Minería de Datos

Master en Ciencia de Datos e Ingeniería de Computadores

Universidad de Granada

Preprocesamiento y Clasificación

Autores:

Brian Sena Simons.
Miguel Garcia Lopez.
Álvaro Santana Sánchez.
Ana Fuentes Rodríguez.

Grupo:

Data Mavericks.

Índice

1. Introducción

Se ha realizado un análisis y comparativa entre diferentes modelos para la detección de anomalías y predicción de vida útil restante (RUL por sus siglas en inglés) en compresores del sector ferroviario[1]. Para ello, se ha utilizado el conjunto de datos (dataset) "MetroPT-3" [1]. Está publicado en "UCI Machine Learning Repository" [2] y, según la descripción, MetroPT-3 [1] es un conjunto de datos multivariantes de series temporales. Los datos provienen de sensores analógicos y digitales instalados en un compresor de tren, que miden 15 señales como presiones, corriente del motor, temperatura del aceite y señales eléctricas de las válvulas de entrada de aire. La información fue registrada a una frecuencia de 1 Hz entre febrero y agosto de 2020 (véase Tabla 1)

Variable	**Tipo**	**Mín.**	**Q1**	**Q2**	**Media**	**Q3**	**Máx.**
TP2	Numérico	-0.032	-0.014	-0.012	1.368	-0.010	10.676
TP3	Numérico	0.730	8.492	8.960	8.985	9.492	10.302
H1	Numérico	-0.036	8.254	8.784	7.568	9.374	10.288
DV pressure	Numérico	-0.032	-0.022	-0.020	0.05596	-0.018	9.844
Reservoirs	Numérico	0.712	8.494	8.960	8.985	9.492	10.300
Oil temperature	Numérico	15.40	57.77	62.70	62.64	67.25	89.05
Motor current	Numérico	0.020	0.040	0.045	2.050	3.808	9.295
COMP	Numérico	0.000	1.000	1.000	0.837	1.000	1.000
DV eletric	Numérico	0.000	0.000	0.000	0.1606	0.000	1.000
Towers	Numérico	0.000	1.000	1.000	0.9198	1.000	1.000
MPG	Numérico	0.000	1.000	1.000	0.8327	1.000	1.000
LPS	Numérico	0.000	0.000	0.000	0.00342	0.000	1.000
Pressure switch	Numérico	0.000	1.000	1.000	0.9914	1.000	1.000
Oil level	Numérico	0.000	1.000	1.000	0.9042	1.000	1.000
Caudal impulses	Numérico	0.000	1.000	1.000	0.9371	1.000	1.000

Tabla 1: Información básica de los diferentes tipos de datos presentes en MetroPT-3 [1]

Este conjunto de datos tiene como objetivo principal mejorar la detección de fallos y la predicción de mantenimiento. Aunque no contiene etiquetas directas, se dispone de informes de fallos que permiten evaluar la efectividad de los algoritmos de detección de anomalías, predicción de fallos y estimación de RUL (véase la Tabla 2). De las variables presentes, las siete primeras son analógicas y las demás digitales. La descripción de cada una viene dada en la siguiente lista:

1. TP2 (bar): La medición de la presión en el compresor.
2. TP3 (bar): La medición de la presión generada en el panel neumático.

[1]El código se puede encontrar en `https://github.com/briansenas/MineriaMetroPT-3`

3. H1 (bar): La medición de la presión generada debido a la caída de presión cuando ocurre la descarga del filtro separador ciclónico.

4. Presión DV (bar): La medición de la caída de presión generada cuando las torres descargan los secadores de aire; una lectura de cero indica que el compresor está operando bajo carga.

5. Reservorios (bar): La medición de la presión aguas abajo de los reservorios, que debería ser cercana a la presión del panel neumático (TP3).

6. Corriente del Motor (A): La medición de la corriente de una fase del motor trifásico; presenta valores cercanos a 0A (cuando está apagado), 4A (cuando trabaja sin carga), 7A (cuando trabaja bajo carga) y 9A (cuando empieza a trabajar).

7. Temperatura del Aceite (^{o}C): La medición de la temperatura del aceite en el compresor.

8. COMP: La señal de la válvula de admisión de aire del compresor; está activa cuando no hay admisión de aire, lo que indica que el compresor está apagado o funcionando sin carga.

9. DV eléctrico: La señal que controla la válvula de salida del compresor; está activa cuando el compresor funciona bajo carga e inactiva cuando el compresor está apagado o funcionando sin carga.

10. TORRES: La señal que define la torre responsable de secar el aire y la torre responsable de drenar la humedad eliminada del aire; cuando no está activa, indica que la torre uno está funcionando; cuando está activa, indica que la torre dos está en operación.

11. MPG: La señal responsable de arrancar el compresor bajo carga activando la válvula de admisión cuando la presión en la unidad de producción de aire (APU) cae por debajo de 8.2 bar; activa el sensor COMP.

12. LPS: La señal que detecta y activa cuando la presión cae por debajo de 7 bares.

13. Interruptor de Presión: La señal que detecta la descarga en las torres de secado.

14. Nivel de Aceite: La señal que detecta el nivel de aceite en el compresor; está activa cuando el nivel de aceite está por debajo de los valores esperados.

15. Impulso de Caudal: La señal que cuenta los pulsos generados por la cantidad absoluta de aire que fluye desde la APU hacia los reservorios.

Número	Inicio	Fin	Duración (mín)	Importancia
1	4/12/2020 11:50	4/12/2020 23:30	700	Alta
2	4/18/2020 00:00	4/18/2020 23:59	1440	Alta
3	4/19/2020 00:00	4/19/2020 01:30	90	Alta
4	4/29/2020 03:20	4/29/2020 04:00	40	Alta
5	4/29/2020 22:00	4/29/2020 22:20	20	Alta
6	5/13/2020 14:00	5/13/2020 23:59	599	Alta
7	5/18/2020 05:00	5/18/2020 05:30	30	Alta
8	5/19/2020 10:10	5/19/2020 11:00	50	Alta
9	5/19/2020 22:10	5/19/2020 23:59	109	Alta
10	5/20/2020 00:00	5/20/2020 20:00	1200	Alta
11	5/23/2020 09:50	5/23/2020 10:10	20	Alta
12	5/29/2020 23:30	5/29/2020 23:59	29	Alta
13	5/30/2020 00:00	5/30/2020 06:00	360	Alta
14	6/01/2020 15:00	6/01/2020 15:40	40	Alta
15	6/03/2020 10:00	6/03/2020 11:00	60	Alta
16	6/05/2020 10:00	6/05/2020 23:59	839	Alta
17	6/06/2020 00:00	6/06/2020 23:59	1439	Alta
18	6/07/2020 00:00	6/07/2020 14:30	870	Alta
19	7/08/2020 17:30	7/08/2020 19:00	90	Alta
20	7/15/2020 14:30	7/15/2020 19:00	270	Media
21	7/17/2020 04:30	7/17/2020 05:30	60	Alta

Tabla 2: Intervalos de tiempo con problemas en la compresión del aire. Nos permite evaluar la capacidad de detección anomalías de nuestros modelo.

2. Análisis Exploratorio de Datos

Se tienen más de un millón de observaciones correspondientes a distintos momentos en el tiempo que capturan datos de distintos sensores. Todas las variables son continuas a excepción del *timestamp*, que es la fecha de registro de cada valor en las variables. No hay nulos, por lo que no se requiere ningún tratamiento especial (como imputaciones) para ese tipo de casos. Lo que sí ocurre es que hay pequeños intervalos de tiempo vacíos, sin datos, pero estos no aparecen como nulos, solo pasa de un intervalo a otro en un salto temporal que se salta parte del tiempo. En ese caso se ha considerado válido eliminar ese espacio temporal por ser mínimo y por no tener información de si podría haber anomalía o no. Otra opción habría sido imputar ese espacio temporal con datos sintéticos que repliquen los datos del espacio temporal anterior, pero se consideró más válido eliminar ese intervalo.

2.1. Transformaciones y Visualizaciones

Se estandarizan los datos para poder visualizarlos y que las escalas no afecten demasiado a estas visualizaciones. No se normaliza por la desviación típica, ya que se desea conservar la variación de los datos. De esta forma no se pierde su relación original de escalas.

Como puede observarse en la Figura 1 y en la Figura 2, se visualizan los rangos donde según expertos, se produjo una anomalía. Gracias a esto es posible observar con facilidad que tipo de forma toma cada variable cuando una anomalía ocurre.

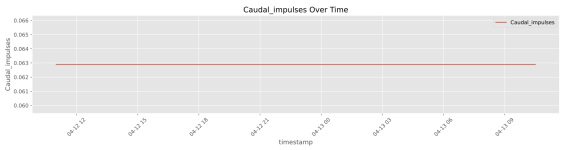
En la Figura 3 se muestran todas las anomalías (centradas gracias al estandarizado sobre la media) y cómo se comportan en un rango anómalo. Se preserva la varianza de cada una de ellas de forma que pueda observarse su rango de valores completo real.

Si se muestra otro rango temporal, se puede observar el comportamiento esperado de cada variable. De hecho, se podría intuir que son series de naturaleza **estacionaria**. Este fenómeno puede observarse en la Figura 4. Una serie estacionaria es una secuencia de datos temporales cuyas propiedades estadísticas, como la media, la varianza y la autocorrelación, son constantes a lo largo del tiempo. Esto significa que su comportamiento no cambia dependiendo del momento en el que se analice, lo que facilita su modelado y predicción.

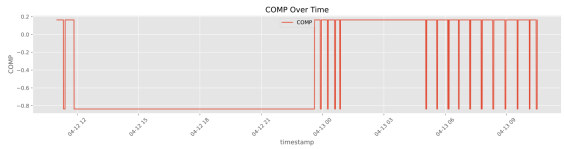
2.2. Test estadísticos

Se realizan pruebas estadísticas a las series temporales con el objetivo de determinar si son estacionarias. En este contexto, si el valor crítico de la prueba es mayor que el valor estadístico obtenido, se concluye que la serie no es estacionaria. Entre las pruebas utilizadas, destaca la prueba de *Dickey-Fuller Aumentada* (ADF), un test estadístico diseñado para evaluar la presencia de una raíz unitaria en una serie temporal. La existencia de una raíz unitaria indica que la serie no es estacionaria.

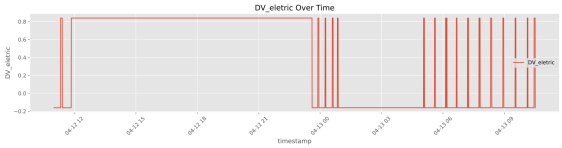
Ya que se tienen muchos datos, hacer la prueba de *adfuller* con todos en cada columna no



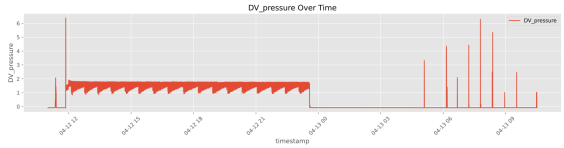
Caudal Impulses



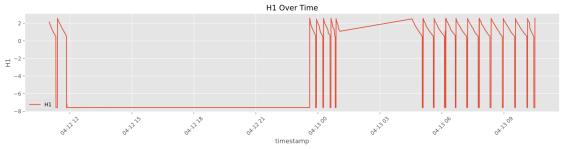
COMP



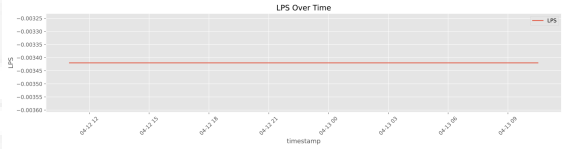
DV Electric



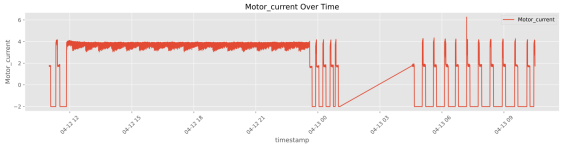
DV Pressure



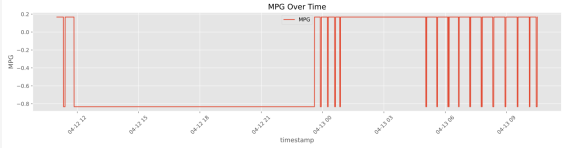
H1



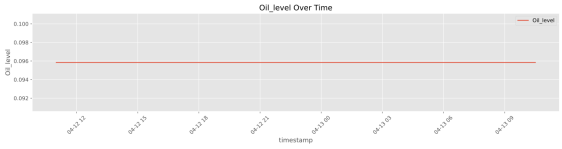
LPS



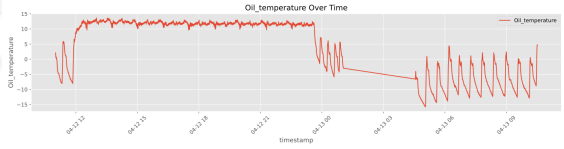
Motor Current



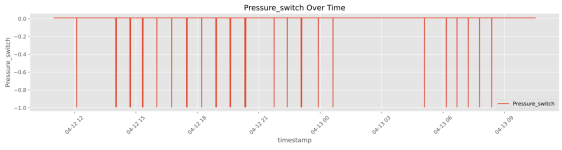
MPG



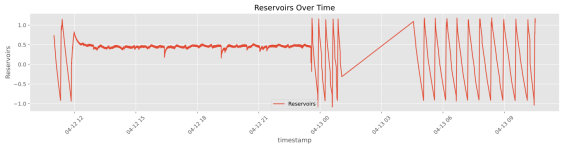
Oil level



Oil temperature

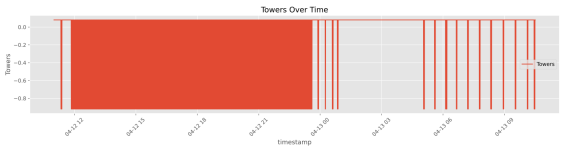


Pressure switch

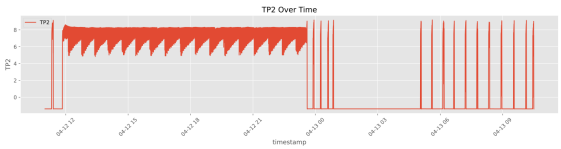


Reservoirs

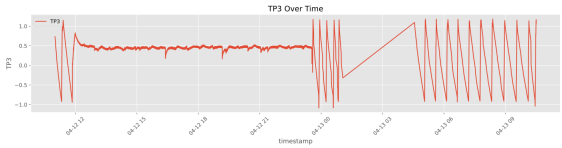
Figura 1: Variables en rangos donde hay anomalías. Se observa cambios bruscos en la naturaleza cíclica del uso habitual del motor de compresión.



Towers



TP2



TP3

Figura 2: Variables en rangos donde hay anomalías.

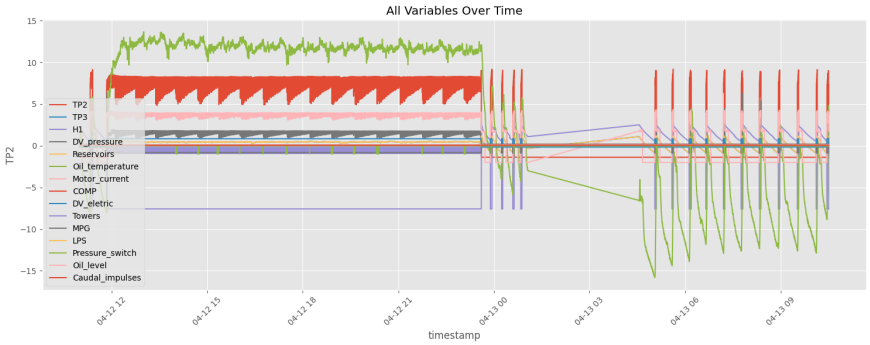


Figura 3: Todas las variables en funcionamiento normal y durante anomalías.

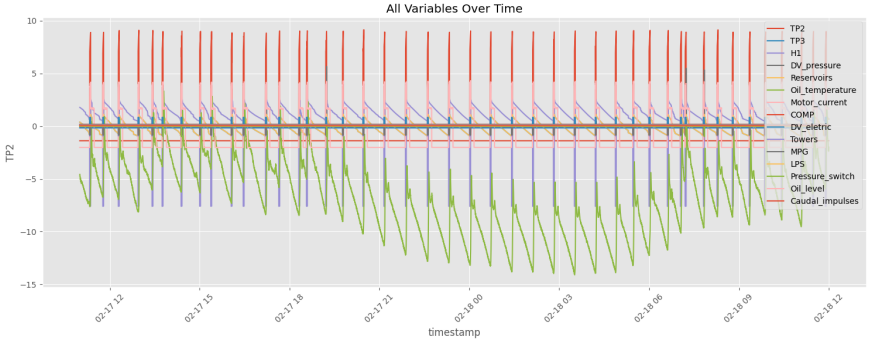


Figura 4: Todas las variables en rangos normales. Se observa la naturaleza cíclica de todas las variables debido a que iteraccionan en conjunto.

Feat	Mean Statistic	Std Statistic	Mean p-value	Std p-value	Stationary
LPS	-4.7913	–	0.0001	–	Yes
Oil_level	-1.4199	–	0.5727	–	No
TP2	-6.0272	0.3595	0.0000	0.0000	Yes
Motor_current	-4.5241	0.5570	0.0006	0.0006	Yes
Towers	-8.0871	2.1996	0.0000	0.0000	Yes
TP3	-4.7383	1.3977	0.0475	0.1492	Yes
COMP	-5.9092	0.3207	0.0000	0.0000	Yes
Caudal_impulses	-12.5275	–	0.0000	–	Yes
DV_pressure	-22.7337	12.0472	0.0000	0.0000	Yes
MPG	-5.8787	0.2662	0.0000	0.0000	Yes
DV_eletric	-5.9815	1.0184	0.0000	0.0001	Yes
H1	-5.8030	0.3944	0.0000	0.0000	Yes
Reservoirs	-4.7148	0.7602	0.0006	0.0008	Yes
Oil_temperature	-5.0249	0.9065	0.0009	0.0022	Yes
Pressure_switch	-24.7790	11.1539	0.0000	0.0000	Yes

Tabla 3: Estadísticos medios de los test de estacionalidad tras diez experimentos.

es viable. Por tanto, de manera aleatoria se escogen muestras de tramos aleatorios de cada variable. De esta manera se puede evaluar si la serie es estacionaria en gran parte de sus tramos y deducir si la serie completa es posible que sea estacionaria (veáse la Tabla 3).

2.3. Preprocesamiento

2.3.1. Enfoques del problema

Existen dos enfoques principales para abordar el estudio de este problema, los cuales varían dependiendo de si se incorporan o no valores temporales. El **primer enfoque** se basa en el uso de **instantáneas** y prescinde de la información temporal. En este enfoque, en cada iteración, los sensores del compresor generan un vector de valores representativos del estado del sistema en ese momento específico. Este vector se puede utilizar para predecir la presencia de posibles anomalías en el compresor, sin considerar las variaciones temporales previas.

Este método tiene la ventaja de permitir una detección rápida de anomalías, ya que, si es capaz de identificar correctamente los fallos, proporcionaría una respuesta con el menor retardo posible, e incluso en tiempo real. La ventaja principal de este enfoque radica en su simplicidad y capacidad de ofrecer una alerta inmediata ante cualquier fallo en el compresor, lo que resulta crucial en aplicaciones donde la rapidez en la respuesta es fundamental.

El **segundo enfoque** se basa en la **incorporación de información temporal** y utiliza valores dentro de una ventana de tiempo determinada para identificar posibles fallos en el compresor. A través de este enfoque, el modelo encargado de la detección de anomalías es

capaz de realizar un análisis más detallado de las tendencias a lo largo del tiempo. Este análisis temporal permite captar patrones que, de otro modo, podrían pasar desapercibidos al considerar únicamente instantáneas.

Es razonable suponer que ciertos fallos no se manifiestan mediante cambios abruptos en los valores de los sensores, sino que se desarrollan progresivamente. Por ejemplo, una disminución gradual del nivel de aceite, que ocurre a una velocidad mayor de la esperada, podría ser suficiente para señalar el inicio de un fallo. En estos casos, el análisis de los valores temporales sería esencial, ya que permite detectar anomalías antes de que los valores de los sensores alcancen niveles extremos. Esto, en un contexto predictivo, posibilitaría adelantarse al fallo y, potencialmente, evitarlo.

No obstante, este enfoque no está exento de desafíos. Una de las principales limitaciones es que los valores anómalos pueden quedar "opacados" o diluidos por el resto de los datos dentro de la ventana temporal, lo que podría dificultar la identificación precisa de anomalías. Sin embargo, el equipo considera que la información temporal ofrece una ventaja significativa en la detección temprana de fallos, por lo que ha decidido centrar su trabajo en este enfoque. A continuación, se abordarán con mayor detalle las características de este modelo y cómo se gestionan los posibles problemas derivados de la utilización de información temporal en el proceso de detección.

2.3.2. Definición de la ventana deslizante

Para resolver este problema estudiamos los tiempos de activaciones de los motores para poder definir una ventana deslizante que pueda recoger información de la activación de los mismos. Para ello, se ha calculado la mediana del tiempo de activación de los motores. Para ello, se ha detectado la activación y apagado del motor por medio de la variable "*Motor current*", cuyos valores para apagado son inferiores a 0.05, veáse la Figura 5.

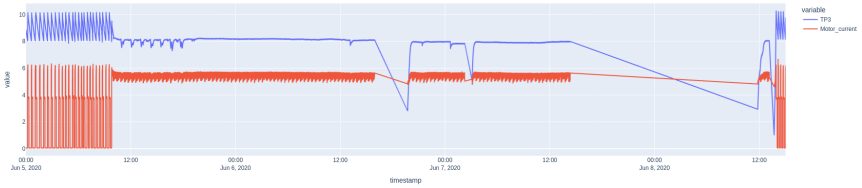


Figura 5: Observamos los datos y los valores de la presión en el panel neumático (TP3, línea azul) y la corriente del motor (línea roja).

Se han recogido los resultados en la Tabla 4. La mediana se calcula sobre los intervalos de tiempo no anómalos. No obstante, aunque el conjunto de datos no presenta valores pérdidos en ninguna de las columnas, sí que presenta saltos temporales. Los datos de los sensores se mostrean cada 10 segundos, pero se ha encontrado saltos temporales de incluso días, véase Figura 6. Se planteó la posibilidad de interpolar los datos, pero dada la naturaleza

del problema, serie temporal pseudo-cíclica pero con intervalos distintos, es difícil obtener resultados prometedores para saltos grandes sin la posibilidad de "ensuciar" la calidad de los datos. Por ello, se ha calculado el tiempo mediano de ciclo de motor tras eliminar los saltos temporales. Para comprobar, se ha calculado también la mediana para todos intervalos sin anomalías de la Tabla 2. Los valores obtenidos están muy cerca del especificado en la Tabla 4.

Mediana del tiempo de ciclo
1260 segundos

Tabla 4: Resultado obtenido del cálculo de la ventana deslizante. Se acerca a los obtenidos en el artículo original de detección de fallos de este dataset [3].

Para la asignación de los grupos se ha utilizado dos veces la mediana del tiempo de ciclo del motor. Esto es debido a que así se puede asegurar contener información de almenos más de la mitad de la activación del motor, asegurando que predecimos con la mayor información posible del estado del motor. Se puede observar la asignación de grupos en la Figura 6.

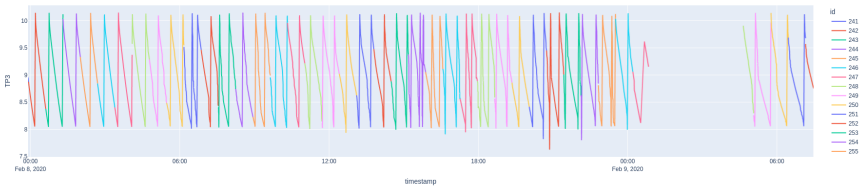


Figura 6: Observamos los grupos asignados de utilizar 2 veces el tiempo de ciclo.

Durante el análisis EDA y prepración del conjunto de datos de entrenamiento y evaluación se han recogido nuevas anomalías, veáse la Tabla 5 y Figura 7. Sería interesante volver a consultar con un experto del campo para que valide dichas anomalías. No obstante, presentan un perfil suficientemente cercano al de las anomalías clasificadas. Por ello, consideramos oportuno la inclusión de dichos ejemplos como anomalías para ayudar a resolver el problema del bajo número de ejemplo de casos positivos.

Tras observar y analizar el conjunto de datos, seguimos un acercamiento similar a [4, 3] para tratar a la serie temporal, se ha optado por la transformación de los intervalos de las ventanas deslizante en obtener el promedio, mínimo, máximo y varianza de cada variable durante el intervalo de tiempo mostreado. Para resolver el problema de saltos temporales, se ha eliminado aquellos conjuntos en los que se estiman estas variables para un número de puntos inferior a (tiempo de ciclo)/10 = 126. Ya que esos ejemplos son estimados con menos puntos que el tiempo de activación del motor, lo cuál puede generar estimaciones del promedio y varianza sub-óptimos.

Número	Inicio	Fin	Duración (min)	Importancia
22	2020-03-06 21:42:15	2020-03-06 23:14:00	92	-
23	2020-03-11 05:15:10	2020-03-11 06:25:00	70	-
24	2020-03-12 00:15:56	2020-03-12 11:59:00	704	-
25	2020-03-26 04:00:20	2020-03-26 05:20:00	80	-
26	2020-03-27 07:12:00	2020-03-27 12:01:00	289	-
27	2020-04-17 08:50:28	2020-04-17 23:59:00	909	-
28	2020-04-25 00:07:15	2020-04-25 01:10:00	63	-
29	2020-05-19 01:35:28	2020-05-19 02:40:00	64	-
30	2020-06-12 01:41:07	2020-06-12 17:06:00	925	-
31	2020-07-21 13:32:48	2020-07-21 22:03:00	510	-
32	2020-07-22 06:40:46	2020-07-22 13:10:00	389	-
33	2020-07-31 00:57:33	2020-07-31 02:09:00	71	-

Tabla 5: Intervalos de tiempo encontrados con valores constantes y fluctuaciones extrañas, un patrón similar al de las anomalías, sin etiquetado.

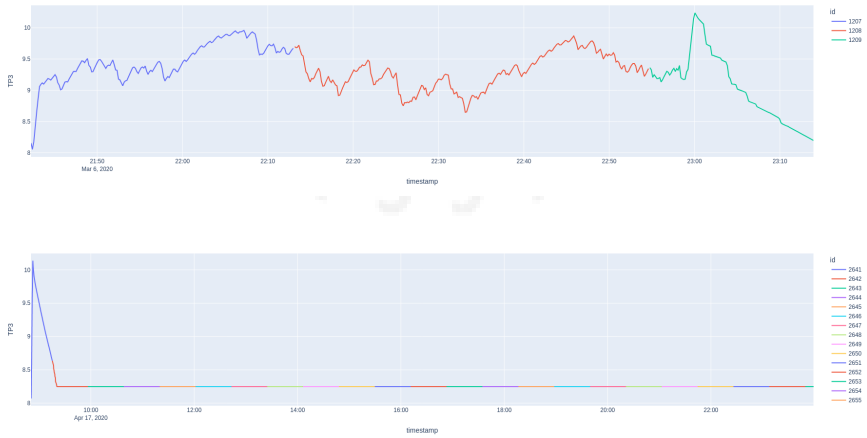


Figura 7: Ejemplo de nuevos intervalos anómalos encontrados. Observamos valores constantes o fluctuaciones fuera de lo habitual para el motor apagado o encendido.

2.3.3. Generación de conjuntos

Una vez determinado la ventana deslizante y las características a extraer, se genera las diferentes instancias y se divide en los conjuntos de entrenamiento y evaluación. Se debe determinar por tanto si un intervalo de la ventana deslizante es una anomalía o no, para lo cuál es utilizado se utilizado un criterio de votación en el cuál gana la mayoría.

Uno de los aspectos cruciales a considerar en este enfoque es la similitud de los datos generada por la ventana deslizante en ciertos momentos. Por ejemplo, en el caso de anomalías cuya duración se extiende por un periodo de tiempo considerable, como un día completo (por ejemplo, de 6/05/2020 10:00 a 6/05/2020 23:59), se generan ventanas de 21 minutos en cada iteración. Esto puede dar lugar a la aparición de instantes temporalmente muy similares entre sí, lo que podría influir en la evaluación de los modelos de detección de anomalías.

Una situación problemática podría ocurrir si las ventanas se distribuyeran de manera completamente aleatoria a posteriori. En ese caso, se podría dar el escenario en el cual un periodo como **6/05/2020 10:00 - 6/05/2020 10:21** pertenezca al conjunto de entrenamiento, mientras que el siguiente periodo **6/05/2020 10:21 - 6/05/2020 10:42** esté en el conjunto de test. Esta división podría generar evaluaciones poco representativas, ya que las ventanas de tiempo consecutivas podrían estar separadas en diferentes conjuntos de datos, lo que afectaría la validez de la evaluación de la capacidad predictiva del modelo. Véase la figura 6.

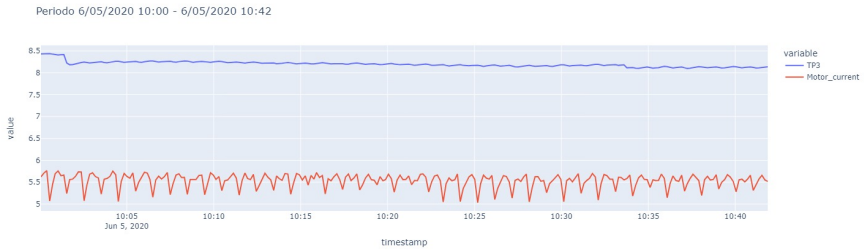


Figura 8: Observamos la similitud de los valores de algunos sensores durante la anomalía.

Para mitigar este riesgo y garantizar una evaluación más precisa y coherente, las instancias se agrupan temporalmente. De esta forma, se asegura que dos instancias pertenecientes al mismo grupo no se distribuyan entre los conjuntos de entrenamiento y test si dichos conjuntos se utilizan con fines de validación y entrenamiento. Se define un **grupo** como un conjunto de ventanas que comparten el mismo tipo (anomalía o no anomalía) y no presentan saltos temporales, es decir, no existe un periodo sin datos entre las ventanas. Este enfoque asegura que la información utilizada en el entrenamiento y la validación sea coherente y representativa, mejorando así la robustez de las evaluaciones del modelo.

El proceso de generación de los conjuntos de datos consiste en barajar los grupos de ventanas deslizantes para asignar de forma aleatoria diferentes grupos de intervalos de tiempo en cada partición, evitando así pliegues más fáciles o difíciles (obtener unos resultados más balanceados en general). Es decir, agrupamos los grupos en grupos de mayor tamaño, pero

esta vez mediante la aleatoriedad y teniendo en cuanta la proporción de grupos anómalos y grupos no anómalos.

Concretamente, se decide dividir los datos en 9 pliegues. Se estima el número de anomalías que pertenecería a cada pliegue y se genera conjuntos lo más equilibrado posible (véase la Tabla 6). Por último, se asigna el pliegue 1 y 8 (elección aleatoria) como el conjunto de test. Los resultantes serán agrupado en 4 pliegues para el entrenamiento de la validación cruzada.

Pliegue	Negativo	Positivo	Conjunto
0	635	31	Evaluación
1	635	31	Entrenamiento pliegue 1
2	635	31	Entrenamiento pliegue 2
3	635	31	Entrenamiento pliegue 3
4	635	31	Entrenamiento pliegue 4
5	635	31	Entrenamiento pliegue 3
6	635	31	Entrenamiento pliegue 2
7	638	31	Entrenamiento pliegue 1
8	641	43	Evaluación

Tabla 6: Distribución de los 9 pliegues generados. Se asigna de forma aleatoria el 1 y 8 a test. Se agrupan los demás hasta forma 4 pliegues usando el primer y último de los restantes: 0 y 7, 2 y 6, 5 y 3, 4.

3. Métricas de evaluación

Debido a que tratamos de evaluar el rendimiento de diferentes modelos para la resolución de un problema de detección de anomalías, el uso de métricas como la **precisión**, la **sensibilidad** (*recall*) y la **puntuación F1** es fundamental para evitar evaluaciones sesgadas. Cada una de estas métricas aporta una perspectiva única sobre cómo el modelo maneja tanto las clases mayoritarias (no anomalías) como las minoritarias (anomalías).

- **Precisión**: mide la proporción de verdaderos positivos entre todas las instancias clasificadas como positivas. Esto permite evaluar qué tan confiable es el modelo al detectar anomalías sin producir demasiados falsos positivos.
- **Sensibilidad**: evalúa la capacidad del modelo para identificar correctamente todas las anomalías presentes en el conjunto de datos. Esta métrica es crucial en aplicaciones donde no detectar una anomalía puede tener consecuencias graves.
- **F1**: como métrica armonizada entre precisión y sensibilidad, es especialmente útil en tareas de detección de anomalías, ya que equilibra ambos aspectos, permitiendo evaluar el rendimiento del modelo en contextos donde el costo de los falsos positivos y falsos negativos debe ser considerado conjuntamente.

Es por ello que en las diferentes experimentaciones se ha hecho uso de esas métricas. La evaluación es el promedio del rendimiento de los modelos en los diferentes pliegues de validación.

4. Máquinas de Vectores de Soporte (Ana Fuentes)

A continuación, se ha implementado un modelo SVM (Support Vector Machine) para trabajar con los conjuntos de datos de entrenamiento y test.

El clasificador basado en Máquinas de Soporte Vectorial (SVM) es un potente algoritmo de clasificación supervisada ampliamente utilizado, no sólo en tareas de clasificación, sino también en tareas de regresión. Este algoritmo se basa en la idea de encontrar un hiperplano óptimo que separa las diferentes clases en el espacio de características, maximizando así el margen entre los datos más cercanos de cada clase, (los vectores de soporte). Este enfoque asegura que el modelo generalice bien a nuevos datos.

El modelo SVM destaca principalmente por su capacidad para tratar problemas no lineales mediante el uso de funciones kernel. Estos kernels transforman el espacio original de características en un espacio con mayor dimensionalidad, donde las clases son linealmente separables. Además, hay diversos tipos de funciones kernel, desde lineales a radiales (rbf), entre otras.

Otro punto a favor de este algoritmo es su alta efectividad en problemas de alta dimensionalidad, como ocurre en el conjunto estudidado. No obstante, el SVM puede ser menos eficiente en datasets con muchas instancias debido al costo computacional asociado. A pesar de esta limitación, SVM sigue siendo una herramienta robusta y versátil con el que se pueden obtener buenos resultados.

4.1. Preprocesamiento

Antes de construir el modelo de SVM, se ha tenido que realizar un pequeño preprocesamiento específico para este algoritmo. En el preprocesamiento general previo, ya se han eliminado los saltos temporales y no hay valores faltantes, por lo que a este respecto no hay que hacer nada. Sin embargo, en las características extraídas hay mucha variabilidad, por lo que es altamente relevante escalar estos valores, debido a la gran sensibilidad del SVM ante la magnitud de las características. De esta manera, se han escalado los datasets de entrenamiento y de test aplicando la función `StandardScaler` para normalizar estos datos.

4.2. Detalles de experimentación y resultados

Para implementar el algoritmo SVM, se ha utilizado la librería "*sklearn.svm*" para poder trabajar con el método de los vectores de soporte. Para conseguir el modelo con mejor rendimien-

to, se han optimizado tres hiperparámetros:

- C: este parámetro controla la penalización por errores de clasificación. Se elige entre 0.1, 1 y 10.
- γ: este define cómo influye una sola muestra en el modelo, lo que resulta relevante para kernels no lineales. Se elige entre "auto", "scale" y 0.1.
- Kernel: también se elige entre dos tipos de kernel, "rbf" (radial) o "linear".

Para conseguir el mejor modelo, se han entrenado las 72 combinaciones posibles generadas y se ha realizado la validación cruzada, separando en 4 plieques el conjunto de entrenamiento (esto se ha explicado en el apartado 3). De esta forma, la mejor combinación ha resultado ser la mostrada en la Tabla 7.

C	γ	**kernel**	F1 promedio
10	auto	rbf	0.94196

Tabla 7: Mejores hiperparámetros tras entrenar el modelo SVM.

Una vez entrenado y obtenido el mejor modelo (como se pueder observar, se ha obtenido un kernel no lineal), se ha evaluado este modelo de SVM con el conjunto de prueba. Así pues, se han reflejado estos resultados en la Tabla 8.

Clase	**Precisión**	**Recall**	**F1-score**	**Soporte**
False	0.99	1.00	1.00	1279
True	0.98	0.86	0.92	74
macro avg	0.99	0.93	0.96	1353
weighted avg	0.99	0.99	0.99	1353

Tabla 8: Resultados obtenidos de la evaluación del modelo SVM con el conjunto de prueba.

Como se puede observar en esta Tabla, se han obtenido unos resultados de precisión y recall para la clase "False" (no es anomalía) excelentes, lo que tiene sentido, al ser la mayoría de los datos no anómalos, ha aprendido mejor el modelo a identificar los valores que no son anomalías.
Por otro lado, la clase "True" (sí es anomalía) presenta un recall más bajo (86 %), lo que indica que 10 anomalías no fueron detectadas como tal.
A pesar de esto, el modelo alcanza un F1-score de 92 % en la clase "True", lo que refleja un buen balance entre precisión y recall.

Para finalizar, se ha generado la matriz de confusión que se puede observar en la Tabla 9.

En conclusión, el modelo SVM tiene un desempeño excelente con una precisión general del 99 %. Para la clase "True" presenta mayor dificultad de detección debido a la desbalanceada proporción entre las clases (muchos más datos no anómalos que anómalos).

1278	1
10	64

Tabla 9: Matriz de confusión para el modelo implementado de SVM.

5. Regresión Logistica (Álvaro Santana Sánchez)

La regresión logística es un algoritmo de aprendizaje supervisado ampliamente utilizado en problemas de clasificación binaria. A diferencia de los métodos de regresión lineal, que predicen valores continuos, la regresión logística se enfoca en estimar la probabilidad de que una instancia pertenezca a una de las dos clases posibles. Este algoritmo utiliza una función logística, también conocida como sigmoide, para transformar la combinación lineal de las variables independientes en un valor entre 0 y 1, el cual representa la probabilidad de pertenecer a la clase de interés.

La parte central de la regresión logística radica en su función de transformación, la función sigmoide. Esta función toma como entrada la combinación lineal de las variables independientes y produce una salida que varía entre 0 y 1 (Figura 10).

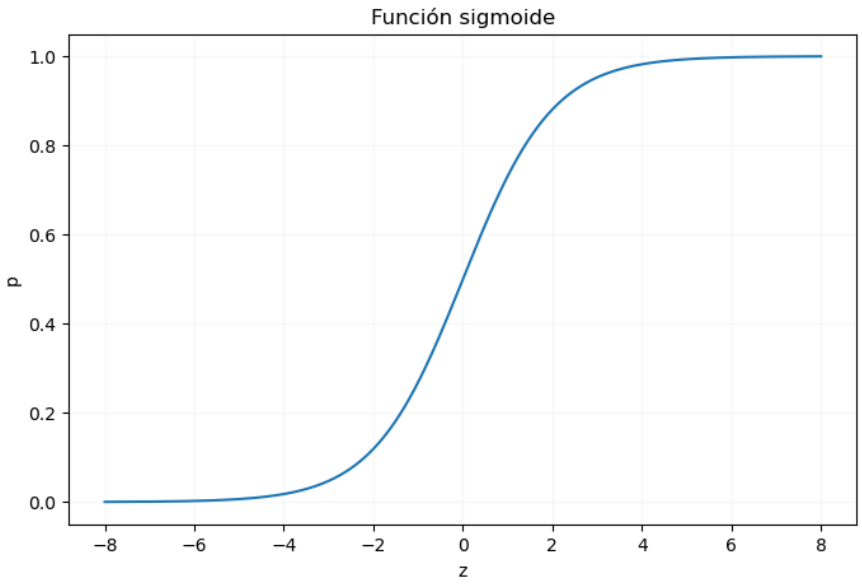


Figura 9: Funcion sigmoide.

5.1. Preprocesamiento especifico

Además del procesamiento general realizado, para mejorar el rendimiento del algoritmo de regresión logística se proponen las siguientes modificaciones:

5.1.1. Eliminación de ruido

El ruido puede introducir sesgos en el modelo, ya que los valores atípicos o erróneos pueden influir desproporcionadamente en la función de coste de la regresión logística durante el entrenamiento, influyendo fuertemente durante la optimización del modelo. Esto puede llevar a una sobreestimación o subestimación de los parámetros del modelo, lo que dificulta además la interpretación de los resultados.

Para ello será utilizado el filtro *EnsembleFilter* propuesto en clase, el cual detecta y elimina ejemplos ruidosos en un conjunto de datos utilizando un enfoque de conjunto (ensemble) de clasificadores. Combina las predicciones de varios modelos para identificar instancias mal etiquetadas o inconsistentes. Los modelos a utilizar son los siguientes: DecisionTreeClassifier, KNeighborsClassifier y LinearDiscriminantAnalysis.Los datos se dividen en 5 particiones, los clasificadores son entrenados en cada subconjunto y se cuenta cuántas veces cada instancia es clasificada incorrectamente. Finalmente el ruido es eliminado mediante el consenso, que elimina instancias clasificadas incorrectamente por todos los clasificadores. Tras la aplicación del filtro el tamaño del dataset varía según los datos de la Tabla 10. Como se puede ver fueron eliminados dos datos normales y 8 anomalías.

Dataset	Tipo	Count
Original	Anomalía	209
Modificado	Anomalía	217
Original	No Anomalia	4445
Modificado	Anomalía	4443

Tabla 10: Tamaño de los datos antes y después del procesamiento procesados.

5.1.2. Escalado de los datos

La estandarización, realizada con *StandardScaler*, es fundamental en regresión logística ya que transforma las variables para que tengan media 0 y desviación estándar 1, lo que ofrece múltiples beneficios. En primer lugar, acelera el descenso de gradiente durante el entrenamiento, haciendo que el modelo converja más rápidamente hacia una solución óptima. Además, evita sesgos en él, ya que sin estandarización, las variables con escalas más grandes podrían dominar el entrenamiento, distorsionando los coeficientes y la capacidad predictiva. También facilita la interpretación, permitiendo que los coeficientes reflejan de manera equitativa la importancia relativa de cada variable en la predicción. Por último, garantiza que las técnicas de regularización (L1 o L2) funcionen correctamente, ya que estas penalizan los coeficientes de manera proporcional, algo que solo es posible si las variables están en la misma escala. Por todo ello, la estandarización asegura un entrenamiento más eficiente, un modelo más equilibrado y resultados más generalizables.

5.1.3. Análisis de componentes principales

El Análisis de Componentes Principales es una técnica de reducción de dimensionalidad que *mapea* un conjunto de variables correlacionadas en un otro conjunto en un espacio diferente. Al hacerlo, PCA ayuda a reducir la multicolinealidad entre las variables, ya que los componentes principales resultantes son combinaciones lineales de las variables originales no correlacionadas. Aplicando PCA reducimo de más de 70 características a solo 30, manteniendo la varianza explicada en un 99 %.

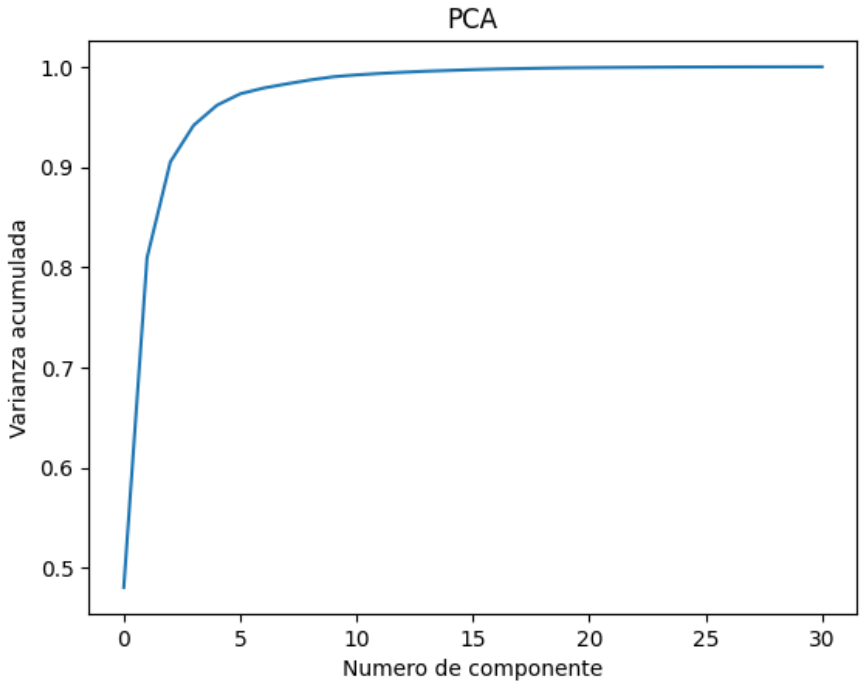


Figura 10: Varianza acumulada.

5.1.4. Hipérparámetros

Se propone un técnica de búsqueda de hiperparámetros mediante *GridSearch* utilizando el siguiente grid de parámetros:

Parámetro	Valores
C	10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}
penalty	l1, l2
class_weight	balanced, {0: 1, 1: 10}, {0: 1, 1: 20}

Tabla 11: Cuadrícula de hiperparámetros.

- C: Este hiper parámetro controla la regularización en el modelo,permitiendo explorar desde una regularización fuerte (valores bajos - $10^{-}3$) hasta una regularización débil (valores altos 10^3). Esto ayuda a determinar el equilibrio ideal entre ajuste del modelo y penalización de los coeficientes. Los valores a probar vienen dados por una escala logarítmica de 7 puntos equiespaciados entre el máximo y mínimo comentados.
- penalty: Define el tipo de regularización aplicada al modelo. **l2 (Ridge)** penaliza/regulariza los coeficientes al elevarlos al cuadrado y **l1 (Lasso)** penaliza los coeficientes utilizando su valor absoluto.
- class_weight: Se propone ya que ajusta la importancia de las clases en el modelo. Se prueban tres configuraciones: 'balanced': Asigna pesos inversamente proporcionales a las frecuencias de las clases, útil cuando las clases están desbalanceadas. Por otro lado se utilizan los balances {0: 1, 1: 10} y {0: 1, 1: 20}, que especifican pesos personalizados para las clases, aumentando la penalización de errores en la clase minoritaria. De esta manera se da una mayor o menor importancia a las anomalías.

Tras el ajuste de hiper parámetros mediante el cross-validation propuesto por el equipo, se encuentra que para la media de f1-score con un valor 0.95, los mejores parámetros encontrados son los presentados en la Tabla 12.

C	class_weight	penalty
1000	{0: 1, 1: 10}	l1

Tabla 12: Mejores hiperparámetros tras entrenar el modelo de Regresión Logística.

5.1.5. Evaluación de métricas es test

Con estos parámetros se realiza el entrenamiento del modelo, para el cuál se utilizan ya si todos los datos de entrenamiento. Entrenado este se realizan las predicciones sobre test y se calculan algunas métricas, véase la Tabla 13.

Métrica	Valor
F1	0.91
Recall	0.9
Precision	0.93
Accuracy	0.99

Tabla 13: Medidas de test.

El modelo presenta un desempeño sobresaliente, evidenciado por una F1 de 0.91, lo que indica un buen equilibrio entre precisión (0.92) y recall (0.89). Esto significa que identifica correctamente la mayoría de los casos positivos mientras minimiza los falsos positivos. Además, su

alta exactitud (0.99) desbalance de clases evidencia aún más el buen rendimiento aunque se ve afectado por el desbalance de clases. La matriz de confusión correspondiente a los anteriores valores puede verse en la Tabla 14.El modelo tiene un alto número de aciertos (1274 VP y 66 VN) y muy pocos errores (5 FN y 7 FP), lo que indica un buen rendimiento en la clasificación.

1274	5
7	66

Tabla 14: Matriz de confusión para el modelo de Regresión Logística.

6. Clasificador Bayesiano (Brian Sena)

El clasificador bayesiano (*Naive Bayes*) es un algoritmo de clasificación basado en el Teorema de Bayes, el cual calcula la probabilidad posterior de una clase dado un conjunto de características de entrada. A pesar de su simplicidad, es una técnica poderosa y ampliamente utilizada en problemas de clasificación, como el filtrado de correos no deseados, la clasificación de textos y el análisis de sentimientos. El modelo recibe el calificativo de "*naive*" (ingenuo) debido a su principal asunción: **independencia condicional entre las características**, es decir, se supone que todas las variables predictoras son independientes entre sí dado el valor de la clase. Aunque esta asunción rara vez se cumple en escenarios del mundo real, *Naive Bayes* sigue funcionando de manera sorprendentemente efectiva en muchos contextos, especialmente en dominios donde las relaciones entre variables no son complejas.

6.1. Asunciones del Naive Bayes

- **Independencia condicional**: Se asume que las características son independientes entre sí. En este caso, se asume que la variabilidad de una característica no depende del valor de otra. No obstante, ya vimos que los distintos sensores iteractúan entre sí, lo que hace débil a esta asunción.
- **Distribución de los datos:** *Naive Bayes* asume que las características siguen una distribución normal dentro de cada clase. En nuestro caso, solamente se cumple para "*Pressure Switch*".
- **Balance de clases**: El modelo puede ser sensible al desequilibrio de clases (cuando una clase es mucho más frecuente que otra), lo que puede sesgar las predicciones hacia la clase mayoritaria si no se toma en cuenta.

6.2. Preprocesamiento

Para implementar el *Naive Bayes* de manera efectiva, es crucial preparar los datos adecuadamente. En este caso, no disponemos de variables categóricas que necesiten codificarse numéricamente ya que todas las características son numéricas y continuas. Además, se ha eliminado los saltos temporales y no se dispone de valores faltantes. El algoritmo, dado que calcula distribuciones de probabilidad para cada clase [5] sin basarse en distancia, es invariante a la escala de los datos y, por ello, no necesitamos escalar los datos. No obstante, aunque en el preprocesamiento definido anteriormente se ha intentado crear un conjunto de datos equilibrado, disponemos de un drástico desequilibrio debido a la baja probabilidad de anomalía del sistema. Es por ello que se ha incluido experimentación con métodos de submuestreo y sobremuestreo con técnicas como "*CondensedNearestNeighbour* (CNN)" y "*SMOTE + TomekLinks* (SMOTETomek)". Por último, dado la primera asunción de independencia entre características se experimenta también con técnicas de selección de características, utilizando pruebas como la "chi-cuadrado" o "información mutua".

6.3. Detalles de experimentación y resultados

En el caso de la experimentación con submuestreo o sobremuestreo se ha hecho uso de la librería "*imblearn*" [6] y "*scikit-learn*" [7]. La primera nos permite crear un *pipeline* en el cual solo se realiza las operaciones de submuestreo o sobremuestreo en los pliegues que correspondan al conjunto de entrenamiento en el bucle de validación. De esta forma, las estimaciones de la precisión no se ve afectada por estos cambios, permitiendo estimar mejor la capacidad de generalización del modelo. La segunda nos permite añadir otras operaciones si son necesarias (por ejemplo, una normalización) además de disponer de la implementación de los modelos. Se ha realizado 4 pliegues de validación. Las métricas que utilizamos para evaluar el rendimiento se han discutido en la Sección 3. Para el clasificador bayesiano simples, el único hiperparámetro que tenemos que optimizar es el suavizado (α) de la varianza para alargar la distribución de probabilidad de las clases. Experimentaremos con los valores espaciados linealmente desde $1e^{-11}$ a $1e^{-7}$ con 25 valores.

El mejor modelo sería la combinación de submuestreo con CNN y selección de características con el test estadístico "chi-cuadrado", donde obtenemos una puntuación F1 promedio de 0.7819 (véase la Tabla 15). La matriz de confusión de ese modelo se puede observar en la Figura 11. Los resultados sobre el conjunto final de evaluación con este modelo se pueden observar en la Tabla 16, con su matriz de confusión correspondiente en la Figura 11.

Modelo	Parámetros	F1 Promedio
Bayesiano	$\alpha = 4^{-9}$	0.6802
Bayesiano + MRMR	$\alpha = 1^{-8}$	0.6894
Bayesiano + CHI	$\alpha = 1^{-11}$	0.6847
Bayesiano + CNN	K=10, $\alpha = 4^{-9}$	0.7088
Bayesiano + CNN + MRMR	K=10, $\alpha = 1^{-11}$	0.7320
Bayesiano + CNN + CHI	K=10, $\alpha = 2^{-8}$	**0.7819**
Bayesiano + Smote	$\alpha = 4^{-9}$	0.6811
Bayesiano + Smote + MRMR	$\alpha = 4^{-8}$	0.7047
Bayesiano + Smote + CHI	$\alpha = 5^{-8}$	0.6865

Tabla 15: Resultados promedios obtenidos sobre las diferentes combinaciones de modelos bayesianos. Se observa que equilibrar el conjunto de datos es crucial para las mejoras del modelo. Además, la selección de variables realmente significativas, eliminando redundancia también facilita la generalización del mismo.

Modelo	Parámetros	Precisión	Sensibilidad	F1
Bayesiano + CNN + CHI	$\alpha = 2^{-8}$	0.8084	0.8396	0.8231

Tabla 16: Resultados finales obtenidos en el conjunto de test con el mejor modelo bayesiano.

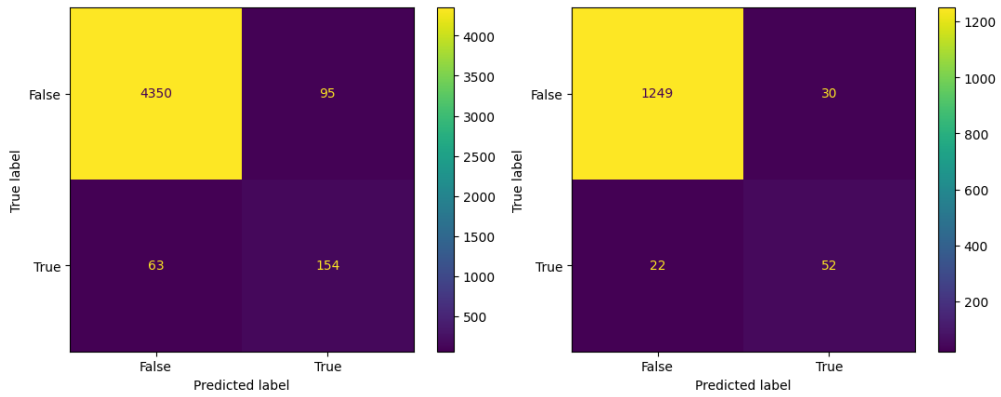


Figura 11: A la izquierda se observa la matriz de confusión de entrenamiento del mejor modelo bayesiano obtenido según validación cruzada. Es notable la cantidad de falsos negativos, podrían ser demasiados en entornos reales donde esta equivocación puede incurrir en algún riesgo. Observamos un comportamiento similar en la evaluación final (derecha).

7. Árboles de clasificación (Miguel García)

7.1. Contexto

Los árboles de decisión, ya sean para regresión o clasificación, son un tipo de algoritmo de aprendizaje automático que representa la toma de decisiones como un árbol hasta llegar a la clase o etiqueta correspondiente al *input* dado (en el caso de clasificación).

- Los nodos **decisión**, que representan una condición sobre las variables. Es una forma de dividir los datos.
- Los nodos **hoja**, que representan la etiqueta o categoría final.

La intuición básica detrás del algoritmo es la de intentar obtener los nodos de decisión más puros posibles (maximizar), esto es, los que sean más discriminatorios entre clases o los que aporten más información.

7.2. Parámetros

El principal parámetro que ha de escogerse para crear un árbol de decisión es el de la función de pérdida. Existen varias, pero las más usadas son la función de **Entropía** y la función de **Gini**.

La función de entropía es la siguiente:

$$Entropy = \sum_{i}^{N} -p_i log(p_i) \tag{1}$$

Donde p_i es la probabilidad de la clase i. El valor más alto de entropía es 1 que significa máxima impureza y el más bajo 0, que es la mayor pureza (o división perfecta).

La función de *Gini* es:

$$I_G(p) = \sum_{i=1}^{J}(p_i \sum_{k \neq i}^{J}(1 - p_i)) = 1 - \sum_{i=1}^{J} p_i^2 \tag{2}$$

Normalmente *Gini* es más utilizado por su eficiencia, pero ambas funciones son válidas y rápidas.

Otros parámetros a tener en cuenta para definir el árbol de decisión son:

- **max_depth**: Profundidad máxima del árbol. Limita el crecimiento para evitar sobreajuste.

- **min_samples_split**: Mínimo de muestras para dividir un nodo. Evita divisiones no significativas.
- **min_samples_leaf**: Mínimo de muestras en una hoja. Garantiza predicciones confiables.
- **max_features**: Número máximo de características consideradas en cada división. Reduce sobreajuste y acelera entrenamiento.

7.3. Características del algoritmo

Los árboles son un tipo de algoritmo sencillo en cuanto a preprocesamiento se refiere. Estos son capaces de lidiar con datos nulos, categóricos y numéricos sin necesidad de transformarlos. No es necesaria la normalización o estandarizado, ya que los árboles de decisión son invariantes en escala porque las separaciones lineales que buscan tratan de maximizar la homogeneidad entre clases considerando una variable a la vez, por lo que se anula el posible efecto que tendría una correlación con otra variable de diferente escala.

No son todo ventajas, ya que los árboles de decisión son un tipo de algoritmo que tiene una varianza muy alta, por lo que cualquier cambio en los datos o parámetros pueden dar resultado a modelos muy distintos.

Tampoco asumen distribuciones especiales sobre los datos (más allá de que son separables, aunque se puede establecer un criterio de parada), por lo que son perfectamente aplicables a características no normales o con otro tipo de distribuciones.

7.4. Algoritmo de clasificación

Se ha utilizado la versión de *scikit-learn==1.5.2*. Dentro de esta librería se ha escogido el clasificador *DecisionTreeClassifier*.

Se escoge por defecto el modo **"balanced"**, que utiliza los valores de y para ajustar automáticamente los pesos de manera inversamente proporcional a las frecuencias de las clases en los datos de entrada. Esto se calcula como:

$$weights = \frac{n\_samples}{n\_classes \cdot np.bincount(y)} \tag{3}$$

Donde:

- **n_samples**: Número total de muestras.
- **n_classes**: Número de clases únicas en y.
- **np.bincount**(y): Cuenta el número de ocurrencias de cada clase en y.

Este enfoque ayuda a manejar desequilibrios en las clases, ya que el problema que se trata de resolver es una clasificación de anomalías, con un alto grado de desbalance. Así se asignan mayores pesos a las clases minoritarias y menores pesos a las clases mayoritarias.

7.5. Búsqueda de hiperparámetros

Se realiza una búsqueda de hiperparámetros sobre las variables de *max_ depth, min_ samples_ split, min_ samples_ leaf* y *criterion*. Todas estos hiperparámetros han sido explicados en la subsección anterior.

Se utiliza el algoritmo de *Random Search* de *scikit-learn* buscando maximizar la métrica de precisión.

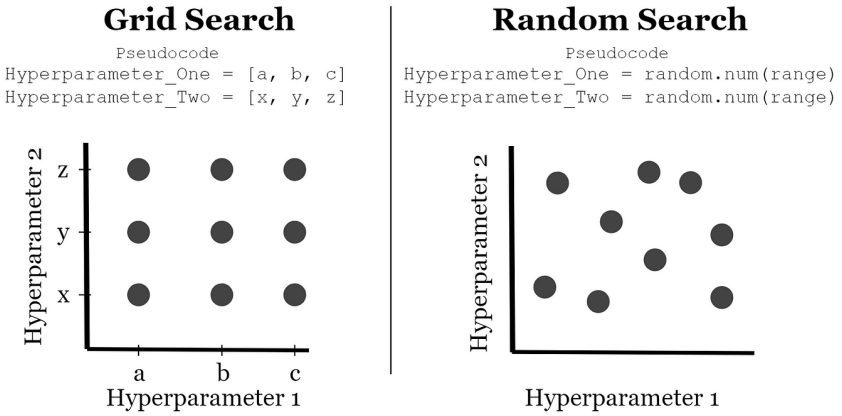


Figura 12: Visualización de *Grid Search vs Random Search*.

Se ha escogido el algoritmo de búsqueda de *Random Search* en vez de *Grid Search* ya que este es capaz de encontrar valores muy intersantes dado un espacio de búsqueda. En *Grid Search* se ha de definir los valores que se deben probar, por lo que es posible perder ciertos tipos de combinaciones muy interesantes.

Estudios empíricos han demostrado que *Random Search* tiende a encontrar combinaciones de hiperparámetros óptimas o cercanas a óptimas con menos iteraciones que *Grid Search*. Esto se debe a que, en muchos casos, no todos los hiperparámetros son igualmente importantes, y *Random Search* explora de manera más efectiva las regiones más relevantes del espacio de búsqueda.

La **precisión** es una métrica de evaluación de modelos de clasificación que mide la proporción de predicciones positivas que son correctas, tal y como se explica en el apartado de métricas del principio de la documentación. Se define como:

$$precision = \frac{TruePositives(TP)}{TruePositives(TP) + FalsePositives(FP)} \tag{4}$$

Donde:

- **True Positives (TP)**: Son los casos en los que el modelo predice correctamente la clase positiva.
- **False Positives (FP)**: Son los casos en los que el modelo predice incorrectamente la clase positiva (falsos positivos).

La precisión es especialmente útil en problemas donde los **true positives** son críticos y los **false positives** tienen un costo asociado. En este caso, captar anomalías es un ejemplo donde los **TP** son esenciales y los **FP** pueden tener un costo, ya que detectar una falsa anomalía en este caso puede llevar a la toma de decisiones como revisiones asociadas a un mal gasto.

Parámetro	**Valor**
criterion	entropy
max_depth	15
min_samples_leaf	2
min_samples_split	17

Tabla 17: Mejores parámetros encontrados para el árbol de decisión.

Se pueden observar los mejores parámetros encontrados para el árbol de decisión en la tabla 17. Hay que tener en cuenta que los árboles son muy variantes, como se ha explicado anteriormente, por lo que cada ejecución puede variar significativamente de parámetros, sobre todo si se usa *Random Search* y sin semilla para la reproducibilidad. Los experimentos lanzados dan, pese a la variabilidad, resultados muy similares.

7.6. Resultados

	Precision	**Recall**	**F1-Score**	**Support**
False	1.0000	0.9960	0.9980	4445
True	0.9234	1.0000	0.9602	217
Accuracy		0.9961		
Macro Avg	0.9617	0.9980	0.9791	4662
Weighted Avg	0.9964	0.9961	0.9962	4662

Tabla 18: Reporte de resultados en *training*.

Tal y como se puede observar en los reportes 18 y 19 los resultados son muy buenos. El árbol ha sido capaz de reconocer la gran mayoría de anomalías. Obviamente el *accuracy* es prácticamente del 100 % ya que hay que tener en cuenta que casi todas las instancias son normales, por lo que discernir este tipo de observaciones es mucho más fácil. En *test* el modelo

	Precision	Recall	F1-Score	Support
False	0.9976	0.9945	0.9961	1279
True	0.9103	0.9595	0.9342	74
Accuracy		0.9926		
Macro Avg	0.9540	0.9770	0.9651	1353
Weighted Avg	0.9929	0.9926	0.9927	1353

Tabla 19: Reporte de resultados en *test*.

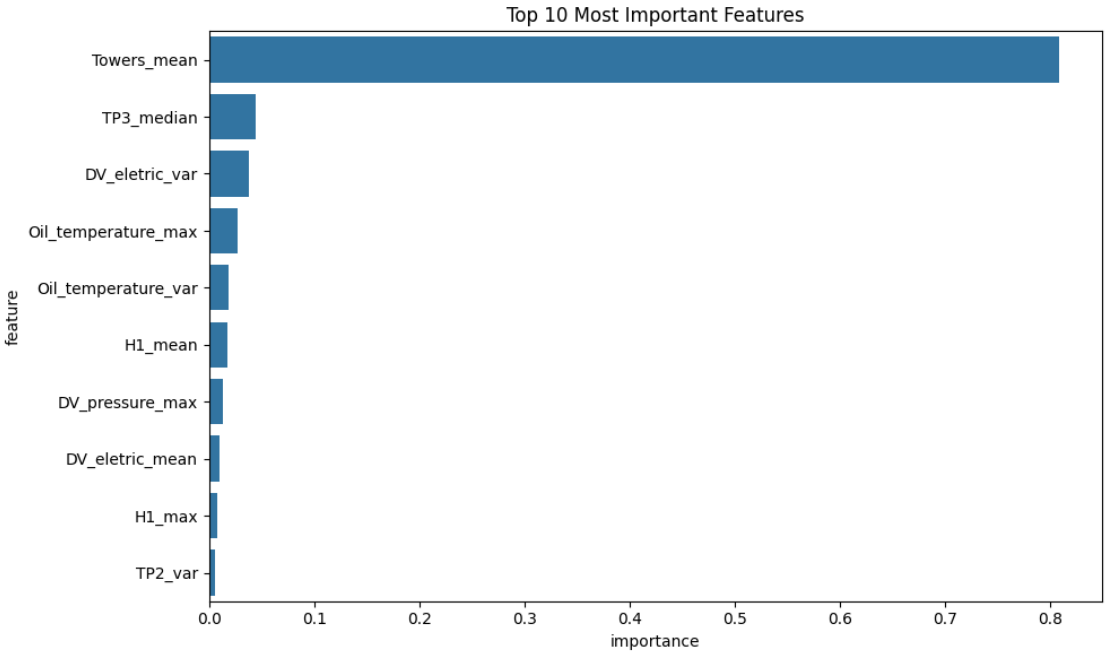


Figura 13: Visualización de importancia de cada característica dado el árbol de decisión entrenado.

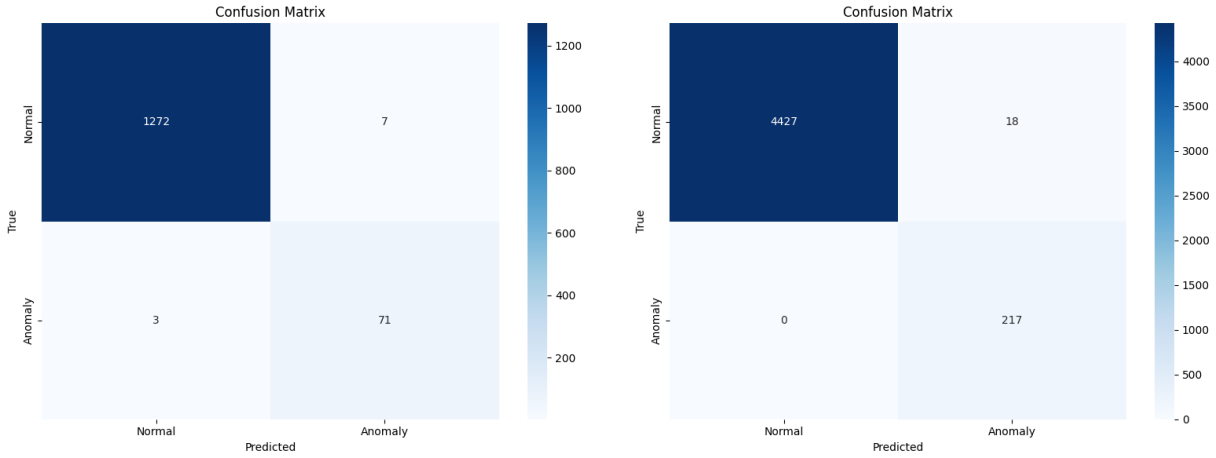


Figura 14: Matriz de confusión de los resultados obtenidos.

es capaz de discernir casi todas las anomalías, alcanzando un porcentaje de precisión del 91 %. En las matrices de confusión definidas en la figura 14 puede verse como los falsos positivos son mínimos, y la mayoría de predicciones sobre anomalías son verdaderos positivos.

Una característica importante y muy relevante de los árboles de decisión es que son capaces de extraer información sobre la relevancia de cada característica. Esto es visible en 13, donde se ve que la variable más relevante para el árbol ha sido *Towers_mean*.

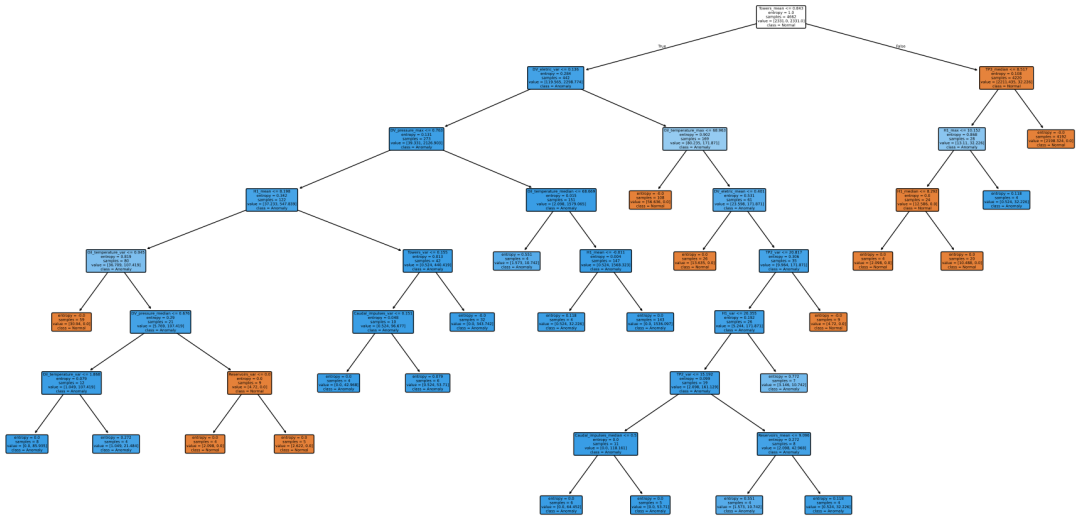


Figura 15: Conjunto de reglas del árbol de decisión entrenado.

Por visualización, también es posible extraer las reglas que constituyen el motor de inferencia de los árboles. El árbol extraído se puede analizar en la figura 15

8. Gradient Boosting (Miguel García)

8.1. Contexto

Se ha escogido el algoritmo de **XGBoost** como algoritmo de *ensemble* con técnicas de *gradient boosting*.

El *gradient boosting* es una técnica de aprendizaje automático que se utiliza para problemas de regresión y clasificación. Consiste en construir un modelo de predicción en forma de un conjunto de modelos débiles, típicamente árboles de decisión, de manera secuencial. Cada nuevo modelo intenta corregir los errores cometidos por los modelos anteriores, optimizando una función de pérdida.

XGBoost (eXtreme Gradient Boosting) es una implementación optimizada y eficiente de *gradient boosting* que incluye regularización para evitar el sobreajuste.

8.2. Parámetros

Los parámetros principales del algoritmo son:

- **n_estimators**: Número de árboles en el conjunto.
- **max_depth**: Profundidad máxima de los árboles individuales.
- **learning_rate**: Tasa de aprendizaje.
- **subsample**: Fracción de datos de entrenamiento usados en cada árbol.
- **colsample_bytree**: Fracción de características usadas para cada árbol.
- **gamma**: Reducción mínima de pérdida requerida para hacer una partición adicional en un nodo hoja.
- **min_child_weight**: Suma mínima de pesos de instancia necesarios en un hijo.

8.3. Características del algoritmo

Al ser un algoritmo que se basa en modelos como el anteriormente descrito árbol de decisión, ya que es un algoritmo de *ensemble*, comparte todas las asunciones previas de los datos, es decir, ninguna. No es necesario ni escalar/normalizar, ni transformar datos categóricos (que no los hay). Más allá de la extracción de características previas, que era un proceso independiente de los algoritmos usados, no se ha realizado ningún tipo de preprocesado para **XGBoost**.

8.4. Algoritmo de clasificación

Se ha usado la versión de **XGBoost** de la librería *xgboost==2.1.3*.

8.5. Búsqueda de hiperparámetros

Se realiza el mismo proceso que el realizado en el árbol de decisión. Se menciona (por describir todo el trabajo realizado) que se ha probado el uso de un algoritmo de búsqueda de hiperparámetros *bayesiano* mediante la librería *hyperopt*. Por simplicidad, justicia de comparación y coherencia se mantiene el *Random Search* como algoritmo de búsqueda principal.

Parámetro	**Valor**
colsample_bytree	0.8778
gamma	1.5361
learning_rate	0.2528
max_depth	4
min_child_weight	2
n_estimators	448
subsample	0.5743

Tabla 20: Mejores parámetros encontrados para el modelo **XGBoost**.

Los resultados obtenidos son los descritos en la tabla 20.

8.6. Resultados

	Precision	**Recall**	**F1-Score**	**Support**
False	0.9996	0.9996	0.9996	4445
True	0.9908	0.9908	0.9908	217
Accuracy		0.9991		
Macro Avg	0.9952	0.9952	0.9952	4662
Weighted Avg	0.9991	0.9991	0.9991	4662

Tabla 21: Reporte de resultados en *training* de **XGBoost**.

Los resultados mostrados en las tablas 21 y 19 mejoran bastante las métricas obtenidas por solo un árbol de decisión. Los resultados de precisión de *test* mejoran en un 4 %. Dentro de que los resultados ya eran excelentes, **XGBoost** los mejora incluso más.

Al igual que en el árbol de decisión singular, con este método también es posible obtener una gráfica que muestre la importancia de las variables. En la figura 16 se muestra como las variables de importancia cambian con respecto al árbol de decisión. El top k sigue siendo

	Precision	Recall	F1-Score	Support
False	0.9969	0.9977	0.9973	1279
True	0.9589	0.9459	0.9524	74
Accuracy		0.9948		
Macro Avg	0.9779	0.9718	0.9748	1353
Weighted Avg	0.9948	0.9948	0.9948	1353

Tabla 22: Reporte de resultados en *test* de **XGBoost**.

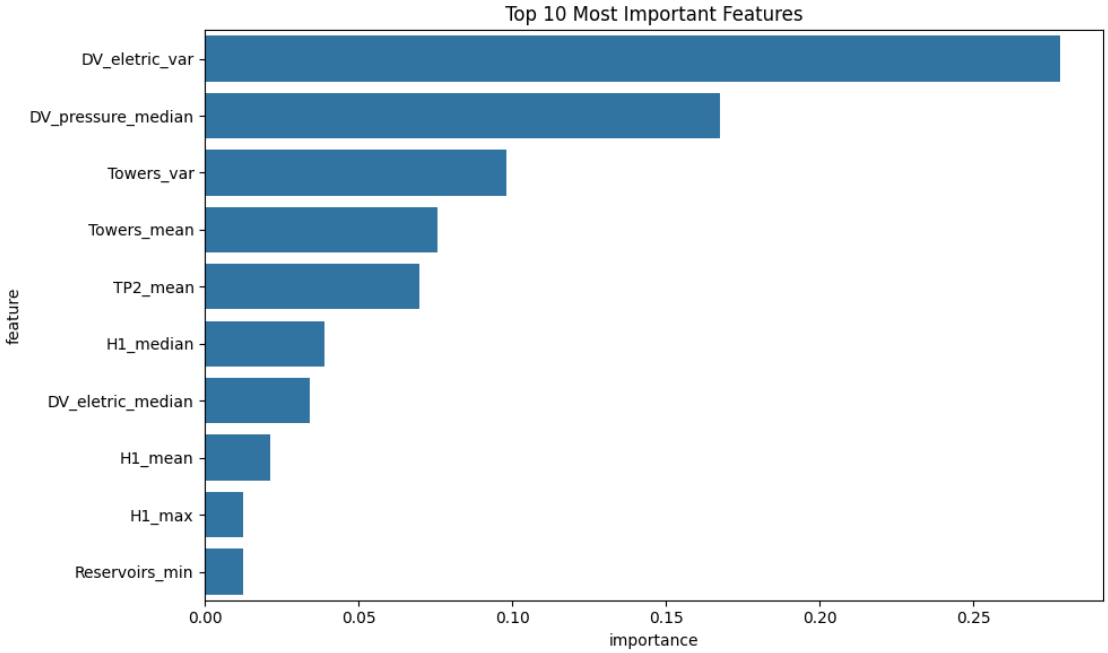


Figura 16: Visualización de importancia de cada característica dado el modelo de **XGBoost**.

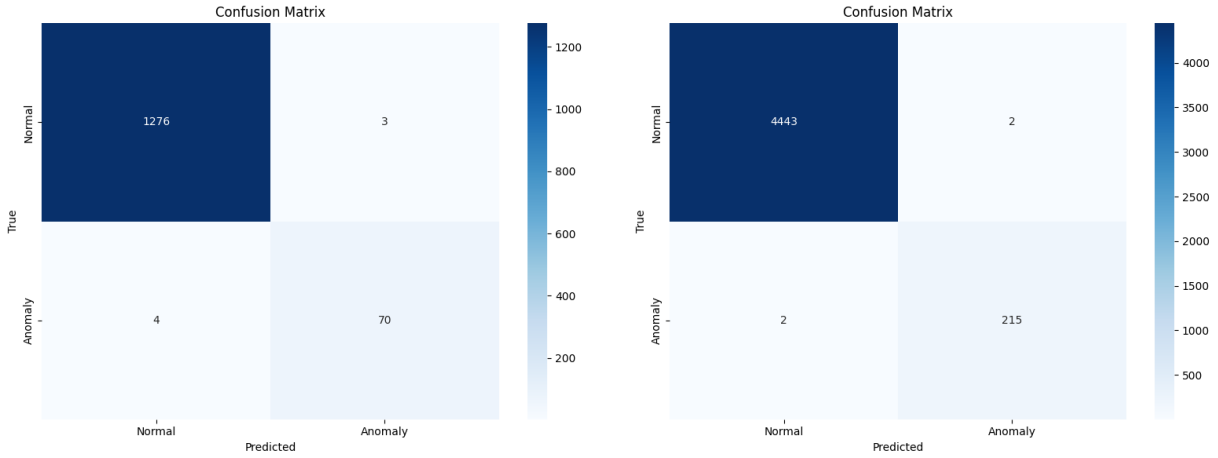


Figura 17: Matriz de confusión de los resultados obtenidos en **XGBoost**.

parecido, aparecen algunas de las mismas variables en ambos gráficos, pero se reordenan y se le dan a las variables un porcentaje más distribuido de importancia.

Se muestran las gráficas de matriz de confusión en la figura 17.

9. AdaBoost (Álvaro Santana Sánchez)

AdaBoost, o Adaptive Boosting, es un algoritmo de aprendizaje automático de la familia de métodos de ensamble avanzados, diseñado para mejorar el desempeño de clasificadores débiles combinándolos en un modelo fuerte. El proceso comienza entrenando un clasificador débil en el conjunto de datos, tras lo cual asigna mayores pesos a las muestras que fueron clasificadas incorrectamente. En las iteraciones posteriores, los clasificadores débiles subsiguientes se enfocan en corregir los errores de sus predecesores. Este ajuste adaptativo de los pesos permite que el modelo sea particularmente eficaz al abordar los ejemplos más difíciles, mejorando de manera incremental su precisión global. AdaBoost es ampliamente utilizado debido a su capacidad para reducir el error de modelos base de aprendizaje, optimizando su desempeño mediante la combinación iterativa y ponderada de múltiples clasificadores débiles. En este caso se ha utilizado el árbol de decisión con estimador dentro de AdaBoost.

9.1. Hipérparámetros

Se realiza la búsqueda de los mejores parámetros mediante *GridSearch*. En ella se utiliza tanto parámetros del propio Árbol de decisión utilizado como los propios de AdaBoost:

Parámetro	Valores
estimator__criterion	gini, entropy
estimator__splitter	best, random
estimator__max_depth	3, 5, 10
estimator__min_samples_split	2, 5, 10
estimator__min_samples_leaf	1, 2, 5
estimator__class_weight	{0: 1, 1: 1}, {0: 1, 1: 10}, {0: 1, 1: 20}, {0: 1, 1: 40}, {0: 1, 1: 100}
n_estimators	1, 2, 10, 50, 100, 200
learning_rate	0.01, 0.1, 0.5, 1

Tabla 23: Cuadrícula de hiperparámetros para búsqueda.

- criterion (DTC) : Determina la función utilizada para medir la calidad de una división en el árbol de decisión.
  - gini: Índice de Gini, que mide la impureza de las particiones.
  - entropy: Medida basada en la entropía, relacionada con la teoría de la información.
- splitter (DTC): Controla el método utilizado para elegir cómo dividir los nodos:
  - best: Selecciona la mejor división posible.
  - random: Elige una división aleatoria entre las mejores opciones.
- max_depth (DTC): Establece la profundidad máxima del árbol de decisión para evitar el sobreajuste (overfitting). Los valores de 3, 5 y 10 ofrecen distintas capacidades de modelado, siendo 3 más restrictivo y 10 más flexible.
- min_samples_split (DTC): Determina el número mínimo de muestras requeridas para dividir un nodo interno. Con valores de 2, 5 y 10, se ajusta la complejidad del árbol; valores más altos restringen el crecimiento del árbol.
- min_samples_leaf (DTC): Establece el número mínimo de muestras requeridas en una hoja del árbol. Valores de 1, 2 y 5 regulan la granularidad de las divisiones finales, evitando que las hojas contengan solo unos pocos datos, lo que puede generar overfitting.
- n_estimators (AdaBoost): Define el número de clasificadores débiles (árboles de decisión) a ser combinados por AdaBoost. Se prueban configuraciones con 1, 2, 10, 50, 100 y 200 clasificadores, donde un número mayor generalmente mejora el modelo, pero también aumenta la complejidad y el tiempo de entrenamiento.
- learning_rate (AdaBoost): Controla la velocidad de aprendizaje, regulando la contribución de cada clasificador débil al modelo final. Los valores de 0.01, 0.1, 0.5 y 1 permiten ajustar la influencia de los clasificadores en el conjunto final, con valores más bajos realizando actualizaciones más pequeñas y valores más altos acelerando el aprendizaje.

- class_weight: aporta mayor valor a los aciertos clase desbalanceada para favorecer las clasificación de los mismo.

Tras realizar el cross validation con todas las combinaciones posibles de parámetros, aquella cuyo f1 medio supero el resto de combinaciones es la que junta los valores del estimador presentes en la Tabla 24 y los valores de AdaBoost presentes en la Tabla 25.

Criterio	max_depth	min_samples_leaf	min_samples_split	splitter	class_weight
gini	3	5	2	best	0:1,1:1

Tabla 24: Mejores hiperparámetros DTC.

learning_rate	n_estimators
1	200

Tabla 25: Mejores hiperparámetros AdaBoost.

9.2. Evaluación de métricas es test

Con estos parámetros, se entrena el modelo de AdaBoost utilizando el conjunto completo de datos de entrenamiento. Después de entrenarlo, el modelo genera predicciones sobre los datos de prueba y se calculan diversas métricas, que se presentan en la Tabla 26.

Métrica	Valor
F1	0.939
Recall	0.945
Precision	0.933
Accuracy	0.995

Tabla 26: Medidas de test.

En la Tabla 27 pueden verse los valores de la matriz de confusión referente a la evaluación del modelo.

1274	5
4	70

Tabla 27: Matriz de confusión para el modelo Ada Boost.

AdaBoost es capaz de identificar la mayoría de las anomalías presentes en el problema, pero la mejora en comparación con clasificadores más simples, como la regresión logística, es

marginal: apenas se obtienen 3 verdaderos positivos adicionales y 4 verdaderos negativos más (véase 14). Dado el potencial de los modelos de ensemble y la complejidad de las funciones que pueden representar, este resultado sugiere que el incremento en desempeño podría no justificar la mayor complejidad del modelo.

10. Stacking (Brian Sena)

El *Stacking* (apilamiento) es una técnica de aprendizaje automático que combina las predicciones de múltiples modelos base para construir un modelo final más robusto y preciso. A diferencia de otros métodos de ensamblado, en el *Stacking* se utiliza un modelo de nivel superior, conocido como meta-modelo, que aprende a combinar las predicciones de los modelos base. El *Stacking* es particularmente útil cuando diferentes modelos base capturan distintos patrones en los datos, ya que el meta-modelo puede aprovechar la diversidad para mejorar el rendimiento global.

10.1. Asunciones de Stacking

- **Diversidad de los modelos base**: El stacking asume que los modelos base son suficientemente diversos y que cada uno aporta información única sobre los datos. Si los modelos base son demasiado similares, su combinación puede no aportar mejoras significativas.
- **Relación entre modelos base y meta-modelo**: El meta-modelo debe ser lo suficientemente flexible como para aprender a interpretar las predicciones de los modelos base. Esto incluye identificar cuáles son más confiables para ciertas regiones del espacio de características.
- **Distribución de clases equilibrada**: Si los datos están desbalanceados, tanto los modelos base como el meta-modelo pueden sesgar sus predicciones hacia la clase mayoritaria, reduciendo la capacidad de detectar la clase minoritaria.
- **Independencia de errores**: Aunque los modelos base pueden cometer errores, se espera que estos errores no estén perfectamente correlacionados entre ellos, de modo que el meta-modelo pueda mitigarlos al combinar las predicciones.

10.2. Preprocesamiento

Se ha utilizado como modelo base aquellos implementados en diferentes secciones. Es decir, máquina de soporte vectorial (SVM por sus siglas en inglés), árbol de decisión y clasificador bayesiano simples. Los modelos son suficientemente distintos como para asumir que cada uno aportará información relevante para la resolución del problema. El árbol de decisión es capaz

de modelar no-linealidades, la máquina de soporte una frontera de decisión y, por último, el clasificador bayesiano las probabilidades de las clases. Para ello, se ha creado un pipeline específico para cada modelo para preprocesar los datos conforme a sus necesidades. Por ejemplo, en el caso de las máquinas de soporte vectorial, escalar los datos. No se dispone de variables categóricas ni valores faltantes, por lo cual lo único que difiere es la presencia (o no) del escalado y la posible selección de variables. Para el caso de los árboles no se ha realizado ningún paso previo. Para el caso de SVM, se ha añadido un escalado como paso previo. Para experimentar con el ajuste de la distribución de las clases, al igual que en el clasificador bayesiano, se ha hecho experimentos con "*CondensedNearestNeighbour* (CNN)" y "*SMOTE + TomekLinks* (SMOTETomek)". Aunque la idea es modelar diferentes características en cada modelo, y que sus errores sean independientes, se ha realizado una prueba adicional utilizando los mejores parámetros obtenidos en las experimentaciones previas de cada modelo individual.

10.3. Detalles de experimentación y resultados

Se ha hecho uso de las mismas librerías definidas en la Sección 6.3. Los mismos 4 pliegues de validación cruzada y las métricas de la Sección 3. Los hiperparámetros que se buscan para cada modelo serán similares a los que se han utilizado en las diferentes secciones. Para el modelo de soporte vectorial se buscarán los valores de regularización (C) entre 0.1, 1 y 10. Para el árbol de decisión se ha experimentado tanto con criterio de corte (ct) de "gini" y "entropía". También se estudia profundidad ilimitada frente a una máxima equivalente a 10. Los resultados se recogen en la Tabla 28 y la matriz de confusión de entrenamiento en la Figura 18.

Modelo	**Parámetros**	**F1 Promedio**
Stacking	C=10, $\alpha = 1e^{-9}$, ct="entropía"	0.9376
Stacking + SMOTE	C=10, $\alpha = 1e^{-9}$, ct="gini"	0.9309
Stacking + CNN	C=10, $\alpha = 1e^{-9}$, ct="gini"	0.8726
Stacking + OPT	C=10, $\alpha = 1e^{-9}$, ct="entropía"	**0.9438**

Tabla 28: Resultados promedios obtenidos sobre las diferentes combinaciones de modelos de apilamiento. El término OPT hace alusión a un modelo de apilamiento en el cuál cada modelo base está utilizando la mejor configuración posible (El bayesiano se entrena con un submuestreo por CNN y características seleccionadas por chi-cuadrado). Se observa una posible saturación del conjunto de entrenamiento.

Los resultados del mejor modelo sobre el conjunto final de evaluación se recogen en la Tabla 29 y su matriz de confusión en la Figura 18. Utilizar este conjunto de modelos y un meta-modelo para modelar la iteracción entre ellos resulta ser muy eficaz para la resolución de este problema.

Modelo	Parámetros	Precisión	Sensibilidad	F1
Stacking + OPT	C=10, α=$1e^{-9}$, ct="entropía"	0.9889	0.9387	0.9623

Tabla 29: Resultados finales obtenidos en el conjunto de test con el mejor modelo de apilamiento.

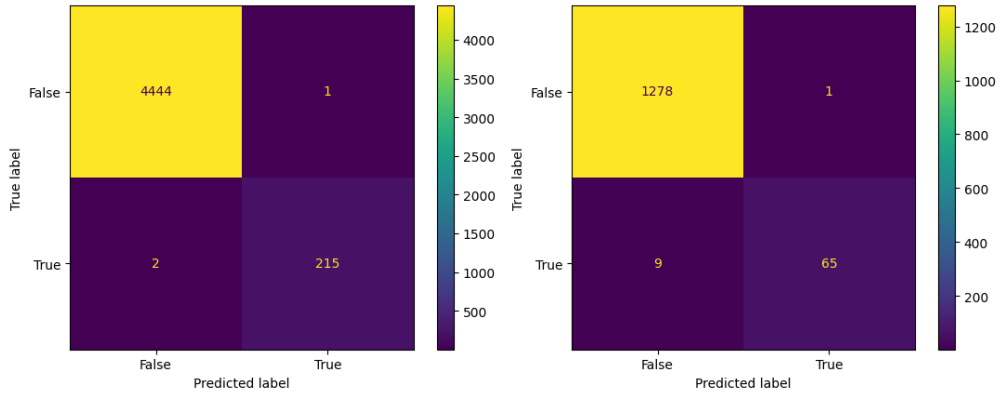


Figura 18: A la izquierda se observa la matriz de confusión de entrenamiento del mejor modelo de apilamiento obtenido según validación cruzada. Observamos un rendimiento de casi el 100 % sobre el conjunto de entrenamiento. En el conjunto final de evaluación presentamos una cantidad razonable de falsos negativos.

11. Bagging (Ana Fuentes)

El método *Bagging* (Bootstrap Aggregating) es una técnica de ensemble diseñada para mejorar la precisión y estabilidad de los modelos de aprendizaje supervisado. Bagging combina múltiples modelos base, usualmente árboles de decisión, entrenados independientemente sobre diferentes subconjuntos de datos generados mediante muestreos por reemplazo (bootstrap sampling). El objetivo principal de esta técnica es reducir la varianza de los modelos base al promediar o votar los resultados individuales. Esto produce un modelo final más robusto y también menos propenso al sobreajuste.

Bagging es ampliamente utilizado en problemas de clasificación y regresión debido a su simplicidad y su capacidad para mejorar el rendimiento de los modelos base en una gran variedad de problemas.

Para trabajar con los diferentes algoritmos de bagging, no ha sido necesario ningún preprocesamiento específico de los datos, ni codificación de variables categóricas (no hay) ni escalado de las variables.

De esta manera, se han implementado tres ensembles; Random Forest, ExtraTrees y Random subspaces.

11.1. Random Forest

Random Forest es un ensemble de árboles de decisión entrenados con subconjuntos de datos obtenidos mediante muestreo con reemplazo. Estos árboles se construyen de manera independiente y las predicciones se combinan mediante el voto mayoritario.
Para este modelo, se han tomado los siguientes hiperparámetros:

- n_estimators: este parámetro es el número de árboles. Se elige entre 100, 200 y 300.
- max_depth: este hace referencia a la profundidad máxima de los árboles. Se elige entre "None", 10 y 20.
- min_samples_split: este es el número mínimo de muestras requeridas para dividir un nodo. Se elige enre 2, 5 y 10.
- min_samples_leaf: es el número mínimo de muestras en una hoja. Se elige entre 1, 2 y 4.

De esta manera, se han generado 324 combinaciones y se han entrenado con la librería "*sklearn.ensemble*" para obtener el mejor modelo. Así pues, la mejor combinación se ha mostrado en la Tabla 30.

n_estimators	max_depth	min_samples_split	min_samples_leaf	F1 promedio
100	None	2	4	0.93851

Tabla 30: Resultados de los mejores hiperparámetros obtenidos tras entrenar el modelo Random Forest.

Tras haber entrenado y haber obtenido la mejor combinación de los hiperparámetros, se ha evaluado el modelo con el conjunto de prueba, obteniéndose los resultados de la clasificación de la Tabla 31. También se ha mostrado la matriz de confusión en la Tabla 32.

Clase	Precisión	Recall	F1-score	Soporte
False	1.00	1.00	1.00	1279
True	0.96	0.95	0.95	74
macro avg	0.98	0.97	0.97	1353
weighted avg	0.99	0.99	0.99	1353

Tabla 31: Informe de clasificación tras la evaluación del mejor modelo de Random Forest para el conjunto de prueba.

1276	3
4	70

Tabla 32: Mattriz de confusión obtenida para el ensemble Random Forest.

Analizando los resultados de estas Tablas, se puede comprobar que hay un mejor equilibrio entre la precisión y el recall. Por otro lado, aunque los resultados son notablemente buenos, para la clase "True" (es anomalía), se han obtenido peores resultados comparados con los arrojados para la clase "False" (no es anomalía), lo que puede explicarse debido al desbalanceo presente entre estas clases (74 frente a 1279).

Se puede concluir entonces que este modelo es muy buena opción para tratar estos datos debido a la robustez de este ensemble.

11.2. ExtraTrees

El ensemble ExtraTrees (Extremely Randomized Trees) es similar al algoritmo Random Forest anteriormente descrito, pero introduce más aleatoriedad durante la construcción de los árboles al seleccionar divisiones aleatorias en las características.
En este caso, los hiperparámetros que se han estudiado han sido los mismos que para el caso de Random Forest, pero en esta ocasión la aleatoriedad en las divisiones es mayor. Una vez entrenadas las 64 combinaciones posibles, los mejores parámetros se han reflejado en la Tabla 33.

n_estimators	max_depth	min_samples_split	min_samples_leaf	F1 promedio
200	10	5	2	0.94488

Tabla 33: Resultados de los mejores hiperparámetros obtenidos tras entrenar el modelo ExtraTrees.

Con este modelo ya entrenado, se ha evaluado con el dataset de test, mostrando en la Tabla 34 el reporte de clasificación y en la Tabla 35 la matriz de confusión.

Clase	Precisión	Recall	F1-score	Soporte
False	1.00	1.00	1.00	1279
True	0.97	0.93	0.95	74
macro avg	0.98	0.97	0.97	1353
weighted avg	0.99	0.99	0.99	1353

Tabla 34: Informe de clasificación tras la evaluación del mejor modelo de ExtraTrees para el conjunto de prueba.

1277	2
5	69

Tabla 35: Matriz de confusión generada tras la evaluación del ExtraTrees.

Observando estas dos Tablas, se puede comprobar que este modelo arroja resultados muy similares a Random Forest, pero con un enfoque más aleatorio en la selección de divisiones. En la matriz de confusión se puede observar que para la clase "False" ha fallado menos que el ensemble Random Forest, pero para la clase "True" ha fallado más el ensemble ExtraTrees. De igual modo, este modelo ha demostrado presentar un alto rendimiento para este dataset.

11.3. Random Subspaces

Finalmente, se ha implementado el modelo de ensemble Random Subspaces. Este algoritmo es una técnica que construye un ensemble de clasificadores base (en este caso, se ha utilizado un ensemble de árboles de decisión) entrenados en subconjuntos aleatorios de características. Los hiperparámetros para este modelo son los siguientes:

- n_estimators: este indica el número de clasificadores base en el ensemble. Se elige entre 50, 100 y 150.
- max_features: este término indica la proporción de características seleccionadas para cada clasificador. Se elige entre 0.5, 0.75 y 1.

- estimator_max_depth: es la profundidad máxima de cada árbol de decisión. Se elige entre "None", 10 y 20.

En esta ocasión, se han generado 108 combinaciones, por lo que después de entrenar los modelos generados ha resultado que los mejores parámetros para este conjunto de datos han sido los mostrados en la Tabla 36 (se ha utilizado la librería "*sklearn.tree*").

n_estimators	max_features	estimator_max_depth	F1 promedio
50	0.5	None	0.93251

Tabla 36: Resultados de los mejores hiperparámetros obtenidos tras entrenar el modelo Random Subspaces.

A continuación, se ha evaluado este algoritmo con el conjunto de prueba, obteniéndose los resultados de la Tabla 37 y la matriz de confusión de la Tabla 38.

Clase	Precisión	Recall	F1-score	Soporte
False	1.00	1.00	1.00	1279
True	0.96	0.93	0.95	74
macro avg	0.98	0.97	0.97	1353
weighted avg	0.99	0.99	0.99	1353

Tabla 37: Informe de clasificación tras la evaluación del mejor modelo de Random Subspaces para el conjunto de prueba

1276	3
5	68

Tabla 38: Matriz de confusión para el modelo Random Subspaces.

Aunque este ensemble también arroja unos resultados excelentes, para la clase "False" se ha obtenido que ha fallado en un valor más, comparado con el ensemble anterior. Este ligero empeoramiento de la precisión en esta clase puede deberse a la selección aleatoria de características.

Finalemnte, los tres modelos de ensemble han obtenido excelentes resultados, aunque el algoritmo Random Forest parece arrojar una ligera mejora al clasificar mejor la clase de anomalía.

Referencias

[1] Narjes Davari et al. *MetroPT-3 Dataset*. https://doi.org/10.24432/C5VW3R. UCI Machine Learning Repository. 2021.

[2] Markelle Kelly, Rachel Longjohn y Kolby Nottingham. *The UCI Machine Learning Repository*. Último acceso el 11/01/2025. 2024. URL: https://archive.ics.uci.edu.

[3] M. Barros et al. «Failure detection of an air production unit in operational context». En: *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning*. Springer, 2020, págs. 61-74.

[4] Narjes Davari et al. «Predictive maintenance based on anomaly detection using deep learning for air production unit in the railway industry». En: *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*. 2021, págs. 1-10. DOI: 10.1109/DSAA53316.2021.9564181.

[5] Vikramkumar, Vijaykumar B y Trilochan. *Bayes and Naive Bayes Classifier*. 2014. arXiv: 1404.0933 [cs.LG]. URL: https://arxiv.org/abs/1404.0933.

[6] Guillaume Lemaître, Fernando Nogueira y Christos K. Aridas. «Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning». En: *Journal of Machine Learning Research* 18.17 (2017), págs. 1-5. URL: http://jmlr.org/papers/v18/16-365.

[7] F. Pedregosa et al. «Scikit-learn: Machine Learning in Python». En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830.