Universitat Oberta de Catalunya

uoc.edu

Detección de eventos anómalos en un entorno industrial mediante el uso de técnicas de Federated Learning.

Darío Martín García Carretero
Máster Universitario en Ciencia de Datos
Área 2

Raúl Parada Medina
Jordi Casas Roma
08/01/2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Detección de eventos anómalos en un entorno industrial mediante el uso técnicas de Federated Learning.*
Nombre del autor:	*Darío Martín García Carretero*
Nombre del consultor/a:	*Raúl Parada Medina*
Nombre del PRA:	*Jordi Casas Roma*
Fecha de entrega (mm/aaaa):	01/2020
Titulación:	*Máster en Ciencia de Datos*
Área del Trabajo Final:	*Área 2*
Idioma del trabajo:	*Español*
Palabras clave	*eventos anómalos, entorno industrial, federated learning*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Un evento anómalo es aquel que se produce de forma repentina y sin previsión. En un entorno industrial, estos eventos, generalmente fallos en las máquinas, pueden provocar grandes daños económicos y personales por lo que su detección puede ayudar a prevenir situaciones irreversibles.

El objetivo de este proyecto es mostrar como detectar anomalías en equipos industriales mediante el uso de un modelo de aprendizaje automático. Para enseñar a distinguir a un modelo entre un comportamiento normal y otro anormal se necesitan datos, contra más datos, mejor.

Hoy en día la mayoría de los componentes dentro de un entorno industrial están monitorizados mediante el uso de dispositivos de medición especializados. Siempre podemos disponer de los datos suministrados por nuestros dispositivos de medición para entrenar al modelo, pero ¿y si pudiéramos disponer de más datos? Muchos equipos industriales son de uso genérico y pueden ser usados para muchas tareas y en muchos tipos de instalación. Si pudiéramos tener acceso a los datos de todos esos dispositivos podríamos crear un modelo mucho más robusto.

Por diversas razones a las compañías rehúsan compartir sus datos. Por este motivo en este trabajo se propone el uso del Federated Learning (FL). Gracias a el FL se pueden construir modelos aprovechando toda la información disponible manteniendo la privacidad de los datos.

Abstract (in English, 250 words or less):

An anomalous event is one that occurs suddenly and without foresight. In an industrial environment, these events, generally machine failures, can cause great economic and personal damage, so their detection can help prevent irreversible situations.

The objective of this project is to show how to detect anomalies in industrial equipment using a machine learning model. To teach a model to distinguish between normal and abnormal behaviour, data is needed, the more data, the better.

Today, most components within an industrial environment are monitored by specialized measuring devices. We can always have the data provided by our measuring devices to train the model, but what if we could have more data? Many industrial equipment is of generic use and can be used for many tasks and in many types of installation. If we could have access to the data of all those devices, we could create a much robust model.

For various reasons companies refuse to share their data. For this reason, this paper proposes the use of Federated Learning (FL). Thanks to FL, models can be built taking advantage of all available information while maintaining data privacy.

Índice

Lista de figuras

Lista de ecuaciones

Lista de tablas

1. Introducción

1.1 Contexto y justificación del trabajo

Un evento anómalo es aquel que se produce de forma repentina y sin previsión. En un entorno industrial estos eventos, generalmente fallos en las máquinas, pueden provocar grandes daños por lo que su detección puede ayudar a prevenir situaciones críticas.

El objetivo de este proyecto es mostrar una forma de detectar las posibles anomalías que se pudieran producir en equipos industriales mediante el uso de un modelo de aprendizaje automático. Para enseñar a distinguir a un modelo entre un comportamiento normal y otro anormal se necesitan datos, contra más datos, mejor. Hoy en día la mayoría de los componentes dentro de un entorno industrial están monitorizados mediante el uso de dispositivos de medición especializados. Siempre podemos disponer de los datos suministrados por nuestros dispositivos de medición para entrenar un modelo, pero ¿y si pudiéramos disponer de más datos? Muchos equipos industriales son de uso genérico y pueden ser usados para muchas tareas en muchos tipos de instalación. Si pudiéramos tener acceso a los datos de todos esos dispositivos podríamos crear un modelo mucho más robusto.

Por diversas razones a las compañías rehúsan compartir sus datos, por este motivo se propone el uso del Federated Learning (FL). Con el FL se pueden construir modelos aprovechando toda la información disponible manteniendo la privacidad de los datos.

Fig. 1 Los fallos no controlados pueden tener resultados catastróficos

1.2 Objetivos del Trabajo

El objetivo es mostrar un caso de uso para la aplicación del Federated Learning en la detección de eventos anómalos en entornos industriales, en este caso particular, fallos en máquinas. La finalidad es que la metodología aquí presentada pueda servir como base para el desarrollo e implantación de soluciones similares en diferentes tipos de situaciones y/o entornos.

1.3 Enfoque y método seguido

El método seguido se describe a continuación:

Proponer un caso de uso

Se introducirá un caso de uso que cumpla con las premisas del objetivo del trabajo.

Diseño del experimento

Se diseñará y expondrá el procedimiento que ayudara a evaluar el éxito o el fracaso de la propuesta.

Preparación del entorno de trabajo

Se instalarán las herramientas necesarias para la generación de los conjuntos de datos, para su gestión, para su análisis y para la creación de los modelos necesarios.

Simulación de un entorno industrial

Se implementará un software que permitirá simular instalaciones industriales. De esta simulación será de donde se obtendrán los datos necesarios para la creación de los modelos.

Construcción del modelo base

Se llevarán a cabo los pasos necesarios para la creación de un modelo que, servirá de base para el desarrollo tanto de modelos locales para cada una de las factorías como para el modelo de aprendizaje federado. Esto incluirá las siguientes tareas:

- Adquisición (generación) de los datos
- Análisis de los datos
- Elección del tipo de modelo a utilizar
- Procesado de los datos
- Selección del framework para el desarrollo del modelo
- Selección de los criterios para la evaluación del modelo
- Implementación, entrenamiento y validación del modelo

Construcción del modelo federado

Discusión de los resultados y análisis de posibles extensiones

1.4 Planificación del Trabajo

La planificación temporal del trabajo es la siguiente:

- **18 septiembre – 20 octubre**:
    - Definir el alcance del TFM.
    - Decidir el conjunto de datos a usar.
    - Decidir el tipo de modelos/ técnicas a usar en base al estudio del estado de arte.

- **21 octubre – 21 diciembre:**
    - Instalación del entorno de trabajo
    - Obtención del conjunto de datos
    - Procesado del conjunto de datos
    - Generación de los modelos
    - Análisis de la calidad de los modelos
    - Análisis de los resultados y estudio de posibles extensiones
- **22 diciembre – 8 enero:**
    - Redacción de la memoria
    - Creación de la presentación
- **9 enero – 22 enero:**
    - Defensa pública del trabajo

A continuación, se muestra un diagrama con la programación establecida:

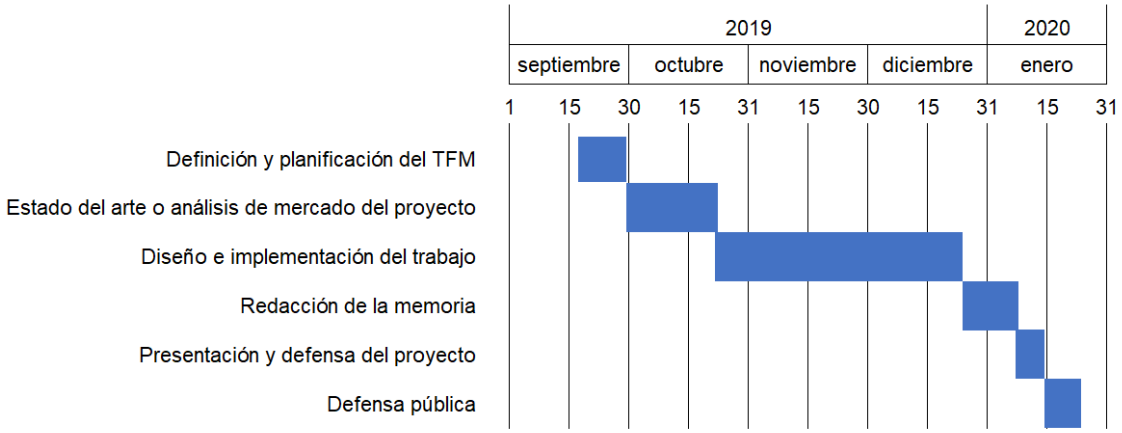

Fig. 2 Planificación del trabajo

1.5 Breve resumen de los productos obtenidos

Como ya se ha mencionado el objetivo del trabajo es mostrar una metodología por lo que el objetivo de los modelos aquí generados no se extiende más allá de su uso a nivel didáctico y su aplicación en entornos reales dependerá mucho del tipo de entorno y de las fuentes de datos disponibles. Si embargo, todo el procedimiento hasta llegar a su construcción puede resultar de gran interés en la resolución de problemas similares y justamente esto, el proceso, es lo que deberá considerarse como el producto final de este trabajo.

1.6 Breve descripción de los otros capítulos de la memoria

En el apartado 2, se presenta un estudio del estado del arte con el objetivo de conocer cómo se han solucionado en el pasado problemas similares. Esto servirá para dos cosas: como base a la hora de construir una solución para el problema objeto de este proyecto y para no repetir estudios o trabajos que ya se realizaron en su momento.

En el apartado 3, se hace una detallada exposición del problema que se desea solucionar incluyendo las necesidades y restricciones a las que habrá que hacer frente. Adicionalmente se realizará una breve introducción al Federated Learning donde se explicará, de una forma introductoria, en qué consiste y cómo funciona.

En el apartado 4, se expondrá el procedimiento que se seguirá para construir y evaluar los modelos.

En el apartado 5, se detallará todo el proceso de construcción y evaluación de estos modelos.

En el apartado 6, se estudiarán los resultados obtenidos y se reflexionará sobre las posibles extensiones o líneas de trabajo futuras.

2. Estudio del estado del arte

Este trabajo aborda dos temáticas que, aunque relacionas entre sí, puede ser estudiadas de forma independiente. Por este motivo se realizará el estudio del arte de manera independiente.

2.1 Detección de eventos anómalos

Un evento anómalo es una observación (o pequeño conjunto de observaciones) que ocurre infrecuentemente y que se desvía o es inconsistente con respecto al resto de observaciones lo suficiente como para indicar que se está produciendo una anomalía en el conjunto de datos.

La detección de este tipo de eventos se ha estudiado ampliamente en la literatura científica desde multitud de enfoques diferentes. A continuación, se muestran algunos de ellos:

IRESE: An intelligent rare-event detection system using unsupervised learning on the IoT edge [1]

Este documento propone un sistema de detección de eventos anómalos basado en el uso de modelos de Machine Learning no supervisados (clustering) en tiempo real. El procedimiento se divide en cuatro fases:

1. Captura de los datos.
2. Extracción de características.
3. Clustering online, se aplica el algoritmo BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) para generar micro-clusters.
4. Clustering offline, se utiliza Agglomerative Clustering que finalmente genera dos macro-clusters, uno que contiene los eventos normales y otro con los eventos raros.

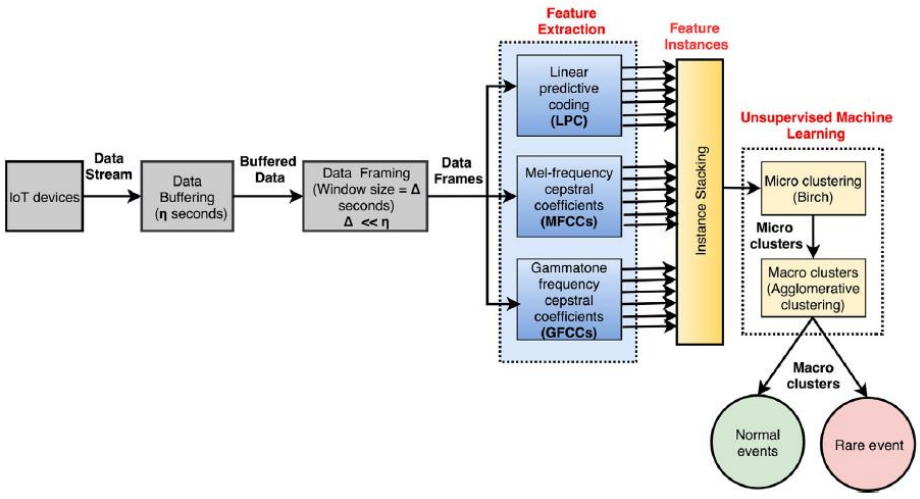

Fig. 3 IRESE framework

Las principales ventajas que ofrece este framework son dos, una es que gran parte del proceso se lleva a cabo en dispositivos locales (edge devices) lo que reduce la sobrecarga en la red y la latencia del sistema. La otra es que no se necesita etiquetar los eventos anómalos debido al uso de algoritmos no supervisados. En cuanto a las

desventajas, la principal es la extracción de características ya que este procedimiento es muy dependiente del tipo de datos que se quieren procesar.

En cuanto a los resultados, el articulo menciona que se han obtenido valores de *precision* y de *recall* por encima del 90% en algunos casos.

Anomaly detection in ECG time signals via deep long short-term memory networks [2]

Este trabajo utiliza la detección de eventos anómalos con fines médicos. En particular se analiza el uso de redes neuronales LSTM (Long Short-Term Memory) profundas para la detección de anomalías en las señales EGC (electrocardiogramas).

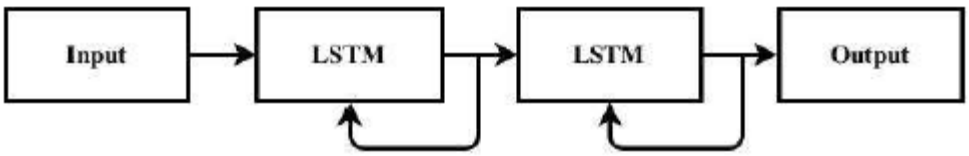

Fig. 4 Arquitectura de la red propuesta

Si denotamos una señal EGC como X = {x(0), …, x(i), …, x(n)} donde cada x(i) representa un valor del EGC en el tiempo i. El procedimiento, a grandes rasgos es el siguiente:

1. Se entrena la red con el siguiente objetivo: predecir los k siguientes valores del EGC para cada x(i).
2. A partir de la red construida se obtiene un conjunto de vectores de error para cada x(i). Con estos vectores de error se ajusta una distribución normal multivariante.
3. A partir de esa distribución se calcula la probabilidad de ocurrencia del vector de error obtenido y en función de un threshold se determina si ese punto anómalo o no.

Este método presenta varias ventajas sobre otros métodos:

1. No se requiere apenas preprocesamiento de los datos.
2. No se necesita saber a priori el tipo de anomalías que pueden aparecer.

El método proporciona resultados bastante buenos llegando valores para el F-score por encima de 0.9 para varias clases de anomalías.

Diagnosing Network-Wide Traffic Anomalies [3]

El documento explora el uso del método PCA (Principal Analysis Components) para la detección de eventos anómalos en una red de comunicación. La idea consiste en dividir el espacio original en dos subespacios, uno que captura el comportamiento normal y otro el anómalo. El espacio normal estará formado por las componentes que capturan más varianza mientras que el espacio anómalo estará formado por los componentes que capturan menos varianza.

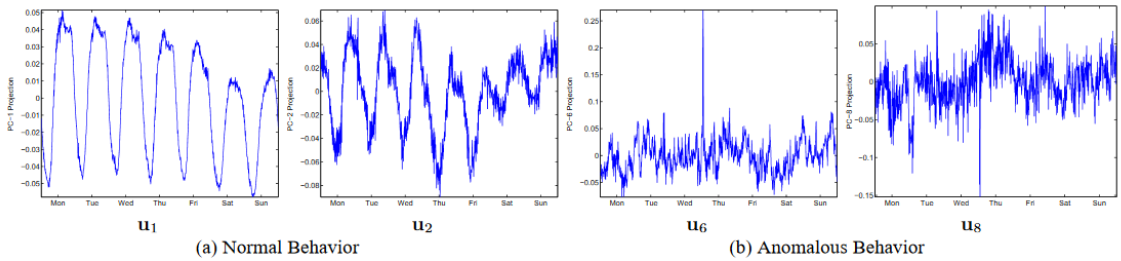

Fig. 5 Proyecciones PCA de los datos temporales

Gracias a esta división podremos descomponer cada instancia en sus componentes normales y anormales. Para determinar cuándo se ha producido un evento extraño se utiliza la norma del vector del espacio anómalo. Si esta norma sobrepasa cierto threshold se considerará como un evento anómalo. En otro caso se identificará como un evento normal.

Los resultados muestran altas tasas de detección y bajas tasas de falsos positivos.

Anomaly Detection with Generative Adversarial Networks for Multivariate Time Series [4]

Este estudio utiliza una arquitectura GAN (Generative Adversarial Network) como mecanismo para la detección de eventos anómalos en un entorno industrial informatizado. Una arquitectura GAN está pensada como un sistema generativo (dada una distribución, el sistema es entrenado para generar datos que sigan esa distribución). Un sistema GAN está formado por dos redes neuronales: una llamada generativa que es la encargada de generar datos que sigan la distribución dada y una llamada discriminante, que se encarga de ayudar al entrenamiento de la primera. El objetivo del método propuesto es entrenar la arquitectura GAN para que aprenda a generar situaciones de uso normal. A continuación, se muestra un esquema del sistema propuesto:

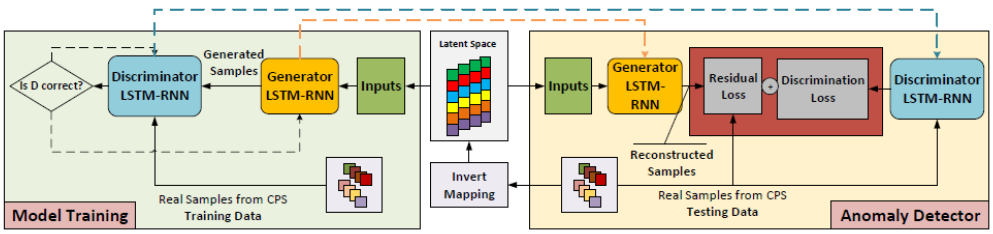

Fig. 6 Arquitectura GAN propuesta

Para determinar cuándo una situación es anómala el sistema se basa en una puntuación (una combinación entre en la diferencia entre el input real y el generado y el valor de salida de la red discriminante).

Una de las principales ventajas de este método es que es no supervisado por lo que no es necesario el etiquetado de las muestras. Por otro lado, su principal desventaja es su alto coste computacional. Con respecto a su rendimiento, el documento menciona que el modelo mejora el desempeño de modelos anteriores en el conjunto de datos utilizado (Secure Water Treatment Testbed).

High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning [5]

Los algoritmos one-class SVM (1SVM) son una herramienta que demostrado ser efectiva en la tarea de la detección de valores anómalos. El problema con este tipo de modelos es que no escalan bien. En este trabajo se propone un método que permite solventar esa limitación gracias al uso de DBNs (Deep Belief Networks). La idea es poder sustituir los kernels no lineales de las 1SVM (muy costosos de calcular) por kernels lineales (menos costosos) gracias al uso de las DBN para la generación de vectores de características.

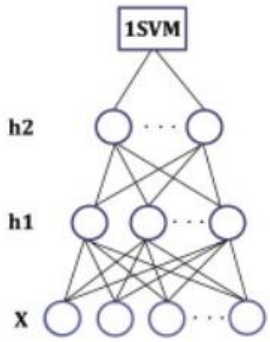

Fig. 7 Modelo híbrido propuesto

Los resultados experimentales demuestran que el modelo híbrido mejora a los modelos 1SVMs en aproximadamente un 20% en algunos conjuntos de datos. Comparado con los métodos basados en autoencoders la diferencia es prácticamente inapreciable sin embargo el método propuesto es tres veces más rápido en el entrenamiento y hasta 1000 veces más rápido en el testing.

Isolation-Based Anomaly Detection [6]

Lo que diferencia a esta propuesta del resto es que, en lugar de tratar de modelar las instancias normales para luego buscar los valores anómalos, el modelo explícitamente identifica las anomalías. Esto se consigue a través de los llamados Isolation Forests. Los Isolation Forest están formados por un conjunto de árboles de decisión. Cada árbol se crea de la siguiente manera, primero se selecciona un atributo aleatoriamente y luego se establece una partición también aleatoria entre el máximo y el mínimo del atributo seleccionado. La idea detrás de este algoritmo es que los puntos que están más aislados estarán en las hojas más cercanas a nodo raíz.

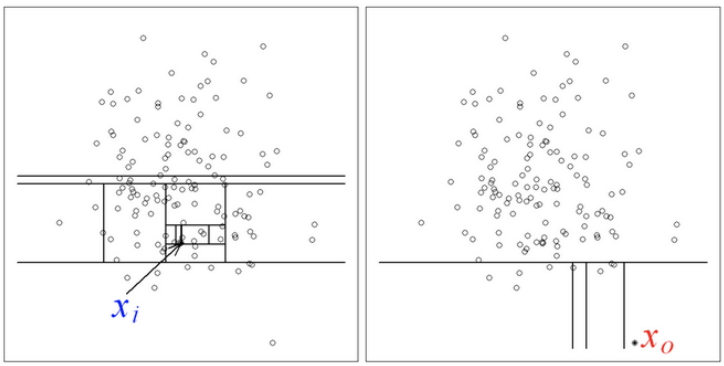

Fig. 8 Identificación de una instancia normal (izquierda) y una anormal (derecha)

Este algoritmo puede tratar con conjuntos de datos grandes y de alta dimensionalidad. Sin embargo, este método presenta ciertos problemas debido a que las particiones se realizan únicamente de manera vertical u horizontal. Para solventar esos problemas se ha propuesto una modificación al algoritmo original en [7] llamado Extended Isolation Forest.

Outlier Detection Using Replicator Neural Networks [8]

La Replicator Neural Network utilizada en este artículo es un MLP (Multi-Layer Perceptron) compuesto por tres capas ocultas. La función de esta red es la de reproducir los datos de entrada en la salida con el mínimo error posible.

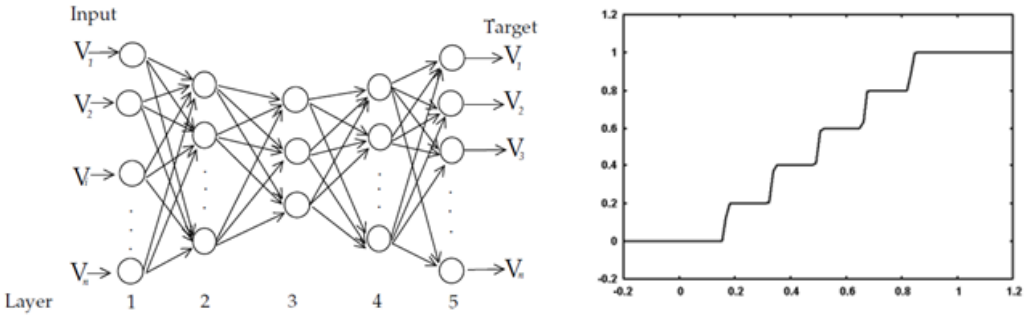

Fig. 9 Arquitectura de una Replicator Neural Network (izquierda) y función de activación de la capa intermedia (derecha)

Una vez entrenada la red a cada punto se le asigna un valor que es la media de los errores de reconstrucción de cada una de las variables. Cuanto más grande sea el error más probable será que se trate de una anomalía. Este método es similar al uso de auto-encoders, la diferencia más llamativa es que este método utiliza una función de activación especial en forma de escalera en la capa intermedia.

2.2 Federated Learning

El campo del Federated Learning es relativamente nuevo y se encuentra en pleno estudio es por este motivo que no existe tantos trabajos como en el caso del Machine Learning tradicional. Se han seleccionado tres trabajos, dos de definición e introducción a esta tecnología y uno centrado en un caso práctico de aplicación.

Communication-Efficient Learning of Deep Networks from Decentralized Data [9]

En este artículo se presenta un algoritmo que permite el uso de modelos de Machine Learning sobre conjuntos de datos descentralizados. En el documento no solo demuestra que esta idea es factible técnicamente presentando un algoritmo llamado *FederatedAveraging* sino que permite crear modelos de una alta calidad. Uno de los aspectos más importante de esta propuesta es que permite incrementar el nivel de privacidad de los clientes ya que en ningún momento se comparten datos entre clientes ni estos son enviados a un servidor central. A continuación, se muestra un esquema del modelo propuesto:

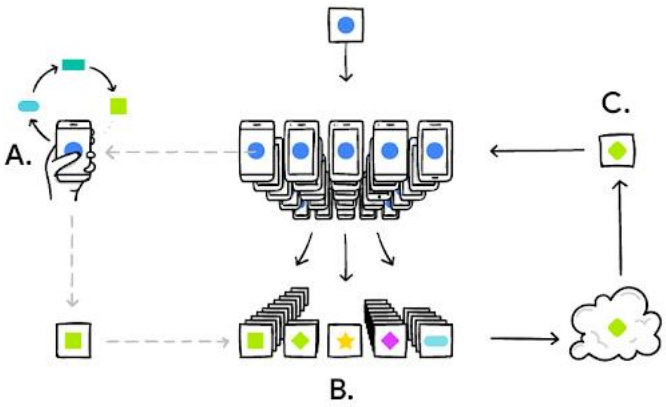

Fig. 10 Cada dispositivo personaliza el modelo localmente (A). Las actualizaciones de muchos usuarios se agregan (B) para actualizar el modelo global que se compartirá con todos los usuarios (C), después de lo cual se repite el proceso.

Federated Machine Learning: Concept and Applications [10]

Este documento presenta los conceptos básicos del Federated Learning sus principales técnicas y arquitecturas. Expone su potencial en varias aplicaciones y compara sus características con otros sistemas de aprendizaje distribuidos (Distributed Machine Learning, Federated Database Systems, etc.). Adicionalmente introduce el concepto de Data Alliance of Enterprises. Una Data Alliance of Enterprises es un conjunto de organizaciones que, mediante el uso de sus datos (sin compartirlos directamente con los demás miembros) colaboran para crear un modelo que aporte beneficios a todos los participantes. Este tipo de alianzas están llamadas a ser el futuro del Machine Learning y se espera constituyan todo un nuevo modelo de negocio [11].

Multi-Institutional Deep Learning Modeling Without Sharing Patient Data: A Feasibility Study on Brain Tumor Segmentation [12]

En este documento se presenta el primer uso del Federated Learning para el tratamiento de imágenes para el diagnóstico médico. En particular se usa esta tecnología para la creación de un modelo distribuido de segmentación de imágenes (U-Net) sin compartir datos de pacientes entre instituciones.

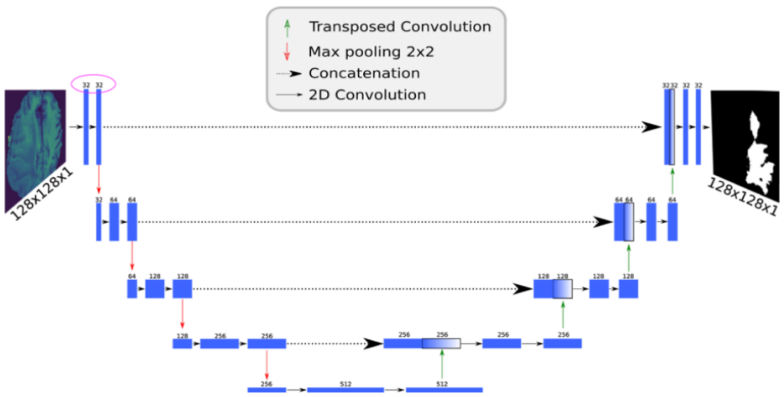

Fig. 11 Arquitectura de la U-Net utilizada

El estudio demuestra que el modelo basado en Federated Learning consigue un rendimiento prácticamente equivalente al modelo creado con el uso de datos compartidos.

2.3 Novedades propuestas

Aunque la detección de eventos anómalos ha sido estudiada en profundidad desde múltiples perspectivas nunca se había abordado el problema de forma colaborativa y descentralizada. Esta es la principal novedad que pretende aportar este trabajo.

3. Escenario

Supongamos que una empresa multinacional dispone de plantas repartidas por todo el mundo. A pesar de pertenecer a la misma compañía cada una de las instalaciones tiene sus propias particularidades en cuanto al tipo de productos que producen, la manera de fabricarlos, etc. A estas diferencias se han de añadir también las condiciones ambientales de cada lugar: temperatura, humedad, presión atmosférica, etc. A pesar de estas diferencias, las máquinas utilizadas en todas las factorías son similares.

La empresa desea implantar un sistema que pueda ayudar a detectar las posibles averías de las máquinas de sus factorías antes de que estas puedan provocar daños tanto al personal como a las propias instalaciones.

Dado que todas las instalaciones están dotadas de sensores que capturan los datos operaciones de las máquinas se dispone de gran cantidad de datos sobre su funcionamiento. Por este motivo se ha optado por la utilización de técnicas de Machine Learning para tratar de resolver el problema.

A pesar de pertenecer a la misma compañía, las plantas trabajan con un alto grado de independencia y de hecho, suelen competir entre ellas en cuestiones como la cantidad de producción, calidad, etc. Debido a esta competitividad, las factorías suelen ser reacias a compartir datos sobre sus técnicas de producción, las configuraciones de sus máquinas, etc. Esto implica que el acceso a sus datos está muy restringido y estos pueden ser únicamente utilizados a nivel interno. Esto motiva que la construcción de modelos de Machine Learning solo se pueda realizar a nivel local con los datos de cada factoría de forma independiente.

Debido a las diferencias en las condiciones de trabajo y ambientales de cada instalación, un modelo construido para una factoría concreta puede no funcionar correctamente en otra instalación diferente. En principio esto no representa un problema ya que cada planta tendrá su propio modelo que será usado internamente.

La compañía está en constante expansión y es habitual que abra nuevas plantas a lo largo de mundo. Obviamente la empresa desea implantar el sistema de detección de fallos que tan bien le ha funcionado en otras factorías. Pero hay un problema, en principio no se podría confiar en que el ninguno de los modelos locales existentes fuera de utilidad en estas nuevas instalaciones. Por lo que sería necesario recolectar datos de esta nueva factoría quizás durante meses o incluso años para poder crear un nuevo modelo local.

La compañía considera esta tardanza inaceptable por lo que exige que diseñe un sistema que permita que las nuevas factorías se aprovechen de los modelos construidos previamente, sin tantas esperas.

La forma tradicional de solucionar este problema sería almacenando los datos de todas las instalaciones existentes en un único lugar y a partir de todos los datos construir un modelo más robusto que pudiera adaptarse mejor a un nuevo comportamiento al haber aprendido de datos de diferentes fuentes.

Este tipo de solución no se puede aplicar en este caso ya los datos de cada instalación son privados y es aquí es donde entra en juego el uso del Federated Learning**.**

El aprendizaje federado (también conocido como aprendizaje colaborativo) es una técnica de aprendizaje automático que entrena un algoritmo a través de múltiples dispositivos o servidores que contienen muestras de datos locales, sin intercambiar sus muestras de entre ellos. Este enfoque contrasta con las técnicas tradicionales de aprendizaje automático centralizado donde todas las muestras de datos se cargan en un servidor, así como con los enfoques descentralizados más clásicos que suponen que las muestras de datos locales se distribuyen de manera idéntica.

El aprendizaje federado permite a múltiples actores construir un modelo de aprendizaje automático robusto y común sin compartir datos, abordando así problemas críticos como la privacidad de los datos, la seguridad de los datos, los derechos de acceso a los datos y el acceso a datos heterogéneos.

El aprendizaje federado tiene como objetivo entrenar un algoritmo de aprendizaje automático, por ejemplo, redes neuronales, en múltiples conjuntos de datos locales contenidos en nodos locales sin intercambiar muestras de datos. El principio general consiste en entrenar modelos locales en muestras de datos locales e intercambiar parámetros (por ejemplo, los pesos de una red neuronal) entre estos modelos locales con cierta frecuencia para generar un modelo global.

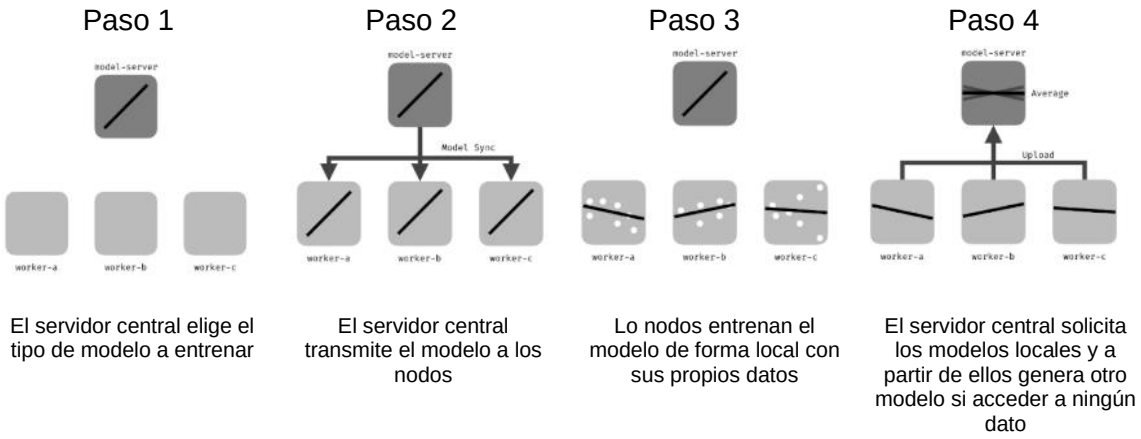

Fig. 12 Esquema de funcionamiento del Federated Learning

Para construir el modelo global existen varias alternativas aunque las más utilizadas son:

Federated Stochastic Gradient Descent (FedSGD)

El entrenamiento de modelos basado en la optimización de parámetros se suele basar en variantes del SGD (Stochastic Gradient Descent), donde los gradientes se calculan en un subconjunto aleatorio del conjunto de datos total y luego se utilizan para hacer un paso del descenso del gradiente.

El FedSGD es la traducción directa de este algoritmo a la configuración federada pero utilizando una fracción aleatoria de los nodos y utilizando todos los datos en estos nodos. El servidor promedia los gradientes proporcionalmente al número de muestras de entrenamiento en cada nodo, y se utilizan para realizar un paso de descenso de gradiente.

Federative averaging (FedAvg)

El FedAvg es una generalización de FedSGD, que permite a los nodos locales realizar más de una actualización de sus parámetros. Posteriormente intercambia estos parámetros con el servidor central en lugar de los gradientes. La razón de que este método funcione es que si todos los nodos locales comienzan desde la misma inicialización, promediar los gradientes es estrictamente equivalente a promediar los pesos mismos. Además, se sabe que promediar pesos ajustados provenientes de la misma inicialización no necesariamente perjudica el rendimiento promedio del modelo resultante. Esta será la opción que se utilizará en este trabajo.

4. Diseño del experimento

El objetivo del trabajo es mostrar la posible aplicación del aprendizaje automático federado en la detección de potenciales fallos en máquinas dentro de un conglomerado de entornos industriales monitorizados. Este tipo de entornos monitorizados hoy en día están muy extendidos gracias al auge de la Industria 4.0 y la implementación de dispositivos IoT (Internet of Things) en las fábricas. Sin embargo, es prácticamente imposible obtener un conjunto de datos real por ser este tipo de datos muy sensible para las compañías. Por este motivo no se utilizarán datos reales y en su lugar, se desarrollará un software que nos permitirá la simulación de estos datos.

Teniendo lo anterior en cuenta, se tratará de demostrar la utilidad del aprendizaje federado en situaciones como las descritas en el apartado 3 siguiendo el siguiente procedimiento:

- Se generarán 4 datasets simulados con diferentes condiciones ambientales y de funcionamiento. Para ello se utilizarán un conjunto de scripts creados para tal fin. El funcionamiento de estos scripts se detalla en el anexo A. Los archivos de configuración para cada conjunto de datos se pueden encontrar en el anexo B.

- De esos conjuntos se elegirá uno y con estos datos se construirá un modelo. Este modelo será el modelo base para todos los demás conjuntos de datos. Notar que lo que nos interesa es la estructura del modelo y no el valor de sus parámetros.

- Después se entrenará un modelo (con la estructura definida en el paso anterior) por cada uno de los conjuntos de datos.

- Se evaluarán los modelos obtenidos.

- A continuación se elegirán 3 de los 4 conjuntos de datos y se entrenará un modelo federado.

- Compararemos el modelo federado global con cada uno de los modelos locales para comparar los rendimientos.

- Finalmente se evaluará el conjunto de datos que se quedó fuera del entrenamiento del modelo federado con este, y comparemos su desempeño con respecto a su correspondiente modelo local entrenado con anterioridad.

5. Implementación

5.1 Preparación del entorno de trabajo

La mayor parte del desarrollo del proyecto se llevará a cabo con el lenguaje de programación Python. Python ofrece, entre otras, las siguientes ventajas [13]:

- **Simplicidad.** Python ofrece la posibilidad de desarrollar programas muy potentes con muy pocas de líneas de código. Identifica y asocia automáticamente los tipos de datos y, en general, resulta un lenguaje fácil de usar y no se requiere mucho tiempo de codificación.

- **Compatibilidad.** Muchas de las tecnologías actuales relacionadas con el Machine Learning están pensadas para ser utilizadas con este lenguaje.

- **Facilidad de aprendizaje.** En comparación con otros lenguajes, Python es fácil de aprender incluso para los programadores con menos experiencia. Principalmente por tres razones:

    - Cuenta con amplios recursos de aprendizaje.
    - Garantiza un código legible.
    - Se rodea de una gran comunidad.

- **Variedad de paquetes.** Python tiene un poderoso conjunto de paquetes para una amplia gama de necesidades de análisis y ciencia de datos (numPy, Pandas, Scipy, Scikit-learn, etc.)

- **Visualización de datos.** Aunque no es el mejor lenguaje en lo que respecta a la visualización de datos. Actualmente existen APIs que pueden ofrecer resultados bastante buenos.

Nuestro entorno de trabajo estará formado por las siguientes herramientas:

- **Microsoft Excel.** Software de hojas de cálculo
- **Python.** En su versión 3.6
- **PyCharm.** Editor para el lenguaje de programación Python

5.2 Simulación de un entorno industrial

5.2.1 Introducción

El objetivo es implementar un software para la simulación de un entorno industrial cuyas instalaciones se encuentran monitorizadas mediante dispositivos IoT. Debido a las características únicas de cada tipo de instalación no nos es posible construir un software que simule cada una de las diferentes máquinas que pudieran existir en dicho entorno industrial. Por lo tanto, consideraremos únicamente un tipo de máquina: máquinas rotatorias genéricas. Basándonos en conjuntos de datos ya existentes (por ejemplo, *Turbofan Engine Degradation Simulation Data Set)* para cada máquina se estudiarán las siguientes variables operacionales[1]:

[1] Las unidades de medida serán: kPa para las presiones, ºC para las temperaturas y rpm para las velocidades

- Velocidad rotacional
- Temperatura
- Presión

Además de estos valores se tendrán en cuenta variables relacionadas con el entorno. En particular la temperatura del entorno y la presión atmosférica.

5.2.2 Caracterización de la degradación de las máquinas

Con el propósito de una generación de datos inspirada en la física se utilizará una aproximación similar a la utilizada en [14]. En particular se usará la siguiente ecuación para simular la degradación de una máquina en función de su tiempo de uso:

$$\boldsymbol{h(t) = 1 - d - exp\{at^b\}}$$

Ecuación 1

Donde, *a* y *b* son coeficientes que determinan la forma de la curva y *d* es la degradación inicial. A continuación se muestra un ejemplo de estas curvas:

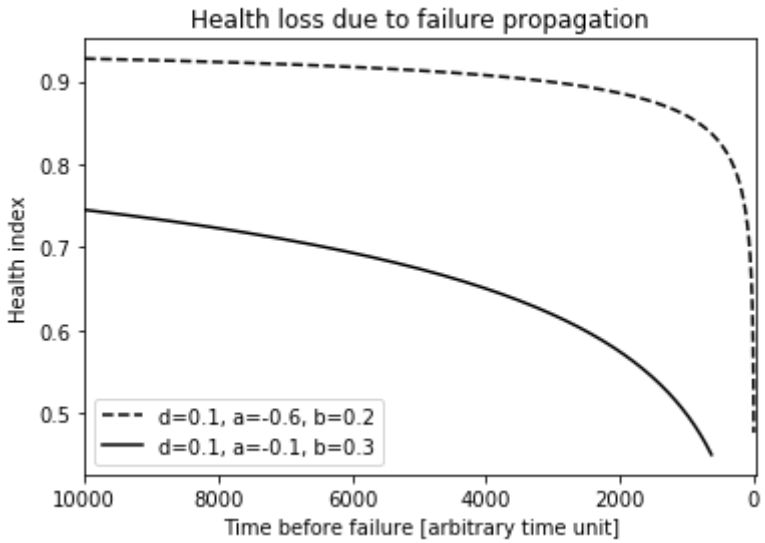

Fig. 13 Curvas de degradación para diferentes parámetros (Fuente: Elaboración propia)

Se puede observar que la curva punteada representa una degradación repentina mientras que la curva continua representaría una degradación más constante (aunque el fallo se produciría antes).

En la simulación se considerarán dos curvas de degradación para cada máquina. Esto equivaldría a decir que cada máquina puede fallar por dos motivos diferentes:

- Uno estrechamente relacionado con la presión y/o la temperatura (esta curva se denominará ***h1).***

- Una relacionada con la velocidad (se denotará por ***h2****)*

5.2.3 Evolución de las condiciones operacionales

Para cada uno de los parámetros operacionales se propone una fórmula[2] que proporcionará su valor en el tiempo *t* en función de:

- Su valor en *t-1*
- Valores del resto de parámetros operacionales
- Valor de la(s) curva(s) de degradación en *t*
- Parámetros ambientales

Formalmente:

$$\boldsymbol{F(t) = f(w, F(t-1), h^*(t), a^*(t))}$$

Ecuación 2

Donde:

- $\mathbf{F(t)}$ es el valor objetivo
- $\mathbf{w}$ valores de los parámetros operacionales actuales
- $\mathbf{F(t\text{-}1)}$ es el valor anterior
- $\mathbf{h^*(t)}$ valor de la(s) curva(s) de degradación
- $\mathbf{a^*(t)}$ condiciones ambientales

Cálculo de la velocidad

La velocidad en *t* será una media entre a velocidad en *t-1* y la velocidad objetivo ponderada con un coeficiente relacionado con la función de degradación ***h2.***

$$\boldsymbol{v_t = (v_{t-1} + (2 - h2_t) * v_{objetivo})/2}$$

Ecuación 3

A continuación, se muestra el efecto de la degradación en la evolución de la velocidad:

[2] Todas las fórmulas propuestas en este apartado están basadas en las fórmulas presentadas en [25].

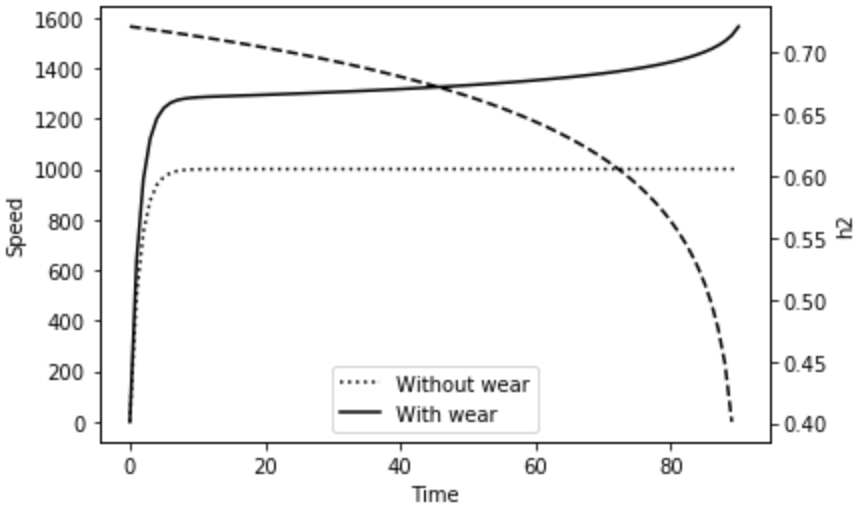

Fig. 14 Efecto de la degradación en la velocidad (Fuente: Elaboración propia)

En este caso, en el que la velocidad objetivo es 100, se puede ver como el efecto del desgaste afecta a la velocidad real. Al principio la diferencia se mantiene más o menos constante empezando a aumentar de forma notable al final de la vida útil de la máquina.

Cálculo de la temperatura

La temperatura del dispositivo estará afectada por la velocidad por lo que en este caso la temperatura estará influenciada tanto por ***h1*** como por ***h2*** (a diferencia de lo que ocurría con la velocidad a la que solo influía ***h2***). La fórmula utilizada para el cálculo de la temperatura es la siguiente[3]:

$$\mathbf{T_t = (2 - h1) * g(T_{t-1}, T_{ambiente}, T_{máxima}, v_t/100, 0.5 * v_t/1000)}$$

Ecuación 4

Donde:

$$\boldsymbol{g(v, v_{min}, v_{max}, obj, r) = max\ (min(v + (obj - v) * r, v_{max}), v_{min})}$$

Ecuación 5

Seguidamente se muestra gráficamente el efecto del desgaste de los componentes en la temperatura de operación:

[3] El valor de las constantes que aparecen en la fórmula podrá variar en función de las condiciones de la instalación que se quiera simular.

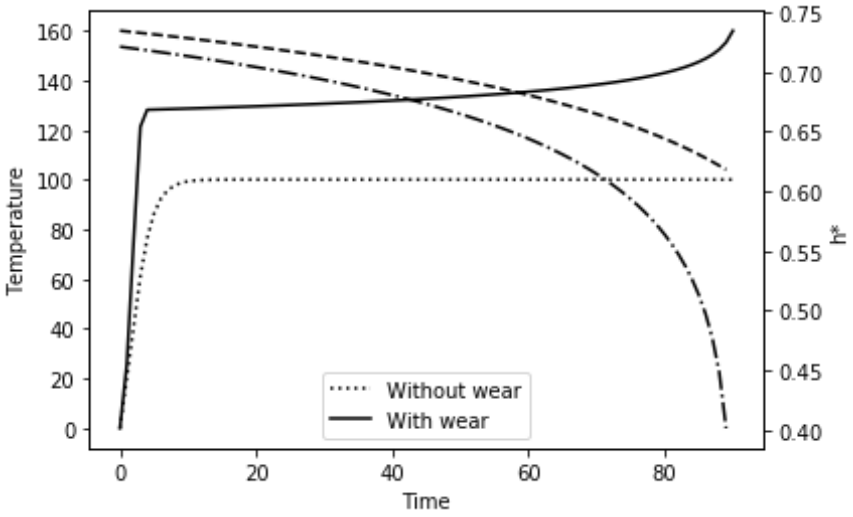

Fig. 15 Efecto de la degradación en la temperatura (Fuente: Elaboración propia)

Como sucedía en el caso de la velocidad se puede ver que la diferencia entre el caso ideal (línea punteada) y del afectado por el desgaste (línea continua) crece de forma moderada hasta que, al final de la vida útil del dispositivo, crece de forma más pronunciada.

Cálculo de la presión

Igual que con la temperatura el cálculo de la presión estará influenciado por ***h1*** y ***h2.*** A continuación se muestra la fórmula usada para el cálculo de la presión[4]:

$$\boldsymbol{P_t = h1 * g(P_{t-1}, P_{ambiente}, P_{máxima}, 2 * P_t, 0.3 * v_t/1000)}$$

Ecuación 6

Donde ***g*** es la función definida en Ecuación 5.

A continuación, se puede observar gráficamente el efecto del desgaste sobre la presión:

[4] Al igual que el caso de la temperatura las constantes pueden variar en función de las condiciones de la instalación que se quiera simular.

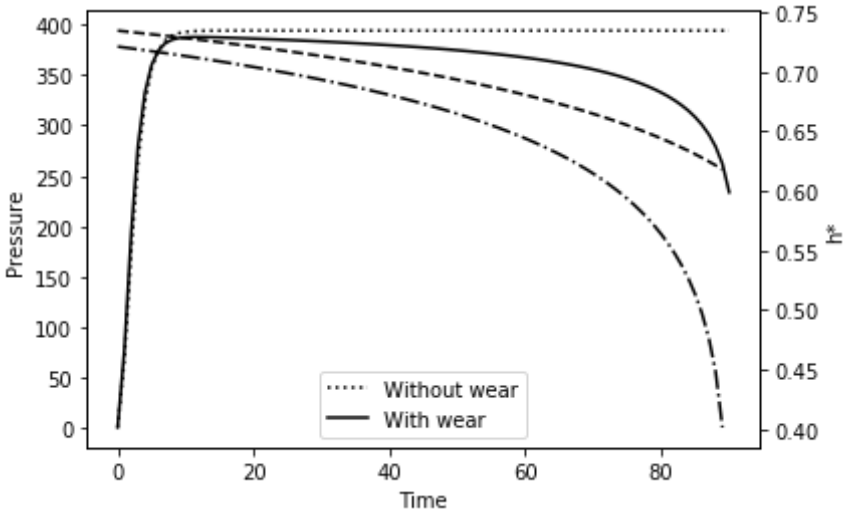

Fig. 16 Efecto de la degradación en la presión (Fuente: Elaboración propia)

A diferencia de lo que ocurría con la velocidad y la temperatura cuando el aparato se encuentra al final de su vida útil el valor real tiende a ser menor que el del funcionamiento óptimo.

5.2.4 Proceso de simulación

El objetivo es simular el funcionamiento de un conjunto de máquinas dentro de una instalación industrial y almacenar tanto los datos de telemetría como los datos referentes al mantenimiento de las máquinas.

Antes de iniciar la simulación a cada una de las máquinas que forma parte de la instalación se les asignan unas curvas de degradación (***h1***, ***h2***) cuyos parámetros son elegidos de forma aleatoria dentro de un intervalo de valores previamente establecido.

Una vez asignadas las curvas se inicia la simulación. Cada simulación (que tiene una duración determinada) se divide en lo que denominaremos periodos. En cada periodo se elige de forma aleatoria un conjunto de máquinas. Estas funcionaran durante un intervalo de tiempo elegido al azar dentro de cada periodo. La velocidad de funcionamiento de cada máquina se establece aleatoriamente dentro unos márgenes establecidos. Durante su funcionamiento se almacenarán sus datos telemétricos. Puede ocurrir, que durante su funcionamiento, en una máquina se produzca un fallo. En este caso esta máquina quedará inutilizada hasta el siguiente periodo y se creará un registro de mantenimiento indicando cuando se produjo el fallo y cuál fue la causa del fallo. Una vez finalice el periodo actual esa máquina será “reparada” asignándole un nuevo par de curvas de degradación y pudiendo participar nuevamente en la selección para el siguiente periodo.

El procedimiento anteriormente descrito se repite hasta que el tiempo de simulación finaliza.

5.2.5 Configuración de la simulación

Cada simulación tiene un total de 20 parámetros configurables. Esta capacidad de personalización será crucial para las siguientes fases del proyecto. Algunos de estos parámetros son:

- Duración del periodo (batch_time)
- Duración de la simulación (simulation_time)
- Número de máquinas en la instalación (machine_count)
- Número de máquinas por periodo (machines_per_batch)
- Temperatura ambiente (temperature)
- Intervalo de velocidad de funcionamiento (operational_speed_min, operational_speed_max)

```
[CONFIGURATION]

name = Facility A
; three months
simulation_time = 7776000
temperature = 15
pressure = 98
cycle_length_min = 1
cycle_length_max = 5
cycle_duration = 60
machines_per_batch = 5
operational_speed_min = 800
operational_speed_max = 900
batch_time = 3600
machine_count = 10
ttf1_min = 5000
ttf1_max = 50000
ttf2_min = 500
ttf2_max = 90000
temperature_max = 100
pressure_factor = 1.8
telemetry_path = ../data/generation/facility A/telemetry
event_path = ../data/generation/facility A/event
```

Fig. 17 Ejemplo de un archivo de configuración de una simulación

5.2.6 Resultados de la simulación

Una vez finalizada la simulación por cada máquina se obtendrán dos tipos de archivos:

Datos de telemetría

Como su nombre sugiere contiene los datos registrados durante el funcionamiento de la máquina. La información contenida en este archivo es la siguiente:

- **id:** identificador único de la máquina
- **time:** marca temporal de la medida
- **speed:** velocidad medida
- **tgtspeed:** velocidad objetivo
- **temp:** temperatura de la máquina
- **press:** presión en la máquina
- **atemp:** temperatura ambiente
- **apress:** presión atmosférica

Tabla 1 Fragmento de archivo de telemetría

id	time	speed	tgtspeed	temp	press	atemp	apress
2e8a9c8c	1125	488.79	873.00	18.22	98.71	15.08	98.06
2e8a9c8c	1126	727.72	873.00	22.14	286.18	15.04	97.94
2e8a9c8c	1127	848.98	873.00	26.85	482.01	14.93	97.98
2e8a9c8c	1128	911.73	873.00	32.51	608.00	14.94	97.90
2e8a9c8c	1129	945.71	873.00	39.37	730.92	15.08	97.91
2e8a9c8c	1130	953.32	873.00	47.71	796.18	14.99	97.96
2e8a9c8c	1131	965.56	873.00	57.69	810.21	14.94	98.02

Es importante notar que durante el proceso de simulación los datos han sido contaminados añadiendo ruido para que el proceso resultara más realista. Este ruido se genera a partir de una distribución de probabilidad uniforme continua y es sumado a los datos calculados con las fórmulas anteriormente descritas.

Registro de mantenimiento de la máquina

Este archivo contendrá información sobre las averías de la máquina y está compuesto por los siguientes atributos:

- **id:** identificador único de la máquina
- **time:** marca temporal del evento
- **code:** código que identifica al evento (**F1** y **F2** son los códigos de cada una de las averías que se pueden producir mientras que el código **FIXED** indica que una máquina ha sido arreglada)
- **severity: CRITICAL** si estamos ante una avería **INFO** en otro caso

Tabla 2 Fragmento del registro de mantenimiento

id	time	code	severity
2e8a9c8c	300061	F1	CRITICAL
2e8a9c8c	302400	FIXED	INFO
2e8a9c8c	2301135	F1	CRITICAL
2e8a9c8c	2304000	FIXED	INFO
2e8a9c8c	2893790	F1	CRITICAL
2e8a9c8c	2894400	FIXED	INFO

5.3 Construcción del modelo base

Antes de poder aplicar técnicas de Federated Learning es necesario construir un modelo base. Este modelo base será la piedra angular de todo el proceso de aprendizaje federado. En los siguientes apartados se detallará todo el proceso necesario para la creación de este modelo.

5.3.1 Adquisición de los datos

Los datos han sido obtenidos gracias al software de simulación cuyo funcionamiento se ha detallado en el apartado anterior. La simulación se corresponde a un periodo de

temporal de aproximadamente seis meses. Los parámetros utilizados para realizar la simulación han sido:

- Número de máquinas: 20
- Número de máquinas por periodo:10
- Duración del periodo: 1h
- Temperatura ambiente media: 20 ºC
- Presión atmosférica media: 101 kPa
- Tiempo mínimo de actividad por periodo: 1 min
- Tiempo máximo de actividad por periodo: 10 min
- Mínima velocidad de funcionamiento: 800 rpm
- Máxima velocidad de funcionamiento: 900 rpm
- Temperatura máxima de funcionamiento: 125 ºC

5.3.2 Análisis de los datos

Debido a la forma de generar los datos de telemetría y mantenimiento no será necesario realizar el análisis sobre todo el conjunto de datos. Se elegirá una máquina al azar y esta se considerará como representante de todo el conjunto. Una vez se ha seleccionado esta máquina representativa se le hará un pequeño interrogatorio a su conjunto de datos asociado:

¿Cuál es el número de registros almacenados?

El conjunto está formado por un total de 743144 registros

¿Cuántos fallos se han producido en estos seis meses?

Se han producido un total de 21 fallos de tipo **F1** y 12 de tipo **F2.** Teniendo esto en cuenta parece que son mucho más frecuentes los errores de tipo **F1**.

¿Cuáles son sus estadísticas más relevantes?

Tabla 3 Datos descriptivos de la máquina

	speed	tgtspeed	temp	press	atemp	apress
count	743144	743144	743144	743144	743144	743144
mean	971.10	835.12	146.25	871.70	20.00	101.00
std	128.77	115.44	18.63	169.02	0.06	0.06
min	5.54	0.00	22.40	9.12	19.90	100.90
25%	943.28	823.00	143.58	801.87	19.95	100.95
50%	980.49	849.00	146.71	904.57	20.00	101.00
75%	1014.11	876.00	151.85	981.73	20.05	101.05
max	1545.38	899.00	223.94	1753.91	20.10	101.10

En interesante destacar las diferencias notables que existen entre los valores medios de la velocidad (*speed*), la presión (*press*) y la temperatura (*temp*) y sus valores máximos. Estos valores extremos podrían estar relacionados con la aparición de fallos en la máquina.

¿Qué relación existe entre las variables operacionales?

Tabla 4 Tabla de correlaciones

	speed	tgtspeed	temp	press
speed	1.000	0.874	0.153	0.539
tgtspeed	0.874	1.000	-0.018	0.387
temp	0.153	-0.018	1.000	-0.061
press	0.539	0.387	-0.061	1.000

Se puede observar una correlación alta entre la velocidad (speed) y la velocidad objetivo (tgtspeed) lo cual es totalmente lógico. Sin embargo, entre el resto de las variables no se observa ninguna relación destacable.

¿Qué aspecto tienen los datos?

A continuación, se muestra una representación gráfica de los datos operacionales:

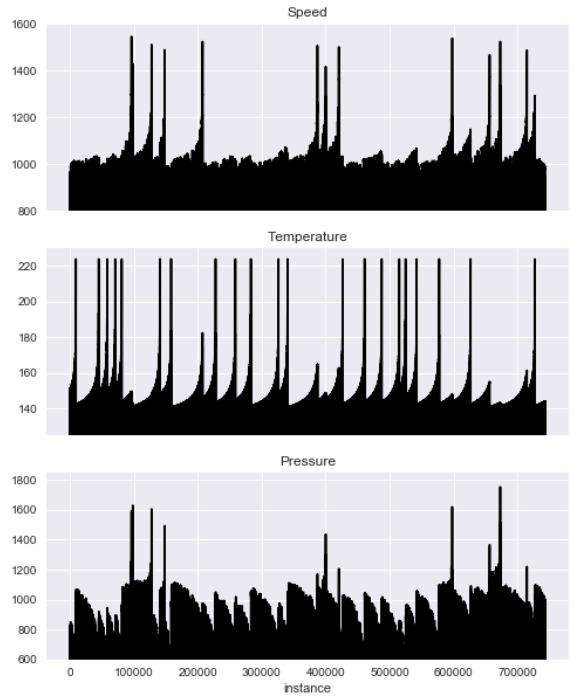

Fig. 18 Representación gráfica de los datos operacionales

Se observan en todas las variables operacionales picos muy marcados. Es muy probable que estos picos estén relacionados directamente con los fallos. Para comprobar que realmente existe esa relación vamos a añadir a estas gráficas la información temporal de los fallos producidos:

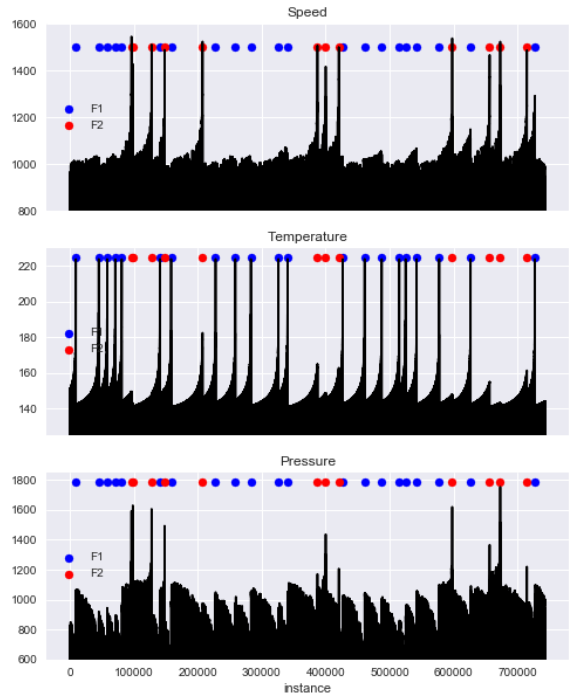

Fig. 19 Datos operacionales junto con los datos de mantenimiento

Efectivamente se puede ver que la hipótesis de que los picos estaban relacionados con los fallos en la máquina era cierta en el caso de la velocidad y la temperatura. Sin embargo, en el caso de la presión se puede ver que, mientras que los fallos de tipo **F2** sí que están relacionados con los picos los fallos de tipo **F1** parece que están relacionados con los valles.

5.3.3 Elección del tipo modelo

El objetivo es poder decidir si en un determinado momento una máquina está en riesgo de rotura o no utilizando únicamente sus datos telemétricos. Dicho de otra manera, se debe clasificar esa máquina como potencialmente peligrosa o como segura. Teniendo en cuenta esto parece claro que será necesario el uso de un modelo de clasificación. Existen multitud de familias de modelos que pueden ser usados para tareas de clasificación: árboles de decisión, máquinas de soporte de vectores, k-vecinos más cercanos, etc. Sin embargo nos hemos decidido por el uso de redes neuronales. Esta elección está motivada principalmente por dos cuestiones:

- Las redes neuronales han demostrado tener un rendimiento excelente en multitud de problemas.

- Aunque en teoría el uso del Federated Learning es aplicable a cualquier tipo de modelo cuyo entrenamiento se base en la optimización de parámetros [15] , es cierto que al ser una tecnología relativamente nueva la mayoría de los frameworks actuales solo permiten el uso de redes neurales en sus implementaciones de aprendizaje federado.

5.3.4 Procesado de los datos

Se dispone del conjunto de datos pero no en el formato adecuado para sacarle el máximo partido. Para que pueda ser usado en el entrenamiento del modelo este debe pasar por un conjunto de transformaciones. A continuación, se describen las transformaciones aplicadas:

Agregación de los datos de telemetría por ciclo

¿Qué son los ciclos?

Muchas máquinas del mundo real funcionan en ciclos. Un ciclo puede considerarse como un período temporal que describe un estado de funcionamiento de una máquina. Por ejemplo, la operación de un motor en un avión puede describirse por los ciclos: motor en funcionamiento (avión en vuelo) o motor apagado (avión en tierra).

¿Porque es importante agregar los datos por ciclo?

Las transmisiones de telemetría sin procesar, si bien pueden ser muy útiles para tareas como el monitoreo en tiempo real, pueden causar problemas a la hora de construir modelos de detección de fallos. Es frecuente que en entornos no controlados (como puede ser una fábrica) los datos no sean del todo precisos. Para mitigar los posibles errores en las mediciones (fallos puntuales en los equipos de medida, ruido, etc.) que pueden afectar a la calidad de los modelos construidos, suele ser una buena opción considerar datos agregados por ciclo.

Se han utilizado dos operaciones de agregación para los datos operaciones (velocidad, temperatura y presión) que han sido: media y máximo.

Cruce de los datos agregados con los datos de mantenimiento

En este punto disponemos de dos conjuntos de datos independientes, pero es necesario juntarlos en un único archivo. Gracias a que se dispone de las marcas de tiempo en las que se registró un fallo de las máquinas y que en el archivo de telemetría existen también esas marcas de tiempo es sencillo relacionar ambos conjuntos de datos y unirlos en un único dataset.

Cálculo del RUL (Remaining Useful Life)

En esta transformación se creará un nuevo campo numérico que indicará en número de ciclos que quedan antes de que se produzca un fallo. A continuación se muestra un ejemplo:

Tabla 5 Cálculo RUL

cycleid	agg1	…	aggN	error	RUL
465	…	…	…		5
466	…	…	…		4
467	…	…	…		3
468	…	…	…		2
469	…	…	…		1
470	…	…	…	F1	0
471	…	…	…		3
472	…	…	…		2
473	…	…	…		1
474	…	…	…	F2	0

En la parte de derecha de la tabla vemos el resultado de las operaciones de transformación de los anteriores pasos. Rodeado en rojo el nuevo campo.

Etiquetado de las instancias

El objetivo es resolver un problema de clasificación por lo que se deben etiquetar las instancias. El proceso para el etiquetado se basa en el método utilizado en [16]. Lo primero será establecer un tamaño de ventana (en este caso se ha utilizado el valor 7) entonces se le asignará la etiqueta 1 o 2 a una instancia cuando su valor RUL sea menor que el tamaño de ventana (1 cuando el fallo sea de tipo F1 y 2 cuando sea de tipo F2) y 0 en otro caso. Esta forma de etiquetado nos permite detectar, no solo las instancias que se corresponden con un fallo sino que también permite identificar las que están muy cerca de ser un fallo. Esto tiene dos ventajas:

- Nos permite tener más instancias de las clases minoritarias[5].
- Detectar un estado de fallo antes de que se produzca. Esto es una ventaja muy interesante a la hora de poder realizar tareas preventivas.

Enriquecimiento de los datos

Una forma de añadir mayor cantidad de información al conjunto de datos disponible es lo que se conoce como feature engineering (ingeniería de características). La ingeniería de características consiste en la creación de nuevos atributos a partir de los ya existentes. En muchos casos este tipo de procedimientos mejora notablemente la calidad de los modelos obtenidos. Aquí se añadirán cuatro nuevos atributos. Cada nuevo atributo se corresponderá con los datos promediados de cierto atributo de las n instancias precedentes. De forma visual:

Tabla 6 Feature engineering

cycleid	f1	…	fN	f1_mean	…	fN_mean
465	…	…	…			
466	…	…	…			
467	…	…	…			
468	…	…	…			
469	…	…	…			
470	…	…	…			
471	…	…	…			
472	…	…	…			
473	…	…	…			
474	…	…	…			

En este caso se ha tomado n=5 y las variables involucradas han sido[6]:

- Temperatura (media y máximo)
- Presión (media y máximo)

[5] Aunque todavía no se ha mencionado, nos encontramos ante un problema de clasificación desbalanceado. Hablaremos de esta problemática en un apartado posterior

[6] Notar que las variables temperatura y presión ya habían sido agregadas en una transformación anterior

Este procedimiento es muy interesante ya que permite añadir una cierta cantidad información histórica a cada registro. El modelo podrá, no solo tener información instantánea si no que tendrá también información sobre la tendencia.

Balanceo del conjunto de datos

Por la propia naturaleza de los datos que estamos manejando es normal que exista una gran diferencia en número entre los casos donde no se detecta ninguna anomalía (clase 0) y los casos donde es posible que se produzca una avería (clase 1 o clase 2). Sin embargo, este tipo de conjuntos de datos suelen ser problemáticos a la hora de entrenar los modelos. Existen multitud de técnicas para balancear conjuntos de datos: oversampling, subsampling, métodos basados en pesos, etc. [17]

Se ha decidido hacer uso del algoritmo SMOTE (Synthetic Minority Over-sampling Technique) [18]. Este algoritmo genera nuevas instancias a partir de las clases minoritarias dejando intacta la clase mayoritaria. Las nuevas instancias no son copias de los casos existentes si no que se calculan como combinaciones lineales de los vecinos más cercanos de esa misma clase.
A continuación, se muestra el efecto de aplicar el algoritmo SMOTE sobre el conjunto de datos:

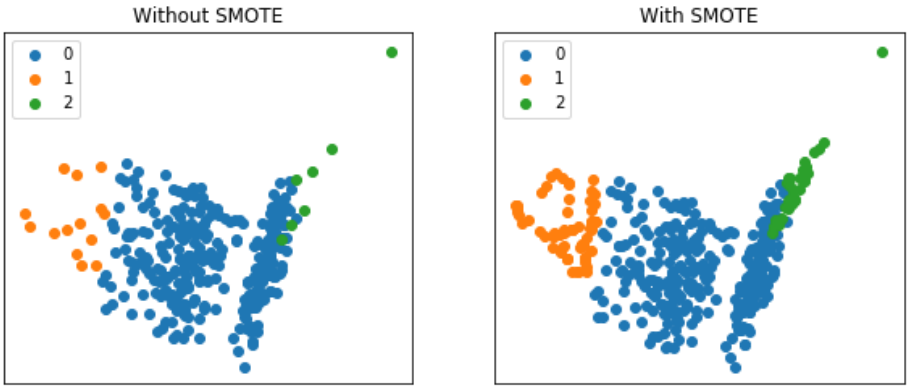

Fig. 20 Efecto de aplicar el algoritmo SMOTE sobre los datos

Debido a la configuración de la simulación y otros factores como por ejemplo, el tamaño de la ventana a la hora de etiquetar los datos, la proporción entre clases puede variar. Sin embargo gracias al uso de SMOTE se puede garantizar que esa proporción se mantendrá constante (se ha establecido una proporción mínima de un 20% de clases minoritarias).

.

Normalización de los datos

A parte del desbalanceo hay otro factor que puede provocar que los modelos no funcionen correctamente, que los datos estén en distintas unidades y que tengan valores demasiado grandes con respecto a los parámetros que el modelo necesita ajustar, esto es especialmente cierto cuando hablamos de redes neuronales [19]. Por este motivo se han normalizado los datos (a cada atributo se le ha restado la media y se ha divido por su desviación típica). De esta manera todos los atributos estarán en la misma escala y además tendrán valores similares a los parámetros entrenables del modelo.

División de los datos para el entrenamiento y la validación del modelo

Esta es la última fase de la preparación del conjunto de datos para el entrenamiento del modelo y su validación. Para cada instalación se construirán dos conjuntos de datos: uno de entrenamiento y otro de validación. Mientras que el primero servirá para determinar el conjunto de parámetros del modelo, el segundo servirá para validar que el modelo se ha entrenado de forma satisfactoria. Es muy importante que entre ambos conjuntos no se comparta ningún tipo de información.

La forma habitual de construir estos dos conjuntos es a partir de un porcentaje. Se determina que porcentaje de datos de todos los disponibles se dedicará al entrenamiento y cual a la validación. Posteriormente se asignan instancias a un conjunto u otro hasta que se cumpla con el porcentaje indicado (el proceso puede ser determinista o aleatorio).

Sin embargo en este caso no se puede aplicar directamente esta técnica ya que debido a la forma de calcular algunos atributos (ver apartado de enriquecimiento de los datos) este método podría provocar una filtración entre ambos conjuntos [20].

Por lo tanto se utilizará la siguiente técnica:

- Se ordenarán los datos de forma temporal

- Se dividirá el conjunto de datos de la siguiente manera: 80% de datos para el entrenamiento y 20% para la validación

- Finalmente se eliminarán del conjunto de entrenamiento los últimos n casos, donde n se corresponde con el valor seleccionado en el apartado de enriquecimiento de datos (en nuestro caso 5)

Este procedimiento garantizara que no existe filtración de datos entre los conjuntos.

5.3.5 Selección del framework para el desarrollo del modelo

En la actualidad existen multitud de librerías que nos permiten la creación de modelos de Machine Learning [21]. Para tomar la decisión de cual elegir se han tenido en cuenta los siguientes factores:

1. Su soporte para trabajar con Federated Learning.
2. El lenguaje de programación que requiere.
3. Su sencillez de uso.

Teniendo en cuenta esos factores se ha decidido que el uso combinado de PyTotch (https://pytorch.org) y PySyft (https://github.com/OpenMined/PySyft) sería lo más conveniente.

5.3.6 Selección de los criterios para la evaluación del modelo

Los criterios de evaluación de un modelo de clasificación dependen mucho del tipo de problema que se esté tratando [22]. Enumeraremos a continuación los criterios más utilizados y veremos cuales son los más idóneos para el problema que se trata de resolver:

***Accuracy*.** Probablemente sea la medida más usada. Es muy fácil de interpretar y se puede utilizar tanto con problemas de clasificación binarios como multiclase. El problema es que su uso no es recomendable para conjuntos no balanceados donde una clase este mucho más representada que las demás.

***Precision*.** Esta medida responde a la siguiente pregunta ¿Qué proporción de instancias clasificadas como "X" son realmente "X"? Se busca maximizar esta medida cuando queremos estar muy seguros de nuestra predicción.

***Recall*.** El recall responde a la siguiente pregunta ¿De todas las instancias "X" que existen, qué proporción han sido clasificadas como "X"? Se busca maximizar esta medida cuando queremos capturar la mayor cantidad de clases "X" posible.

***f1-score*.** Combina las dos medidas anteriores mediante una media armónica. Es muy útil cuando interesa un balance entre ambas medidas.

Desde nuestro punto de vista la medida que debemos tener en cuenta por encima de las demás es el *recall* en las clases 1 y 2 (fallos de tipo F1 y F2 respectivamente). Por una razón: un fallo en una máquina no controlado puede provocar tanto daños materiales graves como daños personales irreparables. En segundo orden de importancia elegiríamos a la f1-score ya que tiene en cuenta tanto la *precision* como el *recall*.

5.3.7 Implementación, entrenamiento y validación del modelo

En esta sección se describirá la construcción del modelo base a partir del conjunto de datos elaborado previamente. Hay que recordar que por los motivos expuestos con anterioridad se usará una red neuronal como modelo base. Dado que la simplicidad siempre una característica muy deseable se empezará por definir un modelo sencillo y comprobar cuál es su desempeño. A continuación se muestra su estructura:

```
class Classifier(nn.Module):

   def __init__(self):
       super().__init__()
       self.fc1 = nn.Linear(10, 5)
       self.fc2 = nn.Linear(5, 5)
       self.fc_out = nn.Linear(5, 3)

   def forward(self, x):

       x = sigmoid(self.fc1(x))
       x = sigmoid(self.fc2(x))
       x = F.log_softmax(self.fc_out (x), dim=1)
       return x
```

Fig. 21 Estructura de la red neuronal (simple)

Como se puede ver la red posee dos capas ocultas de cinco neuronas cada una, la función de activación entre capas es una función sigmoid y para la capa de salida se utiliza la función LogSoftmax que será la que proporcione las probabilidades de

pertenencia a cada clase. Como función de coste se ha utilizado NLLLoss [23] . Los resultados arrojados por este modelo ha sido los siguientes:

	precision	recall	f1-score	support
0	1.00	0.91	0.95	7247
1	0.68	0.99	0.81	651
2	0.39	1.00	0.56	232
accuracy			0.92	8130
macro avg	0.69	0.96	0.77	8130
weighted avg	0.96	0.92	0.93	8130

Fig. 22 Resultados red neuronal (simple)

La siguiente figura muestra diferentes aspectos del entrenamiento de la red:

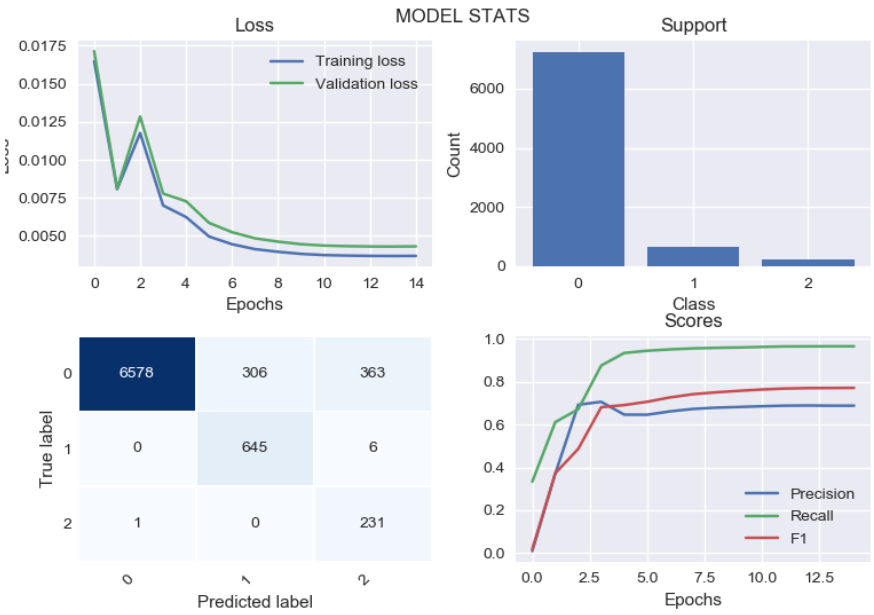

Fig. 23 De izquierda a derecha y de arriba abajo: Evolución de la función de coste, número de instancias de cada clase (test), matriz de confusión y evolución de las principales medidas de calidad del modelo

Podemos ver que el *recall* en las clases 1 y 2 es muy alto (hay que recordad que era la medida que más en cuenta se iba a tener), prácticamente un 100%. Sin embargo, otras medidas como la *precision* son bastante bajas. ¿Qué supondría esto en nuestro caso de estudio? Esto supondría un número bastante alto de falsas alarmas por rotura lo que provocaría por ejemplo, gastos innecesarios en mantenimiento.

Los resultados de esta primera aproximación han sido prometedores pero no del todo satisfactorios se ha decido probar con una red más compleja. A continuación se muestra la estructura de esta nueva red:

```python
class Classifier(nn.Module):

    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(10, 15)
        self.fc2 = nn.Linear(15, 15)
        self.fc3 = nn.Linear(15, 15)
        self.fc4 = nn.Linear(15, 10)
        self.fc_out = nn.Linear(10, 3)

    def forward(self, x):

        x = sigmoid(self.fc1(x))
        x = sigmoid(self.fc2(x))
        x = sigmoid(self.fc3(x))
        x = sigmoid(self.fc4(x))
        x = F.log_softmax(self.fc_out(x), dim=1)
        return x
```

Fig. 24 Estructura de la red neuronal (compleja)

Como se puede ver esta red está formada por cuatro capas ocultas. Cada una de ellas está formada por 15 neuronas salvo la última que esta compuesta por 10 neuronas. Como función de activación se ha utilizado la función sigmoid excepto para la capa de salida en la que se utiliza la función LogSoftmax para el cálculo de probabilidades. Como función de coste se ha utilizado la función NLLLoss.

Este aumento en complejidad lleva asociado una mejora en el rendimiento como se puede ver en los resultados presentados a continuación:

	precision	recall	f1-score	support
0	1.00	0.98	0.99	7247
1	0.88	0.97	0.92	651
2	0.87	0.97	0.91	232
accuracy			0.98	8130
macro avg	0.91	0.97	0.94	8130
weighted avg	0.98	0.98	0.98	8130

Fig. 25 Resultados red neuronal (compleja)

Se han mejorado prácticamente la totalidad de las medidas con respecto al modelo anterior. Ha bajado levemente el *recall* aunque este descenso entra dentro de lo aceptable. En la siguiente figura se muestran gráficamente los resultados obtenidos:

Fig. 26 De izquierda a derecha y de arriba abajo: Evolución de la función de coste, número de instancias de cada clase (test), matriz de confusión y evolución de las principales medidas de calidad

Esta red neuronal ha tenido un rendimiento notable y por este motivo será usada como modelo base para su aplicación en los procedimientos descritos en la sección 4.

5.4 Aplicación del modelo base

Se dispone de la estructura del modelo base. El siguiente paso será aplicar este modelo en todas las instalaciones simuladas para comprobar su rendimiento en estas. El objetivo es disponer de una medida base que nos permita evaluar de una forma fidedigna el rendimiento del modelo de aprendizaje federado, objetivo final de este trabajo. Es importante destacar que el procedimiento para la preparación de los datos de todas las instalaciones sigue un patrón idéntico al descrito en la sección anterior. Se mostrarán en las siguientes subsecciones los resultados obtenidos para cada planta.

5.4.1 Instalación “Piloto”

Fig. 27 Estadísticas planta "Piloto"

	precision	recall	f1-score	support
0	1.00	0.98	0.99	7247
1	0.88	0.97	0.92	651
2	0.87	0.97	0.91	232
accuracy			0.98	8130
macro avg	0.91	0.97	0.94	8130
weighted avg	0.98	0.98	0.98	8130

Fig. 28 Scores planta “Piloto”

Los datos de la planta “Piloto” son los que se han utilizado en la construcción del modelo base y como se ha comentado anteriormente se han obtenido unos valores altos tanto para el *recall* como para la *precision* lo que se ve reflejado en el *f1-score*.

5.4.2 Instalación “A”

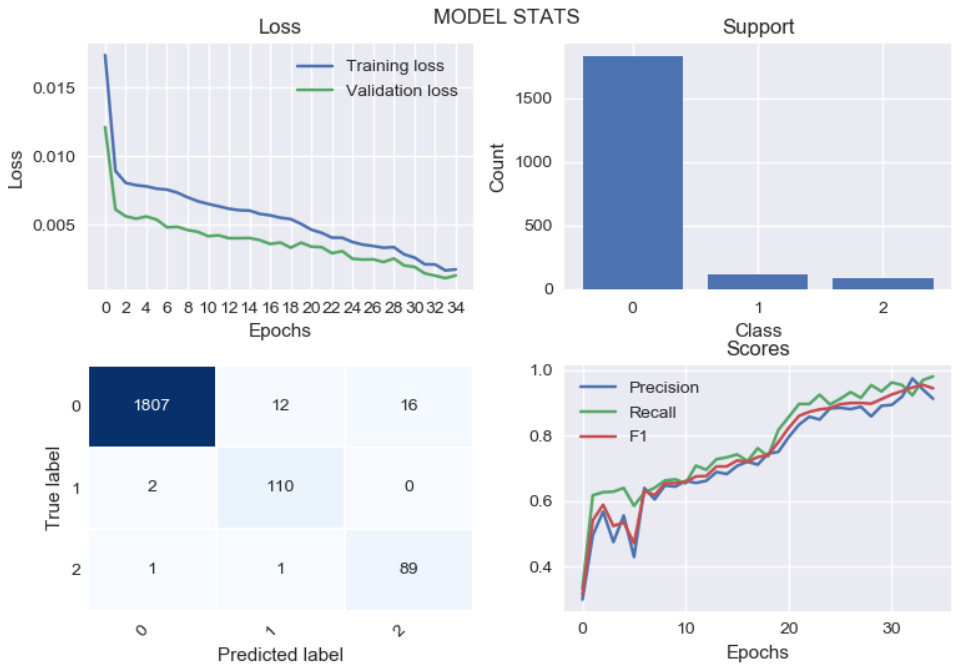

Fig. 29 Estadísticas planta "A"

	precision	recall	f1-score	support
0	1.00	0.98	0.99	1835
1	0.89	0.98	0.94	112
2	0.85	0.98	0.91	91
accuracy			0.98	2038
macro avg	0.91	0.98	0.95	2038
weighted avg	0.99	0.98	0.98	2038

Fig. 30 Scores planta “A”

Se puede observar que los resultados que arroja el modelo entrenado con los datos de la instalación “A” son prácticamente idénticos a los obtenidos en el entrenamiento de la planta “Piloto”. Es posible que si hubiéramos aumentado el número de epochs hubieran mejorado los resultados ya que si nos fijamos en la evolución de las curvas de coste, parece no haberse alcanzado el punto de estabilidad.

5.4.3 Instalación “B”

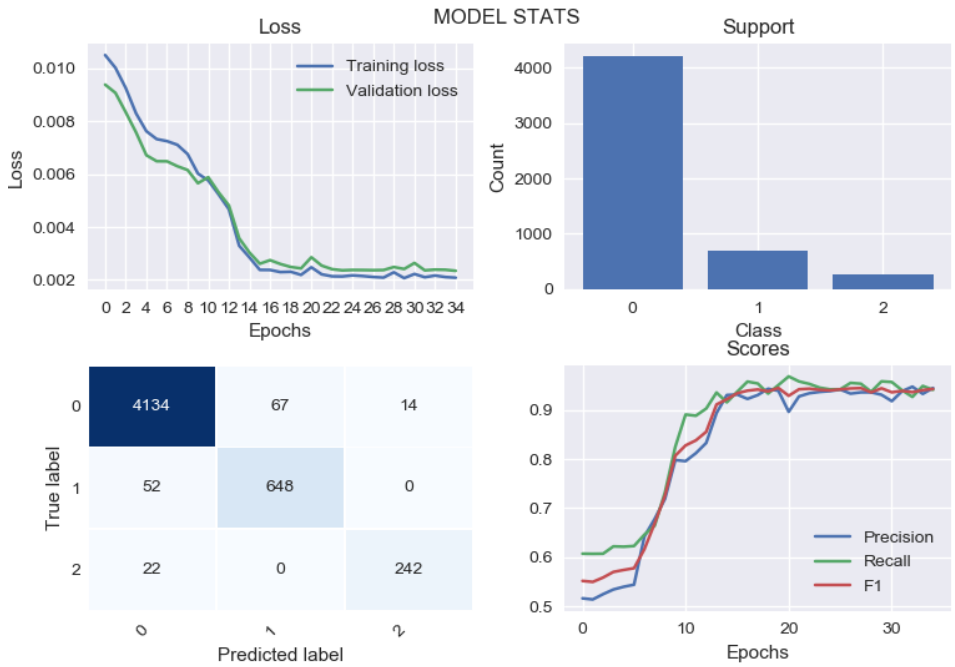

Fig. 31 Estadísticas planta "B"

	precision	recall	f1-score	support
0	0.98	0.98	0.98	4215
1	0.91	0.93	0.92	700
2	0.95	0.92	0.93	264
accuracy			0.97	5179
macro avg	0.94	0.94	0.94	5179
weighted avg	0.97	0.97	0.97	5179

Fig. 32 Scores planta “B”

En este entrenamiento se observa un ligero aumento de la *precision* en detrimento del *recall*. Vemos que ambos se compensan al ya que el *f1-score* se mantiene prácticamente igual que en los casos anteriores.

5.4.4 Instalación “N”

Fig. 33 Estadísticas planta "N"

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1966
1	0.89	0.90	0.89	154
2	0.83	0.88	0.85	56
accuracy			0.98	2176
macro avg	0.90	0.92	0.91	2176
weighted avg	0.98	0.98	0.98	2176

Fig. 34 Scores planta “N”

De todos los casos este podría considerarse el peor de todos ya que el *recall* y la *precision* han disminuido de forma notable con respecto al resto de fábricas. Al igual que ocurría con la instalación “A”, los resultados probablemente hubieran mejorado al aumentar el número de epochs.

5.5 Intercambiabilidad del modelo base

En la sección anterior se ha visto que se puede esperar del modelo base si se entrenase de manera local. En este apartado se pretende responder a la siguiente pregunta: ¿Funciona bien un modelo entrenado en una planta en otra sin necesidad de reentrenarlo? Para resolver esta duda presentaremos a continuación una tabla comparativa del *f1-score* (media de todas las clases) sobre todas las posibles combinaciones train-test. La razón de elegir esta medida es por simplicidad, ya que con un único valor podemos hacernos una idea tanto de la *precision* como del *recall*.

Tabla 7 Train-test f1-score (mean)

		Test			
		Piloto	**A**	**B**	**N**
Train	**Piloto**	0.94	0.91	0.82	0.88
	A	0.89	0.95	0.79	0.84
	B	0.80	0.81	0.94	0.73
	N	0.92	0.93	0.76	0.91

Observando la tabla se puede ver que por norma general el modelo entrenado de forma específica (diagonal) supera ampliamente en rendimiento a los entrenados en otras instalaciones. Se debe destacar el caso de la instalación “N” en la que para algunos casos el score es mucho mejor que para su propio conjunto de entrenamiento. El motivo seguramente guarde relación con el hecho de que el número de epochs no fuera lo suficientemente alto. Aunque este hecho aislado fuera generalizado existiría otro problema, ¿cómo saber a priori que modelo de todos de los que se dispone es mejor para la planta objetivo?

Teniendo en cuenta los datos y reflexiones anteriores no parece posible traspasar un modelo de una instalación a otra sin que esto tenga una repercusión negativa en la calidad de las predicciones del modelo.

5.6 Construcción y aplicación del modelo de aprendizaje federado

Una de las cualidades del Federated Learning es que permite compartir el conocimiento entre plantas manteniendo la privacidad de los datos. Para demostrar que esto es posible se ha entrenado un modelo federado que utiliza únicamente tres de las cuatro instalaciones (“Piloto”, “A” y “B”) para entrenarse pero que se evalúa en las cuatro. Los resultados obtenidos se muestran en los siguientes aparatados.

5.6.1 Instalación “Piloto”

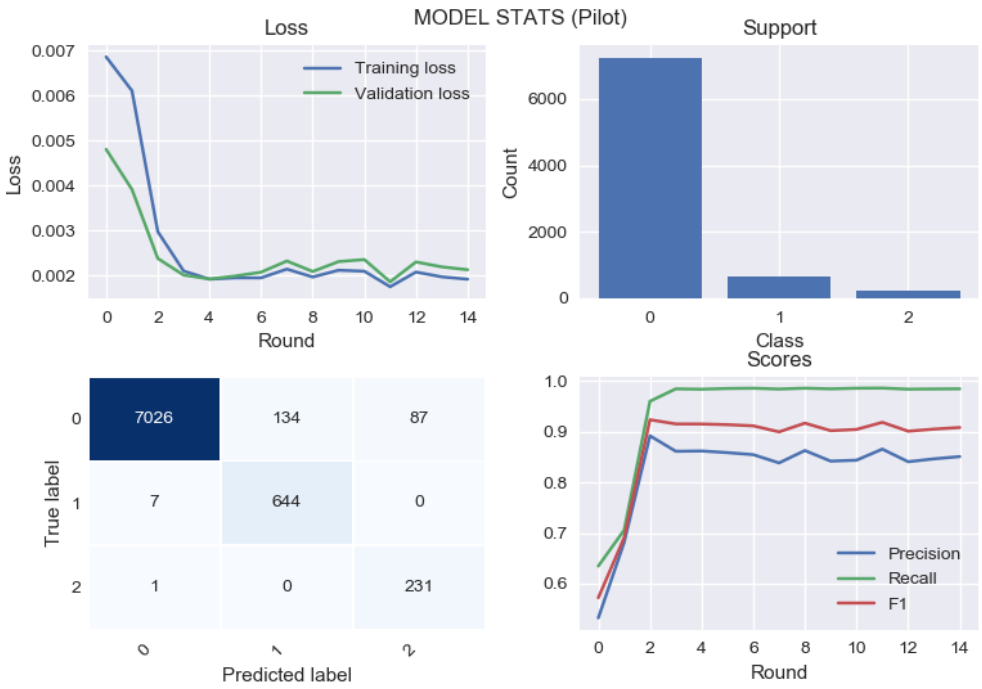

Fig. 35 Estadísticas planta "Piloto" (Federated Learning)

	precision	recall	f1-score	support
0	1.00	0.97	0.98	7247
1	0.82	0.99	0.90	651
2	0.72	1.00	0.84	232
accuracy			0.97	8130
macro avg	0.85	0.98	0.91	8130
weighted avg	0.98	0.97	0.97	8130

Fig. 36 Scores planta “Piloto” (Federated Learning)

Aunque los scores no son tan buenos como los del entrenamiento de forma local vemos que se mantienen en un rango aceptable.

5.6.2 Instalación “A”

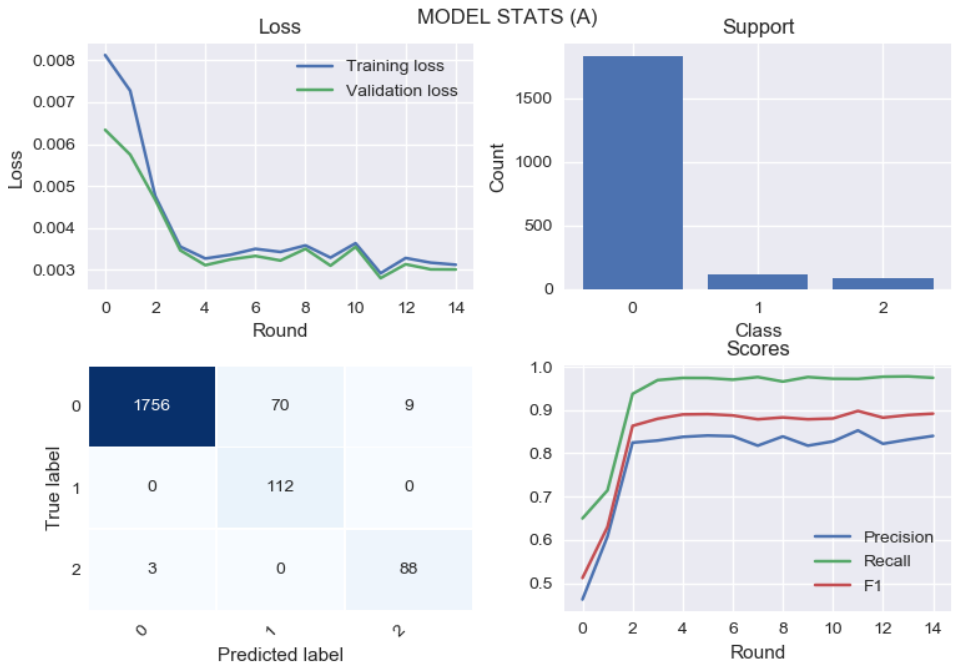

Fig. 37 Estadísticas planta "A" (Federated Learning)

	precision	recall	f1-score	support
0	1.00	0.96	0.98	1835
1	0.62	1.00	0.76	112
2	0.94	0.97	0.95	91
accuracy			0.96	2038
macro avg	0.85	0.98	0.90	2038
weighted avg	0.97	0.96	0.97	2038

Fig. 38 Scores planta “A” (Federated Learning)

No encontramos ante una situación muy parecida al caso anterior los scores bajan pero aun así siguen estando en valores aceptables.

5.6.3 Instalación “B”

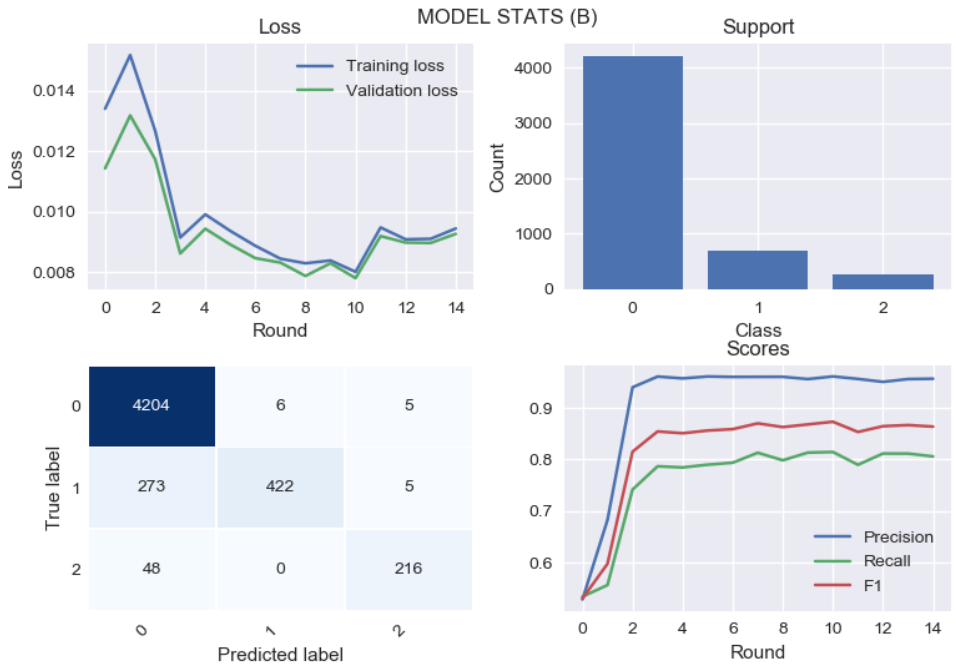

Fig. 39 Estadísticas planta "B" (Federated Learning)

	precision	recall	f1-score	support
0	0.93	1.00	0.96	4215
1	0.99	0.61	0.75	700
2	0.96	0.82	0.88	264
accuracy			0.94	5179
macro avg	0.96	0.81	0.87	5179
weighted avg	0.94	0.94	0.93	5179

Fig. 40 Scores planta “B” (Federated Learning)

En el caso de esta instalación los valores de *recall* que en el entrenamiento local ya eran relativamente bajos bajan aún más siendo sin duda los peores valores de todas las instalaciones.

5.6.4 Instalación “N”

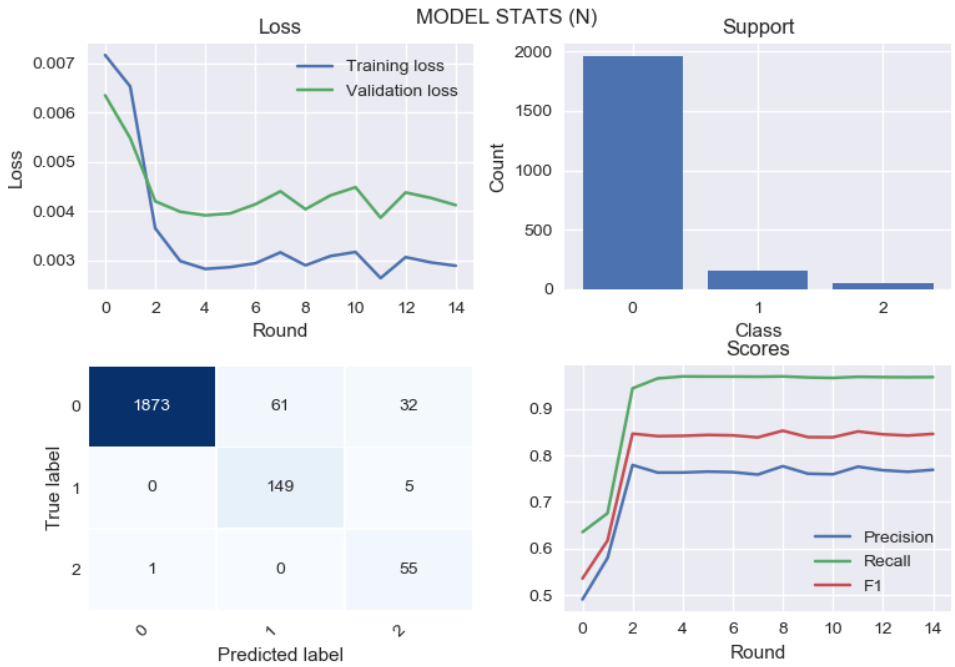

Fig. 41 Estadísticas planta "N" (Federated Learning)

	precision	recall	f1-score	support
0	1.00	0.95	0.98	1966
1	0.71	0.97	0.82	154
2	0.60	0.98	0.75	56
accuracy			0.95	2176
macro avg	0.77	0.97	0.85	2176
weighted avg	0.97	0.95	0.96	2176

Fig. 42 Scores planta “N” (Federated Learning)

Es importante notar que esta instalación no participo en la fase de entrenamiento y eso explicaría los valores tan bajos de *f1-score* y que la curva de la función de coste del conjunto test esté por encima de la del entrenamiento. Aun así son valores bastante buenos.

5.7 Comparación entre las diferentes aproximaciones

A continuación se muestra una tabla de resultados obtenidos:

Tabla 8 Comparación entre las diferentes aproximaciones

	Instalación			
	Piloto	**A**	**B**	**N**
Entrenamiento local	0.94	0.95	0.94	0.91
Otra instalación (peor)	0.80 (B)	0.81 (B)	0.76 (B)	0.73 (B)
Otra instalación (mejor)	0.92 (N)	0.93 (N)	0.82 (Piloto)	0.88 (Piloto)
Federated Learning	0.91	0.90	0.87	0.85

Como se lógico ambas alternativas presentan resultados peores al entrenamiento local que sería el caso ideal. Por otro lado, se puede ver que el método de aprendizaje federado es siempre mejor que el peor de los casos cuando se utiliza otra instalación, incluso en algunos casos supera a la mejor de las opciones. Hay que tener en cuenta que aunque en el mejor de los casos de usar el modelo de otra instalación supera al aprendizaje federado, nos encontramos con el problema adicional de encontrar, a priori, cual de todas las instalaciones disponibles será la más adecuada. Por lo tanto, parece claro que el uso del Federated Learning es de gran utilidad en situaciones como la descrita en este trabajo.

6. Conclusiones

Hay que recordar que el objetivo de proyecto es explorar el posible uso del Federated Learning para la detección de eventos anómalos dentro de un entorno industrial. Para ello se ha descrito un escenario (o caso de uso) que podría corresponderse con las necesidades de una compañía multinacional como podría ser una compañía siderúrgica, minera, un fabricante de productos químicos, etc. En este caso de uso se han expuesto las limitaciones existentes en cuanto a la distribución de datos entre las distintas instalaciones en el ámbito de una organización con una gran dispersión geográfica y se ha puesto de manifiesto la necesidad de una rápida implantación de modelos de Machine Learning en instalaciones de nueva creación.

Para la solucionar el problema presentado se han comparado dos soluciones:

- Una basada en la intercambiabilidad de modelos
- Una basada en el uso del Federated Learning

Se ha mostrado que el método basado en la intercambiabilidad entre modelos puede resultar de utilidad pero añade complejidad al problema. Es necesario crear un método para decidir la planta de origen del modelo ya que como se ha visto, no todos los modelos ofrecen los mismos resultados. Otro problema que habría que añadir a esta alternativa es la propiedad del modelo, una planta podría exigir a otra algún tipo de contrapartida por la cesión del modelo creado con sus datos.

El método basado en Federated Learning ha demostrado ser más eficaz por dos motivos:

- Ofrece resultados similares a la solución óptima basada en el intercambio de modelos y siempre resultados mejores que la peor de las soluciones de intercambiabilidad.

- Todos los participantes son responsables en la creación del modelo por lo que nadie es propietario exclusivo de este.

Gracias a la planificación realizada al inicio de este proyecto se ha podido concluir en tiempo y forma. Sin embargo, también es cierto que debido a que se ha desarrollado con tecnologías relativamente nuevas, los frameworks existentes (en particular el framework pySyft) no disponen de todas las funcionalidades deseables ni de mucha documentación. Por este motivo ha sido necesario tanto estudiar el código fuente para comprender su funcionamiento como modificarlo para obtener las funcionalidades que se consideraban imprescindibles para el desarrollo de este trabajo.

Debido a las particularidades de cada tipo de instalación puede ser complicado que los modelos construidos aquí puedan aplicarse directamente en entornos del mundo real. Posibles líneas de trabajo futuro podrían ser la aplicación de los métodos aquí descritos en entornos industriales reales y no simulados.

7. Bibliografía

[1] Z. Janjua, M. Vecchio, M. Antonini y F. Antonelli, «IRESE: An intelligent rare-event detection system using unsupervised learning on the IoT edge,» *Engineering Applications of Artificial Intelligence,* vol. 84, pp. 41-50, 2019.

[2] S. Chauhan y L. Vig, «Anomaly detection in ECG time signals via deep long short-term memory networks,» de *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Paris, 2015.

[3] A. Lakhina, M. Crovella y C. Diot, «Diagnosing Network-Wide Traffic Anomalies,» *Computer Communication Review,* vol. 34, 2004.

[4] D. Li, D. Chen, J. Goh y S. k. Ng, «Anomaly Detection with Generative Adversarial Networks for Multivariate Time Series,» de *International Workshop on Big Data*, London, 2018.

[5] S. M. Erfani, S. Rajasegarar, S. Karunasekera y C. Leckie, «High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning,» *Pattern Recognition,* vol. 58, pp. 121-134, 2016.

[6] F. T. Liu, K. Ting y Z.-H. Zhou, «Isolation-Based Anomaly Detection,» *ACM Transactions on Knowledge Discovery From Data - TKDD,* vol. 6, pp. 1-39, 2012.

[7] S. Hariri, M. Carrasco Kind y R. J. Brunner, «Extended Isolation Forest,» 2018.

[8] S. Hawkins, H. He, G. Williams y R. Baxter, «Outlier Detection Using Replicator Neural Networks,» *Data Warehousing and Knowledge Discovery,* pp. 170-180, 2002.

[9] H. B. McMahan, E. Moore, D. Ramage, S. Hampson y B. Agüera y Arcas, «Communication-Efficient Learning of Deep Networks from Decentralized Data,» 2016.

[10] Q. Yang, Y. Liu, T. Chen y Y. Tong, «Federated Machine Learning: Concept and Applications,» 2019.

[11] A. Gonfalonieri, «Towards Data Science,» [En línea]. Available: https://towardsdatascience.com/federated-learning-a-new-ai-business-model-ec6b4141b1bf.

[12] M. J. Sheller, G. A. Reina, B. Edwards, J. Martin y S. Bakas, «Multi-Institutional Deep Learning Modeling Without Sharing Patient Data: A Feasibility Study on Brain Tumor Segmentation,» 2018.

[13] U. I. d. Valencia, «www.universidadviu.es,» [En línea]. Available: https://www.universidadviu.es/python-para-big-data-motivos-para-elegirlo/. [Último acceso: 15 12 2019].

[14] K. G. D. S. N. E. A. Saxena, «Damage propagation modeling for aircraft engine run-to-failure simulation,» de *International Conference on Prognostics and Health Management,*, Denver, CO, USA, 2008.

[15] F. Hartmann, «Federated Learning,» [En línea]. Available: https://florian.github.io/federated-learning/. [Último acceso: 15 10 2019].

[16] C. Yang y S. Létourneau, «Learning to predict train wheel failures,» de *KDD '05*, 2005.

[17] machinelearningmastery.com, «8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset,» [En línea]. Available: https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/. [Último acceso: 5 12 2019].

[18] K. W. Bowyer, N. V. Chawla, L. O. Hall y P. Kegelmeyer, «SMOTE: Synthetic Minority Over-sampling Technique,» *CoRR,* vol. abs/1106.1813, 2011.

[19] T. Stöttner, «Why data should be normalized before training a neural network,» [En línea]. Available: https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d. [Último acceso: 10 12 2019].

[20] Microsoft, «Azure AI guide for predictive maintenance solutions,» [En línea]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/cortana-analytics-playbook-predictive-maintenance#time-dependent-split. [Último acceso: 5 12 2019].

[21] O. Kharkovyna, «Top 10 Best Deep Learning Frameworks in 2019,» towardsdatascience.com, 3 Jan 2019. [En línea]. Available: https://towardsdatascience.com/top-10-best-deep-learning-frameworks-in-2019-5ccb90ea6de. [Último acceso: 29 11 2019].

[22] R. Agarwal, «The 5 Classification Evaluation metrics every Data Scientist must know,» towardsdatascience.com, 17 Sep 2019. [En línea]. Available: https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226. [Último acceso: 1 12 2019].

[23] PyTorch, «pytorch.org,» [En línea]. Available: https://pytorch.org/docs/stable/nn.html#nllloss. [Último acceso: 10 12 2019].

[24] Wikipedia, «Federated Learning, Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Federated_learning. [Último acceso: 1 12 2019].

[25] Microsoft, «Predictive Maintenance,» [En línea]. Available: https://gallery.azure.ai/Collection/Predictive-Maintenance-Template-3. [Último acceso: 16 11 2019].

8. Anexos

Anexo A. Descripción del código fuente

Todo el código fuente, además del conjunto de archivos de configuración utilizados durante el desarrollo del proyecto se puede encontrar en un repositorio público en la siguiente dirección (https://github.com/Borogum/masterdatacience.git). El código fuente se divide en cinco partes:

- Generación
- Procesado
- Preparado
- Creación del modelo base
- Creación del modelo federado

En las siguientes secciones se describirá detalladamente cada una de las partes.

Generación (generation)

Esta parte es la encargada de simular los datos de las instalaciones. Dentro de este paquete podemos encontrar cuatro módulos:

- simulation.py
- notify.py
- generate_data.py
- generate_all_data.py

simulation.py

Contiene las clases necesarias para poder realizar las simulaciones de las instalaciones:

- **BrokenMachine:** Excepción que será lanzada cuando una máquina llegue a un punto de ruptura.
- **Clock:** Reloj, cada "tick" será un segundo.
- **HealthIndex:** Permite simular el desgaste de un componente.
- **Machine:** Simula el funcionamiento de una máquina.
- **Facility:** Representa una instalación.

notify.py

Este módulo contiene un conjunto de clases que permiten la comunicación (telemetría) de una máquina con otros dispositivos (archivos, consola, etc.). Se han implementado tres clases:

- **ConsoleNotifier:** Muestra los datos por consola.
- **ParquetNotifier:** Almacena los datos en el formato Apache Parquet (https://parquet.apache.org/).
- **CsvNotifier:** Almacena los datos en formato csv (clase usada por defecto).

generate_data.py

Este script realiza una simulación y almacena los resultados (telemetría y mantenimiento) en archivos csv. Las características de la instalación a simular deben ser establecidas en un archivo de configuración. La forma de invocar este script es la siguiente:

```
python generate_data.py plant.cfg
```

Un ejemplo de archivo de configuración podría ser el siguiente:

```
[CONFIGURATION]

name = Plant
; aprox. three months 3*30*24*3600
simulation_time = 7776000
; ambient temperature
temperature = 10
; ambient pressure
pressure = 98
; cycle characteristics
cycle_length_min = 1
cycle_length_max = 3
cycle_duration = 120
; working machines per period
machines_per_batch = 5
; machine speed limits
operational_speed_min = 1000
operational_speed_max = 1100
; period time
batch_time = 3600
; total number of machines
machine_count = 15
; times to failure
ttf1_min = 7000
ttf1_max = 40000
ttf2_min = 1000
ttf2_max = 50000
; max working temperature
temperature_max = 100
; relation between pressure and speed
pressure_factor = 1.5
; output paths
telemetry_path = plant/telemetry
event_path =  plant/event
```

Con la configuración anterior, se almacenarían los datos de telemetría (por máquina) en la carpeta "plant/telemetry/" y los datos de mantenimiento (también por máquina) en la carpeta "plant/event/".

generate_all_data.py

Este script de conveniencia admite como parámetro de entrada una carpeta. El código escanea esta carpeta en busca de archivos de configuración. Para cada archivo encontrado se ejecutará la simulación correspondiente. Las simulaciones se realizarán de forma paralelizada lo que acelerará el proceso de generación de datos.

Procesado (processing)

Para el procesado se utiliza un único script: "process.py". Este script procesará los datos obtenidos de la etapa de generación. Las trasformaciones que realizará sobre los datos serán las siguientes:

- Unificar los datos de mantenimiento y telemetría en un único archivo (por máquina).
- Agregar los datos a nivel de ciclo.
- Etiquetar las instancias.
- Calcular agregaciones temporales.

La forma de invocar este script es la siguiente:

```
python process.py process.cfg
```

El archivo de configuración deberá tener la siguiente estructura:

```
[CONFIGURATION]

; minimum distance between consecutive cycles
gap = 30
; cycles before fail
w = 7
; number of cycles for rolling stats
rolling = 5
; path to load data
path_in = data/generation/
; ath to store processed data
path_out = data/processing/
```

Preparación (preparation)

Como en el caso anterior, se ha utilizado un único script que realizará toda la tarea de preparación de los datos: "prepare.py". Esta última fase en la transformación de los datos estará centrada en construir un dataset cuyo objetivo es el proceso de entrenamiento y validación de los modelos de Machine Learning. Concretamente las tareas que realizará serán las siguientes:

- Unión de los datos de todas las máquinas en un único archivo.
- División del dataset en dos conjuntos: uno será utilizado para el entrenamiento y otro para la validación.
- Aplicación del método SMOTE para el balanceo de las clases.
- Normalización de los datos.

La forma de invocarlo será la siguiente:

```
python prepare.py preparation.cfg
```

El archivo de configuración deberá tener la siguiente estructura:

```
[CONFIGURATION]

; test size percentage
test_size = 0.2
; Look back it must be equal to rolling value in processing stage
lookback = 5
; Proportion of minor classes (for each class)
p = 0.1
; folders
path_in = data/processing/
path_out = data/preparation/
```

Creación del modelo base (single_model)

Este paquete contiene los módulos necesarios para la creación del modelo que, en una fase posterior, se utilizará como modelo base para la construcción del modelo federado. Este paquete está compuesto por los siguientes módulos:

- datasets.py
- model.py
- utils.py
- workers.py
- start_worker.py
- train_and_validate.py

datasets.py

Este módulo contiene una única clase llamada **MachineDataset** que hereda de la clase **Dataset** (pyTorch). Esta clase facilitará la carga de los datos ya que permitirá la abstracción de la estructura subyacente de los datos.

model.py

Este módulo contiene dos objetos:

- La clase que describe al modelo de clasificación denominada **Classifier**.
- La función de coste que se utilizará para el ajuste del modelo cuyo nombre será **loss_fn**.
- **utils.py**

La finalidad de este módulo es proporcionar funciones que, sin tener un objetivo específico, serán utilizados de manera transversal por varios paquetes. El módulo contiene dos funciones:

- **cm2pred** - Esta función transforma una matriz de confusión en dos vectores. Uno de ellos contendrá las etiquetas que se suponen ciertas (y_true) y el otro las etiquetas predichas por el modelo (y_pred).

- **show_results** - Dado un histórico de matrices de confusión y costes (train y test) crea una representación gráfica de estas. Además, calcula una serie de estadísticas como: *precision*, *f1-score*, *recall*, etc.

workers.py

Dado que el framework utilizado (pySyft) no proporciona ciertas estadísticas que se consideraban de interés a la hora de evaluar los modelos fue necesario añadirlas. Para ello se crearon dos clases:

- **CustomWebsocketClientWorker**
- **CustomWebsocketServerWorker**

que heredan de:

- **WebsocketClientWorker**
- **WebsocketServerWorker**

respectivamente. Estas clases además de todas las estadísticas que proporcionaban las clases originales proporcionan también como método de evaluación la matriz de confusión.

start_worker.py

El objetivo de este script es poner en marcha un **CustomWebsocketServerWorker** con los parámetros especificados en línea de comandos. Su sintaxis sería el siguiente:

```
python start_worker.py id host port train_data test_data --verbose
```

Donde:

- id: es el nombre del servidor
- host: ip del servidor
- port: puerto por el que escuchara el servidor
- train_data: archivo que contiene los datos de entrenamiento
- test_data: archivo que contiene los datos de test
- --verbose: Es un parámetro adicional que controla los mensajes que se muestran por consola

Un ejemplo de uso podría ser el siguiente:

```
python start_worker.py  server 127.0.0.1 8777 "data/train.csv"
"data/test.csv"
```

train_and_validate.py

Este script entrena y valida el modelo implementado en **model.py**. Los parámetros de entrenamiento y del worker encargado de realizar la tarea son especificados en un archivo de configuración. A continuación, se muestra un ejemplo de uso:

```
python train_and_validate.py configuration.cfg
```

Un archivo de configuración podría ser el siguiente:

```
[CONFIGURATION]

;Worker config
worker_id = Pilot
host = 127.0.0.1
```

```
port = 8777
verbose = 0
; Train config
epochs = 35
batch = 32
optimizer = Adam
lr = 0.002
shuffle = 1
; Data config
train = data/train.csv
test = data/test.csv
```

Creación del modelo federado (federated_model)

Para la creación del modelo federado se han diseñado dos scripts:

- start_workers.py
- train_and_validate.py

start_workers.py

Este es el script encargado de iniciar los workers que participarán en la construcción y en la validación del modelo federado. Su invocación se realizaría de la siguiente manera:

```
python start_workers workers.cfg
```

Donde el archivo de configuración debería tener una estructura similar a la siguiente:

```
[WORKER 0]
id = Pilot
host = 127.0.0.1
port = 8800
verbose = 0
train = ../data/preparation/pilot/train.csv
test = ../data/preparation/pilot/test.csv

[WORKER 1]
id = A
host = 127.0.0.1
port = 8801
verbose = 0
train = ../data/preparation/A/train.csv
test = ../data/preparation/A/test.csv

[WORKER 2]
id = B
host = 127.0.0.1
port = 8802
verbose = 0
train = ../data/preparation/B/train.csv
test = ../data/preparation/B/test.csv

[WORKER 3]
id = N
host = 127.0.0.1
port = 8803
verbose = 0
```

```
train = ../data/preparation/N/train.csv
test = ../data/preparation/N/test.csv
```

train_and_validate.py

Este script funciona de forma análoga al descrito en la sección anterior. Su ejecución se realizaría de la siguiente manera:

```
python train_and_validate configuration.cfg
```

Donde el archivo de configuración aunque guarda ciertas similitudes con el descrito anteriormente tiene una estructura propia. Un ejemplo de configuración podría ser la siguiente:

```
[TRAIN]
rounds = 15
epochs = 35
federate_after = 5
batch = 32
optimizer = Adam
lr = 0.002
shuffle = 1

[WORKER 0]
id = Pilot
host = 127.0.0.1
port = 8800
verbose = 0
federation_participant = 1

[WORKER 1]
id = A
host = 127.0.0.1
port = 8801
verbose = 0
federation_participant = 1

[WORKER 2]
id = B
host = 127.0.0.1
port = 8802
verbose = 0
federation_participant = 1

[WORKER 3]
id = N
host = 127.0.0.1
port = 8803
verbose = 0
federation_participant = 0
```

Es importante destacar que el atributo **federation_participant** de las secciones referidas a los workers permite controlar la participación del worker en la construcción de modelo y en su validación. El valor 1 indica que ese worker participará en la creación del modelo y en su validación mientras que el valor 0 indicará que únicamente participará en la fase de validación.

Anexo B. Archivos de configuración para las simulaciones

Planta “Piloto”

```
[CONFIGURATION]

name = Pilot
; aprox. six months 6*30*24*3600
simulation_time = 15552000
temperature = 20
pressure = 101
cycle_length_min = 1
cycle_length_max = 10
cycle_duration = 60
machines_per_batch = 10
operational_speed_min = 800
operational_speed_max = 900
batch_time = 3600
machine_count = 20
ttf1_min = 5000
ttf1_max = 50000
ttf2_min = 500
ttf2_max = 90000
temperature_max = 125
pressure_factor = 1.6
telemetry_path = ../data/generation/pilot/telemetry
event_path = ../data/generation/pilot/event
```

Planta “A”

```
[CONFIGURATION]

name = A
; aprox. three months 3*30*24*3600
simulation_time = 7776000
temperature = 10
pressure = 98
cycle_length_min = 1
cycle_length_max = 3
cycle_duration = 120
machines_per_batch = 5
operational_speed_min = 1000
operational_speed_max = 1100
batch_time = 3600
machine_count = 15
ttf1_min = 7000
ttf1_max = 40000
ttf2_min = 1000
ttf2_max = 50000
temperature_max = 100
pressure_factor = 1.5
telemetry_path = ../data/generation/A/telemetry
event_path = ../data/generation/A/event
```

Planta “B”

```
[CONFIGURATION]

name = B
; aprox. four months 4*30*24*3600
simulation_time = 10368000
temperature = 25
pressure = 103
cycle_length_min = 1
cycle_length_max = 20
cycle_duration = 60
machines_per_batch = 10
operational_speed_min = 1100
operational_speed_max = 1200
batch_time = 3600
machine_count = 25
ttf1_min = 5000
ttf1_max = 55000
ttf2_min = 700
ttf2_max = 100000
temperature_max = 140
pressure_factor = 1.8
telemetry_path = ../data/generation/B/telemetry
event_path = ../data/generation/B/event
```

Planta “N”

```
[CONFIGURATION]

name = N
; aprox. two months 2*30*24*3600
simulation_time = 5184000
temperature = 22
pressure = 100
cycle_length_min = 1
cycle_length_max = 9
cycle_duration = 60
machines_per_batch = 8
operational_speed_min = 1050
operational_speed_max = 1200
batch_time = 3600
machine_count = 10
ttf1_min = 5000
ttf1_max = 50000
ttf2_min = 500
ttf2_max = 90000
temperature_max = 120
pressure_factor = 1.65
telemetry_path = ../data/generation/N/telemetry
event_path = ../data/generation/N/event
```