Monitorización de tormentas geomagnéticas aplicando técnicas de detección de anomalías mediante aprendizaje no supervisado

Enlace a GitHub

Titulación:
Máster en Inteligencia Artificial
Curso académico
2021 – 2022

Alumno/a: Mateos García, Damián
D.N.I: 77557502F

Director/a de TFM: Cesar Augusto Guzmán Álvarez

Convocatoria:

Tercera

Octubre 2022

De:
Planeta Formación y Universidades

Resumen

El campo magnético interplanetario (*Interplanetary Magnetic Field*, IMF) es un vector que representa el campo magnético proveniente del Sol y transportado por el viento solar entre los planetas. Su existencia influye en las misiones espaciales, la comunicación con los satélites y la interacción con el campo magnético de la Tierra.

Conocer y monitorizar el comportamiento de esta magnitud ayuda a aumentar el conocimiento existente sobre el Sistema Solar y las interacciones entre el Sol y los planetas. Además, detectar anomalías en el comportamiento de esta magnitud es útil para estudiarlas con mayor detalle y encontrar una causa. También esta detección de valores anómalos puede servir de alerta para misiones que puedan verse perjudicadas por ciertos valores de la magnitud.

El objetivo principal del presente Trabajo Fin de Máster es la aplicación de técnicas de aprendizaje no supervisado para monitorizar la evolución temporal del campo magnético interplanetario y detectar *outliers* que signifiquen tormentas geomagnéticas.

Se han utilizado diferentes técnicas de *machine learning* para la detección de anomalías como AutoEncoders variacionales, *Isolation Forest*, *Empirical-Cumulative Outlier Detection* o un algoritmo tipo *ensemble* llamado *Large-Scale Unsupervised Outlier Detection*.

Se compara el rendimiento de todos estos algoritmos en diferentes aspectos como el tiempo de entrenamiento y predicción, el número de hiperparámetros que tienen y los *outliers* detectados. Al estar en un entorno no supervisado sin etiquetas reales, se utiliza la definición estadística más extendida de *outlier* para tener unas etiquetas de referencia. También se aplican técnicas de *online machine learning* simulando la llegada de datos en tiempo real y la adaptabilidad en un proceso de entrenamiento del modelo por ventanas.

Por último, se exponen las conclusiones obtenidas con la realización del trabajo y posibilidades futuras a explorar para mejorar y avanzar con lo comenzado en este TFM.

Abstract

The Interplanetary Magnetic Field (IMF) is a vector that represents the magnetic field coming from the Sun and carried by the solar wind between the planets. Its existence influences space missions, communication with satellites and interaction with the Earth's magnetic field.

Knowing and monitoring the behavior of this magnitude helps to increase the existing knowledge about the Solar System and the interactions between the Sun and the planets. Also, detecting behavioral abnormalities of this magnitude is useful for studying them in greater detail and finding a cause. This detection of anomalous values can also serve as an alert for missions that may be affected by certain magnitude values.

The main objective of this Master's Thesis is the application of unsupervised learning techniques to monitor the temporal evolution of the Interplanetary Magnetic Field and detect outliers that mean geomagnetic storms.

Different machine learning techniques have been used to detect anomalies such as variational AutoEncoders, Isolation Forest, Empirical-Cumulative Outlier Detection or an ensemble algorithm called Large-Scale Unsupervised Outlier Detection.

The performance of all these algorithms is compared in different aspects such as the training and prediction time, the number of hyperparameters they have and the outliers detected. Being in an unsupervised environment without real labels, the most widespread statistical definition of outlier is used to have reference labels. Online machine learning techniques are also applied, simulating the arrival of data in real time and the adaptability in a windowed model training process.

Finally, the conclusions obtained with the completion of the work and future possibilities to be explored to improve and advance with what has been started in this Master's Thesis are presented.

Índice

Índice de ilustraciones

Índice de tablas

Nomenclatura

Advanced Composition Explorer	ACE
Autoregressive Integrated Moving Average Model	ARIMA
Autoregressive Moving Average Model	ARMA
Cumulative Distribution Function	CDF
Deep Learning	DL
Deep Space Climate Observatory	DSCOVR
European Space Agency	ESA
Empirical Cumulative Distribution Function	ECDF
Empirical-Cumulative Outlier Detection	ECOD
Gated recurrent unit	GRU
Generative Adversarial Network	GAN
Heliophysics Data Application Programmer's Interface	HAPI
Inteligencia Artificial	IA
Interplanetary Magnetic Field	IMF
International Space Station	ISS
Internet of things	IoT
Isolation Forest	IForest
Jet Propulsion Laboratory	JPL
K Nearest Neighbours	KNN
Laboratory for Atmospheric and Space Physics	LASP
Local Outlier Factor	LOF
Local Reachability Distance	LRD
Long short-term memory	LSTM
Machine Learning	ML

Mars Exploration Rovers	MER
Mean Absolute Error	MAE
Mean Squared Error	MSE
Multilayer perceptron	MLP
National Aeronautics and Space Administration	NASA
Out-of-Limits	OOL
Principal Component Analysis	PCA
Python Outlier Detection	PyOD
Python Streaming Anomaly Detection	PySAD
Reachability Distance	RD
Seasonal Autoregressive Integrated Moving Average Model	SARIMA
Space Physics Data Facility	SPDF
Space Physics Environment Data Analysis Software	SPEEDAS
Stochastic Gradient Descent	SGD
Stream Machine Learning	SML
Variational AutoEncoder	VAE

1. Introducción

¿Puede la inteligencia artificial tomar parte en el estudio del clima espacial? En caso de poder hacerlo, ¿tendría la capacidad de hacerlo sin datos etiquetados? La respuesta a ambas preguntas es afirmativa y se invita al lector a introducirse en este trabajo si quiere conocer cómo puede aportar la IA en el entorno espacial y una aplicación concreta para la monitorización del campo magnético interplanetario con algoritmos de *machine learning*.

El ámbito espacial es un campo muy moderno, altamente tecnológico y todavía ampliamente desconocido por los seres humanos. La primera actividad espacial fue el lanzamiento de Sputnik en 1957. Desde ese momento que tuvo lugar 65 años atrás, la carrera por descubrir, comprender y explotar el entorno espacial ha sido desenfrenada. La evolución ha sido muy notable pero el camino a recorrer sigue siendo largo y tan interesante o más que en su inicio. Para seguir avanzando, es fundamental el uso de las tecnologías más punteras que permitan aumentar el conocimiento y las posibilidades en el entorno.

Una de las tecnologías más novedosas y potentes que han surgido en los últimos años es la inteligencia artificial, sin lugar a duda. Por ello, su incorporación al entorno espacial permite crecer y recorrer el camino con las mejores herramientas posibles. La Agencia Espacial Europea, en inglés *European Space Agency* (ESA), recoge en un artículo [1] publicado en su web la implicación actual de la IA en este campo. En él, se menciona que el *machine learning* ya está siendo utilizado para el análisis de grandes volúmenes de datos provenientes del espacio sobre la Tierra o datos telemétricos en general. Un ejemplo es la aplicación de técnicas de ML en algunos de los Rovers de Marte, conocidos como *Mars Exploration Rovers* (MER) [2]. Con la inteligencia artificial se ha podido enseñar a algunos de ellos a navegar por sí mismos, así como analizar los datos provenientes de sus exploraciones.

Sin embargo, el artículo de la ESA [1] también expone las necesidades que tiene la IA de crecer para que su uso en el ámbito espacial sea ampliamente extendido. Lejos de abstenerse de participar en este desarrollo, la ESA expone actividades que realiza para

continuar incorporando el uso de la IA en el entorno espacial. También se comentan las aportaciones de otras entidades como la Administración Nacional de Aeronáutica y el Espacio, conocida por sus siglas en inglés NASA, que significan *National Aeronautics and Space Administration*. La ESA recoge en su artículo [1] la cooperación de NASA con Google para entrenar algoritmos de IA que permitieron descubrir dos nuevos exoplanetas que los científicos no lograron identificar sin ayuda de la IA.

Parece claro que la IA puede suponer un gran impulso en el entorno espacial y que su aplicación en este ámbito será cada vez más relevante. En otros ámbitos, la gran cantidad de datos históricos etiquetados ha permitido a la IA construir conocimiento e inferir comportamientos futuros. Por ello, su aplicación experimenta un continuo incremento en campos como la medicina, las finanzas y la industria en general. Sin embargo, ¿es posible etiquetar cualquier tipo de dato, incluidos los datos de misiones espaciales?

Un dato no se puede etiquetar si no se tiene un objetivo claro, es decir, si no existen etiquetas claramente diferenciadas. Muchas misiones espaciales son de investigación y tienen como objetivo recopilar datos para ampliar el conocimiento sobre el entorno espacial. Por ello, pueden no tenerse claras unas etiquetas objetivo en las que basar el entrenamiento de un modelo de ML, tal y como se hace cuando se quiere construir un modelo de aprendizaje supervisado que distinga imágenes de gatos de imágenes de perros.

Muchas misiones espaciales recopilan datos telemétricos sobre alguna magnitud o cuerpo en concreto que se quiere conocer con más detalle. Para obtener conocimiento sobre esos datos, una vía interesante es estudiar si el comportamiento que siguen es el esperado o existen anomalías. Pero ¿qué se considera exactamente una anomalía? ¿Dónde se encuentra el límite entre un valor anómalo o *outlier* y uno que no lo es o *inlier*? La respuesta es que depende de lo que representen los datos, por lo que en la mayoría de ocasiones no hay un límite estricto. Por ello, existen distintas definiciones de lo que es un valor anómalo a nivel estadístico y en muchos casos no se disponen de etiquetas que puedan distinguir datos anómalos de datos normales.

Puede surgir en este momento una pregunta muy interesante: ¿Para qué se quiere conocer si un dato es anómalo o no? Mayormente, existen dos opciones para con esta

intención. La primera es detectar esos valores anómalos para eliminarlos del *dataset* porque se traten de datos erróneos. Un ejemplo sería una edad de 3000 años en un humano, pues claramente es un dato incorrecto y su identificación permite eliminarlo. Es muy importante la calidad de los datos para entrenar un algoritmo de *machine learning*, por lo que datos erróneos han de ser eliminados para evitar confundir al modelo.

La otra intención que se puede tener al identificar anomalías es monitorizar el comportamiento de una o varias magnitudes para evaluar si se encuentra dentro de lo esperado. De esta manera, si el comportamiento de los datos se sale de lo habitual, se pueden analizar las causas que han provocado ese cambio. Así, la detección de anomalías permite identificar comportamientos diferentes para poder estudiar con detalle lo sucedido en esos casos.

El entorno espacial es un claro ejemplo de ámbito en el que la detección de *outliers* va encaminada a monitorizar el comportamiento de una magnitud. Las misiones espaciales ofrecen multitud de datos que pueden ser sobre el satélite que está llevando a cabo la misión o sobre el objeto, cuerpo o entorno del espacio que se está analizando, si es una misión de investigación.

De esta manera, la detección de anomalías con algoritmos de *machine learning* es una tarea que está creciendo en el ámbito espacial. Más adelante en este documento, se recopila una investigación del estado del arte donde se dan más detalles sobre la aplicación de algoritmos de IA para monitorizar misiones espaciales. Como se ha comentado, la ausencia de etiquetas en estos datos se explica por la falta de un conocimiento sólido que permita establecer con claridad esas etiquetas.

Según lo explicado, se requiere en muchas ocasiones el uso de algoritmos de detección de anomalías con métodos no supervisados. La motivación de su uso es generar alertas y monitorizar el comportamiento de los datos telemétricos para llegar a comprender el motivo de las anomalías y aumentar el conocimiento sobre la magnitud o parámetro que se investigue. Es decir, el uso de estos algoritmos puede ayudar al objetivo principal de misiones espaciales de investigación, que no es otro que aumentar el conocimiento que se tiene sobre el entorno espacial.

A continuación, se introducen los objetivos del trabajo y la estructura que se va a seguir en el documento para facilitar su lectura y comprensión.

1.1. Objetivos

En este apartado se va a desarrollar el objetivo general del trabajo, así como los objetivos específicos que engloba y que son derivados del primero.

1.1.1. Objetivo general

Monitorizar la existencia de tormentas geomagnéticas por medio de técnicas de detección de anomalías no supervisadas utilizando un *dataset* de datos telemétricos que recoja la evolución temporal del campo magnético interplanetario.

1.1.2. Objetivos específicos

- Recopilar un resumen completo del estado del arte, con las diferentes opciones tanto de algoritmos a utilizar como de *datasets* para aplicarlos.
- Utilizar varios algoritmos de detección de anomalías sobre un *dataset* y demostrar cuál de ellos obtiene mejores resultados sobre el conjunto de datos.
- Encontrar unas etiquetas que se suponen reales a través de la definición estadística de *outliers* para poder comparar el rendimiento de los algoritmos con métricas de clasificación.
- Utilizar un algoritmo *ensemble* no supervisado que permita obtener una detección de anomalías robusta y genérica dados varios algoritmos.
- Aplicar *online machine learning* para realizar la detección de anomalías en tiempo real y con un entrenamiento continuo.
- Recomendar otros posibles métodos a aplicar y próximos pasos para proseguir con la investigación iniciada en este trabajo.

1.2. Estructura del documento

El documento se divide en los siguientes capítulos:

2. Estado del arte: se expone una amplia investigación realizada sobre *online machine learning*, detección de anomalías y su aplicación a series temporales y datos telemétricos. Se investigan librerías, herramientas y algoritmos relacionados con la temática del presente trabajo.

3. Desarrollo: se detallan los pasos realizados para llevar a cabo el trabajo, comenzando por la extracción del *dataset* y siguiendo con los modelos elegidos, los hiperparámetros que estos tienen y las herramientas necesarias para implementar *online machine learning*.

4. Resultados: se expone la elección de hiperparámetros para cada algoritmo junto a las pruebas realizadas para ello. Además, se muestran las gráficas y tablas convenientes para ilustrar los resultados obtenidos con cada método. Se detallan los resultados de un método sencillo, 3 algoritmos, un algoritmo *ensemble* y la técnica de *online machine learning*.

5. Conclusiones: se concluye la realización de este trabajo junto a posibles vías que se abren para continuar con la exploración en el campo de la detección de anomalías, el *online machine learning* y los datos telemétricos.

2. Estado del arte

En esta sección se encuentra un amplio resumen del estado del arte actual del campo de la detección de anomalías en aprendizaje no supervisado con técnicas de *online machine learning* para su uso en datos telemétricos de misiones espaciales. El apartado se divide tres puntos principales: *online machine learning*, detección de anomalías y datos telemétricos.

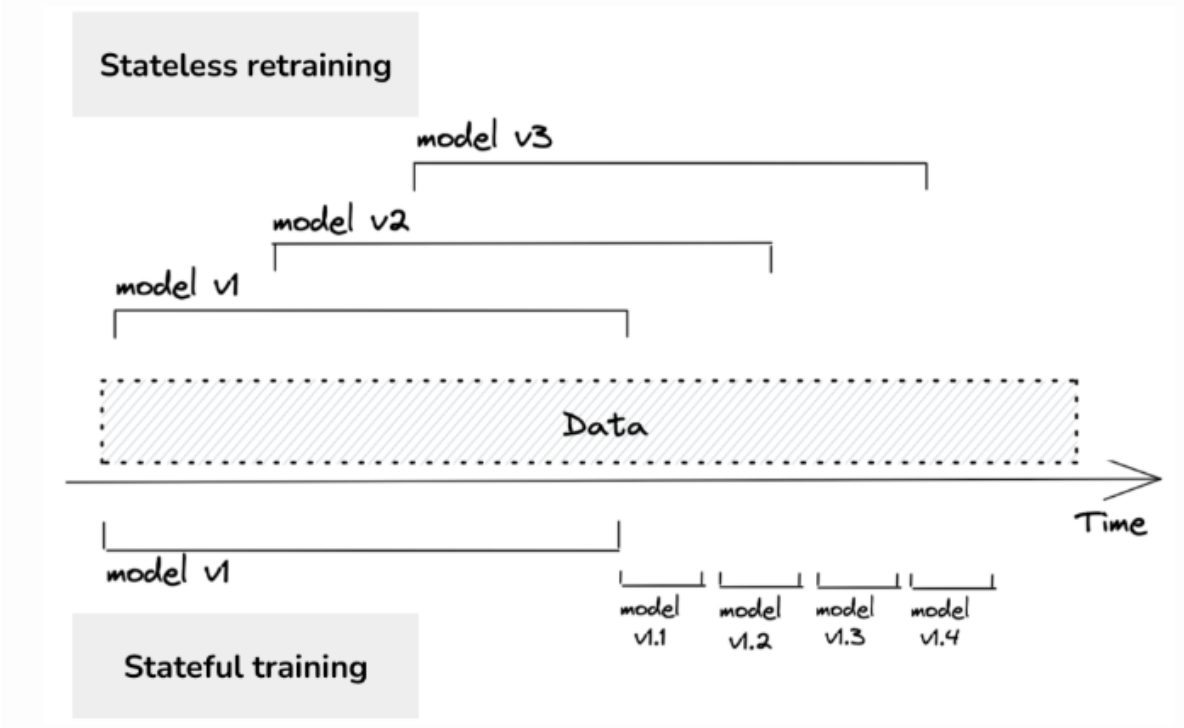
2.1. Online machine learning

El *online machine learning* o *stream machine learning* (SML) es una metodología del aprendizaje automático en la que los datos llegan de manera constante en orden secuencial, es decir, en tiempo real. Las características especiales que aparecen en el entorno del aprendizaje automático en tiempo real son las siguientes:

- Llegada continua de datos, que exige procesarlos rápidamente antes de que aparezcan nuevos. También es muy importante la secuencialidad de los datos, pues van perdiendo importancia conforme quedan alejados en el tiempo.
- Cantidad de datos ilimitada: los *datasets* no tienen un límite, por lo que no todos los puntos van a poder ser almacenados y el método elegido debe ser capaz de trabajar con la limitación de recursos.
- Además, el número de datos que llegan por unidad de tiempo no es constante, sino variable. Esto quiere decir que las distribuciones de datos no son estacionarias y el modelo debe tener esa capacidad de evolución para adaptarse a la variabilidad temporal.

Estas características se resumen en el comportamiento del modelo a lo largo del tiempo, que se ve representado en la Figura 1. Con un entrenamiento continuo del modelo y en pequeños periodos de tiempo, se requieren menos datos para el proceso de aprendizaje y el modelo tiene una evolución más continua.

Figura 1. Entrenamiento continuo frente a entrenamiento sin continuidad [3]



Una vez se han visto las características que tiene el SML, se manifiestan claramente varias de sus ventajas. En primer lugar, se puede contar con una cantidad de datos ilimitada. Por otro lado, el modelo es más adaptable y dinámico frente a los nuevos cambios que se produzcan. Finalmente, la más importante de todas es que se puede obtener una respuesta continua y en tiempo real sobre los datos, para cumplir con la función que requiera el modelo.

En este trabajo se va a utilizar el lenguaje de programación Python, una herramienta muy extendida en la ciencia de datos por su versatilidad y amplio abanico de posibilidades que ofrece gracias a la enorme comunidad de desarrolladores con la que cuenta. A continuación, se exponen librerías de este lenguaje que permiten aplicar SML.

2.1.1. Librerías de tratamiento de datos en tiempo real

Dentro de las distintas plataformas que existen para acceder a datos en tiempo real, la mayoría de ellas son compatibles con Python y tienen paquetes en él. Este lenguaje de programación es el más utilizado en tareas de ciencia del dato y creación de modelos de aprendizaje automático, por lo que permite conectarse con muchas herramientas para obtener datos de manera sencilla.

Redpanda [4] es una plataforma que proporciona acceso a datos en tiempo real. Aparte de la fuente de datos, es vital contar con librerías para completar la tarea de entrenar y mostrar los resultados de un modelo que recibe esos datos. HoloViews [5] es una librería muy interesante que facilita la realización de gráficos dinámicos en tiempo real. Además, construir un *dataflow* desde la plataforma de *data stream* permite alcanzar y procesar los datos que se deseen. Un paquete interesante de Python para llevar a cabo esta tarea es Bytewax [6]. También existe una librería en este mismo lenguaje que es ampliamente utilizada para construir modelos de *stream machine learning* llamada River [7]. Por otro lado, es importante poder leer los datos de la plataforma de *data stream*, para lo cual se puede utilizar kafka-python [8], un paquete que permite cumplir con ello a través de la API de Apache Kafka.

Mientras que las anteriores son herramientas que permiten implementar algoritmos de *machine learning* o usar datos en tiempo real con cualquier fin, existen también librerías específicas para un determinado fin. Es el caso de PySAD (*Python Streaming Anomaly Detection*), una librería destinada a la implementación de algoritmos de detección de *outliers* en tiempo real. Al estar especializada únicamente para esta tarea, cuenta con mayor variedad de modelos y sencillez de aplicación que otras herramientas.

En definitiva, existen numerosas herramientas para el tratamiento de datos en tiempo real. Sin embargo, cada una cuenta con su utilidad y limitaciones. Por ejemplo, River no puede usarse para construir un *dataflow* y aplicarle cualquier algoritmo que se desee, sea de clasificación, regresión o detección de anomalías, pudiéndose solo utilizar los algoritmos que están implementados en la librería. Por otro lado, kafka-python es útil si se quiere procesar los datos en tiempo real a través de Apache Kafka, pero está más orientado al mundo empresarial y entornos de Big Data. De esta manera, la elección de las herramientas vendrá motivada por el uso y requerimientos de la aplicación concreta.

2.2. Detección de anomalías

Desde finales del siglo 19 en el campo de la estadística se comenzó a trabajar con *outliers*. La definición del concepto puede ser muy amplia, pero en realidad se refiere a un dato que es significativamente diferente del resto y que, por tanto, tiene un

comportamiento muy distinto al esperado. También se conocen como anomalías, ruido, valores anómalos, etc.

A lo largo del tiempo se han propuesto muchos métodos para tratar con estos valores, que son una parte muy importante de los datos [9]. En muchos casos, el interés depositado en los valores anómalos se debe a la intención de eliminarlos para que no perjudiquen el comportamiento del modelo de aprendizaje automático creado. Sin embargo, existe la posibilidad de que la información aportada por los *outliers* sea de especial relevancia para diferentes aplicaciones, como pueden ser: detectar el fraude [10], diagnóstico médico de anomalías [11] o detección de anomalías con sensores de redes [12].

En muchas de las aplicaciones mencionadas anteriormente, la detección de los valores anómalos en el momento en que suceden cobra una gran relevancia. Por este motivo, el análisis de observaciones inusuales o *outliers* que tanta importancia tiene en la industria del dato ha llegado al aprendizaje automático online, que se ha explicado anteriormente. De hecho, en esta rama del *machine learning* puede ser incluso más destacable, pues conocer un valor anómalo en tiempo real puede ayudar a prevenir situaciones indeseadas con antelación suficiente [13].

En lo que respecta a los diferentes algoritmos de *machine learning* que se pueden aplicar a la detección de anomalías, las posibilidades son variadas. Cuando el problema se encuentra en un entorno estático, es posible etiquetar aquellos valores que son anómalos durante el procesamiento de datos. Así, los métodos supervisados requieren esas etiquetas que permiten aprender un modelo predictivo para establecer si los nuevos valores son o no anómalos. Otra manera de proceder es con aprendizaje semi-supervisado si se maneja una parte de los datos con etiquetas fiables, como podría ser un conjunto de instancias que se conoce que son valores normales o tener *outliers* identificados. Sin embargo, en la mayoría de ocasiones las etiquetas no están disponibles en lo que respecta a la detección de *outliers.* Menos aún si los datos llegan en tiempo real, pues no tendrán etiqueta alguna. Por ello, para detectar anomalías en problemas estáticos, pero principalmente en online *machine learning,* se suelen dar condiciones que requieren del uso de métodos de aprendizaje no supervisado. A continuación, se exponen numerosas técnicas no supervisadas para la detección de *outliers*.

2.2.1. Técnicas no supervisadas

Los retos a los que se enfrenta la detección de *outliers* en un entorno no supervisado son importantes. Primeramente, la dimensionalidad de los datos puede ser un problema a nivel computacional. Si además de muchos datos se cuenta con un número importante de dimensiones, muchos modelos se vuelven inasumibles computacionalmente. Esto ocurre principalmente en aquellos basados en densidad y en proximidad. Por otro lado, en muchas aplicaciones se requiere que la detección de un valor anómalo vaya seguida de una buena interpretabilidad que lo justifique, algo ausente en la mayoría de algoritmos. Finalmente, no tener acceso a etiquetas dificulta enormemente la evaluación del rendimiento de los modelos utilizados, así como el tuneo de sus hiperparámetros.

Existen muchos métodos propios del aprendizaje no supervisado que han sido utilizados en aras de la detección de los mencionados valores. Entre ellos se encuentran métodos diversos de *clustering*, métodos estadísticos, *K Nearest Neighbours* (KNN), etc. Todos estos métodos tienen en común que no cuentan con una etiqueta para cada dato que indique si el valor es *outlier* o no. Para cumplir con la tarea se basan en realizar hipótesis sobre las relaciones en distancia u otra magnitud entre los datos.

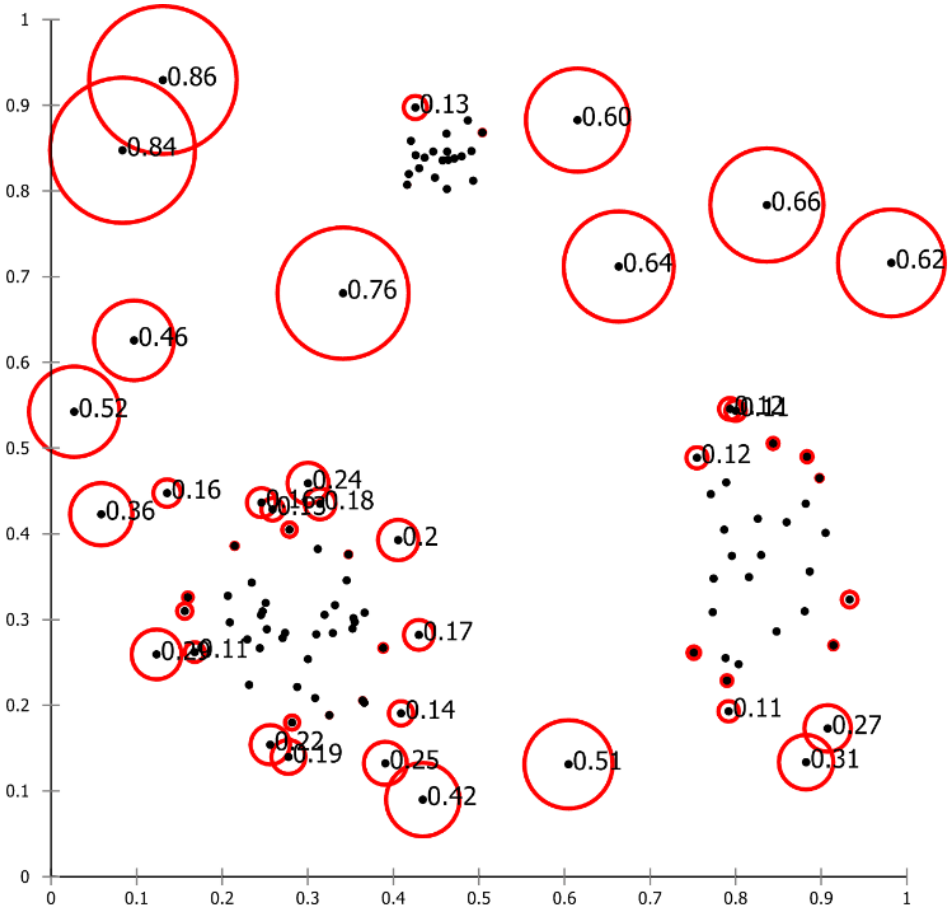
Los métodos de aprendizaje no supervisado utilizados para la detección de anomalías con *online machine learning* en el estado del arte actual se basan en los mencionados anteriormente, pudiéndose clasificar estos en tres grandes grupos:

- Métodos basados en distancias: el algoritmo de aprendizaje no supervisado basado en distancia más extendido es el KNN, por lo que la mayoría de métodos de este grupo se apoyan en él. Para trabajar bajo las condiciones del flujo continuo de datos, se utilizan técnicas para dividir el problema en subconjuntos. Así, se estudia y evalúan las relaciones entre los puntos de ese subconjunto. A ese grupo pertenecen métodos como DUE, Exact-Storm, Abstract-C, Thresh_LEAP, etc. Sin embargo, destaca MCOD por el uso de micro clústeres y HST por utilizar el algoritmo *Isolation Forest* para modelos online.
- Métodos basados en clústeres dinámicos: estos métodos trabajan en dos etapas principales donde en la primera se crean micro clústeres por distancia y en la siguiente etapa se agrupan estos por criterios de densidad. Destacan en este grupo CluStream, DenStream, STREAM y DBSTREAM.

- Métodos basados en densidad: son métodos que se construyen alrededor del *Local Outlier Factor* (LOF). Entre ellos se encuentran iLOF, MiLOF, TADILOF, etc. Al ser LOF la base de todos ellos, se va a comentar la base teórica de este algoritmo con detalle para describir los métodos basados en densidad.

El LOF es un algoritmo que establece para cada dato la probabilidad de que sea un valor anómalo en función a la densidad que existe en sus vecinos. Es un método muy intuitivo cuya representación gráfica en la Figura 2 facilita mucho su comprensión. En ella se muestra la aplicación del método sobre un conjunto de datos, donde se muestra la probabilidad de cada uno de ser outlier con un radio en rojo proporcional a esa probabilidad.

Figura 2. Técnica LOF basada en densidad para detectar anomalías [14]



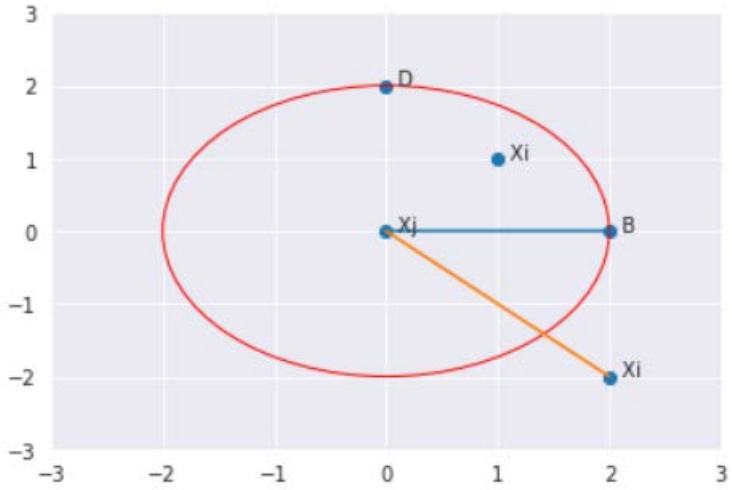
El algoritmo proporciona la probabilidad de que un valor sea outlier como un valor de 0 a 1, siendo un valor cercano a 1 para los *outliers*. Para ello, se va a explicar con detalle la expresión que utiliza. En primer lugar, se debe conocer el concepto de distancia

alcanzable (*Reachability Distance*, RD) de un punto X_i a un punto X_j, que viene dado por la expresión siguiente [15]:

$$RD(X_i, X_j) = \max(K - distance(X_j), distance(X_i, X_j))$$

Así, la expresión de la distancia alcanzable de un punto X_i hasta uno X_j viene dado por el valor máximo entre dos números. El primero de ellos es la distancia K del punto X_j mientras que el segundo es la distancia entre ambos puntos X_i y X_j. La distancia K del punto X_j es aquella que existe entre el punto X_j y su vecino K-ésimo. Esto quiere decir que la RD entre X_i y X_j será la distancia K del punto X_j si X_i está entre sus K vecinos. En caso contrario, la RD entre ellos será la distancia que tienen entre sí. Esa distancia podrá ser euclídea, de Manhattan o de cualquier tipo según se desee en el problema. En la Figura 3, se observa que, si el X_i es el que se encuentra dentro de los K vecinos la RD sería la línea azul mientras que, si se encuentra fuera de ellos, la RD sería la línea naranja.

Figura 3. Representación de la distancia alcanzable RD de un punto X_i hasta uno X_j



A partir de la RD se construye el concepto de densidad alcanzable local (*Local Reachability Density*, LRD) de un punto A, que no es más que el inverso de la media de la RD del punto con sus vecinos. Por tanto, valores bajos indican poca densidad alrededor del punto en cuestión, pues la media de las RD será alta y por tanto el inverso disminuirá. La siguiente expresión es la que se ha explicado, referente a la LRD de un punto A:

$$LRD_k(A) = \frac{1}{\sum_{X_j \in N_k(A)} \frac{RD(A, X_j)}{||N_k(A)||}}$$

Por último, con el LRD se puede obtener el factor de anomalía local (*Local Outlier Factor*, LOF), que no es más que una comparación entre el LRD del punto con la media del LRD de sus K vecinos. Así, la expresión del LOF es la siguiente:

$$LOF_k(A) = \frac{\sum_{X_j \in N_k(A)} LRD_k(X_j)}{||N_k(A)||} \cdot \frac{1}{LRD_k(A)}$$

Si la media del LRD de sus vecinos es similar al LRD del punto, se trata de un valor normal. Por otro lado, si el LRD del punto es muy bajo en comparación con el de la media de sus vecinos, el punto tiene mucha menor densidad alrededor que sus vecinos y se tratará de un outlier. De esto se concluye que cuando el LOF es cercano a 1, la densidad del punto es similar a sus vecinos y no es un outlier mientras que cuando el LOF es bastante mayor que 1 se trata de un outlier.

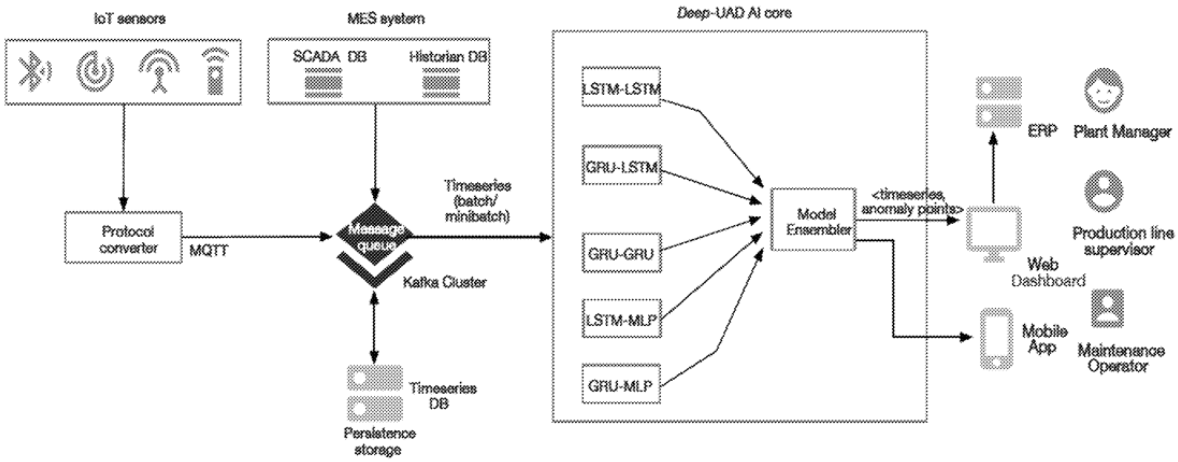
Una vez explicado el algoritmo, los hiperparámetros más importantes de este son bastante claros: el número de vecinos (K) para el cual calcular todas las distancias mencionadas anteriormente y la manera de medir la distancia (euclídea, manhattan, etc.).

Como se ha mencionado, este método tiene un buen comportamiento cuando la densidad es muy diferente a lo largo del *dataset*, detectando en él *outliers* locales. También presenta la ventaja de no asumir ninguna distribución en los datos, mientras que otras herramientas asumen una distribución normal. Además, la probabilidad obtenida para cada dato es adimensional y comparable con la del resto. Sin embargo, los algoritmos basados en densidad como el LOF, al igual que muchos de los mencionados en este apartado, no son adecuados para la detección de anomalías en series temporales, como son los datos telemétricos. Esto es porque la densidad de los datos no es lo suficientemente diferente a lo largo del *dataset*. De hecho, el algoritmo LOF se aplica sobre series temporales para una detección temprana de anomalías o *Novelty Detection*, que se explicará en el apartado 2.3.1 de este documento como herramienta de detección de *outliers* en series temporales.

Por otro lado, este como el resto de algoritmos de detección de anomalías no supervisada tiene la desventaja de no tener un umbral específico en la puntuación que marque la diferencia entre *outlier* e *inlier*. Por ello, se requerirá de un parámetro que refleje el porcentaje de *outliers* en el *dataset* en aras de establecer ese umbral.

Además, existen modelos más complejos que LOF o cualquiera de los anteriores porque combinan otras herramientas del *machine learning* para crear un modelo de aprendizaje no supervisado que los engloba. Es el caso de la patente de Bhardwaj et al. publicada en 2021 [16], donde se creó un método para detectar anomalías mediante aprendizaje no supervisado y *deep learning*. Se toman datos con unos sensores de IoT para procesarlos e introducirlos en el núcleo de DL que contiene 5 *Generative Adversarial Networks* (GANs). Cada una de ellas infiere una puntuación de anomalía y si 3 de las 5 mínimo consideran que el valor es anómalo, el resultado final se toma como tal. Las GANs son diferentes entre sí y cuentan con *multilayer perceptron* (MLP), *long-short-term memory* (LSTM) o *Gated recurrent unit* (GRU). Este modelo ya se ha implementado y usado en la industria, como por ejemplo en una compañía de utilidad de agua en Singapur. En la Figura 4 se muestra un esquema que resume su funcionamiento.

Figura 4. Esquema del modelo de aprendizaje no supervisado para detectar anomalías de Bhardwaj et al. [16]



Otro ejemplo de técnica más compleja para la detección de *outliers* en *data streams* es *Mondrian Pólya Forests* [17]. Este modelo cuenta con procesos *Mondrian*, que consisten en dividir de manera jerárquica y binaria del dominio. Esto se combinar con árboles

Pólya para estimar una función de densidad sobre cada partición y así poder decidir cómo distribuir la masa de datos en el espacio.

Una vez repasados muchos de los algoritmos y técnicas básicas para la detección no supervisada de *outliers*, se expondrán algunas de las librerías existentes en Python para su implementación.

2.2.2. Librerías de detección de outliers

En primer lugar, existe una librería muy conocida en el mundo del ML con Python que se llama scikit-learn. Esta librería cuenta con numerosas funcionalidades para el tratamiento de datos y la creación de modelos de ML de clasificación, regresión y demás. Entre sus posibilidades también cuenta con algoritmos de detección de anomalías.

Existen otras librerías para la tarea de detección de *outliers*, como PyCaret [18]. Es una librería de código abierto en Python que automatiza los procesos propios del ML, reduciendo y simplificando el código necesario para implementarlos. Sus modelos de detección de anomalías son tomados de otra librería más específica de esta tarea, llamada PyOD [19], que quiere decir *Python Outlier Detection*. Al ser una herramienta especializada y dedicada únicamente a detectar *outliers*, cuenta con una enorme variedad de modelos para utilizar. Es la más utilizada para implementar modelos de detección de anomalías en Python.

Para la elección de una librería y concretamente ciertos modelos entre todos los que contenga, se debe tener en cuenta que la detección de anomalías en series temporales no es igual que en imágenes, en textos o en datos multidimensionales sin temporalidad. Es por ello que debe conocerse qué son los datos telemétricos y cuáles son las mejores opciones para detectar anomalías sobre este tipo de *datasets*. Se procede a introducir los datos telemétricos.

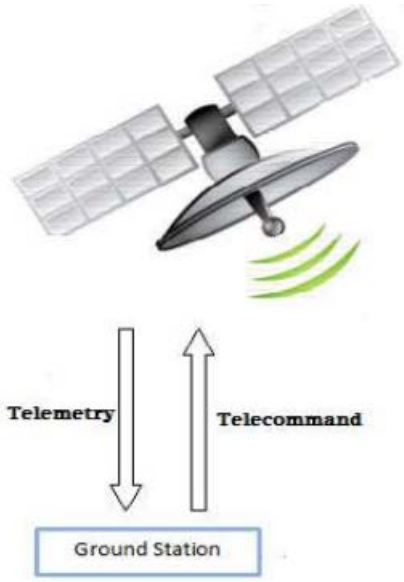
2.3. Datos telemétricos

La detección de *outliers* en el campo específico de los datos telemétricos cuenta con variaciones que permiten que los métodos utilizados se adapten mejor a sus

necesidades. Por ello, se va a introducir el origen de estos datos para conocer posteriormente técnicas específicas de detección de anomalías sobre ellos.

Los datos telemétricos son aquellos provenientes de un satélite y que se envían a la estación en tierra para ser analizados y enviar ciertas instrucciones al satélite en el caso de ser necesario. Este proceso se ilustra en la Figura 5. En este ámbito, algunas veces se da la circunstancia de tener ciertas anomalías etiquetadas y el modelo puede construirse con aprendizaje semi-supervisado. Sin embargo, suele ser más habitual e implica menos trabajo previo tener los datos sin etiquetas y utilizar métodos de aprendizaje no supervisado.

Figura 5. Flujo de datos entre un satélite y la estación en tierra [20]



El motivo por el cual la detección de anomalías sobre los datos telemétricos de un satélite puede ser importante no es otro que el hecho de conocer si algún equipo del satélite va a fallar o si ocurre algún hecho inesperado en el entorno espacial. Si se da esta circunstancia, cuál será el fallo operativo o hecho inesperado, cómo afectará a la misión y cuándo tendrá lugar el hecho será información importante a conocer [21]. También puede ocurrir que la anomalía detectada sea una falsa alarma, que llevará tiempo en estudiarla y será tedioso e innecesario todo el análisis sobre ella. Por ello, es muy importante que las anomalías detectadas casi nunca sean falsas alarmas o el proceso carecerá de sentido práctico. Con la cantidad de datos telemétricos que maneja un satélite en la actualidad, el porcentaje de falsas alarmas debe ser menor al 1% para

que sea viable la detección de valores anómalos. Un reto importante en el campo de la detección de *outliers* sobre datos telemétricos es que el motivo de la anomalía sea claro y manifiesto. Esto evitará que se produzca una falta de confianza en los resultados.

Los datos telemétricos provienen de los sensores de un satélite y cada dato tiene asociado un instante temporal. Por ello, con los datos telemétricos que son tomados cada cierto intervalo de tiempo por el satélite, se construye una serie temporal. Como ya se comentó, muchos de los algoritmos explicados para la detección de anomalías sobre cualquier tipo de *dataset* no son los más adecuados en el caso de tratarse de una serie temporal. Además, el número de falsas alarmas debe ser muy reducido como se ha explicado anteriormente. De esta forma, los métodos para detectar anomalías sobre datos telemétricos deben adecuarse bien a series temporales y no obtener demasiados falsos positivos. En el siguiente apartado se desarrollan varios de los métodos que cumplen con estas características y se utilizan para la detección de anomalías no supervisada sobre datos telemétricos.

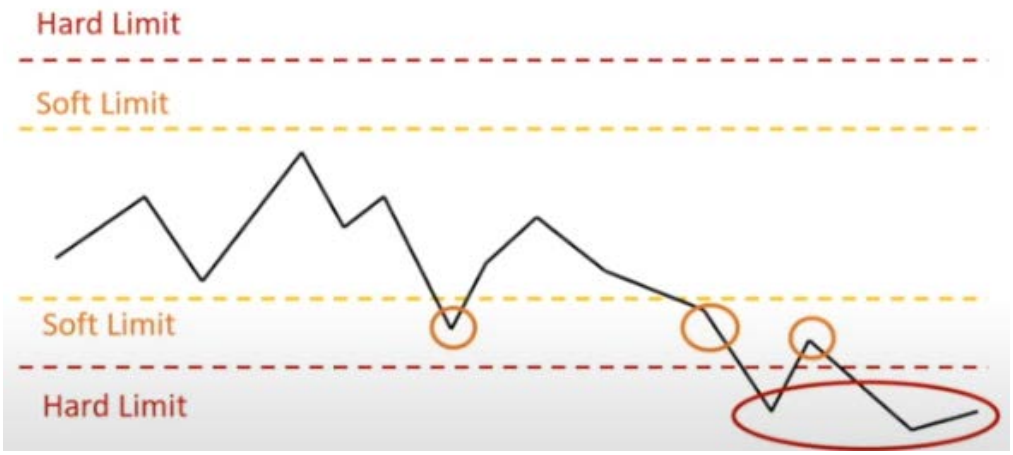
2.3.1. Detección de anomalías con datos telemétricos

Existen métodos usados específicamente en la detección de anomalías para datos telemétricos que no se han nombrado a lo largo de la clasificación más general, como puede ser el enfoque clásico *Out-of-Limits* (OOL). Esta es una solución sencilla que no puede considerarse ni siquiera un algoritmo. Su razonamiento consta simplemente de definir un margen de límites a partir de los cuales se envía una alarma (*soft limits*) y otros límites más estrictos (*hard limits*) que no deberían alcanzarse y pueden poner en riesgo al satélite. Para hacer la solución más inteligente, se puede hacer que estos límites sean dinámicos según ciertas condiciones y la evolución de los datos telemétricos.

Es un modelo basado en probabilidad, estableciendo esos rangos en función de la experiencia que se tiene sobre la magnitud en concreto o sobre su repercusión en el rendimiento de los equipos. Por tanto, es muy adecuado cuando se conocen ciertos rangos entre los que se debe encontrar una magnitud o que no se deben sobrepasar para no comprometer la misión o el conocimiento que se tiene sobre el entorno espacial. Por ejemplo, la temperatura de los equipos electrónicos siempre debe mantenerse en un rango determinado. El *soft limit* puede establecerse con un margen de seguridad respecto al valor que no se puede superar, que sería el *hard limit*. La Figura 6 ilustra

este método, donde se observan los datos y los márgenes que marcan los límites comentados.

Figura 6. Método Out-of-Limits clásico para detectar *outliers* en misiones espaciales [21]



Una versión mejorada de este método es el *Novelty Detection* (detección de la novedad). Esta herramienta permite detectar comportamientos anómalos que ocurren dentro de los límites, pues esto también es posible y puede anticipar una anomalía mayor que se salga de los límites [22]. Es complementario al paradigma anterior de OOL y obtiene muy raramente falsas alarmas. El método ha sido utilizado y validado en diversas misiones como Venus Express, Cryosat 2 o XMM. Se basa en densidad y ha sido patentado por la ESA. De hecho, un algoritmo que se utiliza para *Novelty Detection* es LOF, explicado con gran detalle en el apartado 2.2.1 y cuyo comportamiento se vio representado en la Figura 2.

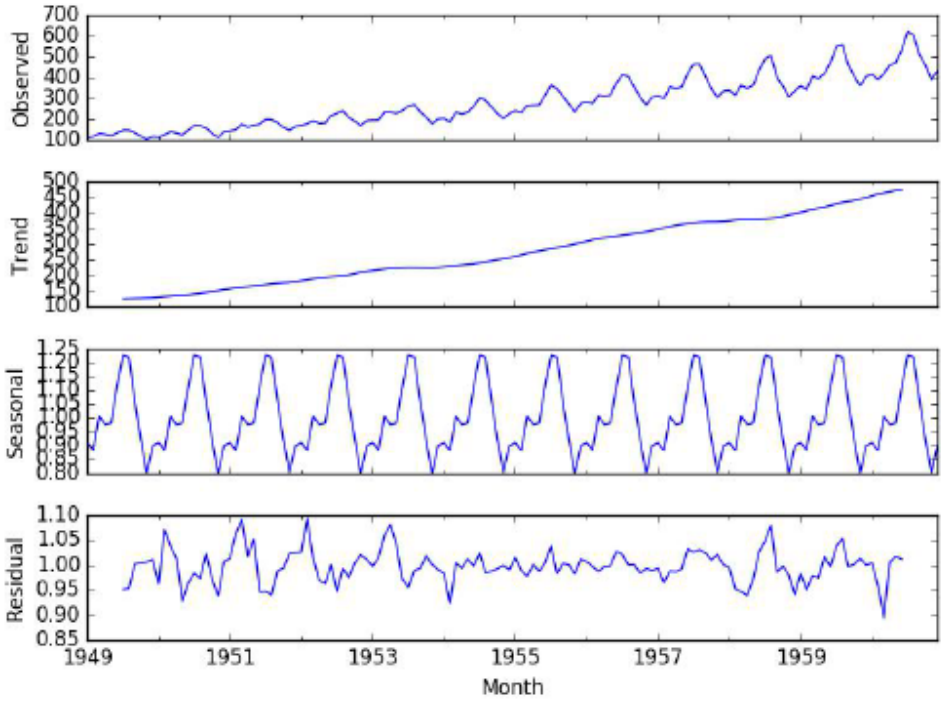
Existen otros métodos aplicables a la detección de anomalías sobre datos telemétricos de misiones espaciales aparte de los métodos no supervisados mencionados para cualquier ámbito y de los recientemente mencionados OOL y *Novelty Detection* para el campo de datos provenientes de satélites. Existe una aplicación llamada WebTCAD desarrollada por el *Laboratory for Atmospheric and Space Physics* (LASP) de la Universidad de Colorado que permite interactuar con el dato en tiempo real y detectar anomalías entre otras tareas [23].

Los datos telemétricos en tiempo real son series temporales, donde la evolución de los datos cuenta con ciertas características como son la tendencia (siempre que no sea una

serie estacionaria), la periodicidad, la estacionalidad y otros elementos propios de su evolución temporal. Una manera común de trabajar sobre series temporales para detectar *outliers* de manera no supervisada divide el proceso en dos etapas. En la primera de ellas, se trata de determinar cuál es el comportamiento estándar dentro de esa serie temporal que tiene tendencias, periodicidades y demás patrones. Para ello, existen diferentes técnicas que se basan en convertir en estacionaria la serie temporal. Una vez realizada esta primera etapa, se trata de determinar qué datos difieren del comportamiento normal del modelo y por tanto pueden clasificarse como anómalos.

Dentro de los métodos que se pueden utilizar para cumplir con la primera etapa, se puede destacar a ARIMA o SARIMA, *AutoEncoders* y métodos estadísticos como *rolling mean*. En primer lugar, ARIMA o SARIMA son algoritmos que permiten crear un modelo matemático para separar las características mencionadas en una serie temporal en tendencia, estacionalidad, ruido blanco, etc. La diferencia entre ARIMA y SARIMA es que este último incluye la estacionalidad. En la Figura 7 se muestra la aplicación de ARIMA sobre una serie temporal.

Figura 7. Descomposición de una serie temporal en diferentes características con ARIMA



Por otro lado, los *AutoEncoders* son modelos que permiten comprimir las características en un vector latente de dimensión mucho más reducida a través del *encoder* para que

luego el *decoder* vuelva a expandir las dimensiones para aproximarse a los datos iniciales. Se pueden aplicar para muchas otras cuestiones, pero en el caso de la detección de *outliers* en series temporales se comportan muy bien aunque son una caja negra, mucho menos explicables que los provenientes de la familia ARMA como ARIMA y SARIMA.

Por último, *rolling mean* es un método estadístico que divide el conjunto de datos en ventanas de N elementos. Esas ventanas no son fijas, sino que la siguiente ventana elimina al último elemento para coger uno nuevo colocado en el primer lugar. En cada una de las ventanas se calcula la media, construyendo así la *rolling mean*. Es, con diferencia, el método más rápido computacionalmente de los mencionados. Sin embargo, no suele ser el que mejores resultados otorga por lo que es adecuado para un primer análisis o si se requiere gran rapidez en el proceso.

Una vez realizado el primer paso, se tiene una serie estacionaria. Lo siguiente es definir los valores anómalos como aquellos que difieren considerablemente del comportamiento normal. La clave en la anterior frase es el término “considerablemente”, que vendrá determinado por un umbral. El umbral podrá ser difuso si se asigna a cada dato una puntuación de *outlier*, de manera similar a lo que se describió para el método LOF. Sin embargo, también puede ser un umbral claramente definido que si se supera el valor se asume como anómalo (etiqueta binaria de *outlier* o no). Sea el caso que sea, la colocación del umbral influirá en que se detecten muchas falsas anomalías o se escapen bastantes de ellas. La decisión a tomar en este *trade-off* dependerá del problema donde se aplique, por lo que es importante tener un alto conocimiento del campo y los objetivos para establecer adecuadamente el umbral.

Existen varias formas de comparar un valor con datos de la serie para clasificarlo como *outlier* o no (o darle una puntuación). Si se quiere asignar una etiqueta binaria de anomalía, se presentan varias opciones:

- La primera se basa en un modelo de predicción que pretende recrear uno original. Para clasificar un valor como *outlier*, compara la desviación estándar del dato predicho con la de N veces la desviación del dato de la serie real.
- Otro método es comparar la desviación estándar de la puntuación de anomalía de un dato con N veces la desviación estándar de la puntuación de anomalía de

la serie. Para este último método, se ha debido anteriormente utilizar otro que permita dar una puntuación de *outlier* a cada dato.

- Por último, existe un método propuesto por el *Jet Propulsion Laboratory* (JPL) de la NASA que obtiene etiquetas binarias dada la serie temporal y un modelo predictivo. Su nombre es *nonparametric dynamic thresholding* [24].

En cuanto a los algoritmos que existen para, una vez la serie es estacionaria, detectar outliers en la serie temporal, se van a destacar tres de ellos entre una gran cantidad de opciones. Dos de ellos son muy conocidos y utilizados para esta tarea, que son *Isolation Forest* (IForest) y *Variational AutoEncoder* (VAE). El último se denomina *Empirical-Cumulative Outlier Detection* (ECOD) y es muy reciente en el estado del arte, pues fue publicado en 2022.

Isolation Forest (IForest)

Este algoritmo publicado a finales de 2008 trabaja por conjuntos, tratando de aislar aquellos valores que son anomalías [25]. En su momento fue muy novedoso y tuvo gran impacto por cambiar el enfoque con el que se trabajaba. Hasta entonces, los algoritmos pretendían conocer el comportamiento normal de los datos para luego descubrir los *outliers*. Sin embargo, este método trata directamente de agrupar y aislar las anomalías, en lugar de centrarse en los datos normales. Para ello, se basa en lo que distingue a los *outliers*: que son pocos y diferentes del resto de los datos.

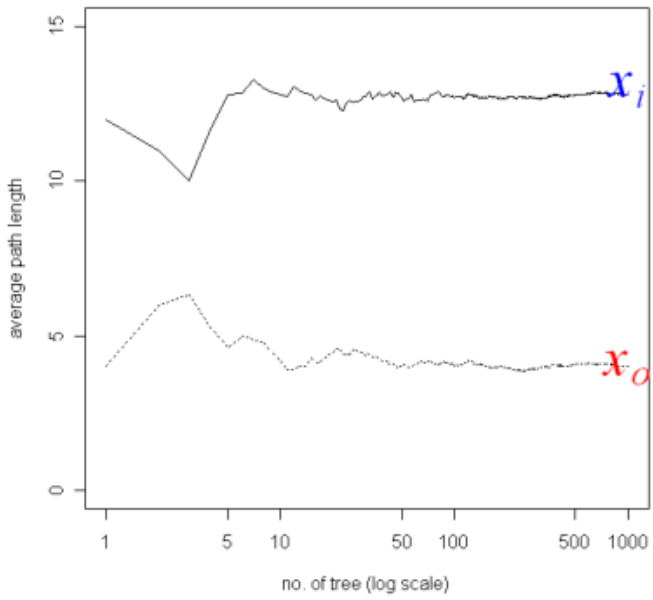
La forma de aislar o agrupar los datos será a través de árboles, de ahí el nombre del algoritmo. Estos árboles tenderán a tener las anomalías cerca de la raíz, puesto que presentan mayor facilidad para aislarse, mientras que los datos normales se encontrarán en las profundidades de cada árbol. Por tanto, el método construye un conjunto de árboles (bosque) que contarán con *outliers* cerca de sus raíces y valores normales conforme se recorren las profundidades de ellos.

Existen una serie de características que diferencian IForest de la mayoría del resto de modelos. En primer lugar, es capaz de construir modelos parciales para detectar anomalías. Esto es porque no se requiere construir un árbol completo para conocer los *outliers*, ya que estos se encontrarán en la raíz y no será necesario profundizar en el árbol para alcanzarlos. Por otro lado, el algoritmo no utiliza distancia ni densidad para hallar las anomalías, lo que reduce el coste computacional y elimina la elección de la

medida de distancia. Esta última característica, junto a otras, provoca que la complejidad sea lineal con una constante baja y sin elevados requerimientos de memoria. Por último, es un algoritmo que escala bien ante *datasets* de gran tamaño y muchas dimensiones, aunque varias de estas sean irrelevantes.

Para aislar un valor del resto de instancias, el algoritmo realiza particiones en árboles aleatorios de manera recursiva. Para aislar *outliers*, se requieren menos particiones pues tienden a separarse mejor en las primeras particiones además de estar menos rodeados, dejando así menos opciones para la próxima división. Por ello, se requerirán menos particiones para aislar valores anómalos (alrededor de 4) que para hacerlo en valores normales (unas 12 en una distribución gaussiana). Además, cada iteración termina cuando se aíslan todos los valores del *dataset*. De esta forma, el algoritmo cuenta con un hiperparámetro claro: el número de iteraciones que se lleven a cabo, terminando cada una al aislar todas las muestras. Conforme aumenta el número de iteraciones (árboles realizados), la media de las particiones necesarias para aislar un valor converge. En la Figura 8 se puede observar la evolución de la media de particiones aleatorias que requieren dos valores, x_o y x_i, para ser aislados en función de número de árboles o iteraciones realizadas. En este caso, el valor x_o es un *outlier* mientras que x_i es un valor normal.

Figura 8. Media de particiones de dos valores según el número de árboles en IForest



Viendo las particiones bajo una estructura de árbol, el número de particiones para aislar un valor es la longitud del recorrido desde la raíz hasta el propio valor. Un árbol estará completo en el momento en que todos los datos estén aislados. Por tanto, si existen n datos serán necesarias $n - 1$ divisiones y la complejidad será lineal de orden $O(n)$. La puntuación de anomalía de cada dato vendrá determinada por la media de divisiones necesarias para aislarlo entre todos los árboles realizados. Conforme crezca está media, la puntuación de anomalía del dato disminuye.

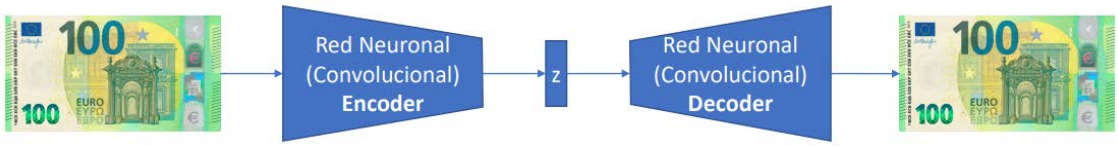
Como conclusión, es un algoritmo que tiene muy pocos hiperparámetros, donde el principal es el número de árboles o iteraciones a realizar. Requiere un parámetro que especifique el porcentaje de anomalías que existen en el *dataset* en caso de usarse sin etiquetas en un entorno no supervisado. Este valor servirá para elegir el umbral que distinga *outliers* de valores normales. Además, la complejidad es lineal por lo que el coste computacional permite aplicar el método a grandes *datasets* con numerosas dimensiones. De esta forma, se espera que su rendimiento sea bueno sobre una serie temporal de datos telemétricos, a pesar de ser un algoritmo de 2008.

Variational AutoEncoder (VAE)

El *Variational AutoEncoder* o VAE [26] es una modificación del tipo de entrenamiento que se realiza sobre el *AutoEncoder* básico. El objetivo de la modificación o mejora es obtener un espacio latente de probabilidades que permita reducir el coste computacional en las iteraciones y generar muestras nunca vistas a partir del espacio latente. Para entender su estructura y funcionamiento, conviene primero conocer el *AutoEncoder* básico.

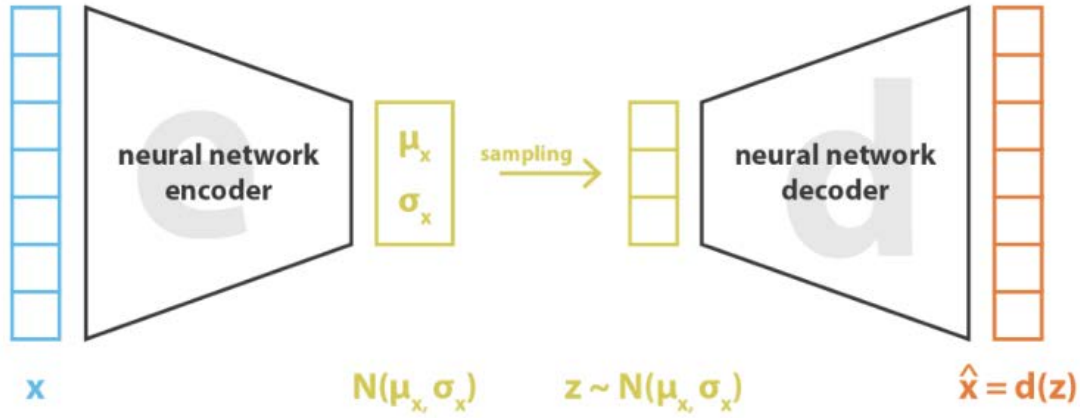
Un *AutoEncoder* es una estructura que acopla dos módulos: un *encoder* y un *decoder*. Entre ellos se ubica el espacio latente. El *encoder* es una red neuronal que recibe unos datos de entrada y obtiene una representación comprimida en muchas menos dimensiones de ellos, que es el espacio latente z. A partir de ese z, el decoder reconstruye los datos originales con la mayor precisión, basándose también en una red neuronal. El entrenamiento del *AutoEncoder* se basa en minimizar los errores de reconstrucción, por medio de una función de pérdidas, entre los datos originales y los que obtiene el *decoder* a partir del espacio latente z. La estructura de esta solución de aprendizaje no supervisado puede observarse en la Figura 9.

Figura 9. Estructura de un AutoEncoder básico



El VAE cuenta con la misma estructura de *encoder*, espacio latente y *decoder* pero su espacio latente no se trata de un vector en el espacio latente, sino de una distribución de probabilidad generalmente gaussiana. La distribución es de esta simplicidad porque se asume que las relaciones entre las variables son más sencillas en el espacio latente comprimido. De esta manera, los valores obtenidos por el *encoder* no son los de un vector, sino la media y la desviación estándar de una distribución de probabilidad gaussiana. El *decoder* tomará un valor aleatorio de esa distribución de probabilidad y reconstruirá los datos de entrada. Así, su estructura es la que se observa en la Figura 10.

Figura 10. Estructura de un VAE [27]



Se puede usar tanto un *AutoEncoder* básico como un VAE en aras de detectar anomalías. Para esta tarea, se basan en el error de reconstrucción como puntuación de *outlier*. Así, si el error de reconstrucción es alto quiere decir que el dato es un valor anómalo. Al igual que el IForest, requiere un parámetro que especifique el porcentaje de anomalías que existen en los datos al tratarse de un entorno no supervisado, para así poder establecer el umbral que distinga *outliers* e *inliers*.

Los hiperparámetros que requiere el modelo son numerosos, pues se va a entrenar una estructura con dos redes neuronales. Los primeros son los parámetros que definen la estructura del *encoder* y el *decoder*, tanto en capas como en neuronas en cada una de ellas. Además, se puede especificar el número de dimensiones del espacio latente, así como las funciones de activación de las capas ocultas y de la capa de salida. También existen otros parámetros propios del entrenamiento de la red, como la función de pérdidas, número de épocas, tamaño del *batch*, optimizador, *dropout*, etc. En definitiva, el entrenamiento de este modelo será costoso a nivel computacional y es importante la elección de sus hiperparámetros, aunque existen muchos estudios donde se presentan las opciones más destacadas que se han encontrado. Aun siendo su entrenamiento costoso, su comportamiento en series temporales es bueno por lo que es una buena opción para detectar anomalías de manera no supervisada en este tipo de *datasets*.

Empirical-Cumulative Outlier Detection (ECOD)

ECOD es un algoritmo de detección de anomalías no supervisado, interpretable y sin parámetros que ha sido publicado muy recientemente, en agosto de 2022 [28]. Mientras que la mayoría de algoritmos requieren tuneo de hiperparámetros, este método no lo necesita, lo cual es una gran ventaja cuando se trata de un entorno no supervisado. El algoritmo se basa en la propia definición de anomalía, aquel valor extraño que no sigue el comportamiento normal. Así, se basa en la estadística tal y como lo hace el método ya descrito OOL. Sin embargo, no se basa simplemente en la media y desviación estándar o en los cuartiles como podría hacerse con OOL. Se pretende disponer de una mayor información que la que aportan los dos parámetros mencionados, para lo que se utiliza la función de distribución acumulada empírica (*Empirical Cumulative Distribution Function*, ECDF).

Una dificultad importante al trabajar con ECDF ocurre en caso de una gran dimensionalidad de los datos. Conforme crece el número de dimensiones, el algoritmo converge más lento y puede llegar a resultar inviable su utilización. Por ello, en ECOD se propone trabajar con cada dimensión de los datos de manera separada e independiente, combinándolas luego para hallar la probabilidad conjunta. Es necesario asumir que las variables son independientes entre sí para multiplicar sus probabilidades y obtener la conjunta. Esa asunción es bastante restrictiva y apenas se cumple en la realidad. Pese a ello, el comportamiento del algoritmo en la práctica es bastante bueno.

Las ventajas principales que presenta el algoritmo son su efectividad, eficiencia e interpretabilidad. Es efectivo por obtener mejores resultados que los 11 algoritmos más conocidos y utilizados para la detección de anomalías. Además, su eficiencia se basa en que su orden de complejidad es $O(nd)$, donde n es el número de datos y d el número de dimensiones. Por último, es interpretable en cuanto a conocer la contribución de cada dimensión sobre la puntuación de *outlier* de un dato.

La idea en la que se basa el modelo es que un valor será *outlier* cuando se encuentre en alguna zona de baja densidad en la distribución de probabilidad. La función de distribución acumulada (*Cumulative Distribution Function*, CDF) mide la probabilidad para cada dato de que un punto generado con su misma distribución sea menor al dato. Así, si esta función en el dato es muy baja, quiere decir que será muy difícil que bajo la misma distribución de probabilidad se genere un dato menor. Por tanto, el dato corresponde a un valor que se presta a ser outlier. Todo esto sirve estudiar la cola izquierda de la distribución de probabilidad, que es aquella zona con poca probabilidad que queda a la izquierda de la distribución. Para estudiar la cola derecha (mismo concepto, pero en el lado derecho de la distribución), se utilizará el complementario de la función CDF mencionada. Esta explicación se puede ver reflejada en las expresiones que siguen a continuación:

Cola izquierda: $F_j(z) = P(X_j \leq z)$

Cola derecha: $1 - F_j(z) = P(X_j > z)$

En ellas, j es la dimensión concreta del dato, z es el dato y X_j es un nuevo dato generado en esa misma dimensión j. Como ya se mencionó, para obtener la CDF en todas las dimensiones se asume que las dimensiones o atributos de los datos son independientes entre sí. Por ello, la función para todas las dimensiones se denomina *Empirical* CDF (ECDF), obteniéndose de la siguiente forma gracias a la hipótesis de independencia:

$$F(x) = \prod_{j=1}^{d} F_j(x_j),$$

Con todo lo anterior, el método ECOD se compone de dos pasos principales: el primero en el que se calcula la cola derecha e izquierda para cada dimensión. El segundo paso es agregar la cola izquierda y cola derecha para cada punto, obteniéndose su puntuación de outlier. Esta puntuación de outlier, como en todos los métodos de este documento, será más alta para aquellos valores propensos a ser anomalías. En este algoritmo, la puntuación va desde 0 hasta infinito, sin estar acotada en un rango. Para establecer el umbral de decisión, el algoritmo requiere un parámetro que le indique el porcentaje de *outliers* que existen en el *dataset*. Este parámetro estará presente en la mayoría de algoritmos de detección de anomalías en entornos no supervisados.

Como conclusión, es un algoritmo muy adecuado para su uso en series temporales siempre que no tengan demasiadas dimensiones. Si hubiese demasiadas dimensiones el coste computacional se dispararía, pero no es lo más común en series temporales, que suelen ser unidimensionales o de pocas variables.

Además de todos los algoritmos mencionados en este apartado, existen algoritmos o métodos conocidos con *ensemble*. Este tipo de algoritmos son diseñados para combinar las predicciones o detecciones de varios algoritmos y obtener una única predicción o detección. Estos métodos *ensemble* tienen la capacidad de generalizar mejor la solución de un problema y añadir robustez a un algoritmo, pues detrás de él existirán varios en los que se base. Un ejemplo de este tipo de métodos es el algoritmo SUOD [29], publicado en marzo de 2021 como una solución para la detección de anomalías no supervisada con algoritmos heterogéneos.

Después de repasar muchos de los algoritmos presentes en el estado del arte y aquellos específicos que funcionan mejor con series temporales y datos telemétricos, se van a detallar diferentes opciones para obtener *datasets* públicos y de manera gratuita en aras de aplicar esos algoritmos y comparar su funcionamiento.

2.3.2. Datasets

En cuanto a los *datasets* públicos existentes de datos telemétricos para poder utilizar, se han explorado varias alternativas:

- *Laboratory for Atmospheric and Space Physics* (Universidad de Colorado): laboratorio de investigación en la Universidad de Colorado que genera y posee

diversos *datasets* sobre la atmósfera y el espacio. Llevan almacenando datos desde hace más de 40 años, por lo que muchos de sus *datasets* cubren un importante abanico de tiempo [30]. Es bastante sencillo acceder a los datos y visualizar gráficas de ellos sin tener que descargarlos. Además, la descarga de todos ellos puede realizarse en formato csv o puede accederse a ellos a través de los comandos **wget** o **curl**. De este proveedor existen varios *datasets* interesantes con los que puede cumplirse el objetivo expuesto en este documento.

- *International Space Station* (ISS, *live data*): existe una página web [31] donde se encuentran datos en tiempo real sobre la atmósfera, la regulación térmica, el agua, control de altitud, datos fotovoltaicos y otros datos sobre el planeta, tomados desde la ISS. Se podrían utilizar técnicas de web *scrapping* para construir una serie temporal sobre la que aplicar detección de anomalías. Sin embargo, no se elige este proceso por su necesidad de aplicar durante un periodo largo de tiempo (del que no se dispone) y su aparente complejidad.
- *Planetary Data System* (NASA): amplio archivo de datos sobre misiones planetarias de la NASA, experimentos en laboratorios e incluso misiones de otras entidades. Todas sus bases de datos están accesibles para todo el público en general, incluido investigadores, estudiantes, etc. Se puede realizar una búsqueda por palabras clave con operadores booleanos, comillas para buscar palabras juntas y otras opciones [32]. Sin embargo, no es sencillo visualizar muchos *datasets* en poco tiempo para elegir y tampoco es intuitivo el proceso de descarga. Por estos motivos, se descartó su uso.
- *Space Physics Data Facility* (NASA): archivo de la NASA con todos sus datos que no son sobre la heliofísica solar [33]. Permite acceso a *datasets* de multitud de misiones actualizados regularmente. Además, el acceso a los datos puede realizarse con multitud de herramientas. Destaca entre ellas CDAWeb, un servicio web que permite acceder a los datos de muy diversas formas. Por ejemplo, puede hacerse con REST Web Services, librerías de Python como **ai.cdas** o **cdasws**, Space Physics Environment Data Analysis Software (**SPEEDAS**) o su librería en Python pySPEEDAS, *Heliophysics Data Application Programmer's Interface* (**HAPI**), etc. De todos ellos, HAPI es una forma muy cómoda para extraer datos y pintar gráficos sobre ellos en tan solo unas pocas líneas de código, con acceso a infinidad de datasets.

Dataset elegido

Entre estas opciones, se elige una de ellas para acceder a un *dataset* adecuado para detectar anomalías sobre él, utilizando diferentes algoritmos descritos en este capítulo y comparando su rendimiento.

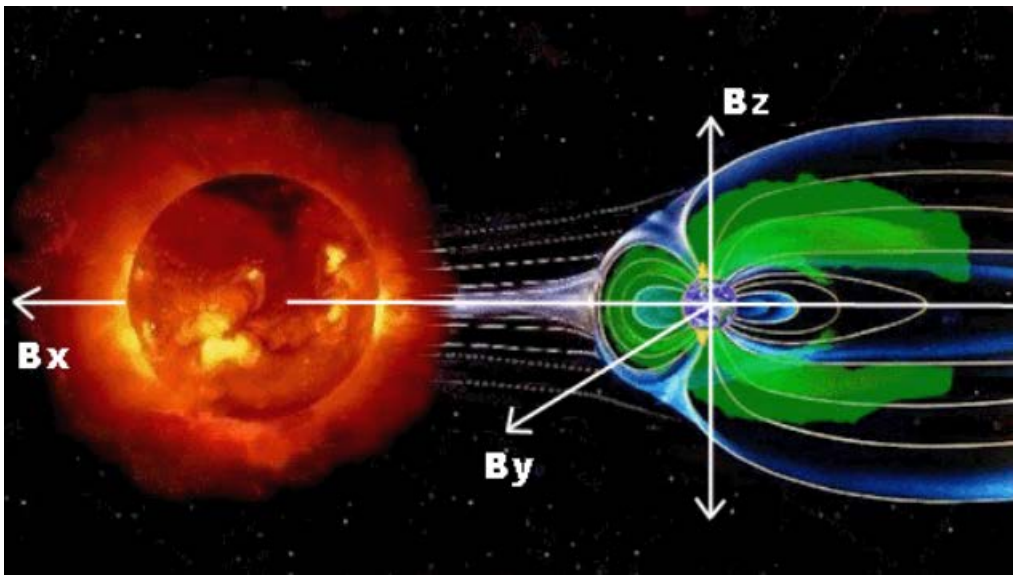
El *dataset* elegido pertenece a la misión *Advanced Composition Explorer* (ACE) [34]. Se trata de una misión de exploración dirigida por el departamento de *Space Science Mission* de la NASA y puesta en órbita con el lanzador McDonnell-Douglas Delta II 7920 en agosto de 1997, desde el centro espacial Kennedy en Florida.

El objetivo de la misión es estudiar las partículas que pueden llegar a la Tierra o se encuentran en el espacio y provienen del Sol u otras fuentes interestelares. Este tipo de estudios contribuyen al entendimiento de los procesos de formación y evolución del Sistema Solar. El satélite orbita alrededor del punto de equilibrio gravitacional L1, por lo que cuenta con un buen acceso al campo magnético interplanetario (*Interplanetary Magnetic Field*, IMF), vientos solares y partículas aceleradas por la heliosfera. Así, sus datos en tiempo real permiten conocer, con aproximadamente una hora de antelación a que sus efectos lleguen a la Tierra y alrededores, tormentas geomagnéticas que pueden presentar una amenaza para las comunicaciones en el planeta o para otras misiones espaciales. Con el propulsante diseñado, se espera que el satélite orbite hasta 2024. Actualmente, sigue dando datos en tiempo real sobre diferentes parámetros.

Uno de los parámetros que los instrumentos del satélite proporcionan en tiempo real es el campo magnético interplanetario, que consiste en aquel campo magnético transportado por el viento solar [35]. El viento solar no es un gas sino un plasma, por lo que es un buen conductor eléctrico y transporta este campo. Para analizarlo correctamente, el satélite se sitúa a unos 1.5 millones de kilómetros de la Tierra, quedando así apenas afectado por su campo magnético. Además, la órbita alrededor del punto L1 le permite a ACE mantener una posición relativa constante respecto al planeta. De hecho, en este punto ha sido la principal fuente de datos en tiempo real sobre el IMF hasta 2016, donde la misión *Deep Space Climate Observatory* DSCOVR empezó la fase operacional. Desde ese momento, ACE se usa como apoyo principal a la labor de DSCOVR.

El IMF es un vector con tres componentes ($B_x, B_y\ y\ B_z$). Dos de esos componentes ($B_x\ y\ \ B_y$) son paralelos al plano de la eclíptica (plano de la órbita de la Tierra alrededor del Sol). El último componente, B_z, es perpendicular al plano mencionado y es el componente más importante en cuanto a la afectación a la magnetosfera de la Tierra. Estas componentes se pueden observar representadas en la Figura 11.

Figura 11. Componentes del campo magnético interplanetario



Sin embargo, el *dataset* que se va a analizar primeramente contiene un solo parámetro, que no es otro que la magnitud o módulo de ese vector. Este valor indica la fuerza total del campo, sin importar la dirección en que se dirija. Las condiciones geomagnéticas son más favorables cuanto mayor es este valor. La unidad en que se mide esta magnitud es el nano Tesla (nT). Los valores adecuados en los que suele moverse están entre 1 nT y 20 nT, siendo a partir de 10 nT un IMF moderadamente fuerte, fuerte si supera los 20 nT y muy fuerte en caso de sobrepasar los 30 nT. Es importante detectar las anomalías en este valor porque pueden indicar un comportamiento del campo magnético interplanetario no esperado, que afecte como se ha mencionado a las comunicaciones u otras misiones espaciales. Además, poder alertarlo con una hora de antelación a su llegada al planeta permitiría tomar acciones en caso de ser necesarias al detectar un comportamiento anómalo y de riesgo.

El siguiente capítulo detalla el desarrollo del proyecto una vez se tiene un *dataset* sobre el que trabajar, donde el primer paso será la extracción del conjunto de datos para acceder a los datos telemétricos que se han expuesto en este apartado.

3. Desarrollo

En este capítulo se exponen los pasos necesarios para el desarrollo del proyecto. En la Figura 12 se presenta el esquema general del mismo. Este comienza con la extracción de datos sobre el dataset elegido, detallando su proceso de extracción. El siguiente paso es la preparación del dataset obteniendo los datos de training y de test con los que se entrenarán y detectarán *outliers* respectivamente los modelos detallados al final del capítulo.

Figura 12. Flujo de trabajo en el proyecto

3.1. Extracción de los datos

Este apartado recoge todo el proceso de extracción de los datos, con las herramientas elegidas para ello. Primeramente, se recuerda que el *dataset* elegido pertenece a la misión *Advanced Composition Explorer* (ACE) para el estudio del entorno espacial y su relación con la Tierra. Concretamente, el *dataset* consiste en una serie temporal sobre el módulo del vector IMF desde el punto L1.

Para la extracción del *dataset* se ha optado por utilizar *Space Physics Data Facility,* el archivo de la NASA descrito en el capítulo anterior. Dentro de este servicio, la plataforma CDAWeb proporciona acceso desde multitud de librería a innumerables sets de datos. Concretamente, se utiliza la *Heliophysics* API (HAPI) para extraer los datos desde Python.

Con HAPI se puede acceder a otros servidores aparte de CDAWeb. Para ver la lista completa de servidores, se puede consultar su página web [36]. Desde esa web se pueden mostrar los datos de cualquier *dataset*, parámetro dentro del mismo y

temporalidad elegida. Según la selección realizada, se mostrará la gráfica correspondiente a esos datos. Esto es muy útil para explorar diferentes conjuntos de datos y parámetros dentro de ellos en aras de elegir uno que se preste a detectar anomalías sobre él. De esta manera, se consultaron diversos *datasets* y se eligió uno de ellos.

Para utilizar HAPI en Python, ha de instalarse su cliente, llamado **hapiclient**. Además, si se desee mostrar alguna gráfica sobre los datos extraídos, es muy útil usar hapiplot, pues sus funcionalidades permiten representar los datos extraídos con el cliente de HAPI en una simple línea de código. Así, pueden extraerse diferentes *datasets* y ver su aspecto en una gráfica, tal y como se hacía en el servidor web de HAPI pero ya en código desde Python.

Dentro de la librería hapiclient, la clase hapi es la necesaria para hacer la llamada y extraer los datos. Los parámetros más importantes (hay otros) que se le pueden pasar a esa clase son:

- Servidor a utilizar, que en este caso será el link de CDAWeb.
- Dataset con el que se va a trabajar, que será el de la misión ACE descrita. Su identificador único en la HAPI es AC_H1_MFI.
- Fecha de inicio y fecha de fin sobre los datos a extraer: la fecha inicial tomada es el 1 de junio de 2018 a las 00:00 mientras que el último dato se toma del 11 de julio de 2022 a las 23:56.
- Parámetros a extraer, que pueden ser varios o uno solo dentro de los disponibles en el dataset seleccionado.

Una vez claros los parámetros que se deben especificar a la clase hapi para extraer los datos, es importante saber qué devuelve esa clase. Se devuelve una tupla con dos objetos: los datos y los metadatos. Además, los datos son un *array* unidimensional donde cada uno de sus valores es una tupla donde el primer valor es el tiempo y el resto de valores son los parámetros pedidos en la llamada. En el caso de solicitar solamente un parámetro, cada tupla que compone el *array* unidimensional de datos constará de dos variables: el tiempo y el parámetro pedido.

De esta forma, el formato en el que se devuelven los datos no es el más adecuado para almacenarlos ni trabajar con ellos, por lo que se deben preparar adecuadamente para tener el dataset en un formato en que se le pueda aplicar cualquier algoritmo de detección de anomalías. Este proceso de preparación se narra en el siguiente apartado.

3.2. Preparación del dataset

El *dataset* extraído, según lo comentado en el apartado anterior, representa el módulo del vector de campo magnético interplanetario. Es una serie temporal con esa única variable, viniendo los datos en un *array* unidimensional donde cada valor es una tupla. Esa tupla se compone de dos elementos: el primero es la fecha como tipo de dato *numpy bytes* mientras que el segundo es el valor de ese módulo como un *float* de 64 bits. Las unidades en las que se expresa el IMF son el nano Tesla, nT. Es importante destacar que se tienen datos cada 4 minutos, por lo que desde el 1 de junio de 2018 a la 00:00 hasta el 11 de julio de 2022 a las 23:56 se recogen un total de 540720 valores.

Se pretende almacenar estos datos en un DataFrame, teniendo la fecha en un formato más adecuado. De esta forma, la fecha se transforma a *string* para posteriormente almacenarse en formato datetime en el DataFrame. Por otro lado, existen valores vacíos en la magnitud del IMF, encontrándose alrededor de 300 en la franja de fechas mencionada. Se decide imputar estos valores nulos con la media en ese periodo, puesto que así no serán detectados como anomalías. Por tanto, el objeto queda con una dimensión de 540720 filas y 2 columnas. Su aspecto se observa en la Tabla 1.

Fecha	Magnitud
2018-06-01 00:00:00	10.124
2018-06-01 00:00:00	9.597
2018-06-01 00:00:00	9.818
2018-06-01 00:00:00	9.659
2018-06-01 00:00:00	9.402

Tabla 1. Aspecto del DataFrame donde se almacenan los datos provenientes de la HAPI

Además, se realiza una partición para separar en datos para training y datos para test, dejando un 20% de los datos para test, quedando los datos de test a partir del 14 de septiembre de 2021. Esta división de los datos se realiza con la librería Scikit-learn,

mientras que en el resto de procesos mencionados se han utilizado las librerías Numpy y Pandas, muy conocidas en el tratamiento de datos con Python.

Se muestra en la Figura 13 los datos de entrenamiento y en la Figura 14 los datos de test, representando el IMF en el eje de ordenadas y el tiempo en el eje de ordenadas. Quedan en el *dataset* de **train 432756** datos mientras que en **test** son **108144**.

Figura 13. Datos de training del dataset AC_H1_MFI

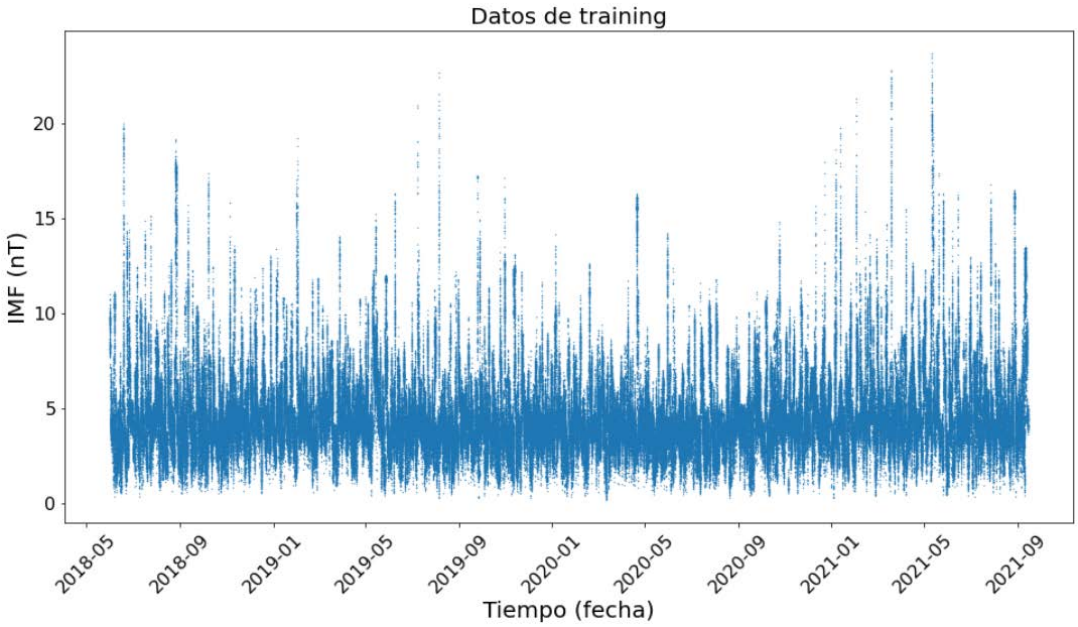
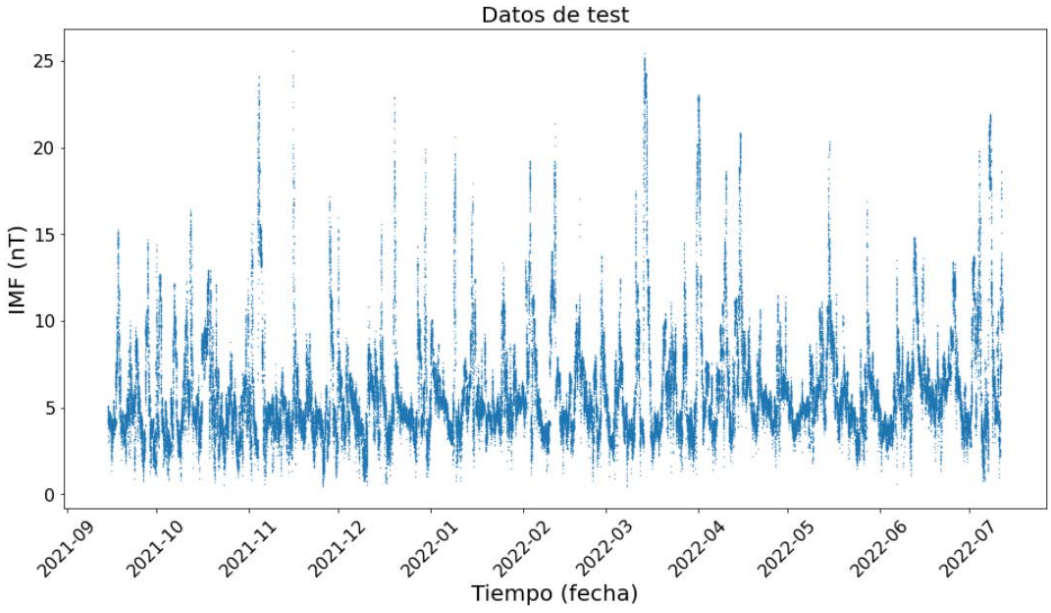


Figura 14. Datos de test del dataset AC_H1_MFI



A continuación, se presentan los modelos a utilizar para detectar las anomalías presentes sobre el conjunto de *test*, entrenando el modelo con los datos de *training*.

3.3. Contextualización de los modelos

Para la detección de anomalías sobre el *dataset* presentado, se van a utilizar una serie de algoritmos que tienen una característica en común: pueden trabajar sin etiquetas de anomalías, por lo que todos ellos trabajarán en un entorno de aprendizaje no supervisado. Se utilizarán varios en aras de comparar los resultados obtenidos y valorar cuál o cuáles de ellos son los más adecuados. Además, el *dataset* AC_H1_MFI es una serie estacionaria, por lo que no requiere de ninguna herramienta previa para convertirla en estacionaria, pudiendo directamente aplicar detección de anomalías sobre la serie.

La elección de los algoritmos a utilizar viene motivada por varios aspectos importantes que rodean la detección de anomalías no supervisada. En primer lugar, se va a utilizar la solución sencilla Out-of-Limits que se presentó en el capítulo 2. Se pretende, en base a sus resultados sobre el *dataset* de training, aproximar el porcentaje de anomalías presente en el *dataset*, pues ese dato es un hiperparámetro necesario que se debe especificar en los 4 algoritmos seleccionados.

Para implementar los algoritmos se ha decidido utilizar la librería PyOD [19], puesto que es la más completa y la única que permitía disponer en Python de todos los algoritmos de detección de *outliers* que se deseaban utilizar. Cada algoritmo de detección de anomalías es una clase dentro de la librería. Todos ellos tienen funciones comunes que se utilizan en este proyecto para entrenar los modelos y predecir con ellos. Las que se usan en la realización de este trabajo son las siguientes:

- **.fit():** entrena al modelo, en este trabajo solo será necesario que reciba el parámetro con los datos de entrenamiento, ya que no existen etiquetas.
- **.decision_function():** predice la puntuación de anomalía de los datos de test que reciba como parámetro, siendo este el único que necesita la función. El modelo debe estar previamente entrenado con el método fit. Además, se puede acceder también en cualquier modelo al umbral (valor decimal) a partir del cual el modelo asigna la etiqueta de outlier a los datos.

- **.predict()**: predice si un valor es outlier (1) o no (0), es decir, la etiqueta de outlier. Al igual que el método anterior, solo se requieren los datos de test como parámetro de entrada.
- **.predict_proba()**: predice la probabilidad de que un valor sea o no outlier transformando la puntuación de outlier linealmente al rango [0,1].

En el momento de la finalización de este trabajo, septiembre de 2022, la librería PyOD [19] cuenta con más de 40 algoritmos. Se hicieron pruebas con bastantes de ellos, descartando aquellos que no eran muy representativos del estado del arte, otros muy sencillos como *Principal Component Analysis* (PCA) o KNN, aquellos que necesitaban de etiquetas para ser entrenados y otros cuya utilización en series temporales no era efectiva.

Entre aquellos algoritmos que resultan de aplicar los filtros mencionados en el párrafo anterior, se eligieron 4. Todos ellos han sido descritos con detalle en el capítulo 2. El primero de ellos es LOF, que se elige por su aplicabilidad a *Novelty Detection* sobre una serie temporal. Otros dos son IForest y VAE, algoritmos muy representativos y utilizados ampliamente para la detección de anomalías en series temporales. Finalmente, se incluye un algoritmo muy reciente, publicado en agosto de 2022 e implementado poco después en PyOD [19]. Se trata de ECOD, cuya valía y rapidez de entrenamiento se probará para la detección de anomalías en el campo magnético interplanetario.

Aparte de ellos, se utiliza el método *ensemble* SUOD, que ya ha sido mencionado en el capítulo 2 de este documento y que se encuentra en la librería de PyOD [19]. Con este algoritmo se podrá obtener una combinación de las soluciones propuestas por los algoritmos anteriores, obteniendo así un modelo más robusto y general.

En la siguiente página se encuentra la Tabla 2 que contiene a los algoritmos, su año de publicación y su base teórica para después de ella exponer el comportamiento esperado de cada uno respecto al *dataset* elegido y los hiperparámetros con los que cuenta.

Para conocer los algoritmos y poder adaptarlos lo mejor posible al dataset concreto, se ha comprendido su funcionamiento y analizado cada parámetro que tiene. Este análisis se expone en los siguientes apartados, en aras de poder elegir los hiperparámetros más adecuados a la hora de implementar cada algoritmo.

Algoritmo	Abreviatura	Año de publicación	Tipo
Local Outlier Factor	LOF	2000	Proximidad
Empirical-Cumulative Outlier Detection	ECOD	2022	Probabilístico
Isolation Forest	IForest	2008	Conjuntos
Variational AutoEncoder	VAE	2013	Redes neuronales
Scale Unsupervised Outlier Detection	SUOD	2021	Ensemble

Tabla 2. Resumen de los algoritmos que se van a utilizar de detección de anomalías

3.3.1. Out of Limits

No es un algoritmo, sino una solución sencilla como ya se ha mencionado. Se pueden establecer los límites en el algoritmo en función de valores que no se deben sobrepasar basándose en el conocimiento que se tiene sobre la magnitud concreta. En el caso del *dataset* AC_H1_MFI que contiene la evolución temporal del campo magnético interplanetario, se ha descrito como un IMF fuerte a partir de 20 nT y muy fuerte a partir de 30 nT. No tendría sentido establecer límites inferiores puesto que la magnitud de este vector siempre va a ser superior a 0 y ningún valor cercano a él supondría algún problema. Sin embargo, el IMF no es una magnitud que se preste a establecerle estos límites manualmente, pues esto sería más propio de magnitudes que deben estar siempre dentro de un rango definido con criterios de durabilidad de materiales o equipos.

Por otro lado, se decide aplicar OOL teniendo en cuenta la definición estadística de outlier más extendida. Todos aquellos valores atípicos son leves en caso de superar por encima o por debajo el valor del cuartil correspondiente más 1.5 veces el rango intercuartílico. Los *outliers* o valores atípicos graves serán aquellos que hagan lo anterior, pero sumándole 3 veces el rango intercuartílico en lugar de 1.5 veces. Las siguientes expresiones representan lo expuesto, donde Q_1 es el primer cuartil, Q_3 el tercer cuartil e IQR el rango intercuartílico:

$$\textbf{Leve (soft limits)}: \; x < Q_1 - 1.5 \cdot IQR \quad \text{ó} \quad x > Q_3 + 1.5 \cdot IQR$$

$$\textbf{Grave (hard limits)}: \; x < Q_1 - 3 \cdot IQR \quad \text{ó} \quad x > Q_3 + 3 \cdot IQR$$

De esta manera se establecen los límites *soft* y *hard*, tanto por arriba como por abajo. Además, todos los valores que superen los límites *hard* se tomarán como *outliers* para

calcular el porcentaje de estos en los datos de train y así estimar el porcentaje de *outliers* en el *dataset*. Con ello, se tendrá información suficiente para construir los siguientes modelos, que ya sí son algoritmos de *machine learning* o *deep learning* y como tales requieren de un tuneo de sus hiperparámetros para obtener el mejor comportamiento del modelo. Por ello, se introduce brevemente los hiperparámetros con los que cuenta cada uno de los algoritmos, muchos de los cuales se tunearán, explicando este proceso en el capítulo 4.

3.3.2. Local Outlier Factor

Como ya se ha mencionado, este no es un algoritmo adecuado para la detección de anomalías en series temporales. Sin embargo, sí que resulta útil para la tarea de detectar de *novelty detection*, es decir, la detección temprana de valores anómalos a través de comportamientos que pueden precederlos. Este algoritmo tiene con un parámetro en PyOD [19] para elegir si se usa para *novelty* o no. Además, el modelo cuenta con hiperparámetros que deberán ser tuneados con el objetivo de obtener el mejor rendimiento posible del modelo. Todos sus parámetros de entrada se encuentran brevemente explicados a continuación, para conocerlos de cara al tuneo de hiperparámetros que se realizará en el capítulo 4.

- **n_neighbors**: número de vecinos para los que calcular las distancias RD, LRD y LOF que componen el modelo, vistas en el apartado 2.2.1. Por defecto, este valor es 20.
- **algorithm**: algoritmo elegido para obtener los vecinos. Por defecto, elige automáticamente el algoritmo en base a los datos con los que se entrene el modelo al llamar al método fit.
- **leaf_size**: tamaño de hoja para los algoritmos BallTree o KDTree, opciones en el parámetro anterior. Afecta a la velocidad y memoria requerida en la construcción de árboles.
- **metric**: métrica usada para la distancia. Por defecto, se trabaja con la distancia de Minkowski pero puede usarse la distancia coseno, de Jaccard, etc.
- **p**: parámetro p de la distancia de Minkowski. Por defecto es 2 para utilizar la distancia euclídea, pero puede establecerse 1 si se quiere usar la de Manhattan u otros valores en caso de desear una distancia de Minkowski diferente.

- **metric_params**: parámetros adicionales a la métrica de distancia, por defecto ninguno.
- **contamination**: porcentaje de outliers o contaminación en el dataset. Fundamental para definir el umbral al entrenar el modelo, por lo que determina enormemente la cantidad de outliers a detectar.
- **n_jobs**: número de ejecuciones en paralelo para la búsqueda de vecinos, 1 por defecto. No afecta a los resultados, simplemente al rendimiento computacional.
- **novelty**: valor booleano para indicar si se usa o no para *novelty detection*. Por defecto no se usa con ese cometido, pero en este trabajo sí se utilizará LOF con este objetivo.

3.3.3. Empirical-Cumulative Outlier Detection

ECOD es un algoritmo de detección de anomalías que se comporta bien ante series temporales. Además, su desventaja al trabajar con datos con varias dimensiones no se manifestará en el *dataset* AC_H1_MFI, puesto que es una serie temporal unidimensional donde solo se expresa la evolución del IMF con el tiempo. De esta forma, se intuye que el algoritmo tendrá un buen rendimiento para detectar anomalías sobre el *dataset* mencionado.

Por otro lado, el coste computacional será del orden del número de datos, pues la dimensión es 1. De esta manera, el coste computacional es muy reducido y se espera que el entrenamiento sea muy rápido. Por tanto, a priori parte como un algoritmo muy adecuado para la tarea y el *dataset* concretos. Además de todo lo anterior, es un algoritmo sin hiperparámetros específicos, por lo que no requiere tuneo alguno. Sin duda, esta es una gran ventaja en cuanto a costes tanto de recursos computacionales como de tiempo empleado. Los únicos parámetros de entrada con los que cuenta la clase del algoritmo ECOD en PyOD [19] son:

- **Contamination**: parámetro presente en todos los algoritmos de aprendizaje no supervisado para definir el umbral, explicado ya anteriormente.
- **n_jobs**: también mencionado con anterioridad, define el número de ejecuciones en paralelo para el entrenamiento y la predicción, siendo 1 por defecto.

Por tanto, este algoritmo no requerirá tuneo de hiperparámetros salvo el valor contaminación o porcentaje de anomalías que se estimará para todos con el método OOL basándose en la definición de outlier, tal y como se comentó en el apartado 3.3.1.

3.3.4. Isolation Forest

El algoritmo IForest tiene muy buen comportamiento para detectar anomalías en series temporales. Al no tener desventajas frente a grandes *datasets* o a series temporales de una sola variable, se espera un buen rendimiento sobre el *dataset* AC_H1_MFI. Sin embargo, no está exento de hiperparámetros como ocurre con ECOD, aunque tampoco cuenta con demasiados. A continuación, los que tiene:

- **n_estimators**: es el número de iteraciones que se van a realizar. Se recuerda que una iteración es la división de todos los datos de entrenamiento. Es un parámetro muy importante en el modelo tanto a nivel de rendimiento computacional como a nivel de resultados.
- **max_samples**: el algoritmo puede entrenarse con cierta cantidad de datos, en cada iteración, puesto que si se entrena con todos los datos en todas las iteraciones no sería asumible en grandes *datasets*. Por defecto, este valor es el mínimo entre 256 y el número de muestras en el *dataset*.
- **contamination**: parámetro ya comentado anteriormente.
- **max_features**: máximo número de atributos de los datos con los que entrenar en cada iteración. Por defecto su valor es 1. Este parámetro se establece para que el algoritmo sea asumible ante *datasets* con muchas características.
- **bootstrap**: valor booleano que indica si los árboles se entrenan con subconjuntos de entrenamiento con o sin reemplazo. Por defecto, se realiza sin reemplazo.
- **n_jobs**: también comentado anteriormente.
- **behaviour**: parámetro obsoleto que determina de qué API se toma el algoritmo.
- **random_state**: las divisiones que se realizan son aleatorias y para que el algoritmo obtenga los mismos resultados siempre que se ejecute se puede fijar este valor semilla. Por defecto es nulo.

- **verbose**: informa o no por pantalla de los procesos del algoritmo durante el entrenamiento.

3.3.5. Variational AutoEncoder

El VAE también es un algoritmo con buen comportamiento para la detección de *outliers* en series temporales. Sin embargo, cuenta con mucho hiperparámetros y su entrenamiento es bastante costoso, más que el de cualquiera de los algoritmos anteriores, pues se tienen que entrenar los pesos de una red neuronal con bastantes capas. Los parámetros con los que se define el algoritmo VAE en PyOD [19] y que se deberán tener en cuenta para elegir los más adecuados y obtener un mejor rendimiento son los siguientes:

- **encoder_neurons**: estructura de capas y neuronas en el *encoder*. Por defecto, se crea un *encoder* cuyas capas ocultas tienen 128, 64 y 32 neuronas.
- **decoder_neurons**: estructura de capas y neuronas del *decoder*. Por defecto, tiene la estructura inversa al *encoder*: 32, 64 y 128 neuronas en tres capas.
- **latent_dim**: número de dimensiones en el espacio latente, por defecto 2.
- **hidden_activation**: función de activación en las capas ocultas, por defecto la más utilizadas en este sentido que es la ReLU.
- **output_activation**: función de activación para la capa de salida. Por defecto es la sigmoide.
- **loss**: función de pérdidas para medir las diferencias entre los datos de entrada y los reconstruidos. Por defecto es '*Mean Squared Error*' (MSE).
- **optimizer**: método optimizador usado para minimizar la función de error, siendo por defecto Adam.
- **epochs**: número de épocas de entrenamiento, por defecto 100.
- **batch_size**: tamaño de *batch*, es decir, número de muestras con las que actualizar el modelo.
- **dropout_rate**: porcentaje de neuronas desconectadas con la técnica de *dropout*, por defecto son el 20%.
- **l2_regularizer**: fuerza del regularizador l2 aplicado en cada capa, por defecto 0.1.

- **validation_size**: porcentaje de datos usados para la validación, siendo por defecto el 10%.
- **preprocessing**: valor booleano que indica si aplicar o no estandarización de los datos, aplicándose por defecto.
- **verbose, random_state, contamination**: parámetros ya comentados.
- **gamma**: coeficiente de peso a la parte beta en el término de pérdidas, este se modifica si se quiere usar el BetaVAE. Por defecto es 1, que es el VAE normal.
- **capacity**: parámetro de capacidad para utilizar el BetaVAE. Su valor por defecto es 0 que corresponde con el VAE sin modificaciones en el término de pérdidas.

3.3.6. Large-scale Unsupervised Outlier Detection

SUOD se mencionó en el capítulo 2 sobre el estado del arte. Se recuerda que es un algoritmo de aquellos conocidos como métodos **ensemble**, que aúnan las predicciones de varios algoritmos para combinarlas y obtener una única más robusta y genérica que las anteriores. El algoritmo cuenta con numerosos hiperparámetros en PyOD [19] para implementarlo. Sin embargo, en este documento solo se van a mencionar los más importantes para una implementación rápida, que coinciden con aquellos que se van a modificar (salvo uno de ellos) respecto a sus valores por defecto. Son los siguientes:

- **base_estimators**: lista de algoritmos que se van a combinar para obtener un resultado único. Por defecto su valor es None. Cabe detacar que la lista debe tener más de un algoritmo y que estos pueden estar configurados con los hiperparámetros que se desee en cada uno.
- **contamination**: porcentaje de anomalías presentes en el dataset. Este parámetro ya se ha mencionado en el resto de algoritmos, pues está presente en los métodos no supervisados para tener una referencia de la cantidad de anomalías que existen. Su valor por defecto es 0.1.
- **combination**: define la forma de agregar y combinar los diferentes modelos. Hay dos opciones: hacer la media de los resultados de los modelos si se elige el valor '*average'* u obtener el valor máximo para cada dato de los que obtiene cada algoritmo. Esta última opción se elige dando el valor '*maximization'* y significa que elegirá como anomalía cualquier valor que al menos uno de los algoritmos lo detecte como tal. El valor por defecto de este parámetro es *'average'*.

- **verbose**: True o False, este parámetro determina si se muestra información del proceso de entrenamiento. Por defecto, su valor es False.

En el siguiente capítulo del documento se presentan los resultados obtenidos con OOL y con cada uno de los algoritmos mencionados. También se explica el proceso de tuneo de hiperparámetros. Es importante destacar la dificultad de este proceso en un entorno de aprendizaje no supervisado, pues al no tener etiquetas no existe un resultado final correcto para cada dato y por tanto no es posible contar con una métrica que evalúe con precisión el rendimiento del modelo. Por ello, se tunearán los hiperparámetros más importantes de cada modelo en base a recomendaciones del estado del arte y a los resultados obtenidos por el modelo.

3.4. Online machine learning con PySAD

Por último, se elige PySAD como librería para aplicar técnicas de *online machine learning* al *dataset*. Esta librería fue mencionada en el capítulo 2 y se recuerda que es una herramienta específica para la detección de anomalías con SML. Se ha probado también bytewax y river, pero la primera no tiene una buena presentación de resultados y la segunda solo permite usar sus algoritmos de detección de anomalías y son escasos. Además, PySAD presenta una gran sinergia con PyOD [19], librería que se ha utilizado para implementar los diferentes algoritmos en Python.

El proceso de aplicar técnicas de *stream machine learning* (SML) se basa en crear un *dataflow* e ir entrenando un modelo con ventanas de datos deslizantes. Si se tratase de un entorno supervisado, las etiquetas permitirían obtener métricas en tiempo real que indiquen el rendimiento del modelo. Sin embargo, en este caso no existe métrica alguna y simplemente se entrena el modelo elegido con una ventana de datos de un cierto tamaño, que se modificará tras un determinado número de pasos.

En la Figura 15 se observa el flujo de trabajo que se sigue en el proceso de detección de anomalías con técnicas de SML. La extracción de datos y preparación del *dataset* va seguida de un entrenamiento con una determinada ventana de datos. A cada dato perteneciente a esa ventana se le asigna una puntuación de anomalía y, llegado un cierto número de datos, se cambia a una próxima ventana con la que se vuelve a repetir el proceso. Se debe tener en cuenta que el número de datos con el que se entrena cada

ventana no tiene por qué coincidir con el número para el que se desliza a la ventana siguiente. Normalmente, el tamaño de la ventana de datos es bastante mayor que los datos necesarios para cambiar de ventana, por lo que con muchos datos se entrena en varias ventanas diferentes. Este proceso se conoce como entrenamiento continuo, cuyo comportamiento se puede ver en la Figura 1 y es propio del SML.

Figura 15. Flujo de trabajo con stream machine learning (SML)

El modelo elegido para aplicar la detección de anomalías proviene de PyOD [19] y es IForest. Se elige este porque es el único de los mencionados en este documento que se ha podido implementar con PySAD. Para llevar a cabo su utilización en PySAD aplicando técnicas de *online machine learning*, se van a utilizar dos clases importantes de la librería:

- **PandasStreamer**: permite crear un *dataflow* a partir de un DataFrame de Pandas. Se usa la función iter de la clase para iterar sobre él.
- **ReferenceWindowModel**: permite realizar el entrenamiento por ventanas, mover las ventanas tras un determinado número de datos y especificar el modelo de detección de anomalías a utilizar. La clase tiene una función fit_score_partial que permite entrenar el modelo con la ventana de datos que corresponde y predecir una puntuación de anomalía.

Es importante tener en cuenta que en *online machine learning* con PySAD no se puede obtener una etiqueta de anomalía, sino que solo se obtiene la puntuación y el umbral debe establecerse posteriormente. Por ello, se probará con distintos umbrales para valorar cuál encaja mejor sobre el *dataset* y modelo utilizados.

En el siguiente capítulo del documento se presentan los resultados obtenidos con OOL y con cada uno de los algoritmos mencionados. También se explica el proceso de tuneo de hiperparámetros. Es importante destacar la dificultad de este proceso en un entorno

de aprendizaje no supervisado, pues al no tener etiquetas no existe un resultado final correcto para cada dato y por tanto no es posible contar con una métrica que evalúe con precisión el rendimiento del modelo. Por ello, se tunearán los hiperparámetros más importantes de cada modelo en base a recomendaciones del estado del arte y a los resultados obtenidos por el modelo. También se incluirá la aplicación del algoritmo IForest con *online machine learning* y se realizará una comparativa entre los algoritmos y análisis de los resultados.

4. Resultados

En este capítulo el lector puede encontrar los resultados obtenidos de la aplicación de los algoritmos sobre el *dataset* para detectar anomalías en el campo magnético interplanetario. Primeramente, se detallan los resultados de cada uno de los algoritmos, así como los hiperparámetros elegidos y los motivos de esta elección. Finalmente, se encuentra una comparativa entre todos los modelos en diferentes aspectos, realizándose un análisis sobre esta comparación.

4.1. Resultados de los modelos

Se exponen en este apartado los resultados de todos los modelos utilizados, comenzando por el método sencillo OOL para tener una primera visión de las anomalías que pueden existir en la serie temporal y estimar ese porcentaje. Después, se continúa con los resultados de los 4 algoritmos utilizados.

4.1.1. Out of Limits

El método sencillo OOL va a ser muy útil para la estimación de *outliers* en el *dataset*, parámetro que debe conocerse para la implementación de los 4 algoritmos que se utilizarán. Para ello, se aplicará el método estableciendo los límites en base a la definición de outlier.

Para comenzar, se obtienen las características principales del IMF en el *dataset* de *train*. Estas se recogen en la Tabla 3. Se tienen 432576 datos donde el valor mínimo es 0.1870 nT mientras que el valor máximo es de 23.7180 nT. Además, dos datos importantes que marcarán los límites de OOL lo cuartiles Q_1 y Q_3, que tienen valores de 3.3800 nT y 5.3490 nT respectivamente. Por lo tanto, el valor de IQR es de 1.9690 nT. Todos los valores se han redondeado a cuatro decimales.

Estadístico	Valor
count	432576
mean	4.6073
std	2.0360
min	0.1870
25%	3.3800
50%	4.2130
75%	5.3490
max	23.7180

Tabla 3. Datos del campo magnético interplanetario en el dataset de train

Con todo lo anterior, los límites establecidos en el método OOL quedan con los valores reflejados en la Tabla 4.

Límite	Valor
Bot soft limit	0.4265
Top soft limit	8.3025
Bot hard limit	-2.5270
Top hard limit	11.2560

Tabla 4. Límites establecidos para OOL en base a la definición más extendida de outlier

Aplicando estos límites, se obtienen que, de los 432576 datos de training, 409805 son valores normales que no superan los límites *soft*. Además, 36 datos están por debajo del límite *soft* inferior y 16825 se encuentran por encima del límite *soft* superior. Estas se consideran anomalías leves, mientras que las anomalías graves son aquellas que superan los límites *hard*. En este caso, solo hay datos que sobrepasan el superior, concretamente 5910, mientras que ninguno supera el límite *hard* inferior. Estos resultados se reflejan en la Tabla 5.

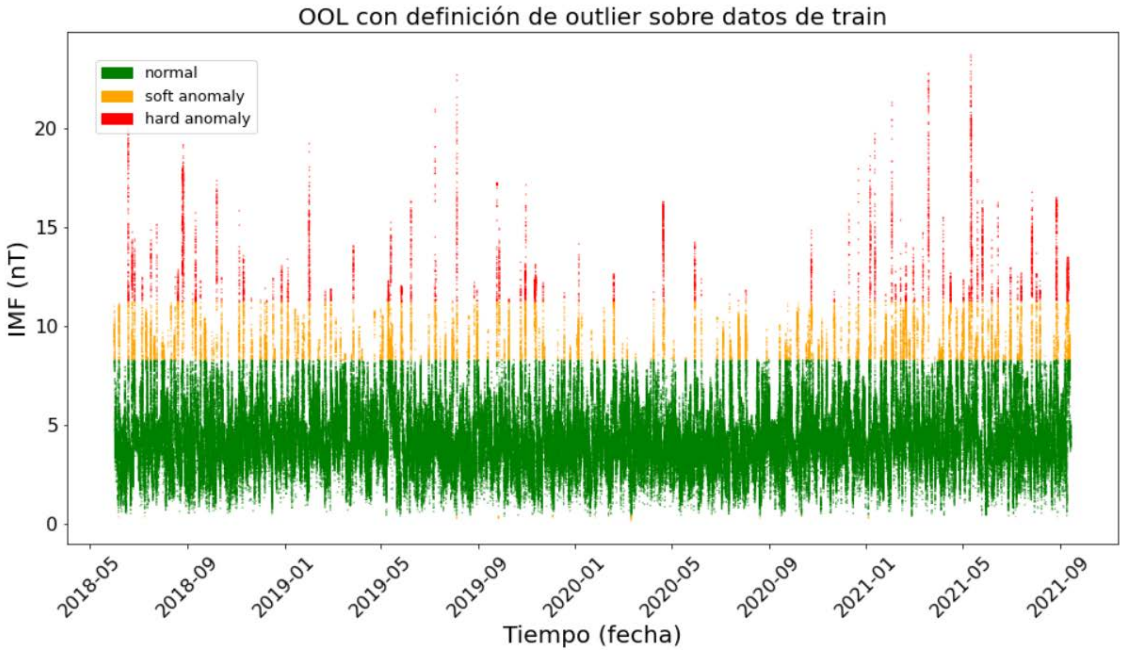
Zonas	Número de datos
Normal	409805
Anomalías leves inferiores	36
Anomalías leves superiores	16825
Anomalías graves inferiores	0
Anomalías graves superiores	5910

Tabla 5. Número de datos en cada rango de los 5 de OOL con la definición de outlier

Se toma el criterio de que las anomalías graves son aquellas que se tienen en cuenta como *outliers* para estimar el porcentaje de ellos en los datos de entrenamiento. Por tanto, según este método existen 5910 *outliers* entre las 432576 instancias. Esto da lugar a aproximadamente un 1.36% de *outliers* en el *dataset*. Teniendo en cuenta que el método es muy aproximado y que no existe una cantidad correcta de *outliers*, este valor se redondea al 1% de *outliers*, que será el valor del parámetro **contamination** en los algoritmos cuyos resultados se mostrarán en los siguientes apartados.

Por último, se muestra la representación gráfica de los datos de *train* por colores en la Figura 16, estando los valores normales en verde, las anomalías leves en naranja y las anomalías graves en rojo.

Figura 16. Aplicación de OOL con la definición de outlier sobre el dataset de train



4.1.2. Local Outlier Factor

Una vez establecido el parámetro **contamination** con un valor de **0.01**, se puede analizar el valor más adecuado para el resto de parámetros que requieren los algoritmos, ejecutarlos y obtener los resultados. En primer lugar, se analiza el algoritmo LOF, del que también se cuenta con otro parámetro establecido como es **novelty**. Como se ha comentado, se va a utilizar el algoritmo para *novelty detection* o detección temprana de las anomalías, así que este parámetro se establece como **True**.

Respecto a la distancia, como **métrica** se va a mantener **Minkowski** puesto que es la métrica más común de distancia y otras pruebas no afectan apenas al resultado del modelo. De hecho, se prueba con diferentes valores de p (1, 2 y 3) y con todos se obtiene el mismo resultado. Esto es lógico, ya que al ser un *dataset* unidimensional (solo se entrena con los valores del IMF), la distancia euclídea es la misma que la de Manhattan y la de cualquier p que se elija. Se elige **1** como valor de **p** por simplicidad. Además, no se va a añadir ningún parámetro adicional a la métrica, por lo que **metric_params** quedará como **None**.

Por otro lado, se realizan pruebas para obtener el valor óptimo de **n_jobs**, parámetro que determina la velocidad de entrenamiento del modelo. En el caso de LOF, este valor no es crítico porque el entrenamiento es muy rápido, pero en otros algoritmos puede ser un parámetro fundamental para ejecutar el modelo en un tiempo mucho menor. En la Tabla 6 se muestran los valores con los que se prueba y el tiempo de entrenamiento con cada uno. Como el tiempo de entrenamiento no es siempre el mismo, aunque se tengan los mismos parámetros, se realizan 3 pruebas con cada valor y en la tabla se añade la media de las pruebas. En vista de los resultados, se elige un valor de **n_jobs** de **16**.

n_jobs	1	2	4	8	16
Tiempo (s)	2.93	1.92	1.84	1.89	1.80

Tabla 6. Tiempos de entrenamiento según el parámetro n_jobs para el algoritmo LOF

En cuanto al **algoritmo** que se usa para obtener los vecinos, se ha comprobado que el que toma por defecto dados los datos de entrenamiento de este *dataset* es KDTree. Se realiza una prueba con BallTree, obteniendo exactamente los mismos *outliers*. Por ello, se deja el algoritmo por defecto, quedando este parámetro como **'auto'**. Además, el **leaf_size** afecta a ambos algoritmos, pero solo a su velocidad de ejecución. Dado que el tiempo de ejecución es de pocos segundos, no se hacen pruebas con este parámetro, dejando su valor por defecto que es **30**.

Por último, el parámetro más determinante en cuanto a los resultados del modelo: el número de vecinos con el que se calculará la probabilidad de cada dato de ser un *outlier*. Para establecer este parámetro, se consulta el *paper* [37] donde se presentó el método en el año 2000. En él, al parámetro **n_neighbors** lo llama MinPts y explica su impacto

y elección en el apartado 6 del documento. Ahí, los autores afirman que, en la mayoría de *datasets* con los que hicieron pruebas, el valor más adecuado estaba **entre 10 y 20**.

Sobre el *dataset* de la evolución del IMF en que se basa este trabajo, se comienza probando con un valor de 1 y efectivamente se obtienen malos resultados, ubicándose los *outliers* dispersos en las zonas más extremas. La siguiente prueba se realiza con un valor de 5, obteniéndose un resultado similar, aunque algo mejor. Con un valor de 10 los resultados se van acercando a lo que se pretende; una detección temprana de las anomalías. Finalmente, se prueba con 15 y los se observa entonces una clara mejora, hasta que el valor de **20** es el que se observa que da **mejores resultados** para la detección temprana o *novelty detection*. En la Figura 17, Figura 18, Figura 19 y Figura 20 se muestran las gráficas de las pruebas realizadas con los diferentes valores de n_neighbors que no han sido elegidos.

Figura 17. Predicción del algoritmo LOF sobre el dataset de test con n_neigbors = 1

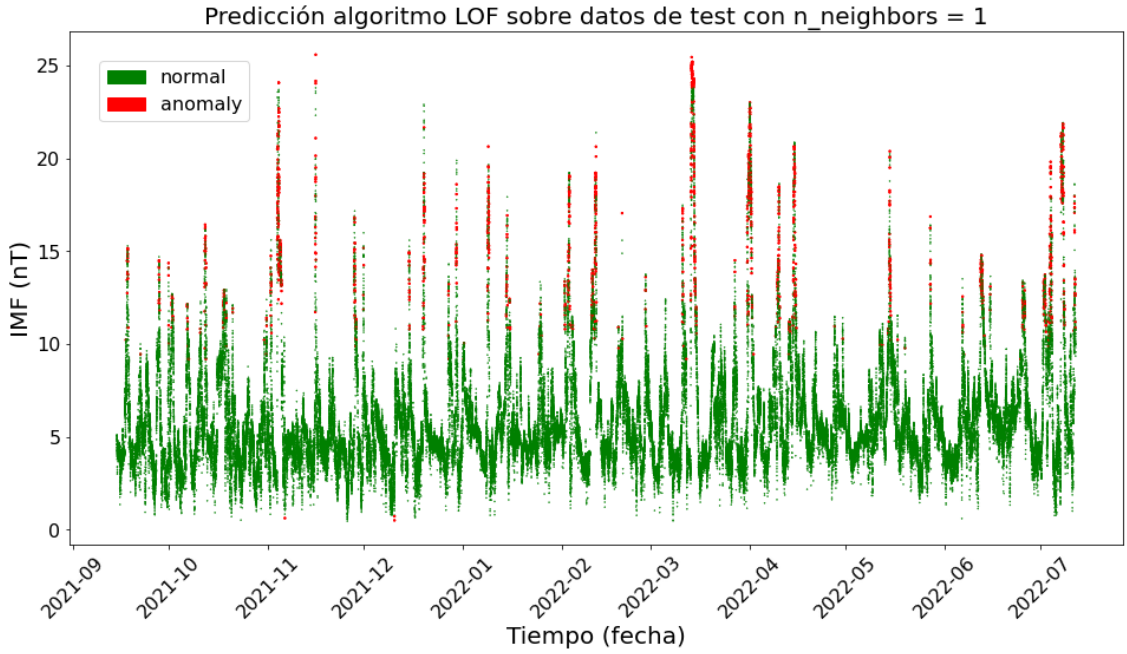


Figura 18. Predicción del algoritmo LOF sobre el dataset de test con n_neigbors = 5

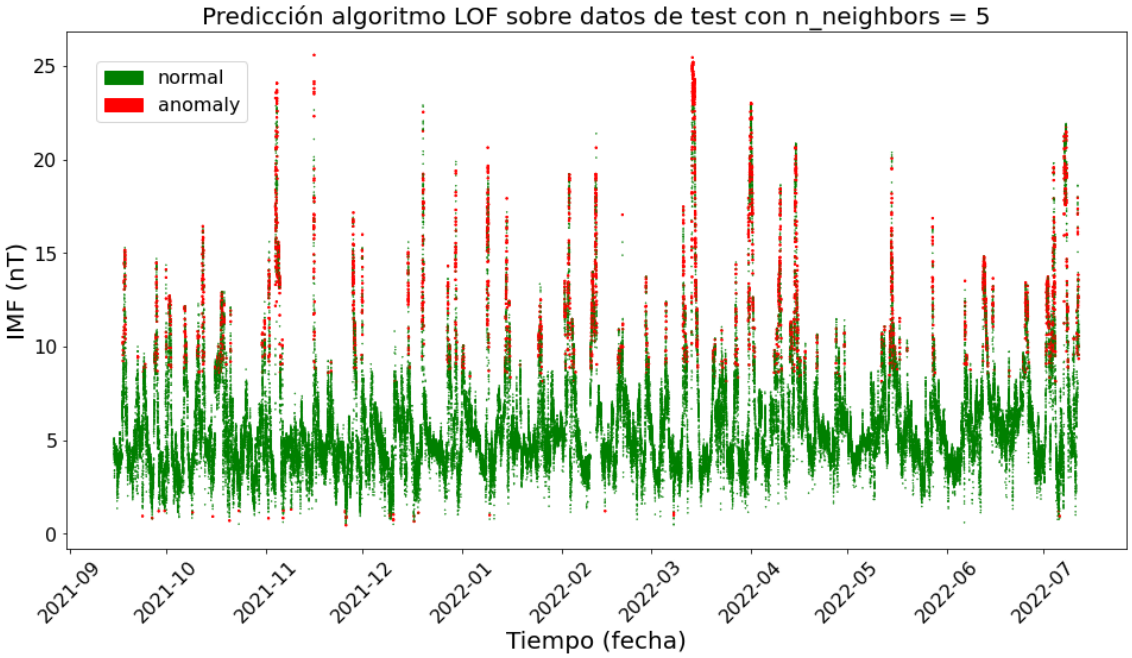


Figura 19. Predicción del algoritmo LOF sobre el dataset de test con n_neigbors = 10

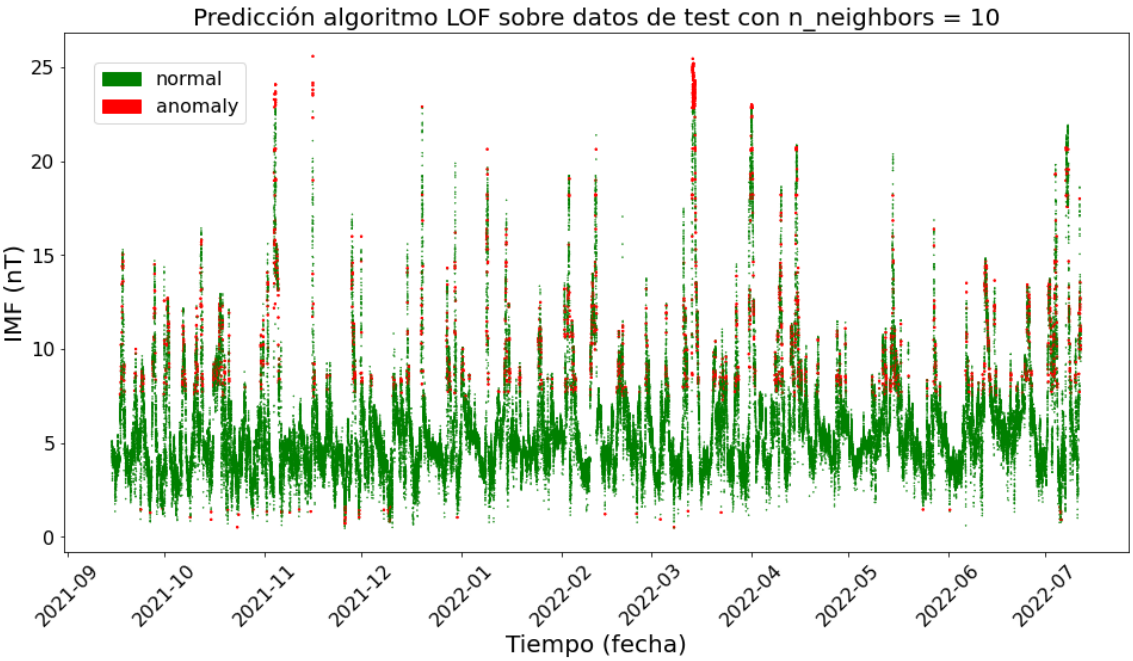
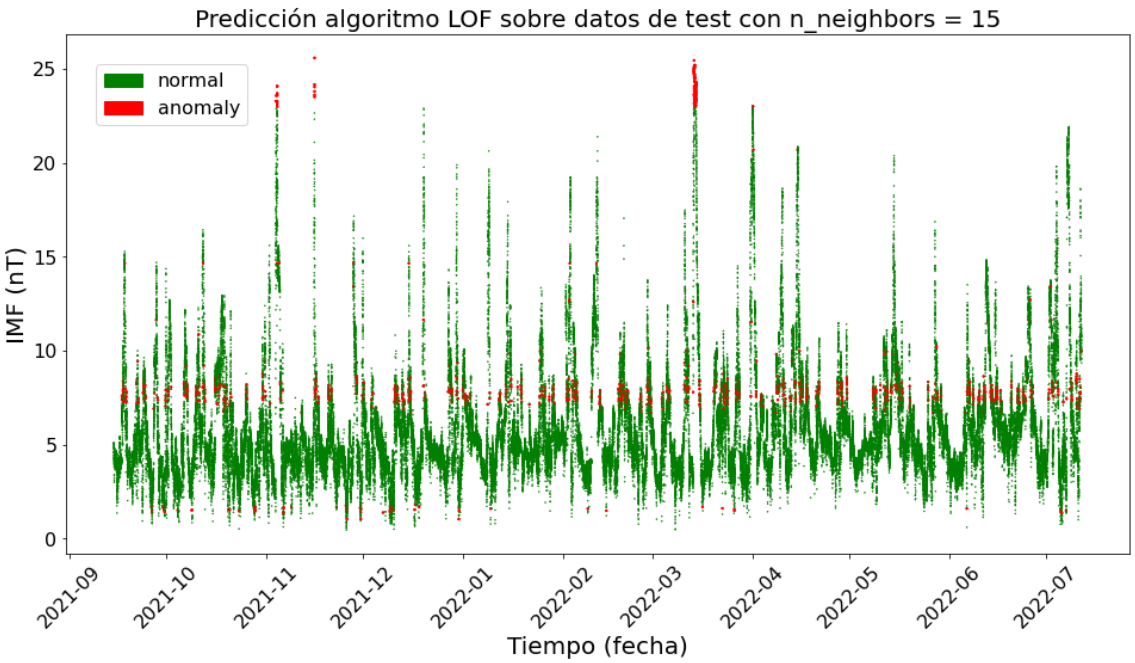


Figura 20. Predicción del algoritmo LOF sobre el dataset de test con n_neigbors = 15



Con este último parámetro, ya se han elegido todos los que tiene el modelo, que quedan recogidos en la Tabla 7:

Parámetro	Valor
n_neighbors	20
algorithm	'auto'
leaf_size	30
metric	'minkowski'
p	1
metric_params	None
contamination	0.01
n_jobs	16
novelty	True

Tabla 7. Parámetros elegidos para el modelo LOF

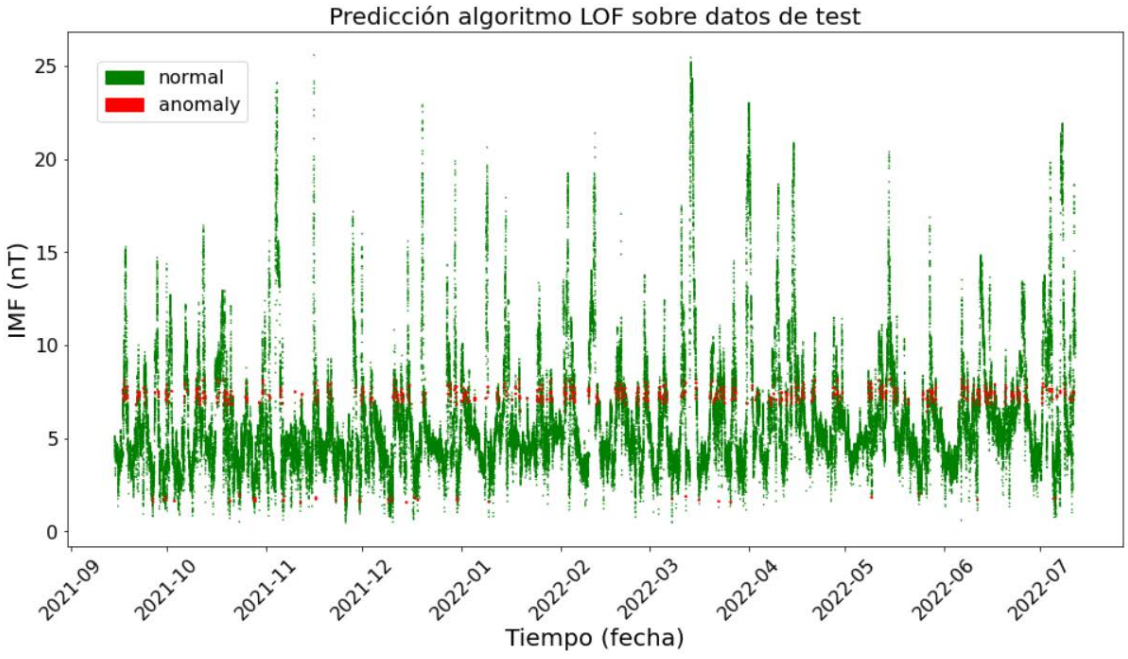
Una vez elegidos todos los parámetros e hiperparámetros del modelo, se obtienen los resultados finales. Estos se presentan en la Tabla 8:

Magnitud	Valor
Número de anomalías tempranas	1421
Porcentaje de anomalías tempranas (%)	1.31
Tiempo de entrenamiento (s)	1.80
Tiempo de predicción (s)	0.61
Probabilidad media de outlier (%)	0.41
Probabilidad media de outlier entre los asignados (%)	29.40

Tabla 8. Resultados obtenidos con el algoritmo LOF

Finalmente, se muestra en la Figura 21. Resultados finales obtenidos con el algoritmo LOF sobre el conjunto de test la gráfica con la predicción final del modelo dados todos los parámetros elegidos. Sobre ella se observa un comportamiento adecuado en cuanto a detección temprana de anomalías, que es el objetivo con el que se ha aplicado LOF sobre el *dataset*. Se puede ver una franja de anomalías en torno a 8 nT que precede a comportamiento que se escapan de lo habitual en el *dataset* y llevan al IMF por encima de 15 o incluso 20 nT. También existe una pequeña franja en torno a 1 nT, aunque el modelo en esa zona cuenta con pocos comportamientos anómalos y por ello no se aprecia tan claramente. En definitiva, LOF ha obtenido buenos resultados aplicando **novelty detection** con los parámetros elegidos.

Figura 21. Resultados finales obtenidos con el algoritmo LOF sobre el conjunto de test



4.1.3. Empirical-Cumulative Outlier Detection

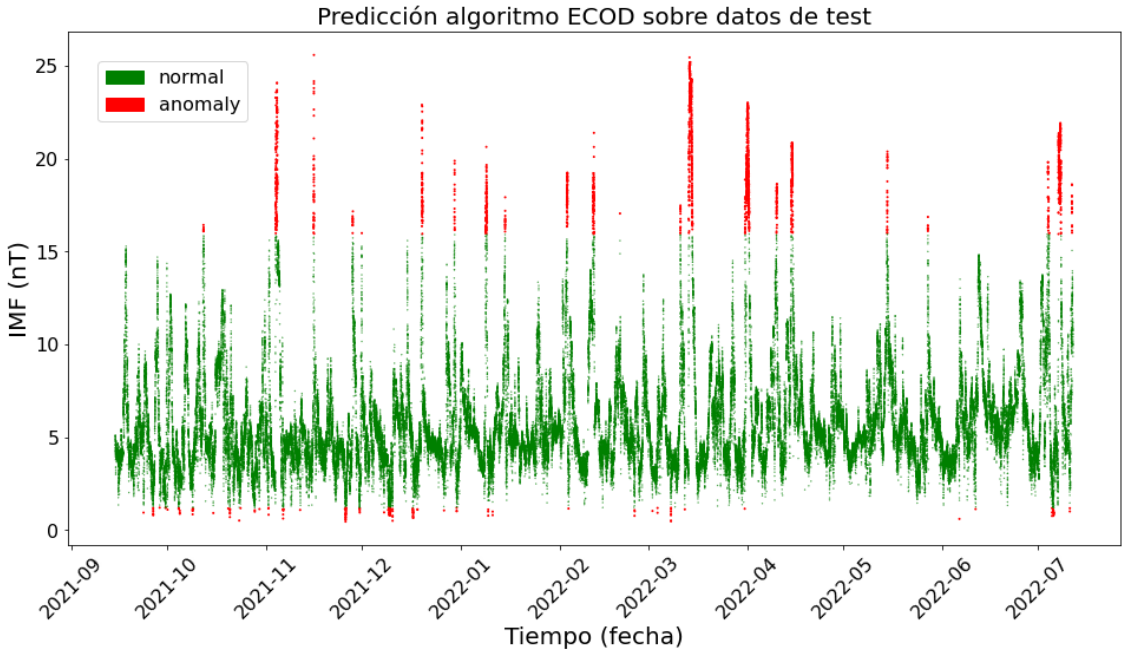
Este algoritmo se utiliza para la detección de anomalías, a diferencia de LOF que tenía la capacidad de usarse para *novelty detection*. El algoritmo ECOD cuenta con una ventaja clara respecto al resto que se utilizan en este trabajo: no requiere tuneo de hiperparámetros porque no los tiene. De hecho, el algoritmo en PyOD [19] solo cuenta con dos parámetros, que son **contamination** y **n_jobs**. Además, su entrenamiento es muy veloz debido a su naturaleza probabilística y a que el *dataset* concreto solo cuenta con una variable, por lo que su complejidad computacional es del orden del número de datos $O(n)$. Por ello, no se va a buscar un valor óptimo de **n_jobs** y se va a dejar el valor **1** que es el que tiene por defecto. Además, el parámetro **contamination** ya fue estimado con ayuda de OOL y su valor es de **0.01**. Por tanto, solo queda entrenar el modelo y obtener los resultados. Los resultados obtenidos entrenando con el conjunto de *train* y prediciendo sobre el de *test* son los expuestos en la Tabla 9.

Magnitud	Valor
Número de outliers	2102
Porcentaje de outliers (%)	1.94
Tiempo de entrenamiento (s)	0.34
Tiempo de predicción (s)	0.43
Probabilidad media de outlier (%)	9.17
Probabilidad media de outlier entre los asignados (%)	46.90

Tabla 9. Resultados obtenidos con el algoritmo ECOD

Como se pueden observar, los resultados de tiempo de entrenamiento y predicción son muy buenos. Además, se puede observar en la Figura 22 la gráfica con la predicción obtenida. En ella se muestra que el algoritmo sin hiperparámetros (solo dándole el porcentaje de *outliers*) es capaz de predecir con muy buenos resultados aquellos valores que se escapan del comportamiento normal en la serie temporal. Además, no muestra apenas *outliers* en el inferior, estando la mayoría en la zona superior, tal y como ocurre con los valores del IMF.

Figura 22. Resultados obtenidos con el algoritmo ECOD sobre el conjunto de test



4.1.4. Isolation Forest

Este algoritmo, a diferencia de ECOD, cuenta con varios hiperparámetros que habrá que ajustar para obtener el mejor comportamiento posible del modelo. En primer lugar, el parámetro de porcentaje de *outliers* ya se estableció con OOL y obviamente se usará el mismo en todos los algoritmos, así que **contamination** sigue siendo **0.01**. Además, **verbose** se asigna como **0** porque no interesan los mensajes informativos del proceso de entrenamiento, ya que se vieron y no son de gran interés. Otro parámetro sencillo es la semilla **random_state** que se establece con un valor cualquiera para obtener siempre los mismos resultados, eligiendo en este caso **42**. También es fácil y poco determinante la elección del parámetro **behaviour**, que solo indica a la API que se conecta para tomar el algoritmo así que se mantiene su valor por defecto **'old'**. Para el parámetro **max_features** no existe opción de elegir pues el *dataset* solo tiene un atributo, por lo que no se podrá entrenar con más ni menos de una característica, pues solo existe **1**.

En cuanto a aquellos parámetros que marcan en mayor medida el resultado, se comienza analizando si debe existir o no reemplazo en los conjuntos que se toman de entrenamiento. Se decide que no debe existir reemplazo, por lo que el parámetro **bootstrap** queda con valor **False**. El motivo es que en cada iteración se pretende coger nuevos datos para abarcar más de ellos y no repetir ninguno, que es la opción por

defecto. Además, se realizó una prueba cambiando solamente este parámetro y permitiendo reemplazo se obtuvieron 4502 *outliers* mientras que sin permitirlo se encontraron 4368. Como el número de *outliers* que se obtiene con el método es grande, interesa reducirlo así que este argumento también refuerza la decisión tomada.

Por otro lado, es importante establecer el número de iteraciones que se van a realizar a a través del parámetro **n_estimators**. Para ello, se consulta el *paper* donde se presentó el algoritmo [25]. En él se afirma que a partir de 100 el algoritmo ha convergido hasta un punto en que los resultados no cambian apenas. Sin embargo, no se va a iterar hasta dividir todas las muestras del *dataset*, por lo que se decide aumentar este número hasta **300**. El parámetro que indica cuantas muestras se van a utilizar en cada iteración es **max_samples**. Por defecto tiene como valor el mínimo entre 256 y el tamaño del *dataset*, por lo que sería en este caso 256. Sin embargo, se han realizado pruebas con otros valores en aras de encontrar el más adecuado. Aumentar el valor del parámetro hace que el entrenamiento sea más lento, pero se tengan en cuenta más datos y por tanto el modelo converja. En la Tabla 10 se observan los valores con los que se ha probado y el número de *outliers* obtenidos con cada uno, tomando 100 iteraciones y no 300 que es la elección final para un entrenamiento más rápido. Al llegar a las 8192 iteraciones, se observa que el número de *outliers* llega a un valor del que apenas cambia, aunque se sumen muchas iteraciones incluso hasta 65536. Por ello, se elige **8192** como valor del hiperparámetro **max_samples**. Además, se puede observar que solo se han hecho pruebas con potencias de 2 para facilitar su encaje con la estructura binaria de la CPU.

max_samples	256	512	4096	8192	16384	65536
outliers	5157	4652	4452	4368	4396	4407

Tabla 10. Número de outliers según el parámetro max_samples del algoritmo IForest

Una vez elegidos todos los hiperparámetros salvo n_jobs, debe estimarse este último según se obtenga menor velocidad de entrenamiento. Para ello, también se realizan pruebas con diferentes valores y se recogen los resultados en la Tabla 11. En ella, el tiempo es una media de tres pruebas con ese mismo valor, pues cada iteración tiene un tiempo diferente aun siendo los mismo hiperparámetros. Se observa que la diferencia entre los tiempos es mínima y, dada la dependencia de estos con otros procesos en la

CPU, se asume que el hiperparámetro no afecta a los tiempos y se mantiene el valor por defecto del **n_jobs** que es **1**.

n_jobs	1	2	4	8	16
Tiempo (s)	107.52	107.27	108.16	106.89	108.26

Tabla 11. Tiempos de entrenamiento según el parámetro n_jobs para el algoritmo IForest

Todos los hiperparámetros del modelo se encuentra ya elegidos, mostrándose en la Tabla 12.

Parámetro	Valor
n_estimators	300
max_samples	8192
max_features	1
behaviour	'old'
bootstrap	False
verbose	0
contamination	0.01
n_jobs	1
random_state	42

Tabla 12. Parámetros elegidos para el modelo IForest

Con la configuración final de hiperparámetros se obtienen los resultados que se muestran en la Tabla 13.

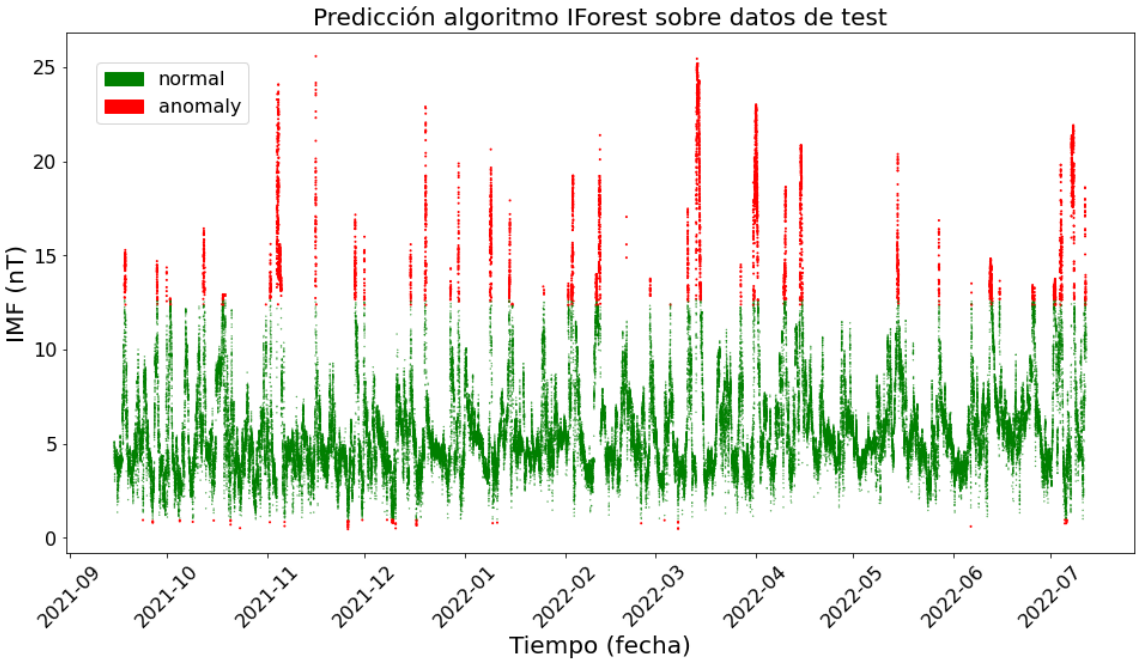
Magnitud	Valor
Número de outliers	4405
Porcentaje de outliers (%)	4.07
Tiempo de entrenamiento (s)	107.52
Tiempo de predicción (s)	0.43
Probabilidad media de outlier (%)	9.17
Probabilidad media de outlier entre los asignados (%)	39.12

Tabla 13. Resultados obtenidos con el algoritmo IForest

Se observa que los tiempos de entrenamiento y predicción son bastante mayores que los de ECOD, aunque siguen siendo completamente asumibles. Además, se obtienen

más *outliers* como puede verse en la gráfica de la Figura 23. Al igual que con ECOD, apenas quedan *outliers* en la parte inferior, situándose la mayoría en la zona superior de la serie temporal.

Figura 23. Resultados obtenidos con el algoritmo IForest sobre el conjunto de test



4.1.5. Variational AutoEncoder

El último algoritmo que se va a utilizar es el VAE. Es el algoritmo que más hiperparámetros tiene de todos los que se han aplicado. Su comportamiento debe ser adecuado para detectar anomalías, pero el tuneo de sus hiperparámetros será a priori importante para su rendimiento. Para exponer la elección de cada uno de sus hiperparámetros, al ser muchos, se van a detallar en una lista:

- **Gamma**: como se quiere trabajar con el VAE estándar y no con Beta-VAE, este hiperparámetro se mantiene en su valor por defecto **1**.
- **Capacity**: de igual manera que con el hiperparámetro anterior, se utiliza el valor del hiperparámetro propio de un VAE estándar, que en este caso es **0**.
- **Contamination**: este parámetro ya fue elegido con OOL y se mantiene en **0.01**.
- **Random_state**: semilla que evita aleatoriedad y se vuelve a establecer en **42**.

- **Verbose**: indica la información mostrada durante el proceso de entrenamiento, estableciéndose en **2** que es la opción de máxima información para conocer el entrenamiento al detalle por época.
- **Optimizer**: se decide utilizar el optimizador **'adam'** por ser el más extendido y mejorar al *Stochastic Gradient Descent* (SGD) según se mostró en el *paper* que se presentó el método [38].
- **Preprocessing**: siempre es más adecuado estandarizar los datos que trabajar con ellos en crudo, aunque al solo tener una variable la estandarización no cambia los resultados, probando con ambos valores de este hiperparámetro y comprobando efectivamente que no cambia el resultado. Además, el tiempo de entrenamiento no se ve afectado. Se elige **True** por ser lo más común en *machine learning*, aunque no afecte al resultado.
- **Validation_size**: en el campo de *machine learning*, este valor suele estar entre 0.1 y 0.2. Se prueba con ambas posibilidades extremas y se obtiene exactamente el mismo número de *outliers* en ambas, por lo que el hiperparámetro no afecta al resultado final en este *dataset*. Se elige **0.1** por ser un *dataset* grande en el que con ese tamaño de validación es suficiente, como se demuestra.
- **Dropout_rate**: durante el entrenamiento no se observa *overfitting*, por lo que a priori no será importante desconectar neuronas de la red. Se prueba con valores de 0, 0.2 y 0.5 y se obtiene exactamente la misma cantidad de *outliers*, por lo que se elige el valor **0** como final.
- **L2_regularizer**: también es un hiperparámetro usado para paliar el sobreajuste u *overfitting*, por lo que se prevé no ser necesario. Se prueba con los mismos valores que en el hiperparámetro anterior y efectivamente no afecta al resultado, por lo que se elige no aplicar regularización y dejar el valor **0**.
- **Epochs**: como ya se ha comentado, no se observa sobreajuste y además el entrenamiento converge muy rápidamente. De hecho, las pérdidas en validación no cambian desde la época 5 hasta la 50, por lo que se elige que el entrenamiento conste solamente de **5** épocas. Incluso con 1 sola época, el número de *outliers* obtenido es el mismo, pero se prefiere mantener 5.

- **Loss**: función de pérdidas para medir el error entre la reconstrucción y el valor real. Se elige mantener **Mean Squared Error (MSE)**, probando también a usar *Mean Absolute Error* (MAE) y obteniendo los mismos *outliers*.
- **Output_activation**: al ser la salida una puntuación de outlier, se considera la función **sigmoide** como la mejor opción para la activación de la capa de salida.
- **Hidden_activation**: para las activaciones de las capas intermedias, se elige la función más utilizada en estas capas que es la **ReLU**, por reducir el coste computacional.
- **Latent_dim**: la dimensión del espacio latente tiene por defecto valor 2, pero en este caso el *dataset* dolo tiene una característica y *encoder* y *decoder* terminan y empiezan respectivamente con una neurona, por lo que la dimensión del espacio latente más adecuada es **1**.
- **Encoder_neurons**: el número de neuronas de la última capa del *encoder* no debe exceder el número de características del *dataset*. Por ello, se debe terminar con una neurona. Se elige una estructura de 4 capas que comienza en 64 neuronas y va reduciéndose en un factor 4 hasta llegar a 1. Sin embargo, también se prueba con dos capas solamente, la primera de 64 neuronas y la segunda de 1. Los resultados obtenidos son exactamente los mismos. Por ello, no importa la elección entre una de las dos estructuras, pero se prefiere una más progresiva aunque tenga más capas, siendo la elección final **[64, 16, 4, 1]**.
- **Decoder_neurons**: una vez elegida la estructura del *encoder*, la del *decoder* será exactamente la inversa que es lo más común en un VAE o un AutoEncoder estándar. Por tanto, este hiperparámetro queda como **[1, 4, 16, 64]**.
- **Batch_size**: conviene que este hiperparámetro tenga un valor que sea una potencia de dos para que encaje con la estructura binaria de la CPU. Se conoce que un batch_size muy pequeño ralentiza mucho cada época y además las actualizaciones de pesos son muy sensibles a los *outliers*. Un batch_size muy grande tiene la desventaja necesitar más épocas para converger y de memoria RAM. Sin embargo, el algoritmo con este *dataset* converge muy rápidamente y además se eligieron 5 épocas cuando solamente con 1 convergía. Se han hecho pruebas con diferentes valores, obteniendo el mismo número de *outliers* en todas ellas por lo que el hiperparámetro se va a elegir en función del tiempo de entrenamiento. Los resultados se recogen en la Tabla 14. Métricas de

clasificación para los diferentes umbrales en IForest con SML. Se elige finalmente un valor de **512** porque al llegar a él se reduce considerablemente el tiempo de entrenamiento y a partir de él el tiempo de entrenamiento disminuye muy lentamente, pero aumentando considerablemente la exigencia a la RAM. Por ello, los beneficios a partir de ese valor no compensan la gran carga añadida en memoria.

batch_size	32	64	128	512	4096
Tiempo (s)	204.85	116.31	74.21	43.74	35.79

Tabla 14. Tiempos de entrenamiento según el batch_size para el algoritmo VAE

Sobre el número de épocas, se muestra en la Tabla 15 la pérdida de reconstrucción en el conjunto de training y en el de validación. Con esta tabla se demuestra que el algoritmo converge con esas 5 épocas, pues en la quinta época el algoritmo apenas mejora sus pérdidas en ambos conjuntos.

época	1	2	3	4	5
pérdida en training	1.1723	1.0788	1.0388	1.0208	1.0118
pérdida en validación	1.1226	1.0640	1.0388	1.0266	1.0202

Tabla 15. Pérdidas de entrenamiento y validación según la época en el algoritmo VAE

Como ha podido observarse, VAE es sin duda el algoritmo que más hiperparámetros tiene de todos los utilizados. La configuración con los elegidos finalmente se resume en la Tabla 16.

Parámetro	Valor
gamma	1
capacity	0
random_state	42
optimizer	'adam'
preprocessing	True
verbose	2
contamination	0.01
validation_size	0.1
dropout_rate	0
l2_regularizer	0
epochs	5
loss	'MSE'
output_activation	'sigmoid'
hidden_activation	'relu'
latent_dim	1
encoder_neurons	[64, 16, 4, 1]
decoder_neurons	[1, 4, 16, 64]
batch_size	512

Tabla 16. Parámetros elegidos para el modelo VAE

Los resultados obtenidos con la configuración de hiperparámetros elegida se presentan en la Tabla 17.

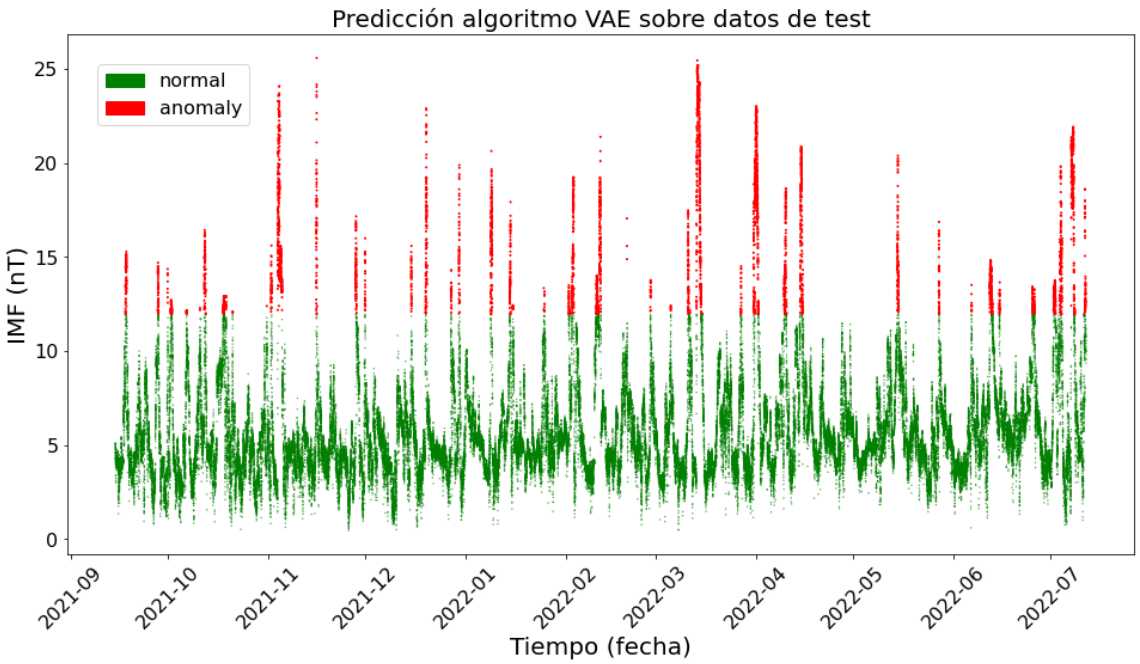
Magnitud	Valor
Número de outliers	5210
Porcentaje de outliers (%)	4.82
Tiempo de entrenamiento (s)	43.74
Tiempo de predicción (s)	10.31
Probabilidad media de outlier (%)	9.18
Probabilidad media de outlier entre los asignados (%)	37.12

Tabla 17. Resultados obtenidos con el algoritmo VAE

Se muestra que el número de *outliers* es el más elevado que se ha obtenido con cualquier algoritmo. Además, los tiempos de entrenamiento y predicción son menores

que en IForest. Por último, se encuentra la gráfica con los *outliers* detectados en la Figura 24. Se observa que identifica mucho *outliers* y todos ellos los encuentra en la parte superior de la serie temporal, sin ubicar ninguno en la zona inferior.

Figura 24. Resultados obtenidos con el algoritmo VAE sobre el conjunto de test



4.1.6. Large-scale Unsupervised Outlier Detection

Como se ha mencionado, este es un método *ensemble* que permite obtener una predicción combinando las de otros algoritmos. A pesar de tener muchos hiperparámetros, solo se van a analizar los mencionados en el apartado 3.3.6. Como se pretende utilizar el método para aunar los algoritmos ECOD, IForest y VAE y obtener una predicción combinándolos, el parámetro **base_estimators** tiene como valor asignado una **lista con estos tres algoritmos**, con la configuración de hiperparámetros elegida para cada uno de ellos y expuesta en la Tabla 12 y Tabla 16 (ECOD no tiene hiperparámetros).

Por otro lado, el parámetro **verbose** se asigna **True** para tener información sobre el proceso de entrenamiento. Además, el parámetro **contamination** se mantiene lógicamente con el valor utilizado de **0.01** en cada uno de los tres algoritmos y estimado a través del método sencillo OOL. Finalmente, para el parámetro **combination** se elige el valor **'average'** para combinar los algoritmos a través de la media. La otra opción es que todos los datos que hayan sido detectados como *outliers* por alguno de los

algoritmos sean asignados como tal por SUOD, pero en este caso no interesa por ser la opción de la media más general y tener en cuenta la participación de todos los algoritmos.

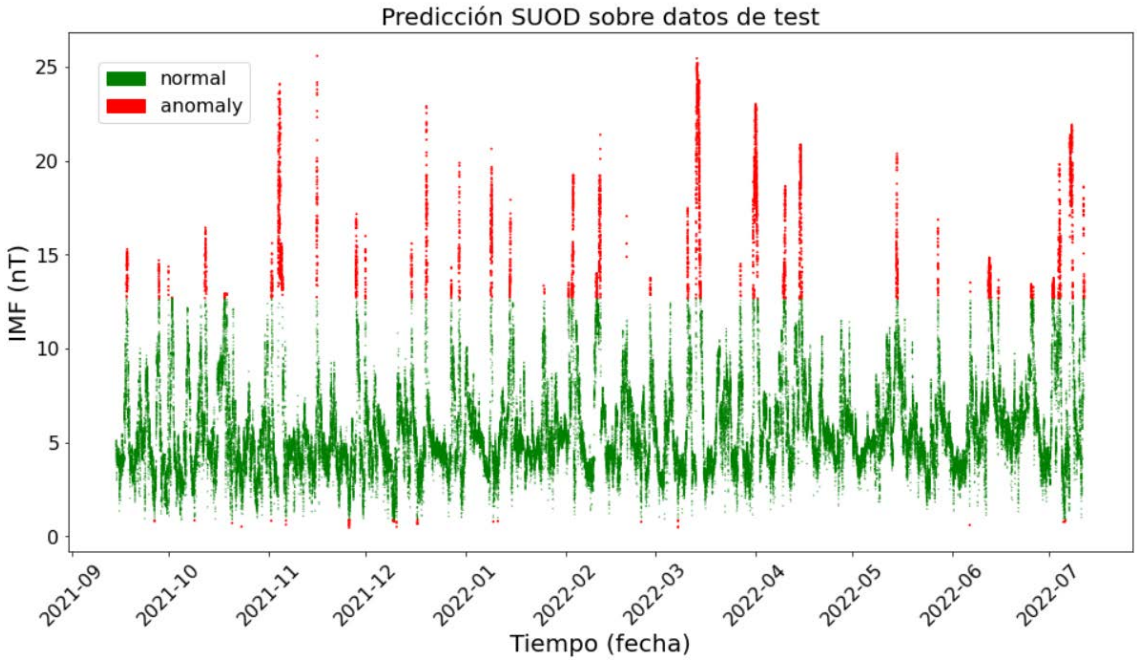
Con la configuración de hiperparámetros expuesta anteriormente, los resultados obtenidos son los que se muestran en la Tabla 18.

Magnitud	Valor
Número de outliers	4241
Porcentaje de outliers (%)	3.92
Tiempo de entrenamiento (s)	164.89
Tiempo de predicción (s)	21.65
Probabilidad media de outlier (%)	9.17
Probabilidad media de outlier entre los asignados (%)	39.41

Tabla 18. Resultados obtenidos con el algoritmo ensemble SUOD

Se observa que el número de *outliers* obtenido es inferior al de VAE e IForest y superior al de ECOD. De hecho, está muy cerca de la media de anomalías obtenidas con los tres métodos. Sin embargo, el tiempo de entrenamiento y predicción es superior al de cualquiera de los algoritmos. Esto es completamente lógico, pues SUOD debe entrenar los tres métodos y predecir con ellos para luego combinarlos en una única predicción. De hecho, tanto el tiempo de entrenamiento como el tiempo de predicción son cercanos a la suma de esos mismos tiempos para los tres algoritmos. Finalmente, en la Figura 25. Resultados con el algoritmo ensemble SUOD sobre el conjunto de test se observa la gráfica obtenida con los *outliers* detectados con el método *ensemble* SUOD.

Figura 25. Resultados con el algoritmo ensemble SUOD sobre el conjunto de test



4.2. Stream machine learning

Se va a implementar el algoritmo IForest con técnicas de SML, simulando que los datos llegan en tiempo real. Se va a utilizar la librería PySAD, con la que se comprobarán las diferencias entre el entrenamiento normal y el entrenamiento continuo propio del *online machine learning*. Para ello, se adelantó en el apartado 3.4 que las clases a utilizar serían **PandasStreamer** y **ReferenceWindowModel**. Con el método **iter** de la primera clase se crea un bucle sobre el que fluyen los datos, es decir, un *dataflow*. La segunda clase sí tiene métodos que cuentan con parámetros más complejos y ha de elegirse un valor para ellos.

Los parámetros iniciales al definir un objeto de la clase **ReferenceWindowModel** que se utilizan en este trabajo son:

- **model_cls**: es el modelo de detección de anomalías de PyOD [19] que se va a utilizar, en este caso **IForest**.
- **initial_window_X**: ventana inicial de entrenamiento para tener una base con la que aplicar el algoritmo en los primeros datos del *dataflow*. En este caso se ha decidido detectar anomalías a partir del dato 50000 del *dataset* de test, para

aplicar SML sobre una parte (algo más de la mitad) de este *dataset*. Así, se decide que la primera ventana de entrenamiento sean los 10000 datos anteriores a ellos, es decir, **desde el dato 40000 hasta el dato 50000 del dataset de test**.

- **window_size**: tamaño de la ventana de datos con la que se entrena. En este caso, se decide entrenar con **10000** instancias en cada ventana. El *dataset* sobre el que se va a aplicar SML tiene 58144 instancias, pues es desde el dato 50000 hasta el último del *dataset* de test. Elegir el valor de 10000 para el tamaño de las ventanas de entrenamiento permite entrenar con un tamaño considerable de datos para que los *outliers* existentes no afecten demasiado al proceso de entrenamiento.
- **sliding_size**: número de datos tras el que se resetea el modelo para entrenarlo con una nueva ventana del tamaño indicado por window_size. Se elige resetear el modelo para volver a entrenarlo cada **5000** datos. Esto hace que el entrenamiento sea continuo y cada 5000 datos se resetee el modelo, por lo que esto ocurrirá más de 10 veces en el proceso de entrenamiento.

Con todo esto, se obtienen las puntuaciones de anomalía de cada uno de los datos presentes en el rango mencionado del *dataset* de test. Se debe establecer un valor de umbral para distinguir anomalías de datos normales según esas puntuaciones. Se prueba con diferentes valores para evaluar el comportamiento del SML y encontrar un valor adecuado. En la Figura 26 se observa la detección realizada implementando IForest con SML sobre la parte mencionada del *dataset* de test con valor del umbral en 0. En la Figura 27 el umbral es 0.1, en la Figura 28 es 0.15 y finalmente en la Figura 29 el umbral es 0.2.

Figura 26. Detección de anomalías mediante SML con algoritmo IForest y threshold 0

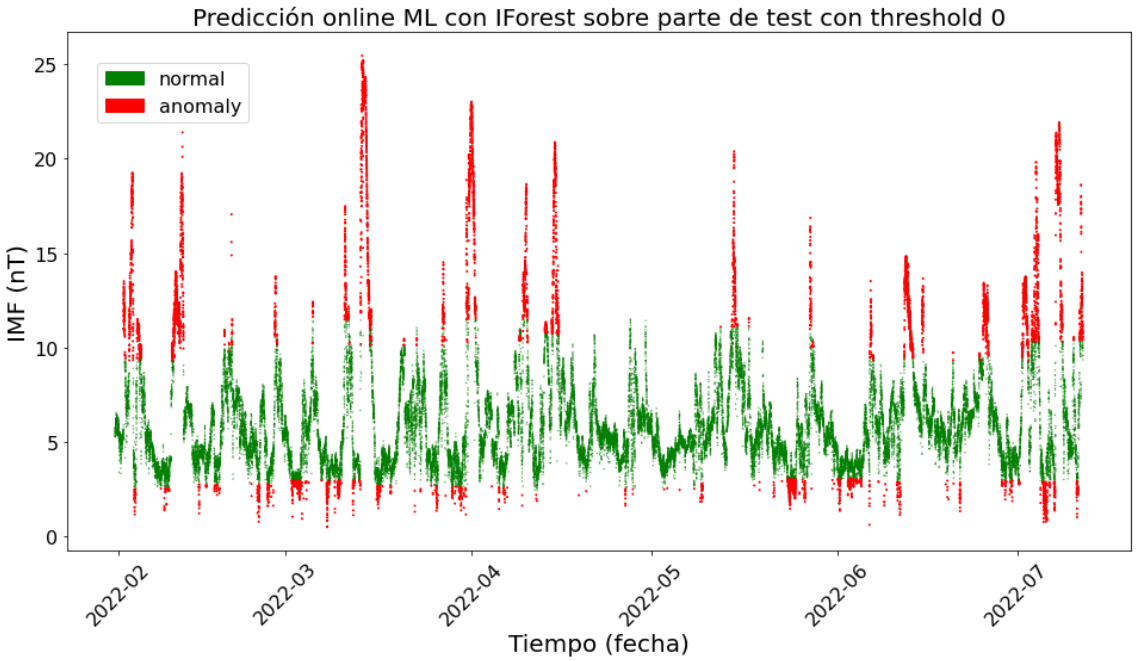


Figura 27. Detección de anomalías mediante SML con algoritmo IForest y threshold 0.10

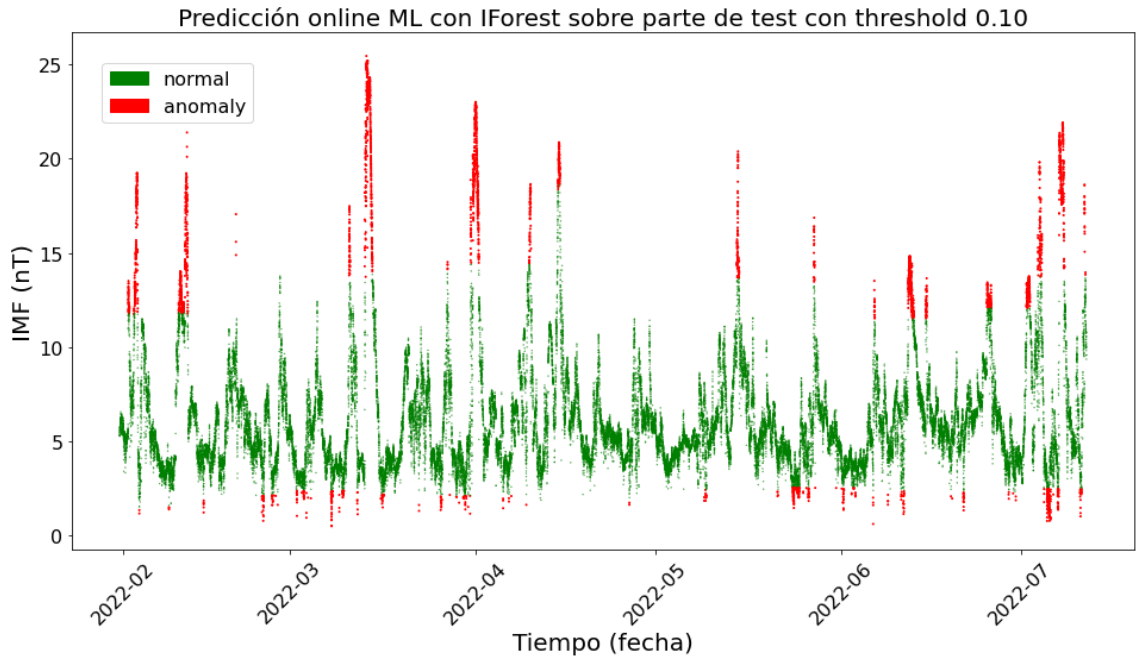


Figura 28. Detección de anomalías mediante SML con algoritmo IForest y threshold 0.15

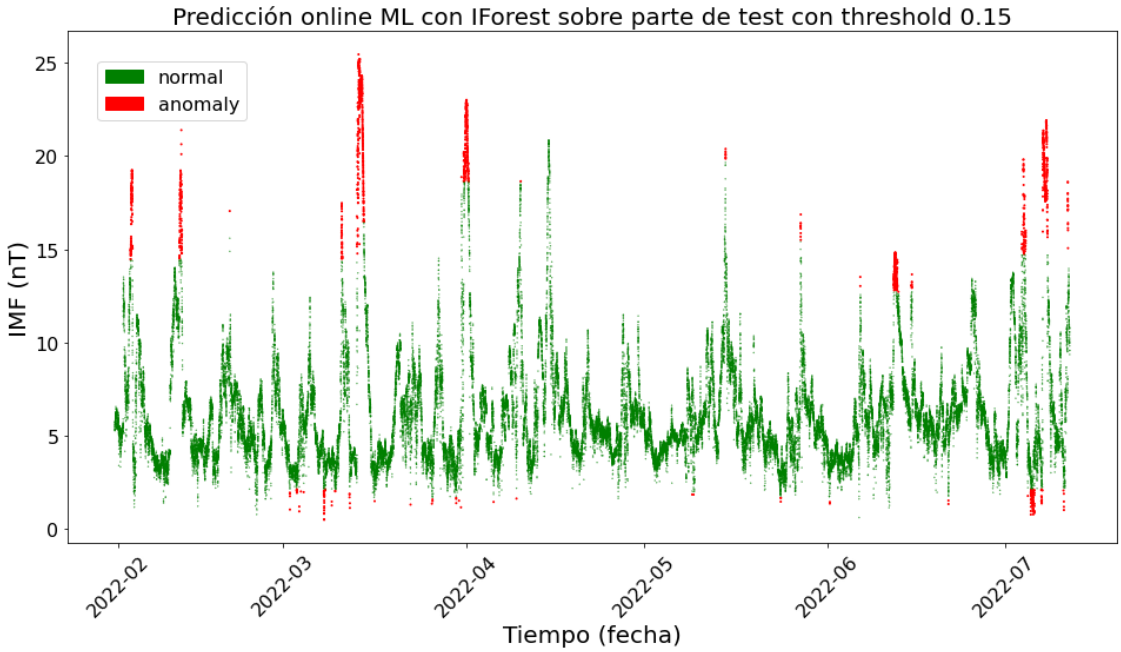
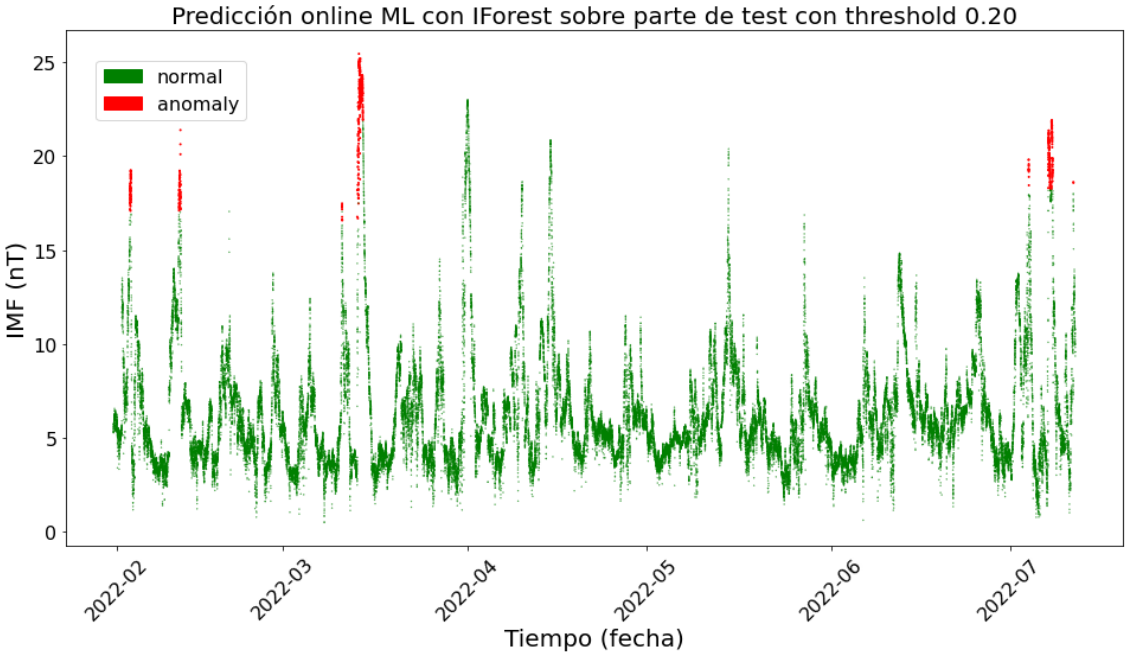


Figura 29. Detección de anomalías mediante SML con algoritmo IForest y threshold 0.20



Lo primero que se puede observar del entrenamiento con SML es que las fronteras de para distinguir *outliers* de valores normales ya no son fijas a lo largo de todo el conjunto de test como ocurría con el entrenamiento habitual en cualquier algoritmo del apartado 4.1. Esto se debe a que el entrenamiento tiene lugar con diferentes ventanas de datos

y experimenta una evolución continua, haciendo que no exista una única frontera. En cuanto a la elección del umbral, esta se realiza en el apartado 4.3.

Con todo lo anterior, se han comprobado las diferencias que existen entre el entrenamiento estático y el dinámico o continuo que caracteriza al SML. Su aplicación sobre el *dataset* ha demostrado establecer unas fronteras dinámicas y una adaptabilidad a los cambios que se puedan producir. Es un método necesario cuando los datos llegan en tiempo real e ideal para detectar anomalías en series temporales que no sean estacionarias, pues ajustando la ventana de entrenamiento se podrá construir un modelo que se adapte a los cambios de tendencia o periodicidad que ocurran en el conjunto de datos. En el siguiente apartado se comparan los algoritmos utilizados y los resultados obtenidos bajo un entorno estático o típico del *machine learning* y se elige el mejor umbral para el SML visto en este apartado.

4.3. Análisis y comparativa de modelos

En este apartado se va a comparar el rendimiento y resultados obtenidos por los tres algoritmos que se han utilizado para detectar anomalías: VAE, IForest y ECOD. También se va a incluir el método *ensemble* SUOD, que obtiene una predicción combinando las de los tres algoritmos mencionados. Se deja a un lado el comportamiento de LOF, pues se ha utilizado con el fin de la detección temprana o *novelty detection* y no tiene sentido compararlo con el resto. Además, sus resultados se expusieron con detalle en el apartado 4.1.2.

Para empezar, es importante visualizar las gráficas de resultados de los tres algoritmos y del método *ensemble*, que son Figura 22, Figura 23, Figura 24 y Figura 25. Se observa que VAE no identifica *outliers* en la zona inferior de la serie temporal, mientras que ECOD y IForest sí lo hacen. Por ello, SUOD muestra también algunos *outliers* en esa zona. En el caso de este *dataset*, tanto por la distribución de los datos y principalmente por el concepto que representan, no hay lugar para *outliers* en la zona inferior. Sin embargo, apenas se identifican *outliers* en esa zona con IForest y ECOD, por lo que no es una diferencia muy importante. En caso de no querer mostrar ningún *outlier* en esta zona, se podría añadir un filtro para ello.

Por otro lado, se observa que los tres métodos establecen una frontera fija para distinguir *outliers*, tanto una superior como una inferior para los dos que identifican *outliers* en esta zona. Se observa que el límite superior en VAE es el más bajo, por lo que identifica más *outliers* que los otros dos. Le sigue cerca IForest y ECOD establece un límite bastante por encima, identificando así mucho menos *outliers*. El método *ensemble* SUOD también establece estas fronteras fijas, situándose la superior solamente por debajo de la de ECOD y la inferior por debajo de ECOD e IForest. En la Tabla 19 se recogen los límites establecidos por cada uno de los modelos, tanto superior como inferior en el caso de haberlo.

	ECOD	IForest	VAE	SUOD
Límite superior (nT)	15.94	12.57	11.96	12.67
Límite inferior (nT)	1.21	0.97	-	0.88

Tabla 19. Límites superior e inferior establecidos por cada algoritmo

Esos límites marcan el número de *outliers* identificado por cada algoritmo. Utilizando la definición de *outlier* con el método OOL se identificó que el *dataset* de training contaba aproximadamente con un 1.4% de anomalías. Con ello, se estableció en 1% el porcentaje de anomalías para entrenar con ello los algoritmos. Sin embargo, todos ellos superan este valor. Los valores concretos sobre *outliers* superiores, inferiores y totales con cada algoritmo se recogen en la Tabla 20, así como el porcentaje sobre las 108144 instancias de test.

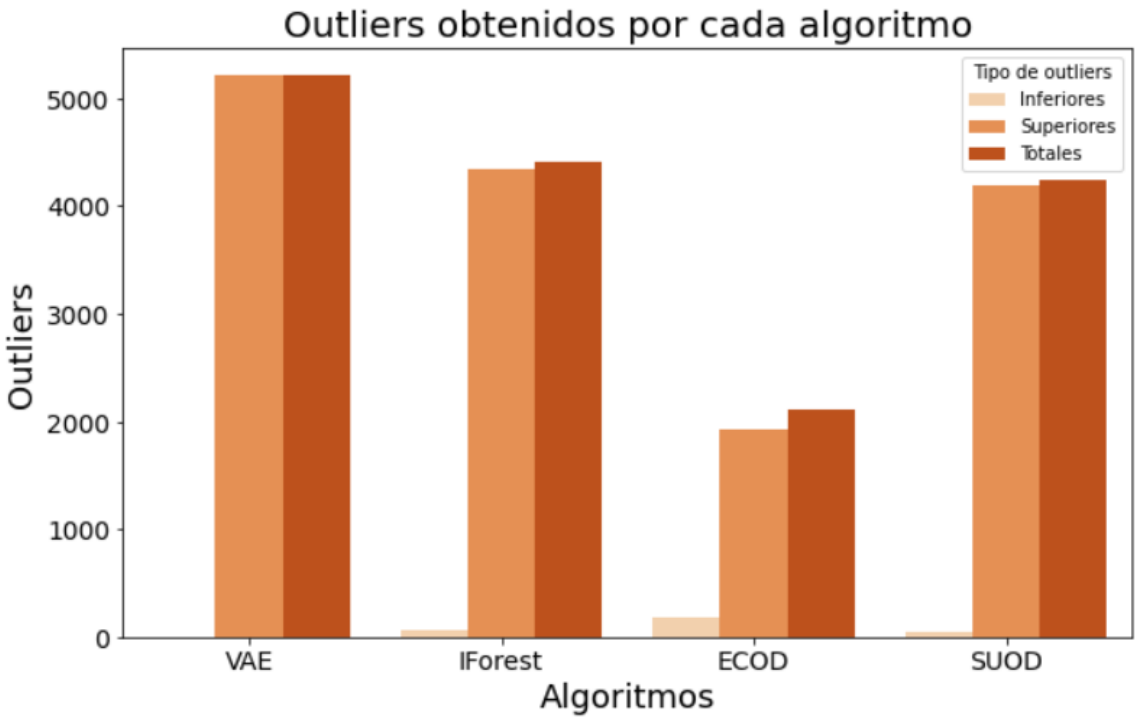
	ECOD	IForest	VAE	SUOD
Outliers superiores	1922	4334	5210	4197
Outliers inferiores	180	71	0	44
Outliers totales	2102	4405	5210	4241
Porcentaje de outliers (%)	1.94	4.07	4.82	3.92

Tabla 20. Outliers superiores, inferiores, totales y porcentaje de ellos por cada algoritmo

Se encuentra que ECOD es el algoritmo que menos *outliers* identifica con mucha diferencia respecto a los otros dos, que tienen más del doble. Por ello, se considera que es el algoritmo que más se acerca a la situación real según la definición más extendida de *outlier* y, por tanto, el que mejor comportamiento tiene. Además, al ser un *dataset* de la evolución temporal del IMF, no interesa un número elevado de falsos positivos, por lo que puede resultar más adecuado la utilización de ECOD. Sin embargo, el hecho de no

tener etiquetas impide identificar un resultado correcto puesto que no hay un valor de verdad ni una métrica a evaluar. Por ello, si se considera adecuado tener un número grande de alertas para analizar o se quiere establecer un umbral que dé lugar a más *outliers*, utilizar IForest o VAE será una mejor opción. El hecho de que ECOD identifique menos de la mitad de valores anómalos que los otros dos, hace que el método *ensemble* SUOD baje su número de *outliers* identificados por debajo de IForest y VAE. Finalmente, se muestran en un gráfico de barras los *outliers* identificados por cada algoritmo, distinguiendo entre inferiores, superiores y totales que es la suma de ambos. Este gráfico se encuentra en la Figura 30 y en él se observa de manera visual y rápida todo lo comentado recientemente.

Figura 30. Gráfico de barras con los outliers obtenidos por cada algoritmo



Además, para hacer esta comparativa entre algoritmos mucho más concreta, se puede suponer que los valores correctos o los valores de verdad son los que aporta el método OOL a través de la definición de *outlier*. Esto es simplemente una suposición que permite comparar los algoritmos entre sí con métricas de clasificación, puesto que no se tienen unas etiquetas reales, si no se entrenaría con ellas y el problema estaría en un entorno supervisado. De esta manera, se puede obtener primeramente la matriz de confusión entre el supuesto valor real obtenido con OOL y cada una de las detecciones realizadas por los algoritmos. La matriz de confusión de cada uno de los algoritmos se

encuentra en la Figura 31 para el VAE e IForest y Figura 32 para ECOD y el algoritmo *ensemble* SUOD. En ellas se puede observar en ambas etiquetas (*outlier* o no) las predicciones respecto a los valores reales.

Sobre las matrices de confusión se observa que ECOD es el algoritmo que menos falsos positivos arroja (valores que detecta como *outliers* pero no lo son, según OOL) con bastante diferencia al resto. De hecho, tan solo obtiene 180 falsos positivos mientras que IForest detecta 2255 y VAE llega a 3060. Este es un dato muy importante en la detección de anomalías y que pone en valor los resultados de ECOD respecto al resto.

Figura 31. Matrices de confusión de VAE e IForest con etiquetas de OOL

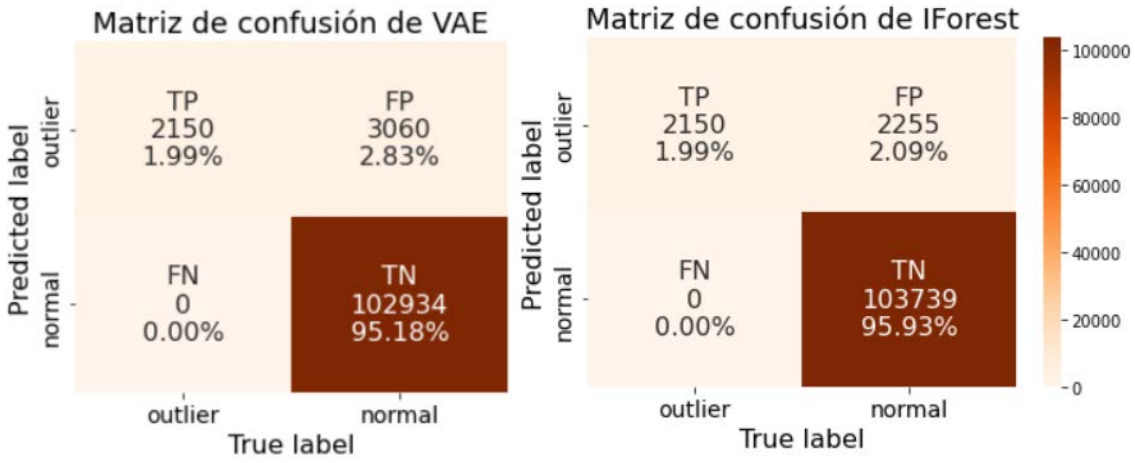
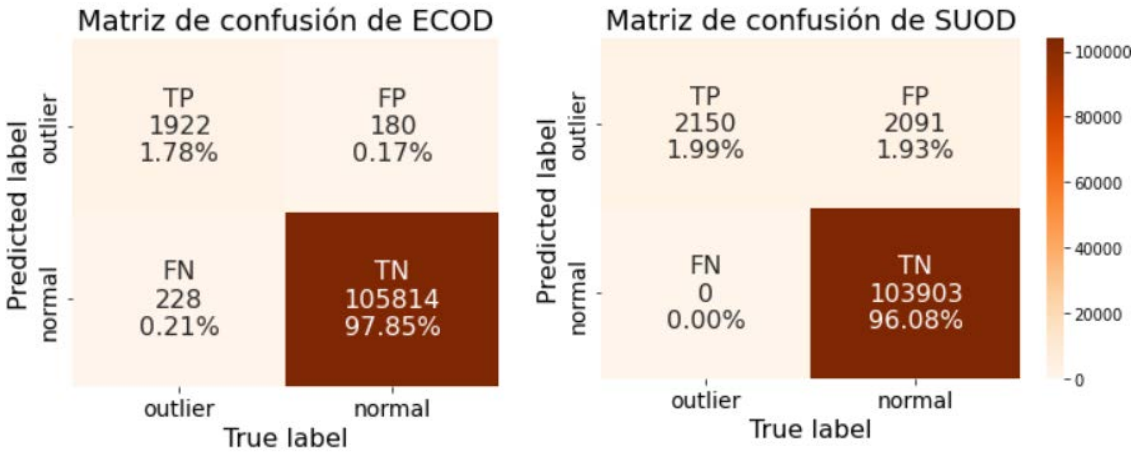


Figura 32. Matrices de confusión de ECOD y SUOD con etiquetas de OOL



Las métricas de clasificación que se van a utilizar en este trabajo se encuentran en el Anexo 1. Siempre que se mencione alguna de ellas, se acompaña de su expresión correspondiente en el anexo. De las métricas que surgen de la matriz de confusión, se van a tener en cuenta tanto *accuracy* (1), *precision* (2), *recall* (3) y F1 (4). Sin embargo, la que tiene mayor importancia en este problema es *precision* (2) si la etiqueta *outlier* es la principal o positiva. Esto es porque con esa métrica se valoran los falsos positivos, que son el error más importante a reducir para no mostrar demasiadas alertas sobre anomalías que en realidad no lo son, pues supondría una gran pérdida de tiempo y esfuerzo investigar sus motivos.

Además, se debe tener en cuenta que el *dataset* está muy lejos de estar balanceado, pues gran parte de los datos son valores normales y solo unos pocos *outliers*. Por ello, la *accuracy* (1) siempre tenderá a ser elevada, ya que simplemente un modelo que diga que todos los valores son normales tendría una *accuracy* (1) de más del 98%.

En el caso de la detección de anomalías, tiene más sentido utilizar la etiqueta *outlier* (valor 1) como la principal, pero también puede hacerse con la etiqueta *inlier* (valor 0). Utilizando ambas formas, se construye el *classification report* de cada algoritmo. Para comprender todos los parámetros que contiene esta tabla, se recomienda consultar el Anexo 1.

Las tablas para cada algoritmo son la Tabla 21para el VAE, Tabla 22 para el IForest, Tabla 23 para el ECOD y Tabla 24 para el algoritmo *ensemble* SUOD. Para el *dataset* de 108144 instancias, según OOL con la definición de *outlier* existen 105994 valores normales y 2150 valores anómalos, siendo estos los que se toman como valores reales o verdaderos bajo la suposición realizada. En las tablas, se utiliza una gama de colores que identifica para cada métrica el orden de los algoritmos respecto a ella. Así, los tonos más oscuros representan aquellas métricas donde el algoritmo es el que mejor resultado obtiene mientras que los tonos más claros representan lo contrario. A continuación, se muestra la gama de colores y su significado en una leyenda.

Nombres (no clasificables)	Cuarto	Tercero	Segundo	Primero

Classification report VAE				
accuracy 0.972	**precision**	**recall**	**F1**	**cantidad de datos**
0 (inlier)	1.000	0.971	0.985	105994
1 (outlier)	0.413	1.000	0.584	2150
macro average	0.706	0.986	0.785	108144
weighted average	0.988	0.972	0.977	108144

Tabla 21. Classification report para el algoritmo VAE respecto a las etiquetas de OOL

Classification report IForest				
accuracy 0.979	**precision**	**recall**	**F1**	**cantidad de datos**
0 (inlier)	1.000	0.979	0.989	105994
1 (outlier)	0.488	1.000	0.656	2150
macro average	0.744	0.989	0.823	108144
weighted average	0.990	0.979	0.983	108144

Tabla 22. Classification report para el algoritmo IForest respecto a las etiquetas de OOL

Classification report ECOD				
accuracy 0.996	**precision**	**recall**	**F1**	**cantidad de datos**
0 (inlier)	0.998	0.998	0.998	105994
1 (outlier)	0.914	0.894	0.904	2150
macro average	0.956	0.946	0.951	108144
weighted average	0.996	0.996	0.996	108144

Tabla 23. Classification report para el algoritmo ECOD respecto a las etiquetas de OOL

Classification report SUOD				
accuracy 0.981	**precision**	**recall**	**F1**	**cantidad de datos**
0 (inlier)	1.000	0.980	0.990	105994
1 (outlier)	0.507	1.000	0.673	2150
macro average	0.753	0.990	0.831	108144
weighted average	0.990	0.981	0.984	108144

Tabla 24. Classification report para el algoritmo ensemble SUOD respecto a OOL

Con la gama de colores sobre las tablas con las métricas de clasificación descritas para cada algoritmo, queda bastante manifiesta la superioridad de ECOD frente a VAE e IForest. De las 13 métricas presentes en cada tabla, ECOD es el que tiene mejor resultado en 10 de ellas. Además, en algunas muestra una gran diferencia respecto al resto, como en la *precision* (2) o el valor F1 (4) con la etiqueta 1 (*outlier*) como positiva. Concretamente, *precision* (2) es fundamental porque reflejan el bajo número de falsos positivos de ECOD comparado con el resto. Respecto a aquellas 3 en las que tiene peor resultado, la diferencia es mínima y la explicación muy lógica: ECOD solo obtiene un 0.21% de falsos negativos pero VAE, IForest y SUOD no cuentan con ningún falso negativo, siendo la etiqueta positiva *outlier*, pues no hay ningún *outlier* que se les escape. Sin embargo, identifican muchos más de los que hay. Un algoritmo que detectara todos los datos como *outliers* también superaría a ECOD en estas métricas y es obvio que no es una mejor solución.

Por otro lado, es importante observar que SUOD es en todas las métricas el mejor o el segundo mejor resultado. Esto quiere decir que, en caso de no tener claro exactamente qué algoritmo utilizar o aproximadamente cuántos *outliers* se quieren detectar, un método *ensemble* como SUOD es muy buena opción. De hecho, la primera exploración de *outliers* en el *dataset* debería hacerse con un método sencillo como OOL o un método *ensemble*, para luego ir al algoritmo particular que mejor funcione cuando se tenga mayor conocimiento sobre los datos y el objetivo de detectar anomalías sobre ellos.

En definitiva, se concluye que ECOD tiene un mejor rendimiento que el resto de algoritmos si se pretende que los resultados se acerquen a la definición estadística de *outliers*. Esto también puede deberse a la naturaleza probabilística del método.

Sobre los resultados obtenidos con los diferentes umbrales aplicando IForest con SML, se va a realizar la elección del mejor umbral en base a las métricas de mayor relevancia para los errores de falsos positivos y falsos negativos, siendo más importante evitar los falsos positivos. Por ello, se utilizan *recall* (3), *precision* (2) y F1 (4), dando mayor importancia a las dos últimas. En la Tabla 25 se muestran los valores de estas métricas para cada uno de los umbrales 0, 0.1, 0.15 y 0.2 con el mismo código de colores anterior para identificar con facilidad el mejor umbral.

Umbral	0	0.10	0.15	0.20
recall	1.000	0.966	0.761	0.409
precision	0.215	0.452	0.719	1.000
F1	0.354	0.616	0.739	0.580

Tabla 25. Métricas de clasificación para los diferentes umbrales en IForest con SML

Se observa que el umbral 0.15 es el valor más compensado y, por tanto, más adecuado. Presenta el mayor valor de F1, por lo que es la mejor combinación de *precision* (2) y *recall* (3) que hay. De hecho, tiene el segundo mejor valor en *precision* (2) porque el umbral 0.20 apenas detecta *outliers* (como se puede ver en la Figura 29) y por ello no tiene falsos positivos, siendo su *precision* (2) de 1. Por tanto, el umbral elegido es 0.15 y su gráfica puede consultarse en la Figura 28. El número de *outliers* y porcentaje de los mismos, sobre las 58144 instancias en las que se aplica SML con el algoritmo IForest, se pueden consultar en la Tabla 26.

Magnitud	Valor
Número de outliers	1709
Porcentaje de outliers (%)	2.94

Tabla 26. Resultados con SML y el algoritmo IForest con threshold 0.15 elegido

En lo que respecta a los tiempos de entrenamiento y predicción, los resultados no son tan interpretables como el número de *outliers*. En este caso, la objetividad es muy clara y menores tiempos de entrenamiento y predicción serán preferibles siempre. En la Tabla 27 se recogen los tiempos de entrenamiento y predicción de los tres algoritmos y el *ensemble* SUOD, así como el tiempo total que es la suma de ambos. Se recuerda que el *dataset* de *train* cuenta con 432576 instancias mientras que el de test tiene 108144.

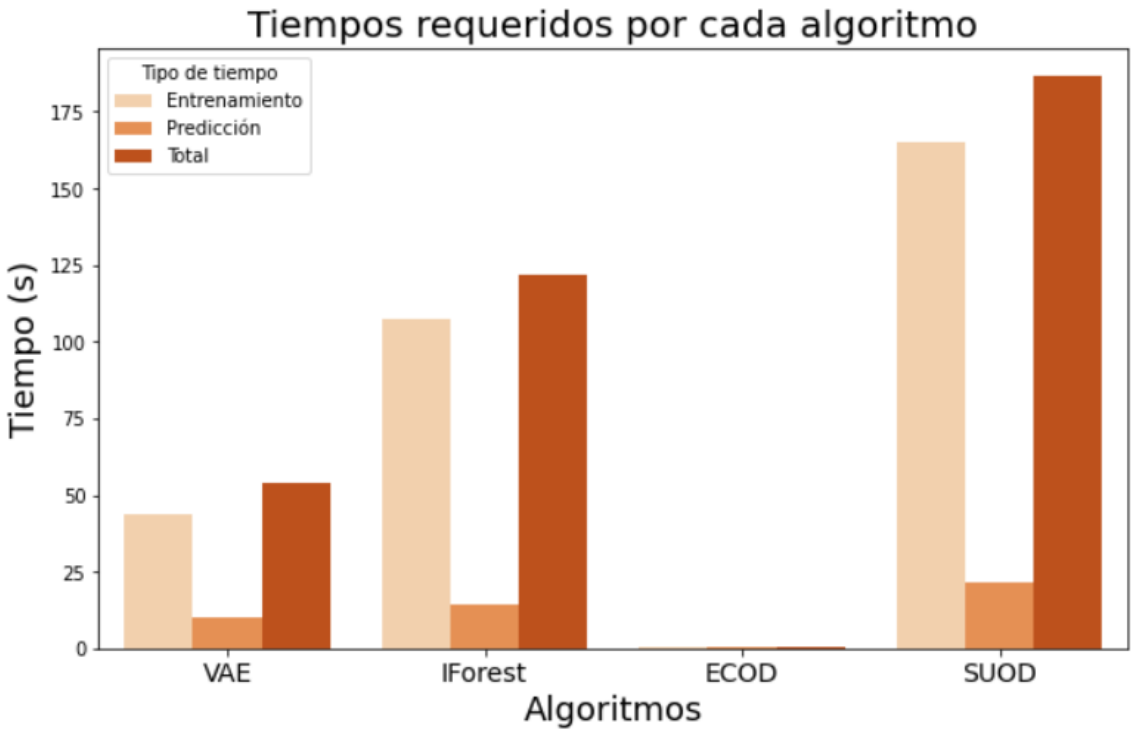
	ECOD	IForest	VAE	SUOD
Tiempo de entrenamiento (s)	0.34	107.52	43.74	164.89
Tiempo de predicción (s)	0.43	14.16	10.31	21.65
Tiempo total (s)	0.77	121.68	54.05	186.54

Tabla 27. Tiempos de entrenamiento y predicción de cada algoritmo

En la tabla queda bastante clara la superioridad de ECOD respecto a IForest y VAE. En menos de un segundo ECOD es capaz de entrenar sobre un conjunto de más de 400000

datos y predecir sobre uno de más de 100000. Para la misma tarea, el VAE con la configuración elegida necesita aproximadamente 70 veces más tiempo e IForest unas 160 veces. La diferencia es muy notoria y puede ser fundamental en un *dataset* mucho más grande o si se quiere implementar la detección en tiempo real. Por ello, siempre que se requiera un entrenamiento o una respuesta rápida, ECOD es el algoritmo más adecuado de los tres. En cuanto a SUOD, los datos son muy lógicos y fáciles de interpretar, pues el *ensemble* requiere entrenar los tres algoritmos para completar su proceso de entrenamiento y predecir con los tres para obtener su predicción final. Así, sus tiempos son cercanos a la suma de los tiempos de los tres algoritmos. Todos estos datos de tiempos se representan en la Figura 33, donde la diferencia se hace muy visual por medio de un gráfico de barras.

Figura 33. Gráfico de barras con tiempos (entrenamiento y predicción) de cada algoritmo



Con todo lo expuesto en esta comparativa, se concluye que ECOD es la mejor opción en la mayoría de los casos para detectar anomalías de manera no supervisada en una serie temporal estacionaria. Primeramente, por su gran facilidad de uso debida a la ausencia de hiperparámetros. Solamente con el porcentaje de anomalías sobre el *dataset* el algoritmo es capaz de realizar el proceso de entrenamiento. Esto evita la ardua tarea de tunear hiperparámetros en un modelo no supervisado, más compleja si cabe que la de hacerlo con etiquetas. Otra ventaja muy clara que presenta son sus bajos

tiempos de entrenamiento y predicción. La diferencia con los otros dos es enorme en este aspecto, que puede ser fundamental en muchas aplicaciones o *datasets*. Por último, el número de *outliers* identificado por ECOD es bastante menor que en IForest y VAE, por lo que puede ser más adecuado siempre que se quiera identificar menos *outliers* o tener un número reducido de falsas anomalías. Sin embargo, esto es relativo al no haber etiquetas o valores correctos. Por ello, dependerá de la aplicación concreta que en este aspecto de los *outliers* detectados sea mejor uno u otro, como sí era muy objetivo y manifiesto en los tiempos de entrenamiento y tuneo de hiperparámetros. En el caso de no tener claro que método utilizar o qué objetivo concreto se tiene con la detección de anomalías en el *dataset*, utilizar el método *ensemble* SUOD es ideal para obtener una detección más general que combine la detección realizada por varios algoritmos como ECOD, IForest, VAE o cualquier otro que se desee incluir.

5. Conclusiones y trabajos futuros

El *dataset* utilizado ha sido relativamente difícil de encontrar dada la poca información disponible para acceder a estos conjuntos de datos telemétricos. Sin embargo, el acceso a través de la Heliophysics API con la librería de Python es muy sencillo. Además, el *dataset* elegido se presta a una detección de anomalías, pero se han explorado muchos otros también de la *Space Physics Data Facility* de la NASA y la mayoría no son adecuados para la tarea. Por ello, no es una tarea sencilla encontrar otro *dataset* como este o más adecuado para detectar *outliers* sobre él.

El primer método utilizado ha sido la definición estadística de *outliers* con los cuartiles, que es un método muy adecuado para conocer la dispersión del *dataset*, el porcentaje de *outliers* que contiene y una primera exploración de los mismos. Además, se demuestra que los algoritmos utilizados posteriormente obtienen resultados similares a este método por lo que la definición estadística de *outliers* puede tomarse como referencia y un primer resultado sencillo.

Se han utilizado tres algoritmos de *machine learning* de detección de anomalías (VAE, IForest y ECOD) y un método *ensemble* que obtiene una salida combinando la detección realizada por los algoritmos mencionados. En cuanto a los algoritmos, se demuestra que ECOD ha demostrado un rendimiento muy superior a VAE e IForest por tres motivos: ausencia de hiperparámetros, menores tiempos de entrenamiento y precisión y mejores métricas comparando con las etiquetas obtenidas con el método OOL ya comentado. Además, se ha demostrado que el método *ensemble* SUOD es una muy buena solución para combinar varios algoritmos y obtener una solución genérica y robusta que los pondere. De hecho, en todas las métricas que se comparan el modelo *ensemble* es el mejor o el segundo mejor de los cuatro.

En cuanto a la implementación de *online machine learning*, se ha mostrado con ella de manera sencilla las diferencias que tiene respecto al aprendizaje automático clásico. En primer lugar, el entrenamiento por ventanas permite al modelo tener más dinamismo y

adaptarse a los cambios que se producen en los datos con el paso del tiempo. Así, no existe una frontera fija que distinga *outliers* de *inliers* en la implementación del IForest con SML, mientras que esta sí existía al implementar cualquier algoritmo bajo el enfoque clásico. Además, la llegada de datos puede ser heterogénea y en tiempo real.

En definitiva, se han aplicado varias técnicas y algoritmos y se ha comparado sus rendimientos sobre el *dataset* de la misión *Advanced Composition Explorer* (ACE) dirigida por la NASA. Se espera que las conclusiones aportadas sean de utilidad para resumir lo realizado. Finalmente, se indican posibles trabajos futuros a realizar basándose en este TFM:

- Incorporar el *dataset* de la misión *Deep Space Climate Observatory* (DSCOVR). Se trata de un satélite que orbita, al igual que ACE, alrededor de L1. Sin embargo, es un satélite más moderno y la principal fuente de datos en tiempo real sobre el clima y campo magnético espaciales desde que en 2016 alcanzó su fase operacional. De hecho, ACE solamente actúa como soporte de DSCOVR desde ese momento.
- Llegada de datos en tiempo real, tal y como los tome el satélite, para el uso del SML con nuevas características y el análisis de las anomalías y sus motivos en el momento en que ocurren.
- Utilizar ECOD como algoritmo para el *online machine learning*, ya que se ha demostrado que presenta el mejor rendimiento y es el más rápido, algo que es fundamental en SML. Al ser un algoritmo tan reciente, no se ha encontrado una librería que permita implementarlo con SML, pero dado su rendimiento debería estar disponible en poco tiempo.
- Probar nuevos algoritmos de detección de anomalías de PyOD [19] u otras herramientas, buscando mejorar las prestaciones de ECOD para el problema.
- Extender el uso a otros conjuntos de datos telemétricos para monitorizar datos de otras misiones que investiguen magnitudes diferentes en el entorno espacial, comprobando así la validez de ECOD y las demás técnicas propuestas. En el estado del arte del capítulo 2 de este documento se exponen procesos y técnicas adecuadas en el caso de que las series temporales no fuesen estacionarias, que también es una vía interesante de explorar.

6. Referencias

[1] European Space Agency. (31 de marzo de 2022). *Artificial intelligence in space.* https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/Artificial_intelligence_in_space

[2] National Aeronautics and Space Administration. *MARS Exploration Rovers*. https://mars.nasa.gov/mer/

[3] Huyen, C. (2022). *Real-time machine learning: challenges and solutions.* https://huyenchip.com/2022/01/02/real-time-machine-learning-challenges-and-solutions.html

[4] Redpanda - Streaming Data Platform - https://redpanda.com/

[5] Holoviews – Data Visualization - https://holoviews.org/

[6] Bytewax – Modern Stream Processing - https://www.bytewax.io/

[7] River – Python package for online ML - https://riverml.xyz/0.13.0/

[8] Kafka-python – Python client for Apache Kafka - https://kafka-python.readthedocs.io/en/master/

[9] Ducharlet, K., Travé-Massuyès, L., Lasserre, J., Le Lann, M., Miloudi, Y. (2022). *Leveraging the Christoffel-Darboux Kernel for Online Outlier Detection*. hal-03562614

[10] Waleed Hilal, S., Gadsden, A., Yawney, J. (2022). *Financial Fraud: A Review of Anomaly Detection Techniques and Recent Advances*. Volume 193, ISSN 0957-4174. https://doi.org/10.1016/j.eswa.2021.116429

[11] Cucchi, M., Gruener, C., Petrauskas, L., Steiner, P., Tseng, H., Fischer, A., Penkovsky, B., Matthus, C., Birkholz, P., Kleemann, H., Leo, K. (2021). *Reservoir computing with biocompatible organic electrochemical networks for brain-inspired biosignal classification.* Volume 7, doi: 10.1126/sciadv. abh0693.

[12] Zubia Garea, A. (2021). *Detección de anomalías en redes IoT mediante Stream Machine Learning* [Trabajo Fin de Grado, Universidad del País Vasco].

[13] Gomes, H. M., Read, J., Bifet ,A., Barddal, J. P., y Gama, J. *Machine Learning for Streaming Data: State of the Art, Challenges, and Opportunities*. SIGKDD Explor. Newsl., vol. 21, no. 2, pp. 6–22, 2019, [Online]. Available: https://doi.org/10.1145/3373464.3373470

[14] Martínez Heras, J. (2020). *Detección de anomalías en espacio*. Iartificial.net,

Local Outlier Probabilities. https://www.iartificial.net/deteccion-anomalias-espacio/#Nuestra_solucion_para_la_deteccion_de_anomalias_en_operaciones_espaciales

[15] Jayaswal, V. (2020). *Local Outlier Factor (LOF) — Algorithm for outlier identification.* Towards Data Science. https://towardsdatascience.com/local-outlier-factor-lof-algorithm-for-outlier-identification-8efb887d9843

[16] Bhardwaj et al. (2021). *System and Method for Unsupervised Anomaly Detection*. United States Patent, US 10,956,808 B1

[17] Dickens, C., Meissner, E., Moreno, P. G., & Diethe, T. (2020). *Interpretable Anomaly Detection with Mondrian Pòlya Forests on Data Streams*. arXiv preprint arXiv:2008.01505. https://doi.org/10.48550/arXiv.2008.01505

[18] PyCaret – Python machine learning library for workflows - https://pycaret.org/

[19] PyOD – Python library for detecting outliers objects - https://pyod.readthedocs.io/en/latest/pyod.models.html#

[20] Taburoğlu, S. (2019). *A survey on anomaly detection and diagnosis problem in the space system operation*. Journal of Intelligent Systems: Theory and Applications, 2(1), 13-17.

[21] Rocket AI. (9 de diciembre de 2021). *Darmstadt - Spacecraft Anomaly Detection*. Vídeo Youtube. https://www.youtube.com/watch?v=UcS4JiVGnPY

[22] European Space Agency. (2021). *Novelty Detection. A New Telemetry Monitoring Paradigm.*https://www.esa.int/Enabling_Support/Operations/Novelty_Detection_br_A_New_Telemetry_Monitoring_Paradigm

[23] Laboratory for Atmospheric and Space Physics. *WebTCAD - A web-based application providing insight into mission telemetry.* https://lasp.colorado.edu/home/our-expertise/data-systems/data-tools-technologies/webtcad/

[24] Hundman, K., Constantinou, V., Laporte, C., Colwell, I., & Soderstrom, T. (2018, July). *Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding*. In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 387-395). https://doi.org/10.1145/3219819.3219845.

[25] Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008, December). *Isolation forest*. In 2008 eighth ieee international conference on data mining (pp. 413-422). IEEE.

[26] Kingma, D. P., & Welling, M. (2013). *Auto-encoding variational bayes*. arXiv preprint arXiv:1312.6114.

[27] Rocca, J. (2019). *Understanding Variational Autoencoders (VAEs).* Towards Data Science.https://towardsdatascience.com/understanding-variational-autoencoders

[28] Li, Z., Zhao, Y., Hu, X., Botta, N., Ionescu, C., & Chen, G. (2022). *Ecod: Unsupervised outlier detection using empirical cumulative distribution functions*. IEEE Transactions on Knowledge and Data Engineering.

[29] Zhao, Y., Hu, X., Cheng, C., Wang, C., Wan, C., Wang, W., ... & Akoglu, L. (2021). *SUOD: Accelerating large-scale unsupervised heterogeneous outlier detection*. Proceedings of Machine Learning and Systems, 3, 463-478.

[30] Laboratory for Atmospheric and Space Physics. *Data Products - Disseminating space and atmospheric data*. https://lasp.colorado.edu/home/our-expertise/data-systems/data-products/

[31] International Space Station – telemetry data - http://www.telemetry.space/

[32] Planetary Data System – Data Search - https://pds.jpl.nasa.gov/datasearch

[33] National Space and Aeronautics Administration - Space Physics Data Facility - https://spdf.gsfc.nasa.gov/

[34] Advanced Composition Explorer – Mission details - https://izw1.caltech.edu/ACE/

[35] SpaceWeatherLive – El Campo Magnético Interplanetario - https://www.spaceweatherlive.com/es/ayuda/el-campo-magnetico-interplanetario-imf.html

[36] Helipysichs Application Programming Interface – HAPI Server - https://hapi-server.org/servers/

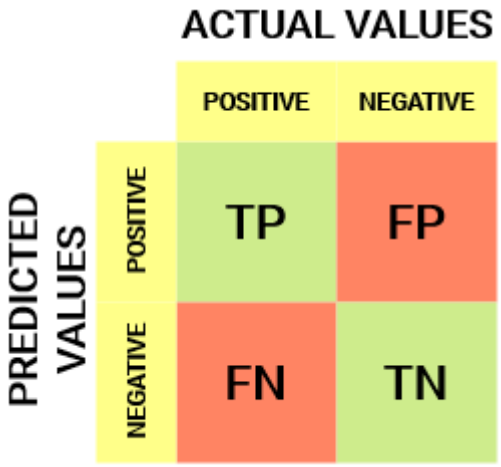
[37] Breunig, M. M., Kriegel, H. P., Ng, R. T., & Sander, J. (2000, May). *LOF: identifying density-based local outliers*. In Proceedings of the 2000 ACM SIGMOD international conference on Management of data (pp. 93-104).

[38] Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980. https://doi.org/10.48550/arXiv.1412.6980

Anexo 1. Métricas

En este anexo se explican las diferentes métricas de clasificación que se utilizan para comparar los diferentes algoritmos con las etiquetas reales, que se suponen por medio del método OOL y la definición estadística de *outlier* con los cuartiles y rango intercuartílico. Las métricas de clasificación son las siguientes:

- **Matriz de confusión**: *confusion matrix* en inglés, es una matriz utilizada para problemas de clasificación, tanto binaria como multiclase. Permite observar en una matriz muy visual las predicciones que ha hecho el algoritmo para cada clase comparadas con el valor real. Así, es fácil visualizar rápidamente si el sistema está confundiendo dos o varias clases. En problemas de clasificación binaria, la matriz de confusión es la siguiente:



  En ella, los aciertos están en verde y los errores en rojo. La traza de la matriz, que es la suma de la diagonal principal, es el número de aciertos totales. Se acierta cuando la predicción coincide con el valor real, sean ambos valores positivos (clase 1) o negativos (clase 0). Un falso positivo es un valor que se predice positivo pero su valor real es negativo, mientras que un falso positivo es un error al contrario.
- **Accuracy**: conocida como exactitud en español, mide el número de aciertos respecto al total de datos. Así, su expresión dados los valores de la matriz de confusión es la (1):

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (1)$$

- **Precision**: en español precisión, es el cociente entre las veces que se predice correctamente la etiqueta positiva y las que se predice una etiqueta positiva, acertando o no. Con los valores de la matriz de confusión, su expresión es (2):

$$precision = \frac{TP}{TP + FP} \quad (2)$$

- **Recall**: en español exhaustividad o sensibilidad, es el cociente entre las veces que se predice correctamente la etiqueta positiva y las veces reales en las que una etiqueta es positiva. Al igual que para las dos anteriores, se muestra en (3) su expresión con los valores de la matriz de confusión.

$$recall = \frac{TP}{TP + FN} \quad (3)$$

- **F1-score**: el valor F1 combina *precision* y *recall* en una única métrica, siendo la media armónica de ellas. Su expresión es la (4):

$$F1\ score = \frac{2 \cdot precision \cdot recall}{precision + recall} \quad (4)$$

A partir de estas métricas, es común en problemas de clasificación obtener un informe conocido como ***classification report***. En él, se organizan todas estas métricas en una tabla, donde en cada una de las filas se toma una etiqueta concreta como la etiqueta positiva alrededor de la que se calculan las métricas de *precision*, *recall* y F1. La *accuracy* no cambia según la etiqueta que se tome como positiva, por lo que aparece una sola vez en la tabla y no en cada fila. Lo interesante de que aporta este informe en un problema de clasificación es el valor de las métricas más comunes de clasificación tomando cada una de las etiquetas como la principal. Esto puede ayudar a distinguir las clases donde el modelo se comporta peor y en qué métrica en concreto.

Además, se suele mostrar también en las dos últimas filas la media aritmética de *precision*, *recall* y F1 de todas las etiquetas, conocida como ***macro average***, junto con la media ponderada de las tres métricas mencionadas de todas las etiquetas. Esta última media tiene en cuenta el número de datos que hay de cada etiqueta y se conoce en inglés como ***weighted average***.