Facultad de Ciencias

Aplicación de técnicas de ML para la optimización de compresores de aire
(ML techniques for air compressors optimization)

Trabajo de Fin de Máster
para acceder al

MÁSTER EN CIENCIAS DE DATOS

Autor: Marcos San Miguel

Director\es: Lara Lloret Iglesias y Rubén Rivas

Septiembre - 2021

Resumen

La industria 4.0 ha llegado con nuevas técnicas para la mejora de procesos de automatización y optimización de recursos en el mundo empresarial. En la empresa Velfair, S.A. (Requejada, Cantabria) se ha llevado a cabo un proyecto en el que se ha desarrollado una plataforma big data capaz de administrar y analizar los datos procedentes de los compresores de aires que comercializa la empresa. Este trabajo presenta el análisis y la construcción de modelos para la optimización del mantenimiento de estos compresores. Se ha presentado un flujo de datos mediante la herramienta Airflow capaz de analizar los datos procedentes del diferencial de presión generado en el filtro de los compresor. Además, este flujo entrena y selecciona el mejor modelo para emitir la predicción de la evolución del estado del filtro, lo almacena en MLflow y lo llama para la obtención de la predicción. Por otra parte se ha desarrollado un detector de anomalías para 7 compresores mediante el uso de autoencoders, LSTM y Bi-LSTM. En esta memoria se muestra el ejemplo de uno de los compresores obtenidos, donde además se realiza otro flujo de Airflow en el que se emite una predicción de las anomalías detectadas. Tanto esta memoria como los archivos adjuntos a ella pueden encontrarse en `https://gitlab.com/marcossanmiguel97/tfm_marcos_sanmiguel_datascience/`.

Palabras clave

Machine Learning, autoencoder, LSTM, detección de anomalías

Abstract

Industry 4.0 has come up with new techniques for optimizing and improving automation in industry. A machine learning project has been developed at Velfair S.A. (Requejada, Cantabria) where a big data platform has been developed to analyze and manage data from air compressors sold by the company. This work shows the analysis and development of machine learning models for the optimization of the maintenance of these compressors. A data flow has been built using Airflow, this flow was able to analyze the pressure difference coming from the compressor filter. Also, this flow trains and chooses the best model to show predictions of the future state of the filter, and persists the best models in MLflow, where they can be called to get predictions. In addition, an anomaly detector was designed for seven compressors through autoencoders, LSTM and Bi-LSTM networks. This report shows an example of the results obtained with a compressor, where another Airflow flow is performed to show a prediction of the detected anomaly. This report and the attached files can be found at `https://gitlab.com/marcossanmiguel97/tfm_marcos_sanmiguel_datascience/`.

Keywords

Machine Learning, autoencoder, LSTM, anomaly detection

Índice

1. Introducción

El nacimiento de la Industria 4.0 está suponiendo una revolución en la forma de fabricación, mejora, control y distribución de los productos de las empresas. La industria es la parte de la economía que fabrica de forma automática y mecanizada bienes materiales. Desde la primera revolución industrial, la aparición de grandes mejoras tecnológicas han provocado grandes cambios en el paradigma de la industria. Estos cambios son conocidos como revoluciones industriales.

Pasando por la mecanización en la primera revolución industrial, el uso intensivo de la electricidad en la segunda y la digitalización en la tercera, el siglo XXI se encuentra ahora en los comienzos de la cuarta revolución industrial, que ya es conocida como la Industria 4.0 [1].

Desde la primera revolución industrial, el constante aumento de la demanda junto con unos recursos naturales limitados, ha supuesto un reto de producción para la industria, donde minimizar el impacto medioambiental y social está a la orden del día. La Industria 4.0 se presenta como una forma de automatización y optimización que puede suponer grandes beneficios en el ahorro y aprovechamiento de recursos [2].

1.1. Objetivos y motivación

El desarrollo de este trabajo se ha realizado en colaboración con la empresa Velfair, S.A. La empresa situada en Torrelavega (Cantabria) trabaja en el área de ingeniería, entre otras cosas, aportando a sus clientes ayuda en la construcción de soluciones reales y prácticas para la industria. Para ello, Velfair cuenta con más 25 años de dedicación a la comercialización, instalación y soporte de mantenimiento de un elevado rango de equipamiento neumático e hidráulico para la industria en general. Actualmente, se ha transformado en una empresa experta en diseño, desarrollo e implementación de soluciones en el ámbito de la Industria 4.0.

En 2019 Velfair llevo a cabo el desarrollo de un sistema integral de monitorización remota de los compresores de aire que comercializa, con la implementación de un sistema de smart monitoring, un módulo de gestión de mantenimiento, más un sistema FSM (Field Service Management). Actualmente Velfair se encuentra en un proyecto innovador que pretende extender la plataforma de monitorización creada en 2019. Esta extensión consiste en dos puntos:

- Plataforma de Big Data: que solucione el problema de la gestión, integración y almacenamiento de datos masivos, así como su explotación analítica.
- Plataforma de Machine Learning: que permita gestionar el ciclo de vida de algoritmos y modelos matemáticos en la identificación de patrones de funcionamiento y anomalías. Dichos modelos deben poder funcionar de manera centralizada (Cloud Computing) o distribuida (Edge Computing)

El objetivo de este trabajo se ha centrado en el segundo punto del proyecto, en la realización de modelos predictivos y de detección de anomalías en los compresores gestionados por Velfair, encontrando las técnicas adecuadas de machine learning y data mining para obtener unos resultados óptimos. Para ello se han manejado herramientas como Docker, Kubernetes, Airflow y MLflow.

1.2. Autoencoders para detección de anomalías

Los compresores de aires monitorizados en Velfair envían a una base de datos de ElasticSearch distintas variables recogidas por sensores de temperatura, presión, vol-

taje, etc. Estas variables permiten conocer el estado del compresor y la calidad de su funcionamiento, por lo que poder detectar comportamientos anómalos en las variables de los compresores puede permitir accionar alarmas de forma automática para efectuar reparaciones o proporcionar asistencia técnica cuando sea necesario, sin la necesidad de requerir de técnicos que estén constantemente vigilando el estado del compresor. Por lo tanto, poder desarrollar algoritmos eficientes de detección de anomalías es crucial, no solo por el ahorro económico en el mantenimiento, sino para la mayor satisfacción de los clientes.

Un método de detección de anomalías es mediante la reducción de la dimensionalidad del sistema (asumiendo que las variables de entrada presentan correlación entre sí). Este método trata de reducir el espacio original a un subespacio latente que permite describir de forma comprimida los datos de entrada. Tras la compresión de los datos, se produce una descompresión, volviendo a las dimisiones de entrada. Aquellas reconstrucciones donde se producen errores mas grandes son las clasificadas como anomalías [3].

Un tipo de reducción de la dimensionalidad aplicada en la detección de anomalías es por el método de Principal Component Analysis (PCA), sin embargo, las transformaciones lineales pueden no ser suficientes para representar las correlaciones no-lineales entre algunas variables de entrada. Esto podría significar el aumento falsos positivos o negativos en la detección de anomalías.

Un método de detección de anomalías por reducción de la dimensionalidad, y el utilizado en este trabajo, es mediante el uso de autoencoders. Este método utiliza redes neuronales para comprimir las variables de entrada en un subespacio óptimo que pueda capturar todo tipo de correlaciones entre variables [3, 4]. Además, debido a la naturaleza de los datos obtenidos, que son aquellos proporcionados por los sensores de los compresores, se han utilizado Recurrent Neural Networks (RNNs) como Long-Short-Term-Memory (LSTM) o Bidirectional-Long-Short-Term-Memory (BiLSTM).

Los datos con los que se ha realizado el trabajo son observaciones que se reciben con orden en el tiempo. Estas secuencias de observaciones ordenadas en el tiempo son denominadas series temporales [5]. Este tipo de datos aparecen en numerosos datasets como los precios de cierre de la bolsa, la evolución de los casos de Covid-19 o del precio de la luz, y un largo etcétera. En el caso que nos ocupa, se trabajan con varias series temporales de las distintas variables de los compresores (presión, temperatura, gradiente de presión el el filtro, etc.)

1.2.1. Funcionamiento

Como se ha explicado anteriormente, se han utilizado autoencoders para comprimir los datos y poder detectar anomalías al descomprimirlos. El funcionamiento básico de un autoencoder consiste por lo tanto en un encoder y un decoder, que puede explicarse de la siguiente manera [3]:

Por una parte el encoder recibe n vectores de entrada x de dimensionalidad d ($x_i \in R^d, i = \{1, 2, ..., n\}$), estos vectores son comprimidos en m neuronas (donde $m < d$) que forman la capa oculta que forma el subespacio latente. La función de activación de una neurona i en el espacio latente sería entonces:

$$h_i = f_\theta(x) = s\left(\sum_{j=1}^{n} W_{ij}^{input} x_j + b_i^{input}\right) \tag{1}$$

Donde x es el vector de entrada, θ son los parámetros $\{W^{input}, b^{input}\}$, W es la matriz de pesos del encoder con dimensiones $m \times d$ y b es el vector bias de dimensión m. En la figura 1 se representa este funcionamiento en la primera mitad, mientras que en la segunda mitad actuaría el decoder [3].

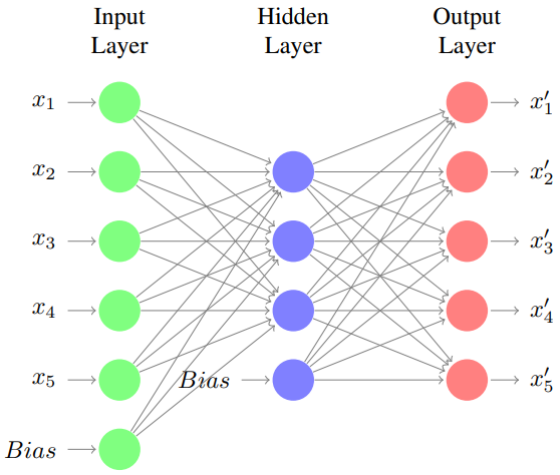


Figura 1: Representación básica de un autoencoder [3].

Tras reducir la dimensión de los vectores de entrada es el turno ahora del Decoder. La codificación resultante del encoder h_i es decodificada para retornar como vector de salida del autoencoder un vector perteneciente al espacio original R^d. La función de decodificación se puede expresar como:

$$x'_i = g_{\theta'}(h) = s\left(\sum_{j=1}^{n} W_{ij}^{hidden} x_j + b_i^{hidden}\right) \tag{2}$$

Donde ahora el set de parámetros $\theta' = \left\{W^{hidden}, b^{hidden}\right\}$. La forma de optimizar el autoencoder será entonces minimizando el error de reconstrucción ε producido entre el vector de entrada y el de salida con respecto a los parámetros θ y θ', obteniendo los parámetros óptimos θ^* y θ'^*:

$$\varepsilon(x_i, x'_i) = \sum_{j=1}^{n} (x_i - x'_i)^2 \tag{3}$$

$$\theta^*, \theta'^* = \arg\min \frac{1}{n} \sum_{i=1}^{n} \varepsilon(x_i, x'_i) = \arg\min \frac{1}{n} \sum_{i=1}^{n} \varepsilon(x_i, g_{\theta'}(f_\theta(h))) \tag{4}$$

Tras un adecuado aprendizaje del autoencoder con un set de datos de entrenamiento cuyo comportamiento es considerado normal, es posible establecer un threshold en el error de reconstrucción ε a partir del cual los datos sean clasificados como anómalos, es decir, al introducir un set de datos de test en el autoencoder, serán considerados anómalos aquellos datos cuyo error de reconstrucción devuelto por el autoencoder sea mayor al threshold establecido. Se obtiene así un detector de anomalías regido por la siguiente ecuación:

$$c(x_i) = \begin{cases} normal & \varepsilon_i < \tau \\ anomalous & \varepsilon_i > \tau \end{cases} \tag{5}$$

1.3. Arquitecturas utilizadas

A continuación se muestran algunas de las celdas utilizadas para la construcción los autoencoders:

1.3.1. RNN

Debido a la naturaleza de los datos, los cuales son entendidos como series temporales, se han utilizado Recurrent Neural Networks (RNNs) para formar la estructura de los autoencoders utilizados. Al igual que las redes neuronales convencionales como las feedforward y las convolucionales, las RNNs utilizan datos para aprender, sin embargo, la característica que las diferencia es que los inputs con los que operan son influidos por el output de la celda anterior, de tal forma que los outputs de las RNNs tienen en cuenta además el comportamiento anterior del sistema.

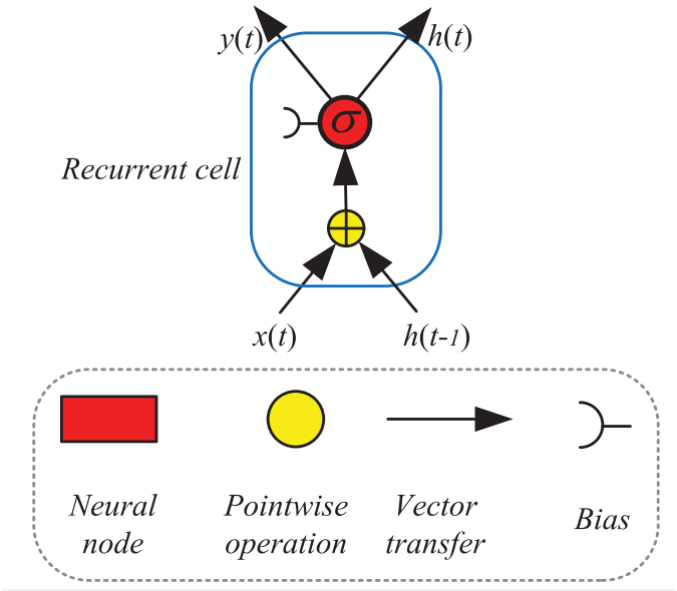


Figura 2: Representación esquemática de una celda recurrente [6].

La representación matemática de la Figura 2 (con función sigma) para el instante de tiempo t viene dada por la siguiente ecuación:

$$\begin{aligned} h_t &= \sigma(W_h h_{t-1} + W_x x_t + b), \\ y_t &= h_t \end{aligned} \tag{6}$$

Donde x_t, h_t e y_t son el input, la información recurrente y el output respectivamente en el instante t de la serie temporal; mientras que W_h y W_x son las matrices de pesos y b es el bias.

1.3.2. Long Short-Term Memory (LSTM)

Para solucionar el problema de las ”long-term dependencies” en 1997 Hochreiter and Schmidhuber propusieron la LSTM, la cual es capaz de tener en cuenta el estados pasados de la serie temporal mediante un término extra de entrada que crea una memoria a largo plazo en la red. Además en el año 2000 Gers, Schmidhuber, and Cummins modifican la LSTM original añadiendo una ”forget gateçon el objetivo de no acumular información irrelevante en la red (Figura 3):

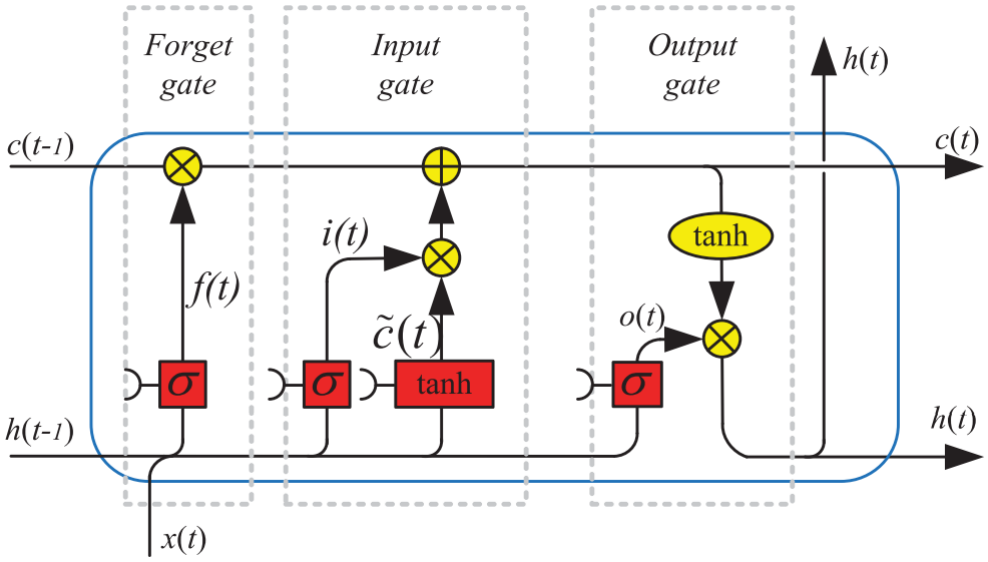


Figura 3: Representación esquemática de una celda LSTM con una forget gate [6].

En la Figura 3 se muestra la representación esquemática de una celda LSTM usada en keras [8], la cual puede ser descrita matemáticamente por el siguiente conjunto de ecuaciones:

$$\begin{aligned} f_t &= \sigma(W_{fh}h_{t-1} + W_{fx}x_t + b_f), \\ i_t &= \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i), \\ \tilde{c}_t &= \tanh(W_{\tilde{c}h}h_{t-1} + W_{\tilde{c}x}x_t + b_{\tilde{c}}), \\ c_t &= f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t, \\ o_t &= \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o), \\ h_t &= o_t \cdot \tanh(c_t) \end{aligned} \tag{7}$$

Donde c_t representa el estado de la celda LSTM o la "memoria a largo plazo, las matrices W los pesos , el operador '$\cdot$' el producto pointwise de dos vectores. Cuando el estado de la celda se actualiza la puerta "input gate" es decide que nueva información se almacena en el estado de la celda c_t y la puerta "output gate"decide la información del output en función del estado c_t. Por otra parte, la "forget gate" se encarga de decidir qué información se desecha en el estado de la celda. Si f_t es 1 la información se mantiene, mientras que si es 0 se olvida.

1.3.3. Bidirectional LSTM Network (BiLSTM)

Las RNNs convencionales solo tienen en cuenta sucesos pasados y actuales. Una herramienta para tener también en cuenta los sucesos futuros la Bidirectional LSTM (BiLSTM). Este tipo de arquitectura permite entrenarse de forma simultánea en la dirección del tiempo y al contrario de la dirección del tiempo, utilizando dos capas separadas para ello. La capa el dirección temporal o "forward layer"se comporta como en la LSTM, calculando la secuencia $(\overrightarrow{h_t}, \overrightarrow{c_t})$ desde t=1 hasta T, mientras que la capa en dirección temporal contraria o "backward layerçalcula la secuencia $(\overleftarrow{h_t}, \overleftarrow{c_t})$ desde t=T hasta 1, tal y como se muestra en la Figura 4:

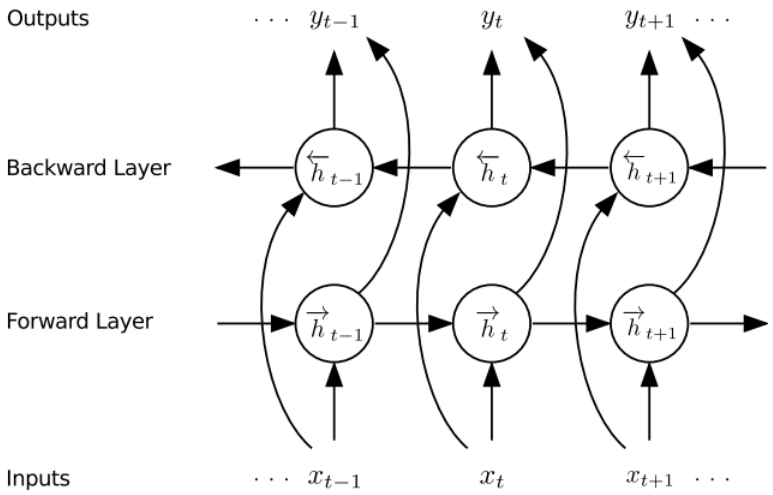


Figura 4: Representación esquemática de una arquitectura BiLSTM [9].

El sistema de ecuaciones que se refleja en la backward layer sería el siguiente:

$$\begin{aligned}
\overleftarrow{f}_t &= \sigma(W_{\overleftarrow{f}\,h} h_{t+1} + W_{\overleftarrow{f}\,x} x_t + b_{\overleftarrow{f}}),\\
\overleftarrow{i}_t &= \sigma(W_{\overleftarrow{i}\,h} h_{t+1} + W_{ix} x_t + b_i),\\
\overleftarrow{\tilde{c}}_t &= \tanh(W_{\overleftarrow{\tilde{c}}\,h} h_{t+1} + W_{\overleftarrow{\tilde{c}}\,x} x_t + b_{\overleftarrow{\tilde{c}}}),\\
\overleftarrow{c}_t &= f_t \cdot \overleftarrow{c}_{t+1} + \overleftarrow{i}_t \cdot \overleftarrow{\tilde{c}}_t,\\
\overleftarrow{o}_t &= \sigma(W_{\overleftarrow{o}\,\overleftarrow{h}} h_{t+1} + W_{\overleftarrow{o}\,x} x_t + b_{\overleftarrow{o}}),\\
\overleftarrow{h}_t &= \overleftarrow{o}_t \cdot \tanh(\overleftarrow{c}_t)
\end{aligned} \tag{8}$$

Dando entonces un output:

$$y_t = W_{\overrightarrow{h}\,y} \overrightarrow{h}_t + W_{\overleftarrow{h}\,y} \overleftarrow{h}_t + b_y \tag{9}$$

2. Desarrollo experimental

Los datos utilizados han sido obtenidos mediante consultas a la base de datos de ElasticSearch de la empresa Velfair. Se han realizado y analizado datos de distintos compresores por separado, en este trabajo se muestran como ejemplo solo algunos de los resultados obtenidos. Para el análisis de datos se ha utilizado Python, con librerías como pandas, numpy, keras o seaborn.

Para familiarizarse con los datos y como objetivo de la empresa, se han procesado los datos de la diferencia de presión producida por el filtro (*Delta*), tratando de encontrar un método que sea capaz de predecir el desgaste del mismo.

Entre otras tareas realizadas en la empresa, se ha diseñado un autoencoder para distintos compresores, encontrando un diseño personalizado para cada compresor y utilizando un dataset de medidas distinto para cada compresor.

Ya que es la primera vez que este tipo de datos son analizados en la empresa, no se disponía de datos anómalos o de estados en los que el filtro entraba en estados e avería (0,6 kPa), por lo que no se ha podido comprobar de forma real los modelos de predicción del estado del filtro y las anomalías han sido inyectadas de forma artificial, modificando los datos a través del código.

Adjunto a esta memoria se presenta el módulo ”lectura”, el código de Airflow realizado para tratar los datos del filtro, un notebook de ejemplo con los datos de un compresor y el entrenamiento del autoencoder y la detección de anomalías.

3. Procesado de Datos para el análisis del filtro

Para la obtención de datos se han realizado consultas a ElasticSearch, filtrando por el identificador del compresor. Se han obtenido de esta forma los datos de los distintos compresores en formato json, para su posterior conversión en dataframe y guardado en csv. A continuación se muestran los datos de un solo compresor como ejemplo del trabajo realizado.

Las columnas que se muestran en los csv son las siguientes:
,_index,_type,_id,_score,sort,_source.metrics.service_time_period.int,
_source.metrics.rated_pressure.flt,_source.metrics.separator_pressure_delta.flt,
_source.metrics.service_level.int,_source.metrics.remote_pressure.flt,
_source.metrics.unloaded_stop_time.int,_source.metrics.coolant_pressure_out.flt,
_source.metrics.loaded_hours.int,_source.metrics.starter_time.int,
_source.metrics.oil_cooler_temperature_out.flt,
_source.metrics.mode_of_operation.int,_source.metrics.running_hours.int,
_source.metrics.online_pressure.flt,_source.metrics.oil_cooler_pressure_out.flt,
_source.metrics.assetName.str,_source.metrics.trip_code.int,
_source.metrics.screw_temperature_out.flt,_source.metrics.warning_code.int,
_source.metrics.sump_pressure.flt,_source.metrics.offline_pressure.flt,
_source.metrics.status.int,_source.metrics.network_pressure.flt,
_source.timestamp

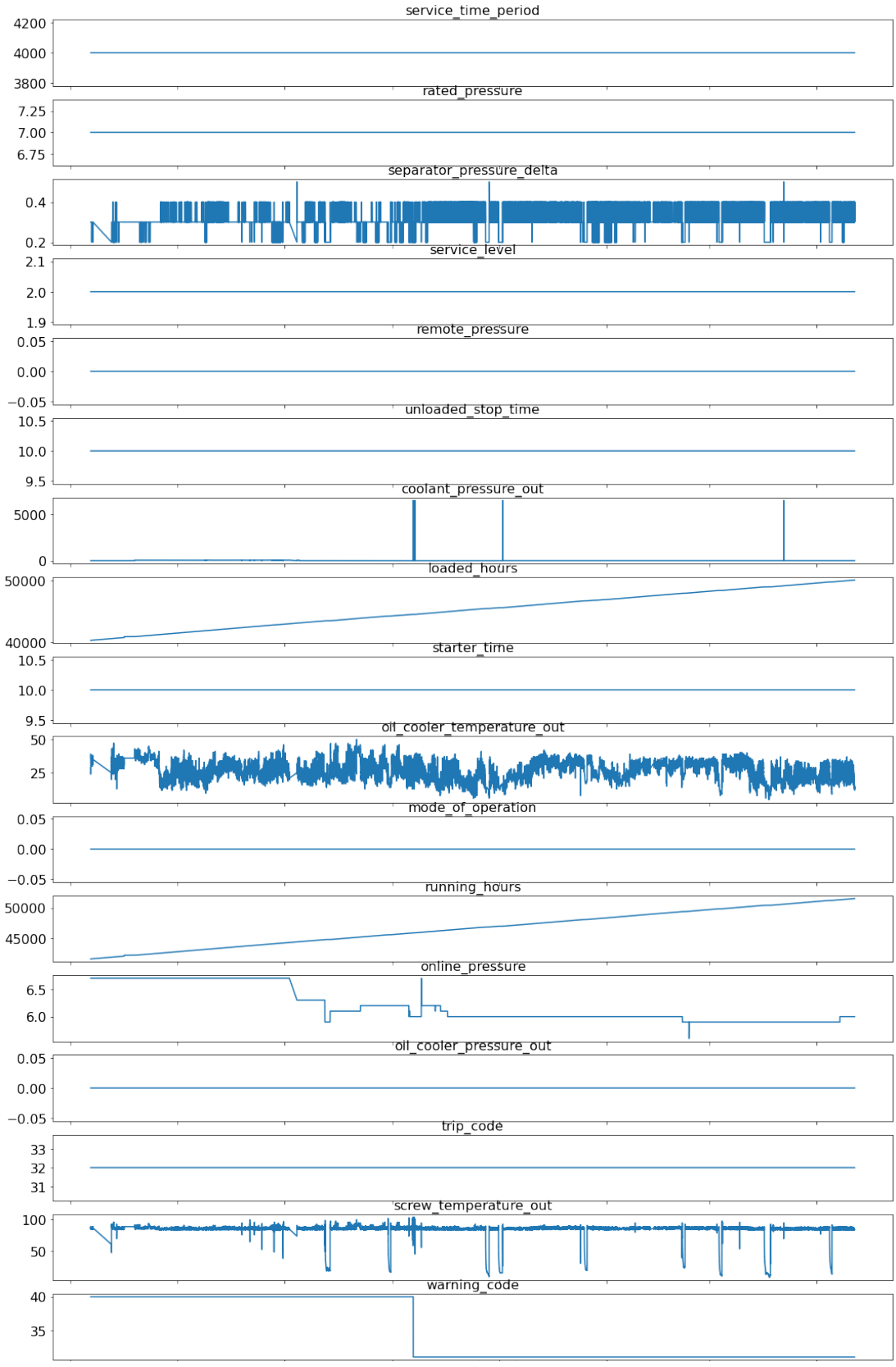
Donde los datos que interesan para el análisis del compresor son aquellos que empiezan por _source.metrics, como por ejemplo la diferencia de presión producida por el filtro del compresor _source.metrics.separator_pressure_delta.flt.

Debido a la complicada legibilidad de los datos y a que para todos los compresores se han obtenido ficheros csv similares, se ha construido un módulo python llamado ”lectura”que contiene tres funciones. La primera función del módulo: ”leer”, recibe como argumento el csv y realiza las siguientes tareas:

- Guarda el csv en formato dataframe.
- Cambia el formato de la fecha de los datos de timestamp a ” %Y- %m- %d %H: %M: %S” y lo establece como índice.
- Elimina variables que no tienen utilidad para el análisis como ’_index’, ’_type’, ’_id’, ’_score’, ’sort’,’_source.metrics.assetName.str’.
- Rellena los valores NaN con el último valor medido (esto es útil para la variable del estado del compresor (status), ya que solo se mide cuando cambia, en lugar de forma continua).
- Elimina las columnas vacías.
- Cambia el nombre de las columnas por uno mas sencillo.
- Modifica la variable status, la cual representa en código de 10 bits el estado del compresor. En concreto el bit 0 y el 1, los cuales se han guardado como dos variables independientes llamadas load y running, que representan el estado de carga y encendido del compresor respectivamente.

La función ”visualizar” representa todas las variables bajo un mismo eje temporal. Por último la función ”correlación” permite observar una gráfica de la correlación que existe entre las variables del compresor

En la Figura 5 se presentan las variables del compresor:



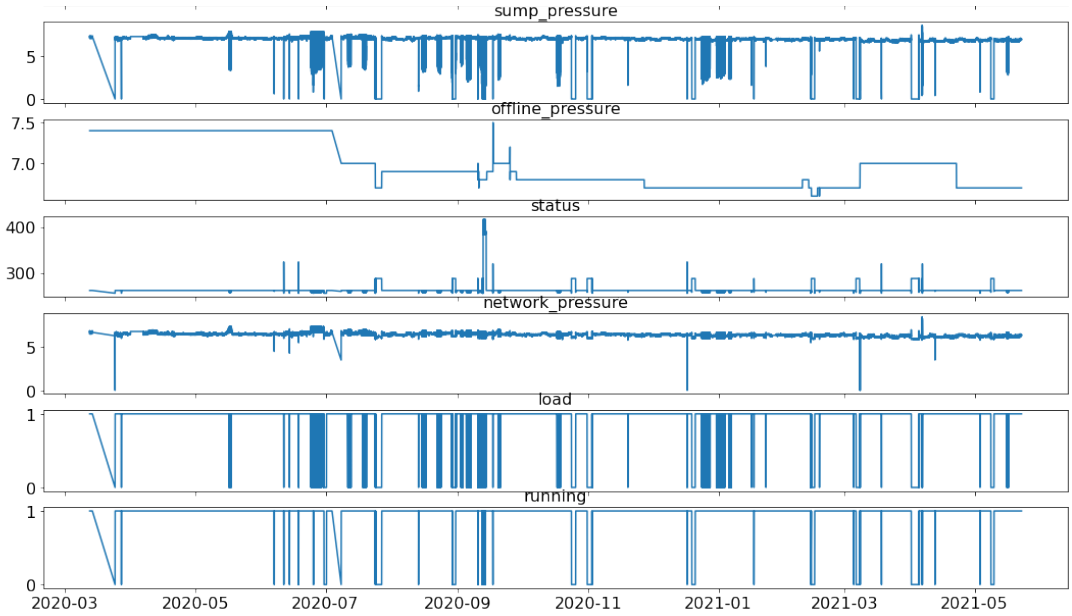


Figura 5: Representación de las variables de un compresor

En la Figura 6 se muestra el comportamiento de la diferencia de presión producida por el filtro (*Delta*). Como se puede observar, cuando el compresor se apaga o deja de funcionar durante un tiempo determinado, la temperatura baja. Es entonces cuando se dan las condiciones para que, al arrancar de nuevo el compresor, el cambio brusco de temperatura genere un aumento brusco en la delta de presión producida por el filtro (tal y como ha sido explicado por los técnicos, en este trabajo no se pretende comprender ni explicar a fondo el funcionamiento de un compresor). Tras el pico de la delta, el filtro se estabiliza y procede a funcionar de manera regular.

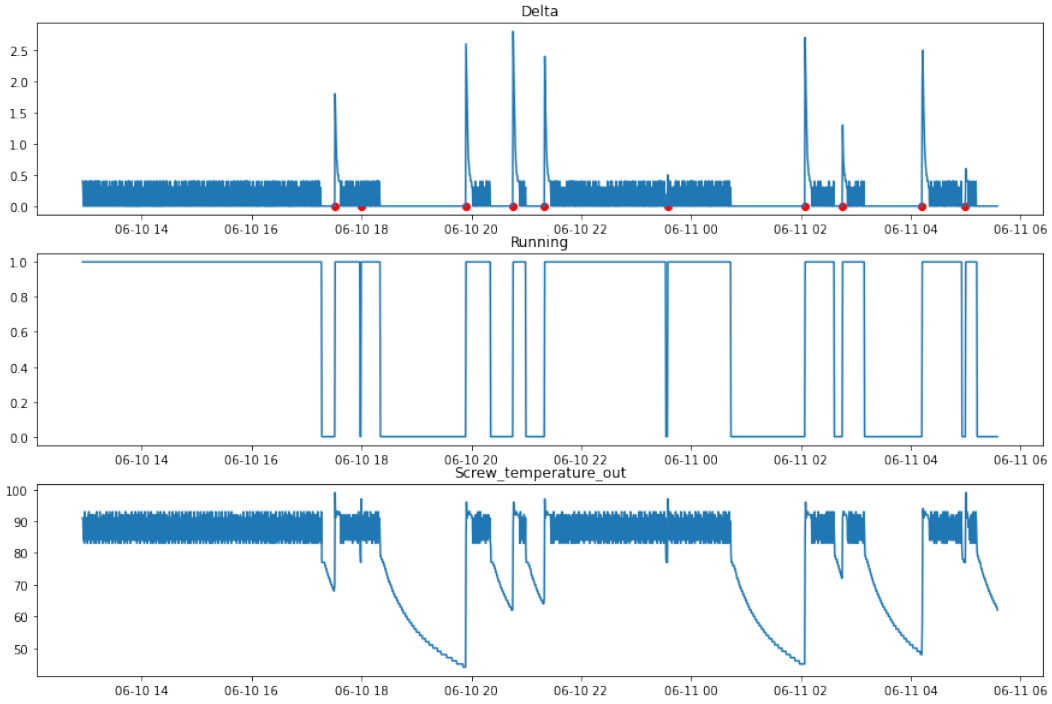


Figura 6: Comportamiento del filtro de aire de un compresor en función de su estado. Se detectan los picos de arranque (en rojo) en la diferencia del presión *Delta* (kPa) producida en el filtro y la variación en la temperatura (ºC).

En esta tarea se pretende analizar el estado del filtro y tratar de predecir su compor-

tamiento en el futuro. Para ello, es necesario eliminar los picos de arranque, y obtener el estado de trabajo regular del filtro del compresor. Para detectar los picos de forma automática se diseñado un alggoritmo que detecta un pico cuando se produce un cambio de 0 a 1 en la variable *Running*:

$$Pico(t) = ((Running(t+1) - Running(t)) == 1) \tag{10}$$

Donde el símbolo == representa una igualdad entre dos expresiones booleanas.

Una vez detectados los picos, estos han sido eliminados junto con los datos del compresor en estado de descarga y en estado apagado (ya que interesa saber el funcionamiento del filtro cuando el compresor está funcionando. Los datos con el compresor apagado no son relevantes). Por último, ya que pueden darse datos perdidos o producirse algún pico que no se haya detectado con este método, se ha tomado el valor medio de ventanas temporales de 7 días cada día. Además para esta media solo se toman los datos dentro de los percentiles 20 y 80 (función mean_percentile descrita abajo). De esta forma, al igual que en los casos de Covid19 donde se toma como medida la incidencia acumulada, se ha asegurado obtener únicamente el estado en el que se encuentra el filtro y eliminar cualquier tipo de ruido o picos de arranque.

```
def mean_percentile(X, minp= 20, maxp = 80, time_steps=1, steps=1):
    ind = []
    m = []
    for i in range(0,len(X) - time_steps,steps):
        v = X[i:(i + time_steps)]
        m.append( v[(v>=np.percentile(v,minp))&(v<=np.percentile(v,90))
            ↪ ].mean())
        format = '%Y␣%m␣%d'
        dt_object = datetime.strptime(v.index[-1].strftime('%Y␣%m␣%d'),
            ↪  format)
        ind.append(dt_object)
    retorno = pd.DataFrame(data = m, index = ind)
    return (retorno)
```

El objetivo final de este proceso es poder obtener el estado actual del filtro y una predicción aproximada del estado del filtro del compresor durante el próximo mes, con el fin de poder detectar cuándo el filtro de un compresor puede llegar a producir una diferencia de presión de 0,6 kPa.

El estado del filtro generalmente es de 0,2 kPa, por lo que realizar un ajuste o entrenar un modelo con esos datos sería contraproducente, ya que el modelo devolvería como predicción que el compresor continuaría en 0,2 kPa hasta el infinito. Para solucionar este problema, se ha propuesto un flujo de tareas en Airflow mediante código python, realizando las siguientes tareas:

- Consulta a la base de datos para obtener los datos de los últimos meses hasta el día de hoy.
- Entrenamiento de varios modelos (exponencial, polinómicos de segundo grado, etc...) y persistencia de aquel modelo con menor error en un set datos de testeo.
- Persistencia de los modelos en MLflow donde pueden ser llamados para realizar predicciones, hasta ser sustituidos al día siguiente donde se reinicia este flujo.

De esta forma se ha realizado un ciclo de entrenamiento y persistencia de modelos con Airflow y Mlflow. En la Figura 7 se muestra el ejemplo de un ajuste polinómico a la variable *Delta* del filtro del compresor.

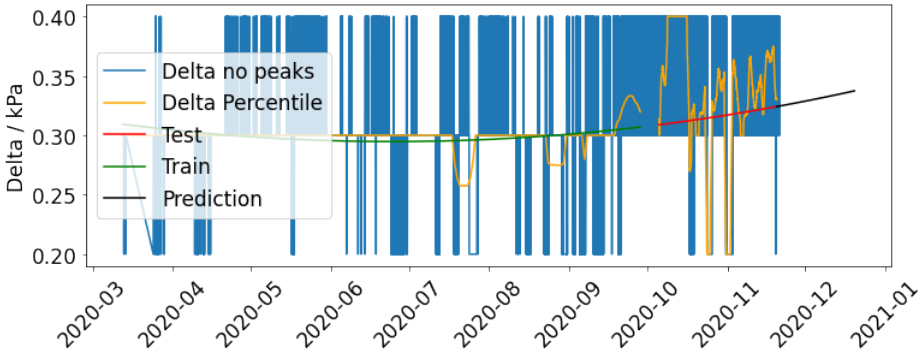


Figura 7: Ajuste polinómico del estado del filtro del compresor. Se muestra en azul la delta del compresor (kPa), en naranja la media de los percentiles (20 %-80 %) de una semana, y en verde, rojo y negro las predicciones en los datos train, test y 30 días en futuro respectivamente.

4. Autoencoders

Para el diseño de los autoencoders de cada compresor se han tomado distintas variables. Debido a que cada compresor funciona bajo unas condiciones determinadas y a que existen compresores fijos (que dan una presión fija) y variables (capaz de proporcionar una presión de aire variable), cada autoencoder recibe y devuelve un set de variables personalizado. Dicho esto, desde la empresa se requería que los autoencoders fueran también capaces de detectar anomalías en la variable de la temperatura. Para ello y como se muestra en esta sección, se ha establecido un threshold para cada variable en función de los datos de error en cada variable a la salida del autoencoder.

La selección de las variables de entrada de cada autoencoder se ha realizado mediante prueba y error. Sin embargo, se ha utilizado la función "correlacion" del modulo de python "lectura" mencionado anteriormente. Con esta función se permite visualizar en una tabla la correlación que presentan las variables de cada compresor (Ejemplo de un compresor en la Figura 8).

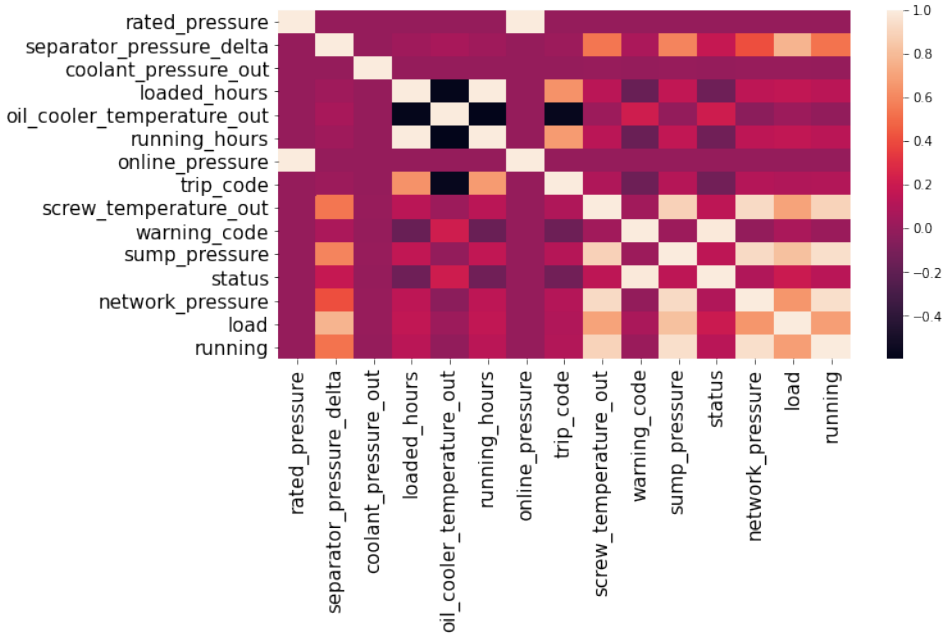


Figura 8: Correlación de las variables de un compresor de ejemplo.

Para este compresor en concreto, las variables de entrada que mejor han funcionado en el autoencoder han sido ['network_pressure', 'separator_pressure_delta', 'screw_temperature_out', 'sump_pressure', 'load']. Tras un limpiado de los datos se ha procedido a dividir los datos en un set de entrenamiento y otro de test.

Los datos han sido escalados con el escalador "MinMaxScaler()" de sklearn para su posterior división en ventanas temporales, en el caso de ejemplo se han tomado ventanas de 30 datos y con un salto de 30, de tal forma que se obtienen dimensiones de (23176, 30, 5) y (5794, 30, 5) para los datos train y test respectivamente. Para este caso se ha utilizado un diseño simple de autoencoder en el que se han introducido los datos de entrenamiento. Como se muestra en el esquema de abajo, el autoencoder utiliza una lstm en la entrada, comprime los datos, y los descomprime pasando por otra capa lstm.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30, 5)             220
_________________________________________________________________
dense (Dense)                (None, 30, 2)             12
_________________________________________________________________
dense_1 (Dense)              (None, 30, 2)             6
_________________________________________________________________
lstm_1 (LSTM)                (None, 30, 5)             160
_________________________________________________________________
time_distributed (TimeDistri (None, 30, 5)             30
=================================================================
Total params: 428
Trainable params: 428
Non-trainable params: 0
_________________________________________________________________
```

En la Figura 9 se muestra el aprendizaje del modelo con los datos de train y una división de validación (10 %), donde se ha obtenido un error en train de $MSE_{train} = 9{,}8967 \cdot 10^{-05}$ y en validación de $MSE_{val} = 1{,}1450 \cdot 10^{-04}$

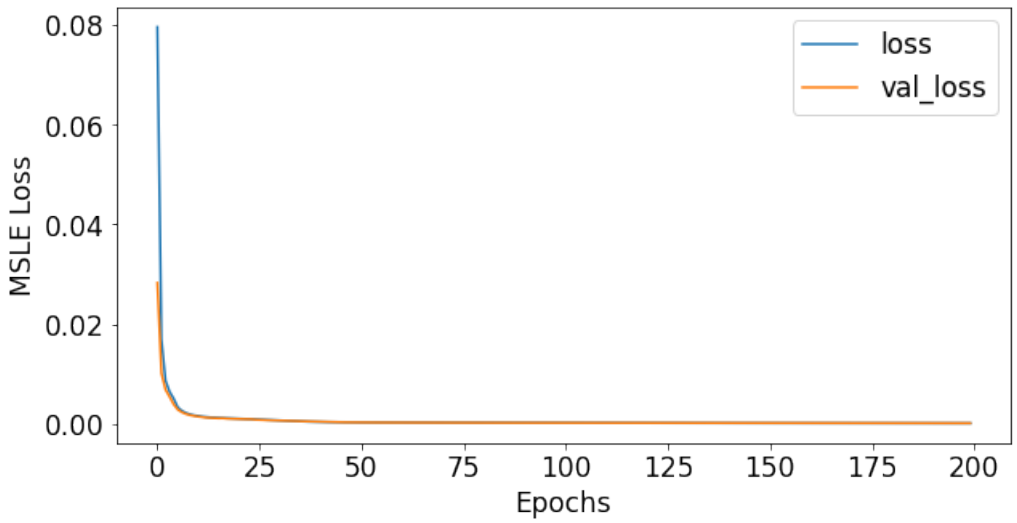


Figura 9: MSE en función de las epocas de entrenamiento para el conjunto de train y de validación.

En las Figuras 10 y 11 se muestran los datos de entrenamiento y las predicciones a

la salida del autoencoder para las variables de la presión de red y de la temperatura respectivamente.

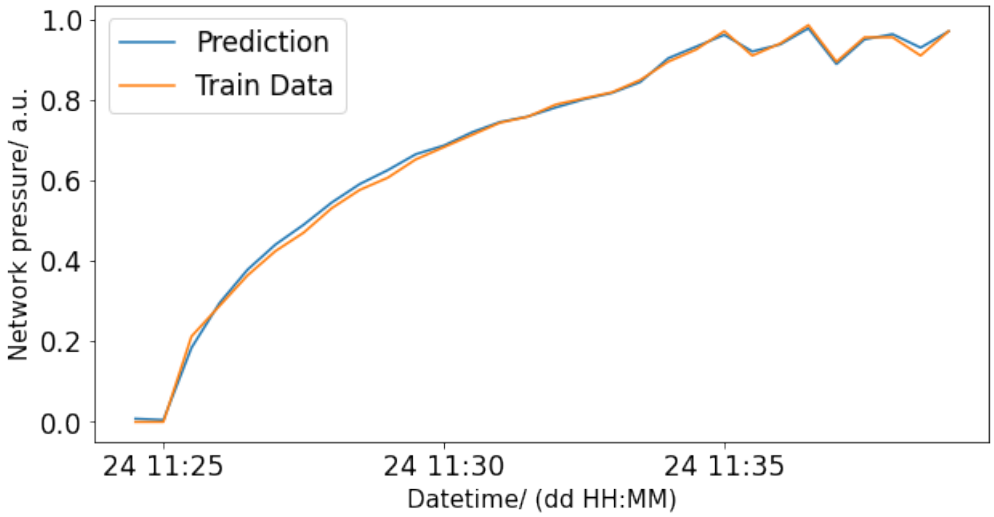


Figura 10: Predicción de la presión de red de un compresor en una ventana temporal de 30 datos junto con la ventana de entrenamiento.

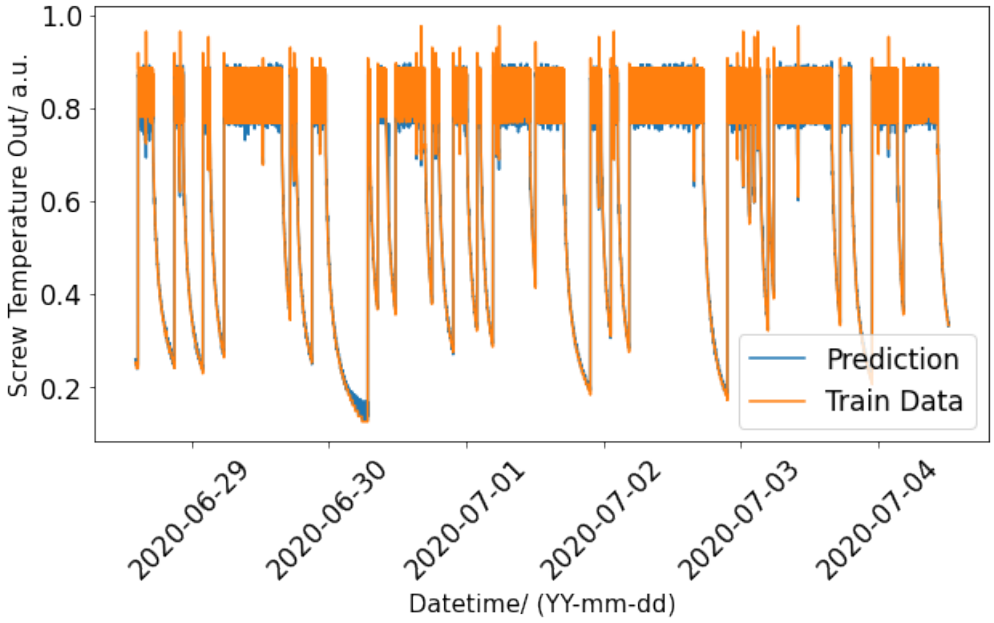


Figura 11: Predicción de la temperatura de un compresor y los datos de entrenamiento.

En las Figuras 12 y 13 se muestran los datos de test y las predicciones a la salida del autoencoder para las variables de la presión de sumidero y de la delta de presión del filtro respectivamente. Se observa que el autoencoder ha aprendido realmente con los datos train y es capaz de devolver la misma entrada que recibe en unos datos de testeo.

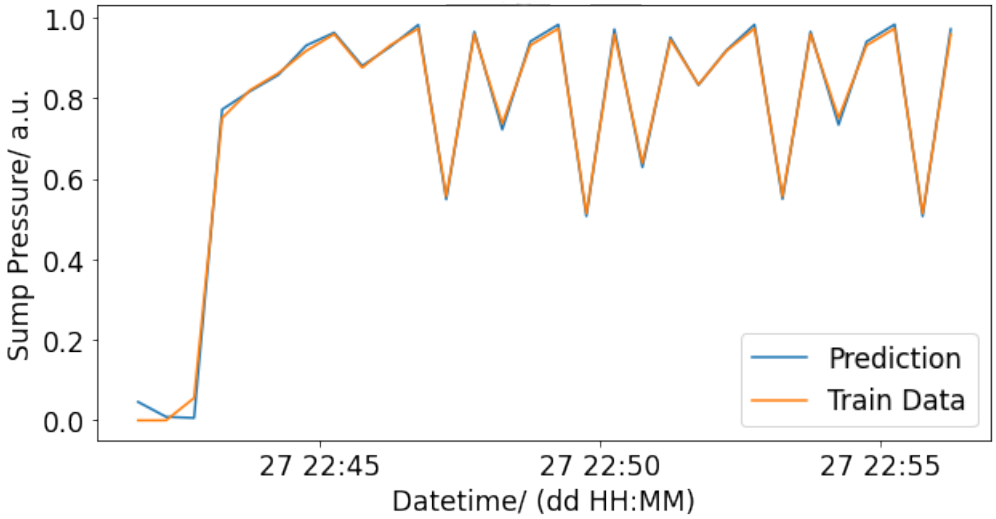


Figura 12: Predicción de la presión de sumidero de un compresor en una ventana temporal de 30 datos junto con la ventana de test.

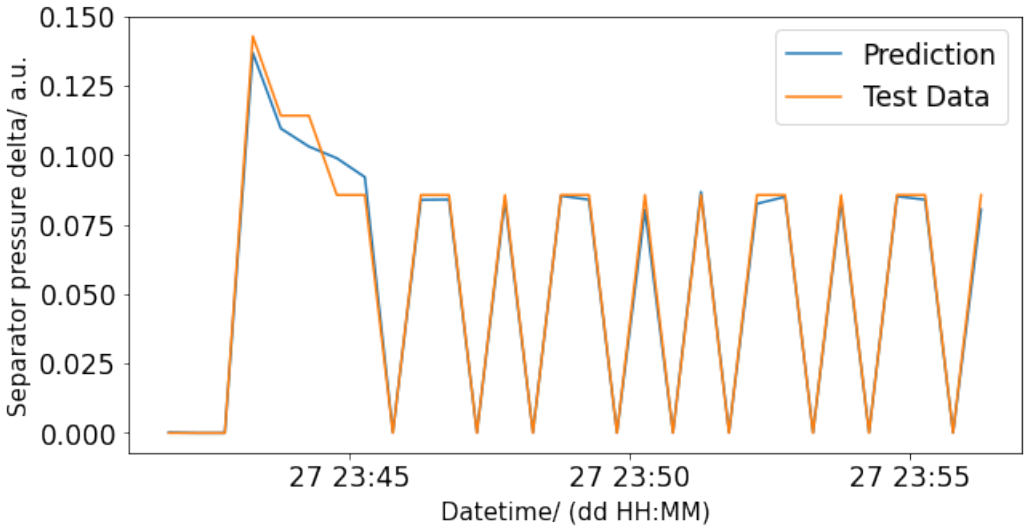


Figura 13: Predicción de la diferencia de presión producida en el filtro de un compresor en una ventana temporal de 30 datos junto con la ventana de test.

Con la ayuda de la Figura 14 se ha establecido un threshold situado en $th = 0{,}013$. Si el error de reconstrucción a la salida del autoencoder de una muestra es mayor que ese th se puede suponer entonces que el modelo esta recibiendo un patrón con el que no es familiar, por lo que esa muestra se clasificará como anómala. Este mismo procedimiento se ha llevado a cabo para cada variable (Figuras 15 y 16), con el objetivo de que modelo sea capaz de detectar en que variable percibe un comportamiento anómalo. Dado a la naturaleza de "caja negra" de las redes neuronales, poder llegar a una arquitectura que satisfaga los objetivos del proyecto, que sería el poder realizar una detección de anomalías en la temperatura, supone un gran reto para el que ha sido necesario muchas pruebas y variaciones en la arquitectura, entrada y dimensión del autoencoder.

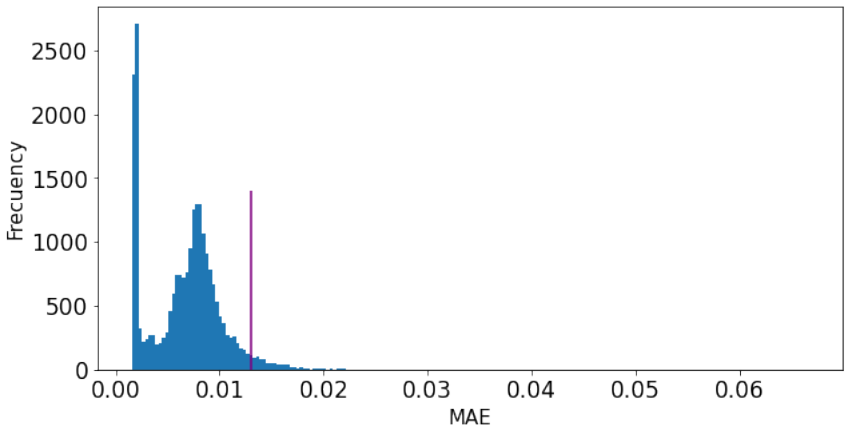


Figura 14: Histograma de MAE en las ventanas temporales de los datos Train y el threshold establecido (línea morada).

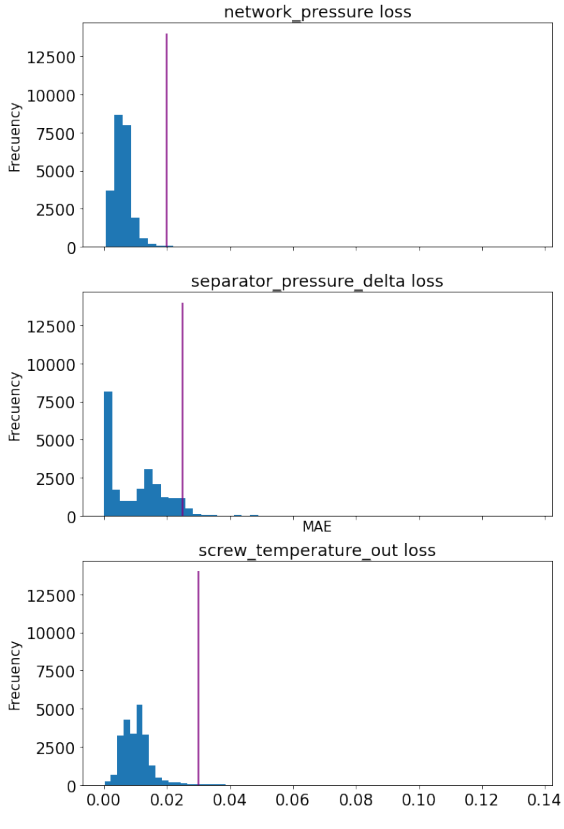


Figura 15: Histograma de MAE en las ventanas temporales de los datos Train y los thresholds establecidos (línea morada).

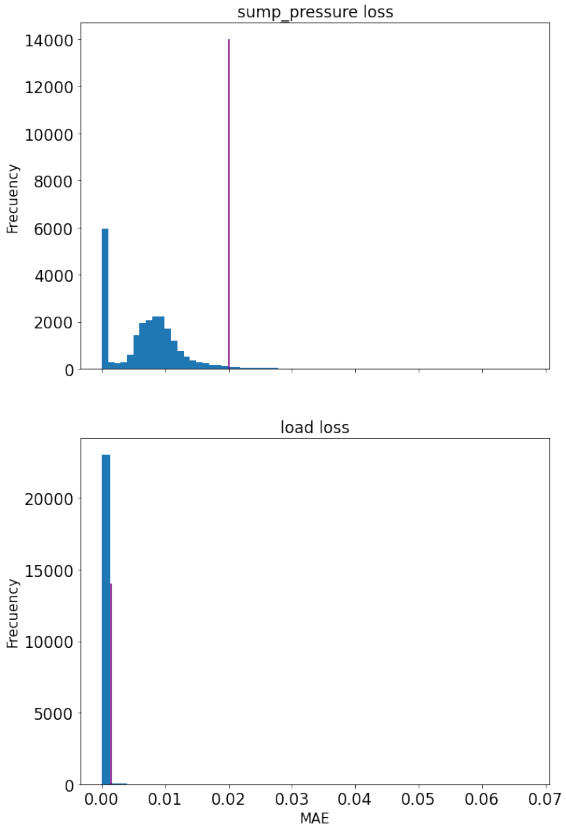


Figura 16: Histograma de MAE en las ventanas temporales de los datos Train y los thresholds establecidos (línea morada).

Con los thresholds establecidos en los datos de entrenamiento, se detectan en los datos test 12 anomalías, que representan un $0,21\,\%$ de los datos. En la Tabla 1 se muestran las anomalías detectadas en cada variable:

Variable	Anomalies Detected	Percentage
network_pressure	8	0,14 %
separator_pressure_delta	2	0,03 %
screw_temperature_out	1	0,02 %
sump_pressure	58	1 %
load	4	0.07 %

Tabla 1: Anomalías detectadas en cada variable en los datos de test.

Tras obtener un autoencoder con un comportamiento adecuado para los datos de test, estos han sido modificados inyectando anomalías artificiales para poder comprobar que el modelo es capaz de detectar comportamientos anómalos en las variables. En la Figura 17 se muestra una ventana temporal con los datos test, los datos test con comportamiento anómalo y la predicción del modelo.

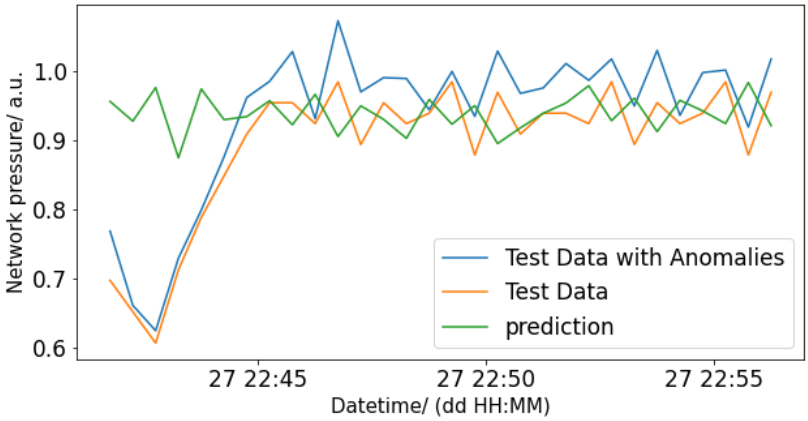


Figura 17: Ventana temporal de los datos test, los datos test con comportamiento anómalo y la predicción del modelo para la presión de red.

Se observa como la predicción ya no sigue de la misma forma a los datos de entrada, por lo que el autoencoder está funcionando peor en la reconstrucción de los datos. Esto puede verse reflejado en la Figura 18, donde se muestra el error MAE de los datos anómalos reconstruidos por el autoencoder y el threshold establecido en los datos de entrenamiento. Las anomalías detectadas ahora alcanzan el 81,84 % de los datos. En la Tabla 2 se muestran las anomalías detectadas en cada variable. Se observa como en la variable de más interés para la empresa, las anomalías detectadas pasan del 0,02 % al 78,25 %.

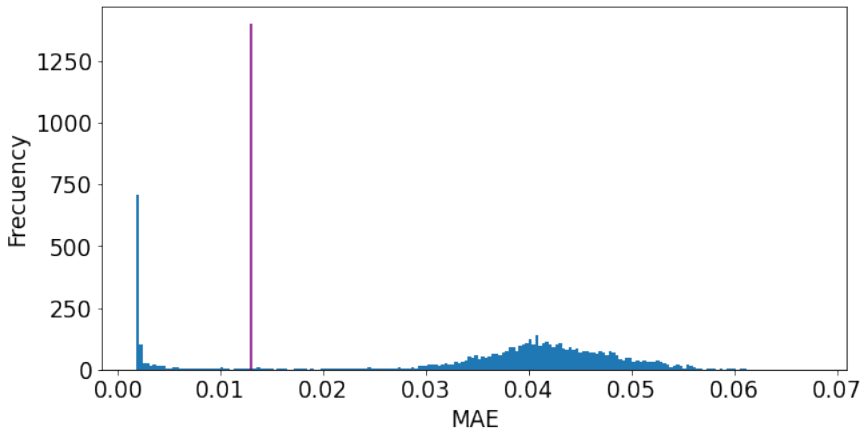


Figura 18: Histograma de MAE en las ventanas temporales de los datos Test con anomalias.

Variable	Anomalies Detected	Percentage
network_pressure	4561	78,72 %
separator_pressure_delta	3405	58,77 %
screw_temperature_out	4534	78,25 %
sump_pressure	4577	80,00 %
load	4782	82,53 %

Tabla 2: Anomalías detectadas en cada variable en los datos de test con anomalías.

Para simular un comportamiento anómalo únicamente en la temperatura, se han inyectado anomalías solo en esta variable. En la Figura 19 se muestra un claro comportamiento de como la predicción se ajusta a los datos no anómalos, produciendo así un error mayor en la reconstrucción, tal y como se deseaba que se comportara el modelo.

Figura 19: Histograma de MAE en las ventana temporal de la temperatura de los datos Test con anomalias.

En la Tabla 3 se muestran las anomalías detectadas cuando solo la temperatura ha sido modificada. Se observa como el autoencoder es capaz de detectar un comportamiento anómalo que destaca en esta variable.

Variable	Anomalies Detected	Percentage
network_pressure	8	0,14 %
separator_pressure_delta	28	0,48 %
screw_temperature_out	4607	79,51 %
sump_pressure	135	2,33 %
load	5	0,09 %

Tabla 3: Anomalías detectadas en cada variable en los datos de test con anomalías únicamente en la temperatura.

Por último y para simular el comportamiento real de una posible avería, se han inyectado anomalías en la temperatura que aumentan de manera exponencial. En este caso la evaluación del modelo no se ha realizado con el porcentaje de anomalías detectadas ya que no es relevante. Se ha buscado detectar el comportamiento y la evolución

de las anomalías en la temperatura, con el objetivo de obtener un flujo similar al de la sección 3 y poder emitir una predicción de como evolucionarán las anomalías en el futuro.

Al igual que en la sección 3, las anomalías detectadas son introducidas en un flujo de datos de Airflow, donde se selecciona el mejor modelo (aquel con menor error en los conjuntos de datos test) entre distintos modelos probados (exponencial y polinómicos de distintos grados) y se registran en MLflow, para su posterior utilización. En la Figura 20 se muestra un entrenamiento donde el modelo que muestra menor error en los datos test es un ajuste polinómico de grado 3.

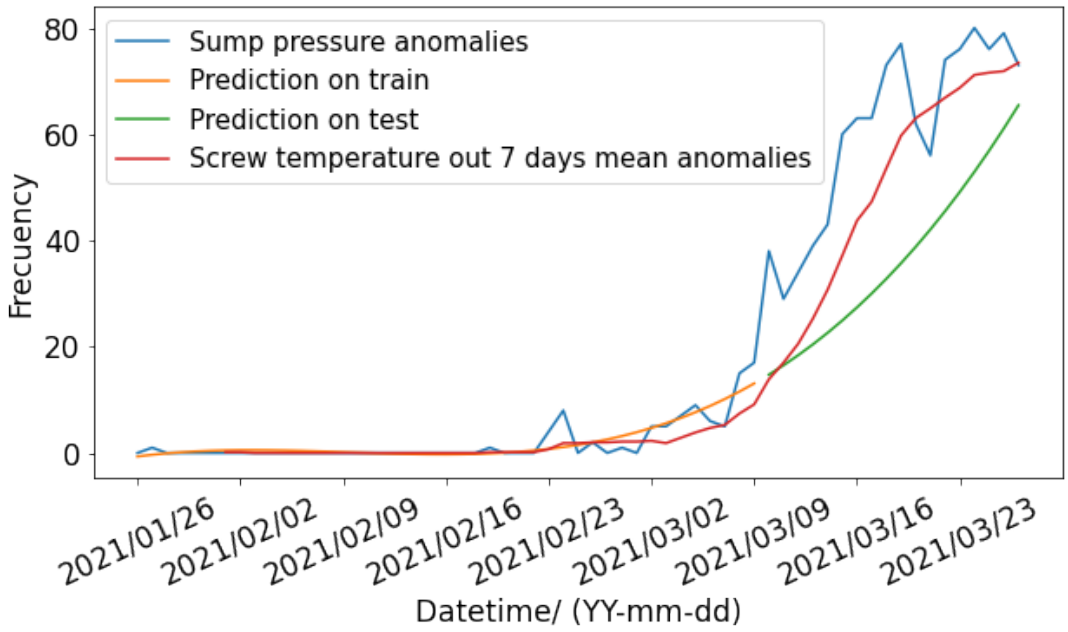


Figura 20: Ajuste polinómico al número de anomalías detectadas por día en la temperatura (azul) cuando estas son aumentadas de forma exponencial con el tiempo. Se muestra en amarillo la predicción a los datos de train, en verde a los datos de test y en rojo la incidencia acumulada media en 7 días.

5. Conclusión

Los métodos utilizados han servido como herramienta a la empresa Velfair para cumplir los objetivos del proyecto. Se han leído, analizado y presentado datos claros que ayudan a entender mejor el comportamiento de los compresores y poder detectar futuros errores en el comportamiento del filtro. El flujo de análisis de datos, persistencia de modelos y emisión de predicción realizado permitirá tener modelos actualizados que informen a los trabajadores del servicio técnico el estado presente y futuro del filtro del compresor.

Además los autoencoders han sido entrenados no solo para detectar comportamientos anómalos a tiempo real, sino para distinguir anomalías en la temperatura. A su vez, se ha diseñado un sistema que permitirá predecir cuando estas anomalías alcanzan un puto crítico donde el compresor pierde funcionalidad y necesite una reparación.

Referencias

[1] Lasi, H., Fettke, P., Kemper, HG. *et al.* Industry 4.0. *Bus Inf Syst Eng* 6, 239–242 (2014). https://doi.org/10.1007/s12599-014-0334-4

[2] Ghobakhloo, Morteza. (2019). Industry 4.0, Digitization, and Opportunities for Sustainability. *Journal of Cleaner Production.* 252. 119869. 10.1016/j.jclepro.2019.119869.

[3] Chen, Zhaomin & Yeo, Chai & Lee, Bu & Lau, Chiew. (2018). Autoencoder-based network anomaly detection. 1-5. 10.1109/WTS.2018.8363930.

[4] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786), 504-507.

[5] Wei, William W. S.. *Time Series Analysis.* (2013).

[6] Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation*, 31(7), 1235-1270. doi:10.1162/neco_a_01199

[7] Lin, T., Horne, B. G., Tino, P., & Giles, C. L. (1996). Learning long-term dependencies in NARX recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6), 1329-1338.

[8] Altché, Florent & de La Fortelle, Arnaud. (2018). An LSTM Network for Highway Trajectory Prediction.

[9] Graves, A., Jaitly, N., & Mohamed, A. R. (2013, December). Hybrid speech recognition with deep bidirectional LSTM. In 2013 *IEEE workshop on automatic speech recognition and understanding* (pp. 273-278). IEEE, doi: 10.1109/ASRU.2013.6707742.