.csv")
# X_test = pd.read_csv(f"{DATA_DIR}/X_test.csv")
# y_test = pd.read_csv(f"{DATA_DIR}/y_test.csv")
```

```
In [5]: data = pd.concat([train_data, test_data])
features = ['volt', 'rotate', 'pressure', 'vibration', 'error1', 'error2', 'error3', 'error4', 'error5',
            'volt_3h_mean', 'rotate_3h_mean', 'pressure_3h_mean', 'vibration_3h_mean', 'volt_24h_mean',
            'rotate_24h_mean', 'pressure_24h_mean', 'vibration_24h_mean', 'error1_count', 'error2_count',
            'error3_count', 'error4_count', 'error5_count']
label = ['RUL']

print(len(train_data))
print(len(test_data))
print(len(data))
```

17280
3600
20880

```
In [6]: feature_scaler = MinMaxScaler(feature_range=(0,1))
label_scaler = MinMaxScaler(feature_range=(0,1))

feature_scaler.fit(data[features])
label_scaler.fit(data[label].values.reshape(-1,1))
```

```
Out[6]: MinMaxScaler()
```

```
In [7]: data_norm = data[features+label].copy()
data_norm[features] = feature_scaler.transform(data[features])
data_norm[label] = label_scaler.transform(data[label].values.reshape(-1,1))
data_norm
```

```
Out[7]:   volt  rotate  pressure  vibration  error1  error2  error3  error4  error5  volt_3h_mean ...  volt_24h_mean  rotate_24h_mean  pres
0  0.384246  0.635590  0.287496  0.550316  0.0  0.0  0.0  0.0  0.0  0.319571 ...  0.370829  0.428252
1  0.196252  0.595322  0.298321  0.255010  0.0  0.0  0.0  0.0  0.0  0.214811 ...  0.341286  0.432249
2  0.395865  0.468667  0.396966  0.388182  0.0  0.0  0.0  0.0  0.0  0.188577 ...  0.332091  0.419061
3  0.487156  0.700909  0.545746  0.547169  0.0  0.0  0.0  0.0  0.0  0.235855 ...  0.322535  0.439945
4  0.479313  0.592491  0.348040  0.421442  0.0  0.0  0.0  0.0  0.0  0.365896 ...  0.326367  0.442053
...
3595  0.590798  0.456078  0.351645  0.786720  0.0  0.0  0.0  0.0  0.0  0.383354 ...  0.375683  0.383491
3596  0.455103  0.375885  0.410713  0.749031  0.0  0.0  0.0  0.0  0.0  0.414840 ...  0.372778  0.371331
3597  0.509347  0.491140  0.574390  0.500942  0.0  0.0  0.0  0.0  0.0  0.454524 ...  0.366536  0.368975
3598  0.638858  0.573666  0.258521  0.588043  0.0  0.0  0.0  0.0  0.0  0.476603 ...  0.381726  0.371157
3599  0.442910  0.657773  0.184867  0.551103  0.0  0.0  0.0  0.0  0.0  0.471001 ...  0.371682  0.392093
```

20880 rows × 23 columns

```
In [9]: train_norm = data_norm[:len(train_data)]
test_norm = data_norm[len(train_data):(len(train_data)+len(test_data))]

X_train = train_norm.loc[:, train_norm.columns != 'RUL']
y_train = train_norm.loc[:, train_norm.columns == 'RUL']
X_test = test_norm.loc[:, test_norm.columns != 'RUL']
y_test = test_norm.loc[:, test_norm.columns == 'RUL']
```

Modelo LSTM

```
In [18]: rangos = 72

train_3d = cut_cycles(train_norm, rangos, 0, features, label)

X_train_3d = train_3d[0]
y_train_3d = train_3d[1]

test_3d = cut_cycles(test_norm, rangos, 0, features, label)

X_test_3d = test_3d[0]
y_test_3d = test_3d[1]

print('X_train_3d:\t', X_train_3d.shape)
print('y_train_3d:\t', y_train_3d.shape)
```

```

print('X_test_3d:\t', X_test_3d.shape)
print('y_test_3d:\t', y_test_3d.shape)

X_train_3d:      (17209, 72, 22)
y_train_3d:      (17209, 1)
X_test_3d:       (3529, 72, 22)
y_test_3d:       (3529, 1)

```

In [11]:

```

%%time
epochs = [100]
batches = [8, 16]
optimizers = ['adam', 'rmsprop']
lrs = [0.01, 0.001, 0.0001]
input_shape=(X_train_3d.shape[1], X_train_3d.shape[2])

for epoch in epochs:
    for batch in batches:
        for opt in optimizers:
            for lr in lrs:
                if opt == 'adam':
                    optimizer = Adam(learning_rate=lr)
                elif opt == 'rmsprop':
                    optimizer = RMSprop(learning_rate=lr)

                model = modelo_lstm(input_shape, optimizer)

                print('=====Training model=====')

                print('Hiperparámetros:')
                print('Optimizer:\t', opt)
                print('Learning Rate:\t', lr)
                print('Epochs:\t\t', epoch)
                print('Batch:\t\t', batch)

                res_m = model.fit(X_train_3d, y_train_3d, validation_data=(X_test_3d, y_test_3d), epochs=epoch,
                pred = model.predict(X_test_3d)
                pred
                pred_list = [x[0] for x in pred]

                print('\nEvaluation:')
                print('R^2 score:\t\t', r2_score(y_test_3d, pred_list))
                print('MSE score:\t\t', mean_squared_error(y_test_3d, pred_list))
                print('MAE score:\t\t', mean_absolute_error(y_test_3d, pred_list))

                print('=====')

```

=====Training model=====

Hiperparámetros:

```

Optimizer: adam
Learning Rate: 0.01
Epochs: 100
Batch: 8
Restoring model weights from the end of the best epoch: 4.
Epoch 14: early stopping
111/111 [=====] - 2s 13ms/step

```

Evaluation:

```

R^2 score: 0.45022060900476635
MSE score: 0.044559361514754245
MAE score: 0.16651966790417883
=====
```

=====Training model=====

Hiperparámetros:

```

Optimizer: adam
Learning Rate: 0.001
Epochs: 100
Batch: 8
Restoring model weights from the end of the best epoch: 9.
Epoch 19: early stopping
111/111 [=====] - 2s 13ms/step

```

Evaluation:

```

R^2 score: 0.5060367010255538
MSE score: 0.04003549346252924
MAE score: 0.1536788379250833
=====
```

=====Training model=====

Hiperparámetros:

```

Optimizer: adam
Learning Rate: 0.0001
Epochs: 100
Batch: 8
Restoring model weights from the end of the best epoch: 5.
Epoch 15: early stopping

```

111/111 [=====] - 2s 14ms/step

Evaluation:

R² score: 0.426638400657786
MSE score: 0.04647068842925923
MAE score: 0.17323612724829365

=====

=====Training model=====

Hiperparámetros:

Optimizer: rmsprop
Learning Rate: 0.01
Epochs: 100
Batch: 8

Restoring model weights from the end of the best epoch: 1.

Epoch 11: early stopping

111/111 [=====] - 2s 13ms/step

Evaluation:

R² score: -2.97351888864436
MSE score: 0.322051840328703
MAE score: 0.49091986091005935

=====

=====Training model=====

Hiperparámetros:

Optimizer: rmsprop
Learning Rate: 0.001
Epochs: 100
Batch: 8

Restoring model weights from the end of the best epoch: 3.

Epoch 13: early stopping

111/111 [=====] - 2s 15ms/step

Evaluation:

R² score: 0.41004842703113575
MSE score: 0.04781529800258636
MAE score: 0.17945221024350452

=====

=====Training model=====

Hiperparámetros:

Optimizer: rmsprop
Learning Rate: 0.0001
Epochs: 100
Batch: 8

Restoring model weights from the end of the best epoch: 7.

Epoch 17: early stopping

111/111 [=====] - 2s 15ms/step

Evaluation:

R² score: 0.34927634454760537
MSE score: 0.052740846754943234
MAE score: 0.18829420043431302

=====

=====Training model=====

Hiperparámetros:

Optimizer: adam
Learning Rate: 0.01
Epochs: 100
Batch: 16

Restoring model weights from the end of the best epoch: 3.

Epoch 13: early stopping

111/111 [=====] - 2s 13ms/step

Evaluation:

R² score: 0.2899622072977417
MSE score: 0.057548229730627266
MAE score: 0.19129440954535235

=====

=====Training model=====

Hiperparámetros:

Optimizer: adam
Learning Rate: 0.001
Epochs: 100
Batch: 16

Restoring model weights from the end of the best epoch: 10.

Epoch 20: early stopping

111/111 [=====] - 2s 16ms/step

Evaluation:

R² score: 0.3101378682429161
MSE score: 0.05591300188363995
MAE score: 0.18090818622730068

=====

=====Training model=====

```

Hiperparámetros:
Optimizer:      adam
Learning Rate:   0.0001
Epochs:         100
Batch:          16
Restoring model weights from the end of the best epoch: 3.
Epoch 13: early stopping
111/111 [=====] - 2s 16ms/step

Evaluation:
R^2 score:      0.2570221060564791
MSE score:       0.06021800947061528
MAE score:       0.20239441684380324
=====

=====Training model=====
Hiperparámetros:
Optimizer:      rmsprop
Learning Rate:   0.01
Epochs:         100
Batch:          16
Restoring model weights from the end of the best epoch: 1.
Epoch 11: early stopping
111/111 [=====] - 2s 14ms/step

Evaluation:
R^2 score:      -2.97351888864436
MSE score:       0.322051840328703
MAE score:       0.49091986091005935
=====

=====Training model=====
Hiperparámetros:
Optimizer:      rmsprop
Learning Rate:   0.001
Epochs:         100
Batch:          16
Restoring model weights from the end of the best epoch: 10.
Epoch 20: early stopping
111/111 [=====] - 2s 16ms/step

Evaluation:
R^2 score:      0.4041971966084157
MSE score:       0.048289537481152715
MAE score:       0.16687301351377531
=====

=====Training model=====
Hiperparámetros:
Optimizer:      rmsprop
Learning Rate:   0.0001
Epochs:         100
Batch:          16
Restoring model weights from the end of the best epoch: 6.
Epoch 16: early stopping
111/111 [=====] - 2s 16ms/step

Evaluation:
R^2 score:      0.4470208276119697
MSE score:       0.044818702294319855
MAE score:       0.164409799681083
=====

CPU times: total: 8h 23min 12s
Wall time: 2h 28min 24s

```

```

In [19]: epochs = 50
optimizer = Adam(learning_rate=0.001)
input_shape=(X_train_3d.shape[1], X_train_3d.shape[2])

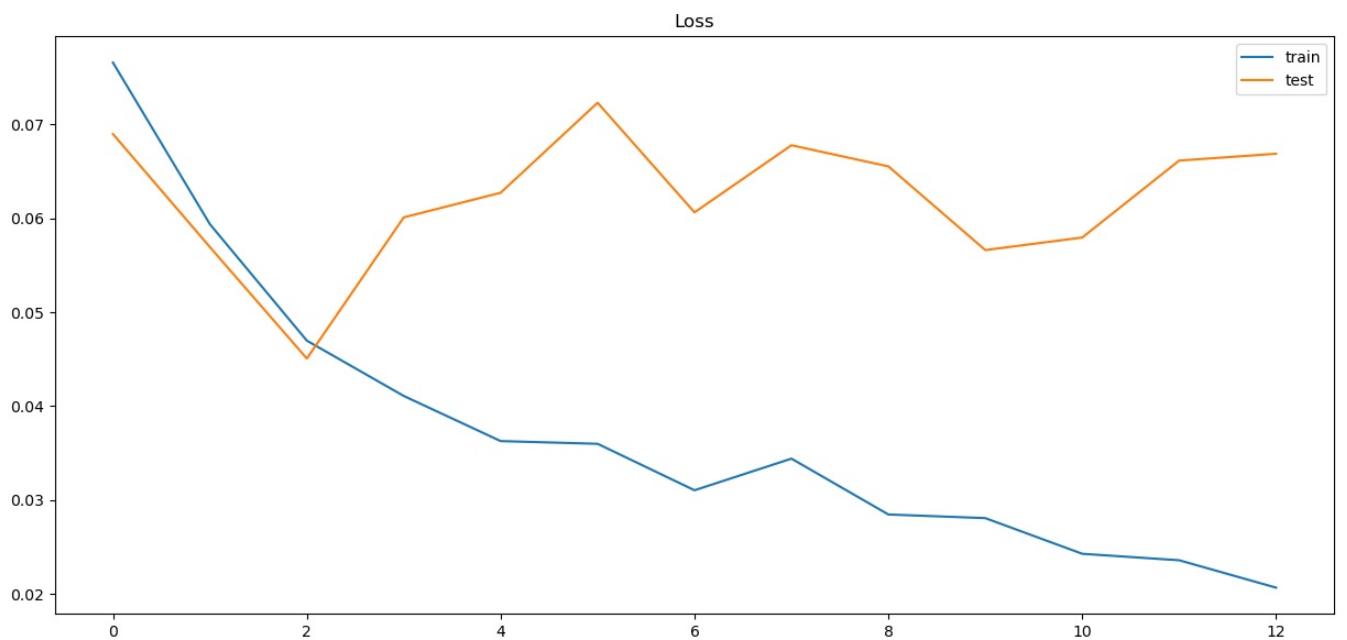
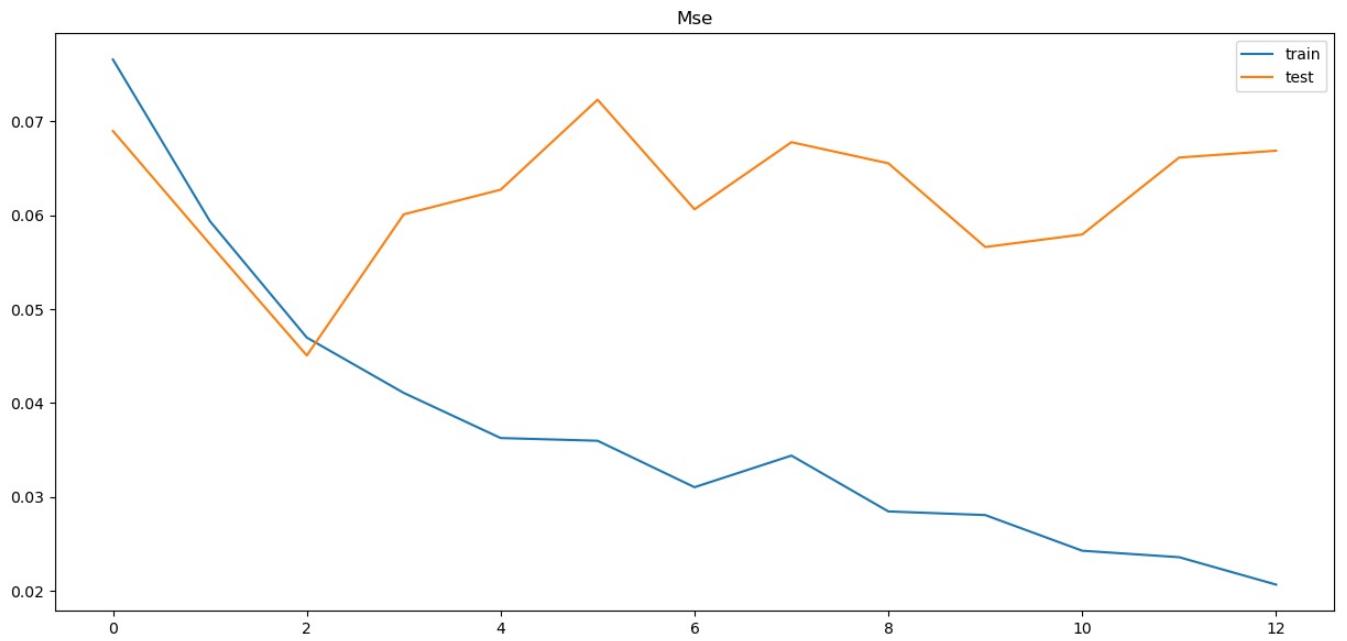
model = modelo_lstm(input_shape, optimizer)

res_m = model.fit(X_train_3d, y_train_3d, validation_data=(X_test_3d, y_test_3d), epochs=epochs, batch_size=8,

```

```
Epoch 1/50
2152/2152 [=====] - 116s 53ms/step - loss: 0.0766 - mse: 0.0766 - val_loss: 0.0689 - val_mse: 0.0689
Epoch 2/50
2152/2152 [=====] - 125s 58ms/step - loss: 0.0594 - mse: 0.0594 - val_loss: 0.0569 - val_mse: 0.0569
Epoch 3/50
2152/2152 [=====] - 117s 54ms/step - loss: 0.0470 - mse: 0.0470 - val_loss: 0.0451 - val_mse: 0.0451
Epoch 4/50
2152/2152 [=====] - 112s 52ms/step - loss: 0.0411 - mse: 0.0411 - val_loss: 0.0601 - val_mse: 0.0601
Epoch 5/50
2152/2152 [=====] - 113s 52ms/step - loss: 0.0363 - mse: 0.0363 - val_loss: 0.0627 - val_mse: 0.0627
Epoch 6/50
2152/2152 [=====] - 112s 52ms/step - loss: 0.0360 - mse: 0.0360 - val_loss: 0.0723 - val_mse: 0.0723
Epoch 7/50
2152/2152 [=====] - 113s 52ms/step - loss: 0.0311 - mse: 0.0311 - val_loss: 0.0606 - val_mse: 0.0606
Epoch 8/50
2152/2152 [=====] - 114s 53ms/step - loss: 0.0344 - mse: 0.0344 - val_loss: 0.0678 - val_mse: 0.0678
Epoch 9/50
2152/2152 [=====] - 116s 54ms/step - loss: 0.0285 - mse: 0.0285 - val_loss: 0.0655 - val_mse: 0.0655
Epoch 10/50
2152/2152 [=====] - 118s 55ms/step - loss: 0.0281 - mse: 0.0281 - val_loss: 0.0566 - val_mse: 0.0566
Epoch 11/50
2152/2152 [=====] - 113s 52ms/step - loss: 0.0243 - mse: 0.0243 - val_loss: 0.0579 - val_mse: 0.0579
Epoch 12/50
2152/2152 [=====] - 111s 51ms/step - loss: 0.0236 - mse: 0.0236 - val_loss: 0.0661 - val_mse: 0.0661
Epoch 13/50
2152/2152 [=====] - ETA: 0s - loss: 0.0207 - mse: 0.0207Restoring model weights from the end of the best epoch: 3.
2152/2152 [=====] - 111s 51ms/step - loss: 0.0207 - mse: 0.0207 - val_loss: 0.0669 - val_mse: 0.0669
Epoch 13: early stopping
```

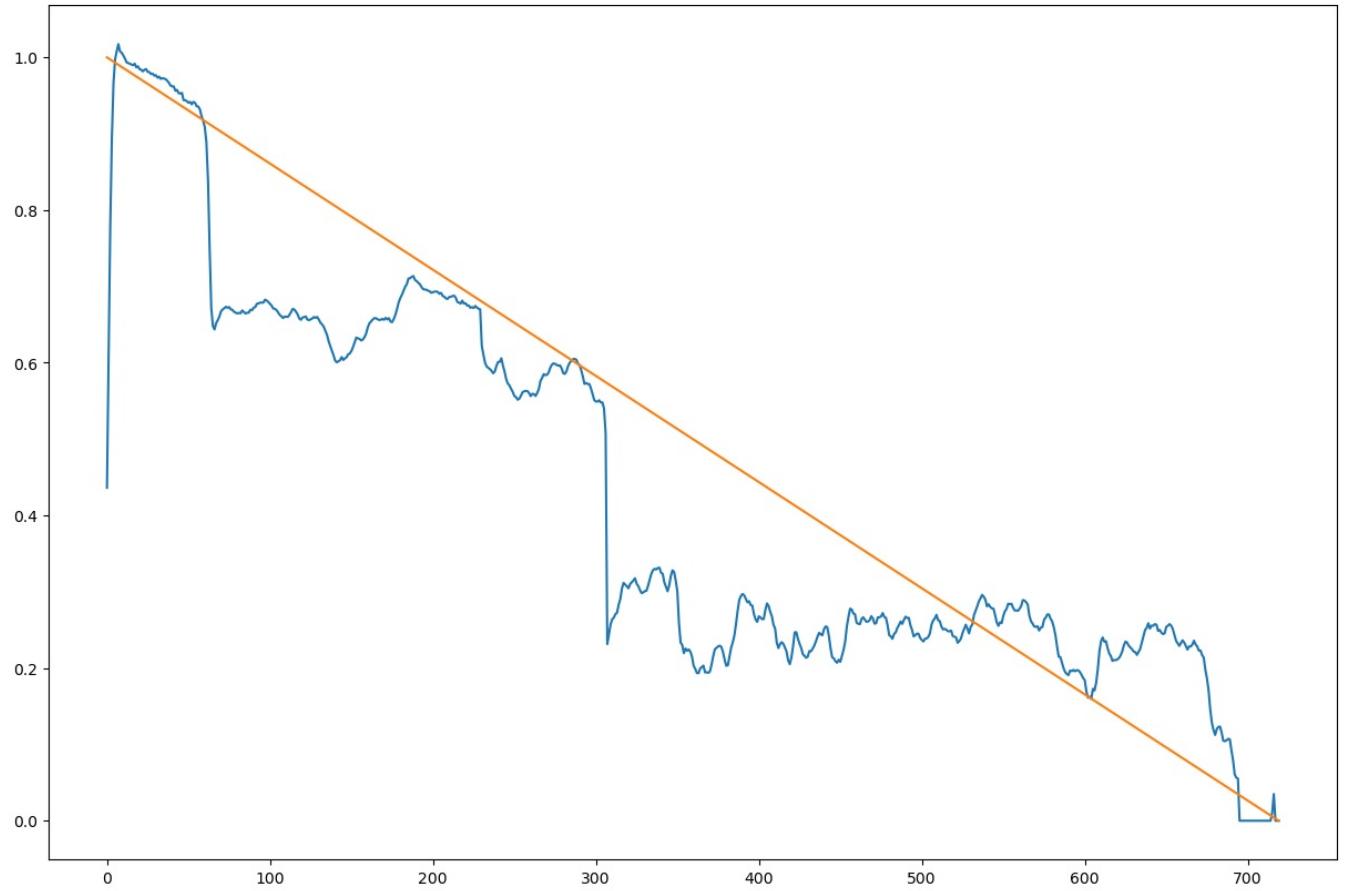
```
In [20]: plot_prediction(epochs, res_m)
```



```
In [22]: y_test_3d_5 = y_test_3d[(1*720)-rangos+1:(2*720)-rangos+1]
pred = model.predict(X_test_3d[(1*720)-rangos+1:(2*720)-rangos+1])
temp_list = []
for i in range(pred.shape[0]):
    temp_list.append(pred[i])

plt.figure(figsize=(15, 10))
plt.plot(temp_list)
plt.plot(y_test_3d_5)
plt.show()
```

23/23 [=====] - 1s 26ms/step



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js