E.T.S de Ingeniería Industrial,
Informática y de Telecomunicación

Desarrollo de un Modelo de Predicción de Parámetros Clínicos Pulmonares mediante Sensores de Temperatura y Humedad usando Técnicas de Inteligencia Artificial

Máster en Ingeniería Biomédica

Trabajo de Fin de Máster

Autor: Ignacio Hernández Jaso

Directores: Daniel Paternain Dallo, Mikel Galar Idoate

Pamplona, Marzo de 2021

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Desarrollo de un modelo de predicción de parámetros clínicos pulmonares mediante sensores de temperatura y humedad usando técnicas de inteligencia artificial

Ignacio Hernández Jaso
ignacio.hernandez@unavarra.es

Resumen

En esta memoria se describe el desarrollo de un modelo de predicción de parámetros clínicos pulmonares mediante la utilización de sensores de temperatura y humedad como alternativa a la espirometría convencional. La toma de datos es realizada durante la respiración en reposo de un sujeto, creando las que denominamos señales espirográficas. La implementación abarcará el preprocesado de los datos para una posterior extracción de características y la construcción, el entrenamiento y la evaluación de modelos de aprendizaje con técnicas de Inteligencia Artificial.

Palabras clave

- Inteligencia Artificial
- Aprendizaje Automático
- Espirometría
- Señales Espirográficas
- Artificial Intelligence
- Machine Learning

Contenido

1. Introducción y Objetivos

El sistema respiratorio es el encargado de suministrar a nuestro organismo del oxígeno necesario para nuestro metabolismo y supervivencia, pero con el paso del tiempo, nuestros pulmones pueden presentar un deterioro que afecta a nuestras capacidades de intercambio de gases con el medio. Ya sea por causas naturales (la edad o la genética) o por factores externos (fumar o respirar aire contaminado), las patologías pulmonares como el asma [1] están presentes en nuestra sociedad, por lo que es necesario contar con métricas y sistemas de detección y diagnóstico [2].

En el entorno médico actual, la prueba realizada a los pacientes para detectar y diagnosticar estas enfermedades o patologías se denomina **espirometría** y, a partir de ella, se extraen diversos parámetros capaces de describir y explicar el estado en el que se encuentran los pulmones del sujeto, como son el ***FEV1*** (volumen de aire espirado de forma forzada en el primer segundo) o el ***FVC*** (volumen de aire total espirado de forma forzada) [3]. Además, resulta una prueba no invasiva y es capaz de realizar un diagnóstico preciso.

No obstante, el análisis espirométrico de un paciente para detectar posibles patologías puede resultar una labor muy costosa. Esto se debe a que el equipamiento es sofisticado y complejo, y requiere de un profesional que sea capaz de guiar al paciente durante la prueba y posteriormente analizar e interpretar los resultados de esta. Además, se le solicita al paciente que colabore efectuando una respiración deliberada y forzada, algo que necesita de un usuario consciente y capacitado. Por ello, estudiaremos una vía alternativa a este proceso mediante las que denominaremos ***señales espirográficas***.

Estas señales, obtenidas a lo largo de una respiración en reposo, no solicitan al sujeto que realice ninguna ventilación forzada, por lo que el paciente no necesita tener esta capacidad o estar consciente. La toma de datos se realiza utilizando tres tipos de sensores que posteriormente serán procesados: **temperatura, humedad y presión**.

Este proyecto tiene como objetivos explorar diversos aspectos tecnológicos, especialmente centrados en estudiar las aplicaciones de herramientas de Inteligencia Artificial (IA) para un posible screening, diagnóstico y seguimiento de patologías pulmonares. En concreto, nos adentraremos en un campo más específico de la IA, el **Machine Learning (ML)**, denominado *Aprendizaje Automático* en castellano.

Este proyecto, por lo tanto, tendrá como objetivo estudiar las **señales espirográficas junto con técnicas de Machine Learning** para encontrar una alternativa a la espirometría convencional. Asimismo, se intentará desarrollar un modelo predictivo eficaz capaz de estimar los mismos parámetros que una espirometría habitual pero con un **bajo coste** y una **mayor accesibilidad** para pacientes con insuficiencia respiratoria o imposibilidad para realizar respiraciones forzadas.

En primer lugar, se afrontará la adquisición, el manejo y la visualización de los datos que se usarán en el proyecto para su estudio y comprensión, incluyendo una fase de **preprocesado** para optimizar los recursos disponibles y las posteriores etapas de **extracción de características**, necesarias para el entrenamiento de los modelos correspondientes. Esta extracción podrá lograrse de forma manual o automática, aspecto que influirá en la clase de modelos que consideraremos utilizar.

Por otro lado, para llevar a cabo este trabajo será necesario construir un entorno de desarrollo eficiente, flexible y altamente interactivo, para así poder generar un flujo de trabajo correcto y permitir una experimentación ágil y adecuada. Puesto que el foco del proyecto será encontrar un modelo predictivo adecuado para la estimación de los parámetros clínicos, será necesaria la **implementación y evaluación** de múltiples algoritmos y técnicas de *Aprendizaje Automático*, además del estudio de sus configuraciones y de diversos tratamientos de datos y etiquetas.

Finalmente, se tratará de interpretar los resultados obtenidos para responder al objetivo principal y determinar cuál puede ser el acercamiento más eficaz a la hora de estimar valores clínicos a partir de las señales extraídas mediante estos sensores de humedad, temperatura y presión. Con esto se espera, por tanto, comprender la naturaleza de los datos, generar aprendizaje de ellos y aplicar e interpretar las soluciones que el *Machine Learning* nos ofrece para este proyecto.

En resumen, el flujo del proyecto se representa en la *ilustración 1* y, para la resolución de los objetivos, hemos planteado las siguientes fases. En primer lugar, se efectuará un preprocesado de los datos adquiridos por los sensores con el fin de hacerlos aptos para el estudio y conformar un dataset limpio del que poder extraer un aprendizaje de forma correcta. Luego, se explorarán varias técnicas de extracción de características, las cuales utilizaremos para entrenar diversos modelos predictivos utilizando técnicas de Machine Learning. Por último, evaluaremos las metodologías experimentadas e interpretaremos los resultados obtenidos para concluir si los objetivos propuestos han sido logrados.

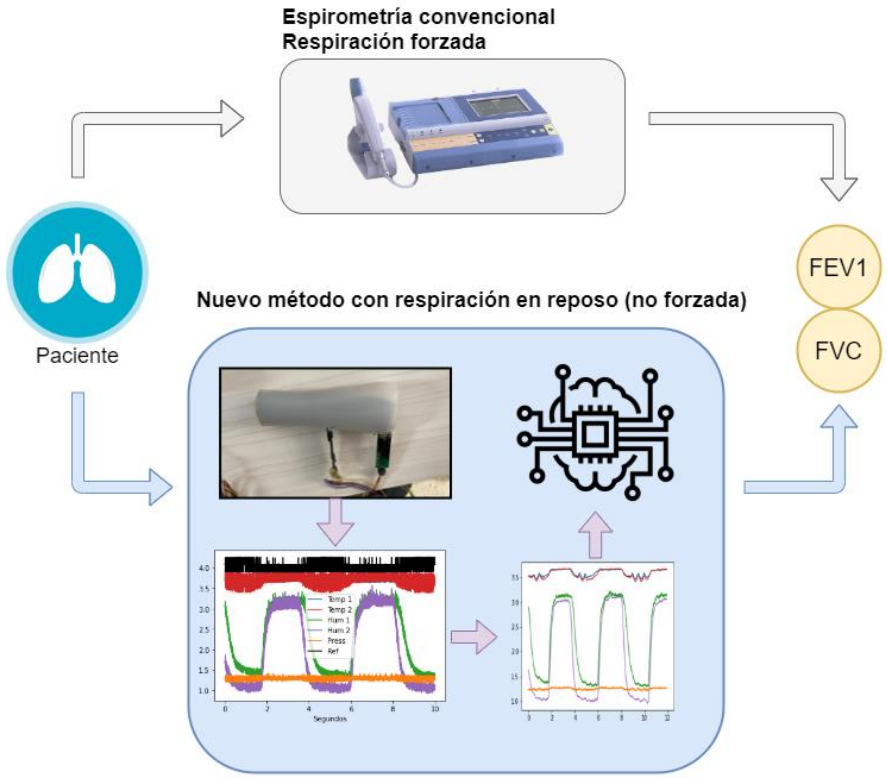


Ilustración 1: *Esquema del objetivo del proyecto.*
Arriba, el método habitual por espirometría.
Abajo, nuestro método con señales espirográficas.

2. Contexto y Materiales

En las pruebas espirométricas realizadas en la actualidad se obtienen parámetros pulmonares que indican el estado de la salud respiratoria del paciente explorado. De las dos métricas más importantes, la primera se denomina **FEV1** (*Forced Espiratory Volume*$_1$) y consiste en la cantidad de aire que el paciente es capaz de movilizar en el primer segundo de una espiración forzada. Por otro lado, tenemos el **FVC** (*Forced Vital Capacity*) que indica el volumen de aire que puede ser movilizado en una inspiración o espiración forzadas. Con estos parámetros relativos al paciente se puede cuantificar la obstrucción pulmonar y conocer su estado respiratorio.

2.1 Obtención de Datos

En una primera instancia, estas medidas espirométricas requieren de un equipamiento sofisticado y de personal entrenado, por lo que se propone un método de adquisición alternativo menos invasivo y mucho más pasivo: las **señales espirográficas**. Para obtenerlas, se coloca sobre la boca del paciente un dispositivo equipado con múltiples sensores y se le solicita que mantenga una respiración en reposo durante varios segundos. El aparato, provisto de sensores de **humedad**, **temperatura** y **presión**, registrará en forma de voltaje las variaciones que se produzcan de estos parámetros a través del tubo que conecta la boca del paciente con el medio exterior.

Esta toma de datos fue realizada tomando dos grupos de pacientes voluntarios, uno de ellos carente de desórdenes respiratorios y el otro con patologías de tipo asmático. En total, se cuenta con un total de 108 sujetos de variada condición física. Las señales conseguidas, mostradas a continuación, representarán de ahora en adelante los datos de los que disponemos para lograr nuestro objetivo.

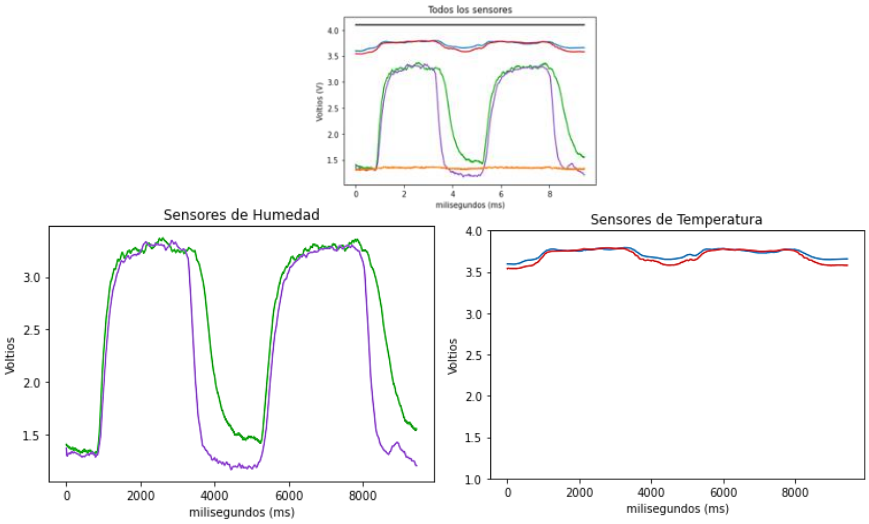


Ilustración 2: *Señales extraídas del estudio.*
De izquierda a derecha, la humedad y la temperatura.

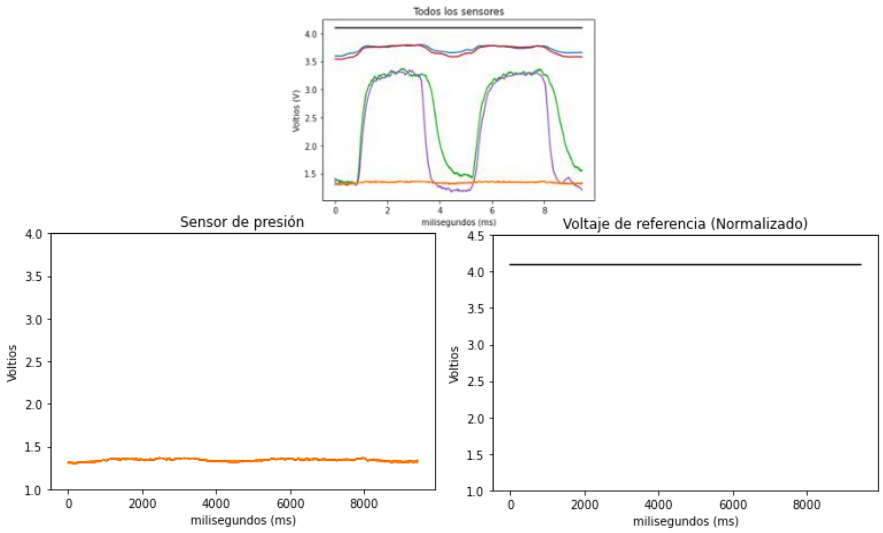


Ilustración 3: *Señales extraídas del estudio.*
De izquierda a derecha, presión y el voltaje de referencia.

Como se puede observar en las *ilustraciones 2 y 3*, contamos con dos señales de temperatura y humedad. Esto se debe a que los sensores del aparato están divididos entre el grupo 1 y el grupo 2, como se indica en la siguiente imagen (*ilust. 4*). Para utilizarlo, el paciente respira de manera natural por el extremo izquierdo, atravesando en primer lugar el grupo de sensores 1 y, antes de llegar al exterior, el grupo 2. Los cambios en las condiciones del aire alrededor de cada sensor son registrados para así generar el dataset del que disponemos.

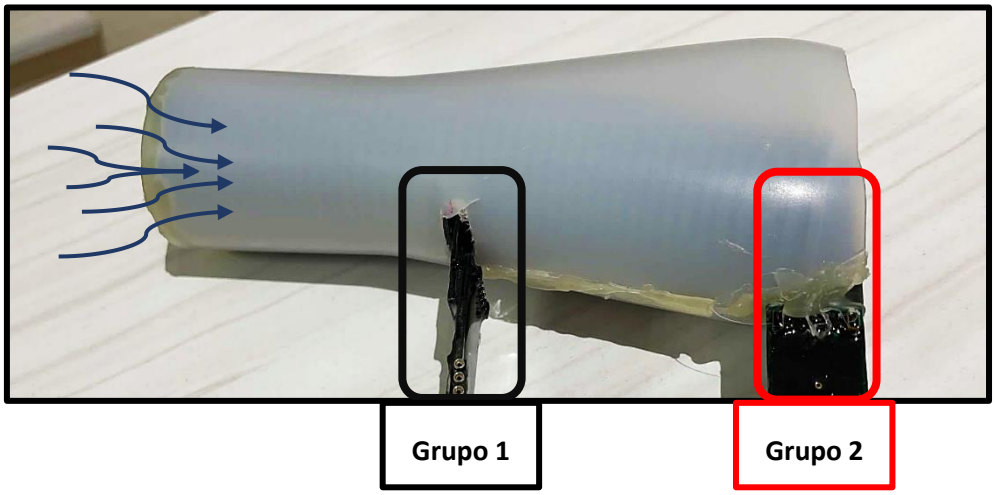


Ilustración 4: *Sensor construido utilizado en la adquisición de señales.*

2.2 Almacenamiento y organización de los datos

Conseguir el aprendizaje de un modelo requiere de un conjunto de datos y de algoritmos capaces de leerlos, organizarlos y administrarlos durante el entrenamiento. Los ficheros correspondientes a estos datos se dispondrán en un directorio en forma de sencillos documentos de texto en plano (.txt) dentro del entorno de desarrollo. En ellos, la información es almacenada de forma tabular, pudiéndose así extraer los valores, interpretarlos y manipularlos con nuestras herramientas de trabajo. En las *ilustraciones 5 y 6* se puede observar el formato en el que los datos han sido almacenados y organizados al comienzo del proyecto.

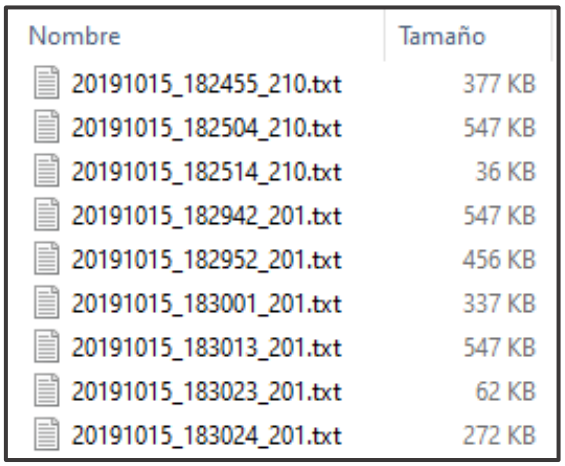


Ilustración 5: *Directorio con los ficheros de datos.*

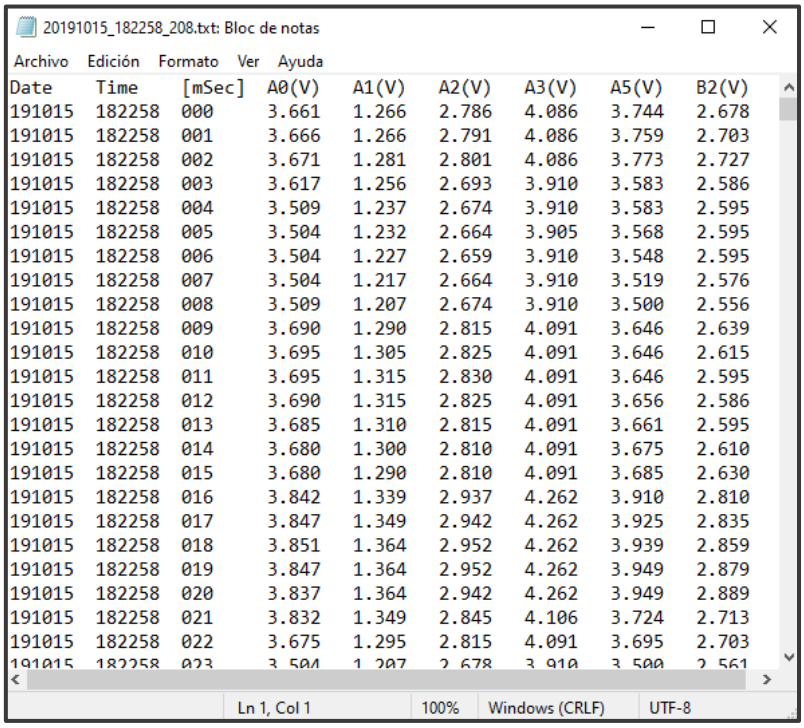


Ilustración 6: *Contenido de un fichero .txt con las señales registradas.*

Cada fichero contiene **9 columnas de datos**: tres de ellas corresponden al contexto temporal en el que se realizó la toma de datos, mientras que las seis restantes corresponden a los valores obtenidos por cada uno de los sensores en **voltios**. Durante la medición, se ha registrado un valor cada milisegundo, por lo que durante 1 segundo de muestreo los sensores obtienen 9 señales de 1000 valores cada una. Las columnas son las siguientes:

1. **Date**: Fecha en la que la medición tuvo lugar. Utiliza un formato YYMMDD por lo que 191015 representa el 15 de octubre del 2019, por ejemplo.
2. **Time**: Hora en la que la medición tuvo lugar. Se almacena en un formato HHMMSS así que 182258 corresponde a las 18:22 y 58 segundos.
3. **[mSec]**: Milisegundo asociado a cada valor muestreado. Esta columna obtendrá valores entre 000 y 999 de forma cíclica conforme el tiempo de muestreo se prolongue. Naturalmente, cuando alcance el valor 1000, regresará al número 000 y el valor de Time incrementará en 1, indicando que ha pasado 1 segundo.
4. **A0(V) | Temp1**: Señal de temperatura del grupo 1.
5. **A1(V) | Press**: Señal de presión del grupo 1.
6. **A2(V) | Hum1**: Señal de humedad del grupo 1.
7. **A3(V) | Vref**: Tensión de referencia del equipo de sensado.
8. **A5(V) | Temp2**: Señal de temperatura del grupo 2.
9. **B2(V) | Hum2**: Señal de humedad del grupo 2.

Si leemos los ficheros desde nuestro entorno mediante la librería *pandas* y renombramos las columnas para darle un mayor contexto semántico obtenemos el **DataFrame** que se muestra en la tabla siguiente, seguida de la representación gráfica de estas mismas señales usando la librería **matplotlib** (*ilust. 7 y 8*):

	Date	Time	[mSec]	Temp1	Press	Hum1	Vref	Temp2	Hum2
0	191015	182258	0	3.661	1.266	2.786	4.086	3.744	2.678
1	191015	182258	1	3.666	1.266	2.791	4.086	3.759	2.703
2	191015	182258	2	3.671	1.281	2.801	4.086	3.773	2.727
3	191015	182258	3	3.617	1.256	2.693	3.910	3.583	2.586
4	191015	182258	4	3.509	1.237	2.674	3.910	3.583	2.595
...	...	...	...	...	...	...	...	...	...
9995	191015	182307	995	3.612	1.281	2.977	3.910	3.675	2.996
9996	191015	182307	996	3.607	1.276	2.972	3.910	3.646	2.967
9997	191015	182307	997	3.607	1.266	2.962	3.910	3.622	2.947
9998	191015	182307	998	3.763	1.310	3.094	4.081	3.822	3.089
9999	191015	182307	999	3.612	1.251	2.972	3.910	3.607	2.893

10000 rows × 9 columns

Ilustración 7: *DataFrame con los datos del fichero leído.*

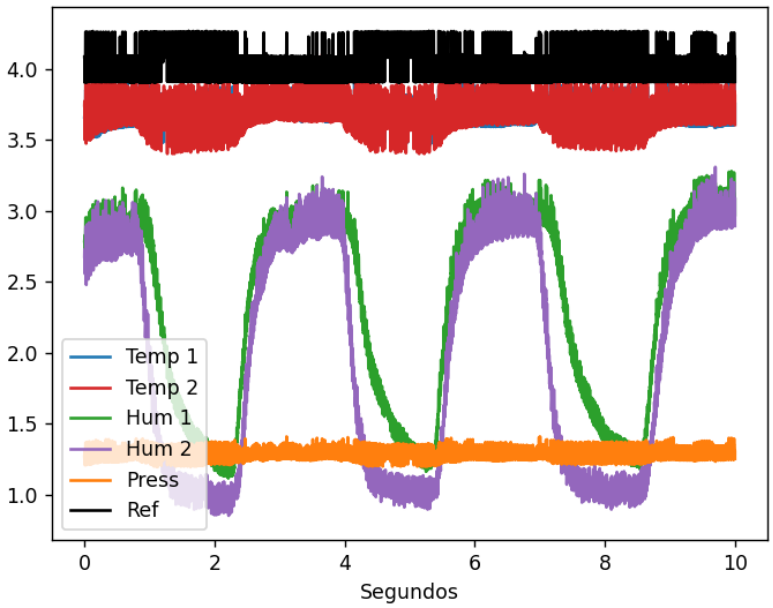


Ilustración 8: *Representación gráfica del fichero leído.*

Por otro lado, disponemos de dos etiquetas principales vinculadas a cada usuario del estudio: FEV1 y FVC. Ambas son variables numéricas y tienen una distribución similar, aunque simbolizan aspectos distintos del estado de salud del individuo, como se ha comentado con anterioridad. En las tablas de la *ilustración 9* podemos ver a grandes rasgos la distribución de los datos de estas variables y varios ejemplos reales.

SUJETO	FEV1	FVC
137	4.18	4.37
232	3.52	3.96
238	3.82	4.14
250	4.40	5.13
260	3.48	3.00
273	4.56	6.23

	FEV1	FVC
Mínimo	1.63	1.94
Máximo	5.5	6.32
Media	3.58	3.99
Mediana	3.52	3.87
STD	0.85	1

Ilustración 9*: Observación de las clases a la izquierda. Características de la distribución de las etiquetas a la derecha.*

2.3 Recursos informáticos

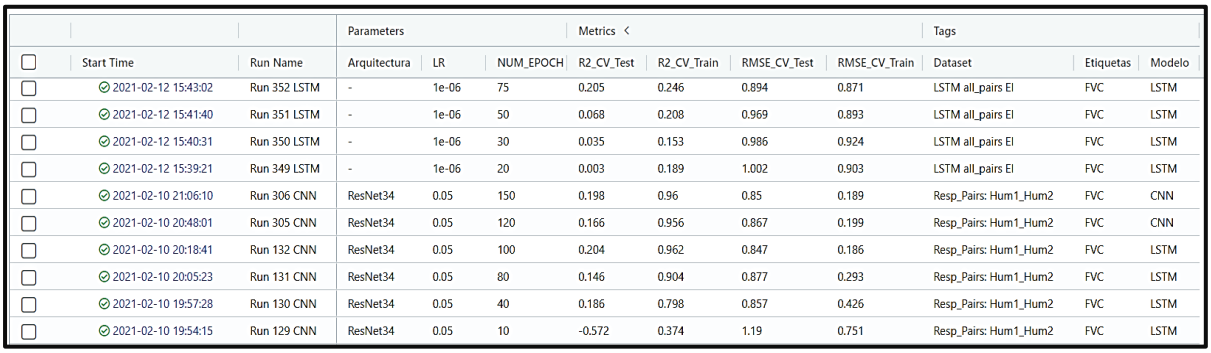
Con respecto al software utilizado, contamos con varios frentes en los que podemos describir las herramientas instaladas y el entorno que se ha establecido para el desarrollo del proyecto. Como lenguaje de programación se ha escogido **Python 3**, el cual nos aporta gran flexibilidad, interactividad y escalabilidad, además de contar con numerosas librerías especializadas en Machine Learning que facilitarán la implementación y experimentación y mejorarán la eficiencia del código. Los módulos que serán usados en el proyecto son:

- **NumPy**: Paquete fundamental para cálculos numéricos en Python.
- **pandas**: Implementa estructuras de datos tabulares y relacionales de forma flexible y organizada. Servirá de utilidad para leer, escribir y almacenar las tablas de datos para manipularlas, transportarlas o procesarlas.
- **Matplotlib**: Librería dedicada a crear visualizaciones de datos en entornos de Python.
- **scikit-learn**: Repertorio muy amplio de herramientas utilizadas en Machine Learning, desde modelos de regresión o clasificación hasta cálculos de métricas.
- **PyTorch**: Framework especializado en Machine Learning de alto nivel para construir redes neuronales de forma altamente configurable.
- **Fastai**: Módulo implementado a partir de PyTorch de aún más alto nivel que nos permitirá entrenar redes convolucionales de forma más eficaz.
- **Keras**: Framework implementado a partir de Tensorflow que nos facilitará la construcción de redes recurrentes.
- **MLflow**: Entorno empleado para el registro de datos relacionados con los entrenamientos que se completen. Con este módulo podremos almacenar y organizar aspectos como parámetros utilizados, métricas obtenidas y modelos entrenados para cada experimento que realicemos.

Para la implementación en sí misma, se ha creado un entorno utilizando **Anaconda**, una plataforma que encapsula a Python junto con todos los módulos que este necesite, como los mencionados anteriormente. De entre ellos, contamos con **Jupyter**, un versátil editor de texto que nos permite desarrollar todo el código desde una interfaz integrada en navegador con la que podemos visualizar datos, crear gráficas y compartir resultados de manera sencilla y eficiente además de programar. Por otro lado, para el desarrollo de clases o módulos externos que nos sean necesarios se ha optado por el IDE de **Atom**, el cual facilita la implementación de .py que posteriormente serán importados desde Jupyter para el entrenamiento de los modelos o el almacenamiento de los resultados en instancias de objetos Python.

Durante el desarrollo, el **control de versiones** del código se llevará a cabo utilizando la herramienta **Git** y almacenando el código en la plataforma GitHub, permitiendo la experimentación colaborativa y un desarrollo más eficiente y organizado. Además, en combinación con Anaconda, configurar y compartir un entorno de trabajo resulta más sencillo, lo que facilita el trabajo en paralelo y la replicación de experimentos.

Una vez logremos entrenar algunos modelos predictivos con los datos necesitaremos hacer uso de métricas para evaluar y comparar la eficacia de todos ellos. Para gestionar estos datos se utilizará la herramienta **MLflow**, la cual hace más accesible la labor de clasificar, organizar, analizar y comparar los modelos que mejor hayan funcionado (*ilust. 10*). Con respecto a la comunicación con los compañeros se opta por la herramienta de **Slack** para un contacto constante e inmediato que permite compartir archivos multimedia y documentos de forma rápida, para los cuales se usarán principalmente los programas de ofimática como Word o Excel para mantener la agilidad de trabajo.



			Parameters			Metrics				Tags		
	Start Time	Run Name	Arquitectura	LR	NUM_EPOCH	R2_CV_Test	R2_CV_Train	RMSE_CV_Test	RMSE_CV_Train	Dataset	Etiquetas	Modelo
	2021-02-12 15:43:02	Run 352 LSTM	-	1e-06	75	0.205	0.246	0.894	0.871	LSTM all_pairs El	FVC	LSTM
	2021-02-12 15:41:40	Run 351 LSTM	-	1e-06	50	0.068	0.208	0.969	0.893	LSTM all_pairs El	FVC	LSTM
	2021-02-12 15:40:31	Run 350 LSTM	-	1e-06	30	0.035	0.153	0.986	0.924	LSTM all_pairs El	FVC	LSTM
	2021-02-12 15:39:21	Run 349 LSTM	-	1e-06	20	0.003	0.189	1.002	0.903	LSTM all_pairs El	FVC	LSTM
	2021-02-10 21:06:10	Run 306 CNN	ResNet34	0.05	150	0.198	0.96	0.85	0.189	Resp_Pairs: Hum1_Hum2	FVC	CNN
	2021-02-10 20:48:01	Run 305 CNN	ResNet34	0.05	120	0.166	0.956	0.867	0.199	Resp_Pairs: Hum1_Hum2	FVC	CNN
	2021-02-10 20:18:41	Run 132 CNN	ResNet34	0.05	100	0.204	0.962	0.847	0.186	Resp_Pairs: Hum1_Hum2	FVC	LSTM
	2021-02-10 20:05:23	Run 131 CNN	ResNet34	0.05	80	0.146	0.904	0.877	0.293	Resp_Pairs: Hum1_Hum2	FVC	LSTM
	2021-02-10 19:57:28	Run 130 CNN	ResNet34	0.05	40	0.186	0.798	0.857	0.426	Resp_Pairs: Hum1_Hum2	FVC	LSTM
	2021-02-10 19:54:15	Run 129 CNN	ResNet34	0.05	10	-0.572	0.374	1.19	0.751	Resp_Pairs: Hum1_Hum2	FVC	LSTM

Ilustración 10: *Interfaz gráfica de MLflow.*

Como se ha descrito anteriormente, el foco de trabajo se encontrará en los Jupyter Notebooks donde se llevará a cabo la implementación y la experimentación de los modelos predictivos. El código se dividirá según la utilidad para la que sean dispuestos, de forma que unos Notebooks se dedicarán al preprocesado, otros a albergar las librerías que se desarrollen e importen posteriormente, y otros a la experimentación con los modelos y datos. También se implementarán clases en ficheros *.py* para hacer más configurables las estructuras de datos o desarrollar modelos de aprendizaje por nuestra cuenta.

2.4 Metodología global

Describiremos ahora a rasgos generales cómo enfocaremos la utilización de las tecnologías de Machine Learning para experimentar con los datos y encontrar un modelo adecuado. En primer lugar, comentaremos a qué tipo de problema nos enfrentamos, tras lo cual definiremos el método que usaremos para evaluar los modelos próximos. Finalizaremos mencionando las métricas que nos servirán para guiar nuestros experimentos y cuantificar los errores cometidos.

Clasificación vs Regresión

Como se ha comentado a lo largo de esta memoria, aunque las tecnologías que utilicemos pueden ser aplicadas tanto en labores de clasificación como de regresión, en este proyecto nos centraremos en la segunda. Razonaremos ahora brevemente las diferencias entre ambos métodos y el motivo por el cual nuestro proyecto trabajará con modelos de regresión.

Una **clasificación** consiste en determinar la categoría a la que pertenece un conjunto de datos de entrada, y puede ser binaria o multiclase dependiendo del número de categorías disponibles en el problema de clasificación. Por lo general, en estos casos, el objetivo es entrenar un modelo capaz de determinar de forma correcta la clase asociada a una muestra, calculando su grado de pertenencia a cada categoría. Este grado es contrastado entre todas las clases para tomar la decisión y clasificar.

En **regresión**, por el contrario, la meta no consiste en determinar a cuál de varias etiquetas discretas pertenece una muestra, sino en aproximar la relación de dependencia que existe entre las variables predictoras y la variable de respuesta, logrando así estimar un valor numérico a partir de los datos de entrada. Aunque en esta memoria no se expongan entrenamientos multivariable, una regresión también puede realizarse con múltiples variables de respuesta.

En nuestro caso concreto, a partir de series temporales, características o imágenes intentaremos entrenar modelos predictivos capaces de estimar los parámetros clínicos pulmonares de relevancia: FEV1 y FVC. Por esta razón, el problema al que nos enfrentamos es de **regresión**, pues nuestro objetivo es conseguir predecir de forma correcta el FEV1 o FVC asociado a cada sujeto partiendo de sus señales espirográficas. A partir de este punto, por lo tanto, exploraremos varios métodos de extracción de características para el posterior entrenamiento de modelos de regresión.

Validación Cruzada

Para la experimentación y evaluación que se llevará a cabo, estableceremos la metodología de evaluación que nos aporte resultados relevantes y fiables, y que asegure una interpretabilidad consistente entre los modelos, los datos y los experimentos entre sí. Se usará, por tanto, la **Validación Cruzada** como técnica de evaluación global [5], la cual explicaremos a continuación.

Cuando disponemos de un conjunto de datos con los que vamos a entrenar un modelo predictivo, resulta algo necesario establecer dos particiones principales de las muestras: *Train* para el aprendizaje y *Test* para evaluar dicho aprendizaje. Esto se hace con el fin de comprobar si el modelo predictivo conseguido ha sido capaz de adquirir un aprendizaje general de los datos, ya que el grupo de datos con el que es evaluado no ha estado presente en la fase de entrenamiento.

No obstante, dependiendo de cómo se generen estas particiones y de sus tamaños, la evaluación llevada a cabo podría no indicarnos correctamente la validez del modelo utilizado. Para evitar esta irregularidad se puede aplicar la Validación Cruzada. En ella se generan N particiones de datos de un mismo volumen y se escogen N-1 particiones para entrenar el modelo. Finalmente, la partición restante es usada como conjunto de *Test* para la fase de evaluación. Este proceso se repite N veces de forma que todas las particiones hayan sido una única vez el grupo de *Test*, como se muestra en la *ilustración 11*. Finalmente, se agregan todos los resultados obtenidos para obtener la evaluación final del modelo.

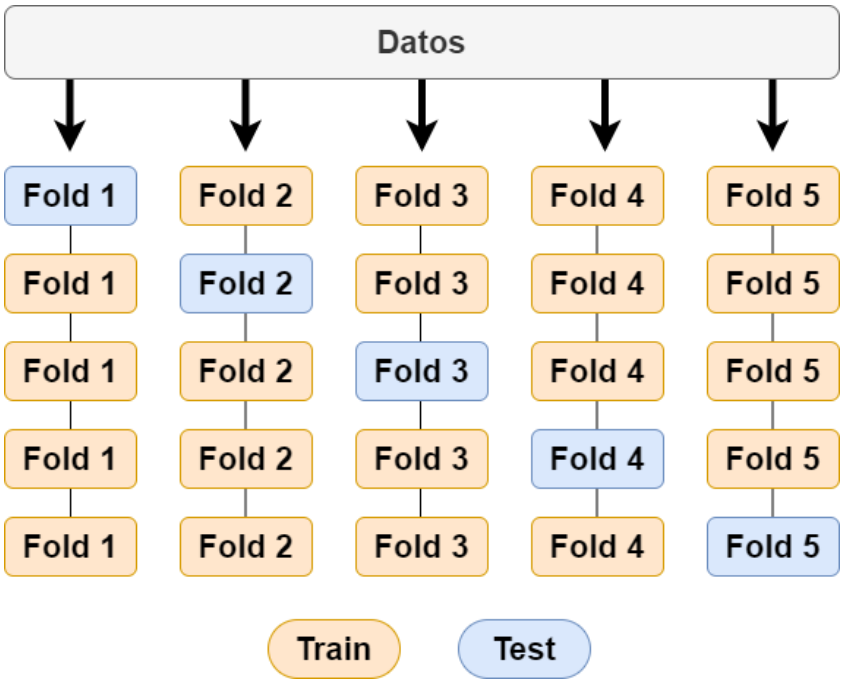


Ilustración 11: *Esquema de 5-Fold Cross-Validation. Cada fila corresponde con un entrenamiento y una evaluación usando las particiones de Train y Test respectivamente.*

Esta técnica nos garantiza un cálculo de métricas adecuado independientemente de la repartición de los datos a la hora de entrenar y posteriormente testear, además de darnos una idea más realista de las aptitudes del modelo que en ese momento estemos entrenando y evaluando.

Métricas

Con el objetivo de medir y cuantificar el error obtenido por el modelo predictivo de regresión y su eficacia, en cada experimento se utilizarán las variables de respuesta y sus predicciones para calcular el *Mean Squared Error* (***MSE***), el *Mean Absolute Error* (***MAE***), el *Root Mean Squared Error* (***RMSE***) y el coeficiente de determinación o R^2. En el caso de encontrarse con un problema de clasificación, la métrica usada será la exactitud o ***Accuracy***. Las tres primeras métricas corresponden a los siguientes cálculos, donde *y* es la clase real del sujeto e *y'* la predicción:

$$\boldsymbol{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i^{'} - y_i)^2 \qquad \boldsymbol{MAE} = \frac{1}{n}\sum_{i=1}^{n}(y_i^{'} - y_i)$$

$$\boldsymbol{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i' - y_i)^2}$$

Por otro lado, para el cálculo del coeficiente de determinación R^2 se utilizarán las librerías comentadas anteriormente mientras que el *Accuracy* se calcula en el caso de utilizar un modelo de clasificación como:

$$Accuracy = \frac{Predicciones\ correctas}{\#\ total\ de\ predicciones}$$

Además, de forma generalizada, las métricas que se usarán con frecuencia para exponer los resultados de los modelos serán el ***RMSE*** y el **R^2**. Por otra parte, debido a que algunos sujetos podrían estar presentes en más ocasiones en el dataset que otros (más tomas de datos o duración del muestreo), se tratarán de **agrupar las predicciones por cada usuario**, para que así cada paciente aporte lo mismo al cálculo final.

3. Preprocesado de los datos

Antes de realizar un entrenamiento con los datos obtenidos del estudio es necesaria una fase de procesado previo, para así reducir el ruido, los artefactos o desperfectos que no representen de forma correcta la naturaleza de las señales. De esta manera, seremos capaces de insertar a los modelos un conjunto de datos limpio que contenga lo necesario para poder extraer las características más relevantes y aprender de ellas. Comenzaremos por retirar las muestras que no nos sirvan para el estudio o buscar maneras de organizarlas y almacenarlas de forma más eficiente. Después describiremos diversas metodologías de normalización para depurar las señales, seguidas de métodos de filtrado para suavizarlas y simplificarlas.

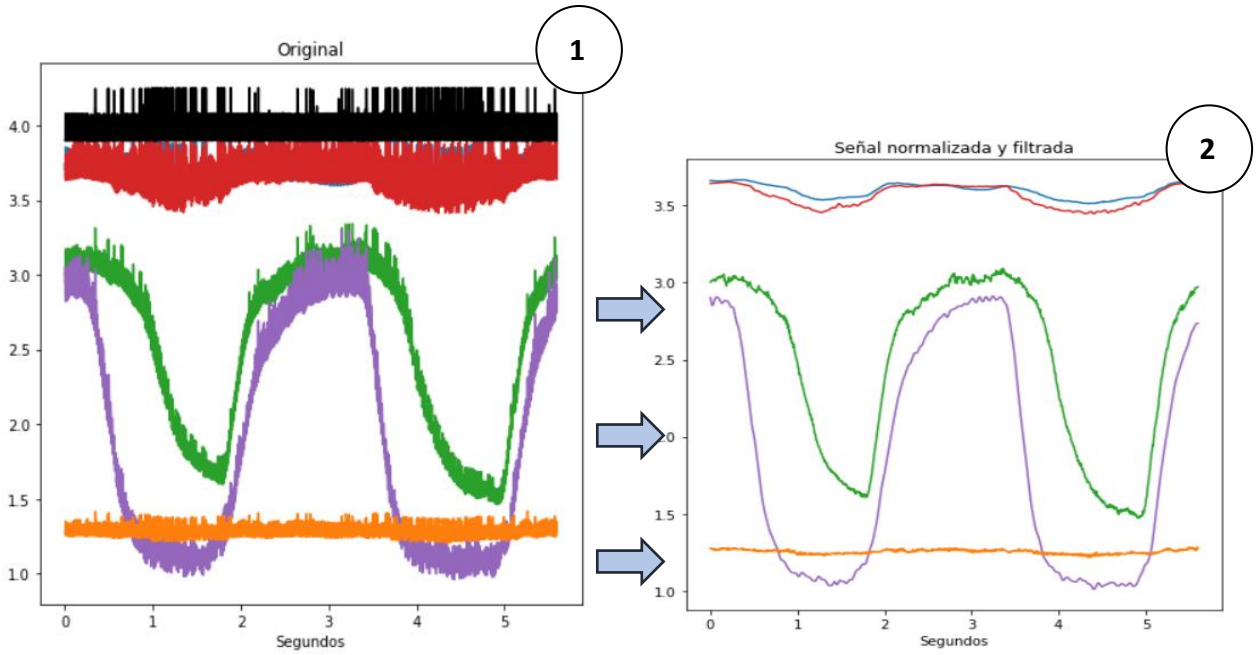


***Ilustración 12**: Preprocesado de una muestra del dataset. **1)** Señal original. Contiene ruido y artefactos. **2)** Señal normalizada y filtrada. Preparada para una extracción de características.*

En la *ilustración 12* se muestra un ejemplo de este preprocesado. Al comienzo, la señal original resulta ruidosa y presenta artefactos, los cuales dificultan su interpretación y complican una labor de aprendizaje por parte de los modelos. Tras esta fase de limpieza, logramos un dataset de señales más limpio, interpretable y preparado para una extracción de características

En una primera instancia, los ficheros con las señales son leídos y sus datos son representados para realizar un primer cribado y así eliminar del repertorio los sensados fallidos, erróneos o que no cumplan unos mínimos de calidad de señal que los hagan viables. Por lo general, los ficheros de los que disponemos son similares a los mostrados en las ilustraciones siguientes (*13* y *14*) y en ellos se pueden apreciar las fases respiratorias que el usuario ha llevado a cabo durante la medición.

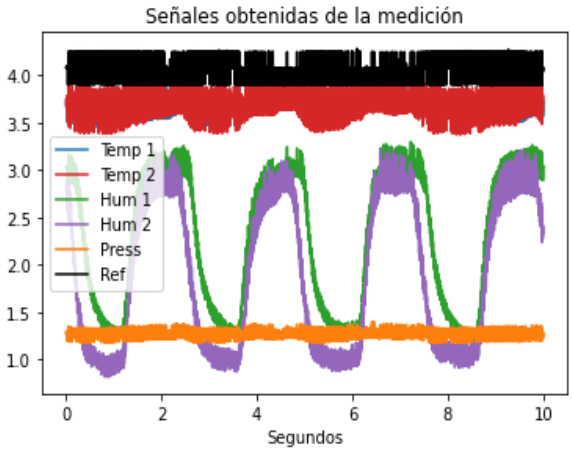


Ilustración 13: *Ejemplo de señales disponibles en el dataset. (1)*

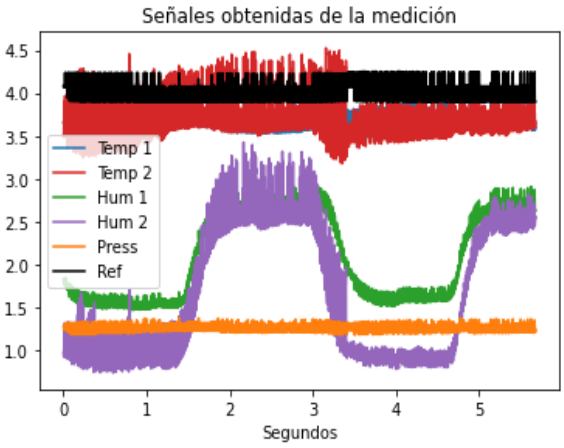


Ilustración 14: *Ejemplo de señales disponibles en el dataset. (2)*

Por desgracia, algunas de las mediciones que se realizaron resultaron fallidas, defectuosas o no apropiadas por diversos motivos, por lo que tuvieron que ser procesadas o descartadas del proyecto. Algunos de estos casos corresponden con **mediciones vacías** (*i. 15*), posiblemente con el propósito de comprobar el correcto funcionamiento del sensor en ausencia de un paciente, **señales excesivamente cortas** (*i. 16*) que imposibilitan cualquier extracción de características o muestras en las algunos de los sensores han resultado **saturados** (*i. 17*) y no pueden ser utilizados.

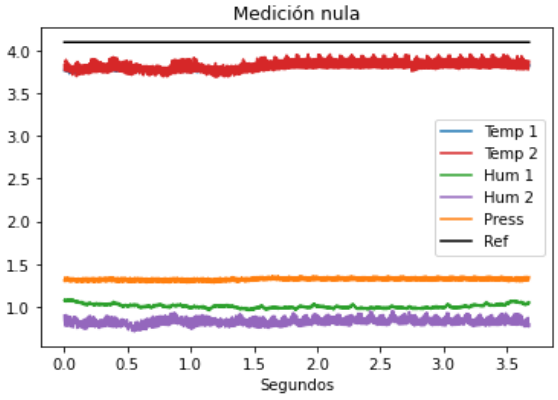


Ilustración 15: *Sensado sin respiración presente por parte del paciente.*

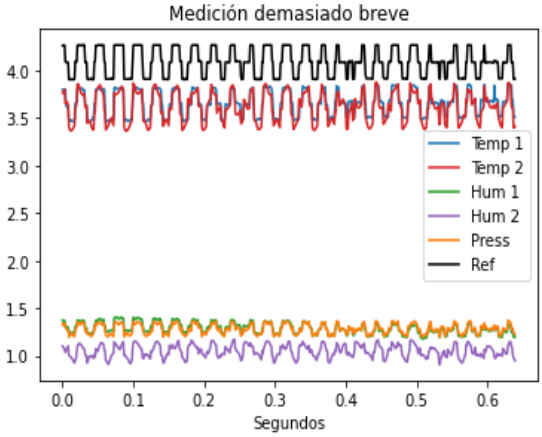


Ilustración 16: *Medición insuficiente de 600ms de duración.*

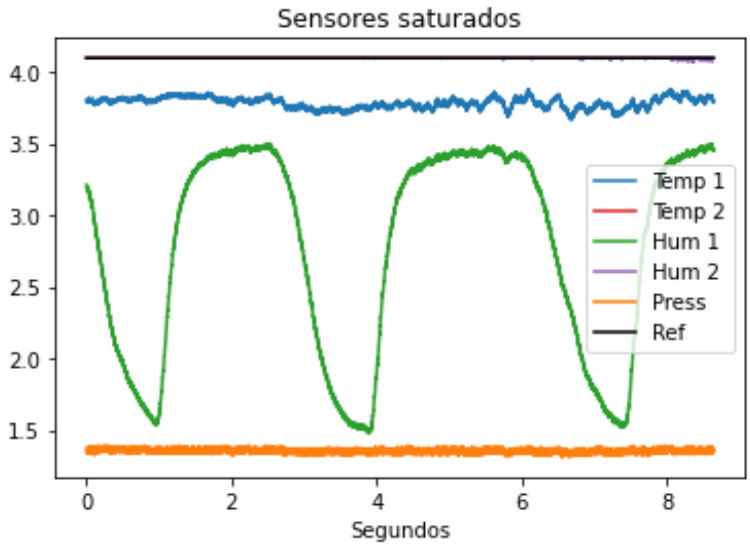


Ilustración 17: *Muestra con los sensores Vref, Temp2 y Hum2 saturados al valor 4.096.*

Para asegurar una consistencia en el repertorio de datos, se ha efectuado un cribado manual consiguiendo así eliminar las muestras incorrectas encontradas en el dataset inicial. Por otro lado, las mediciones parcialmente defectuosas han sido valoradas individualmente y, en los casos pertinentes, procesadas o recortadas para recoger únicamente los fragmentos útiles y aprovechar al máximo los datos disponibles.

Durante la limpieza de los datos se observa que la **longitud máxima** que almacena cada fichero es de **10 segundos** de muestreo, lo que nos lleva a deducir una limitación del sensor: las mediciones más largas resultan fragmentadas en múltiples archivos en tramos de un máximo de 10 segundos. Esto conlleva que, por ejemplo, un sensado continuo e ininterrumpido de 23 segundos finalice con 3 ficheros de longitudes de 10s, 10s y 3s respectivamente, por lo que ciertos muestreos aparentemente muy breves podrían resultar ser el tramo final de otras mediciones más extensas. Para solventar esta situación, se recorre el dataset completo en busca de tomas contiguas de un mismo paciente y se trata de unir estas, consiguiendo un conjunto de datos más compacto y organizado.

Se muestra en la *ilustración 18* cómo se ha llevado a cabo esta unión en un ejemplo real. En la fila superior, se observan dos tomas de datos que pueden ser unificadas para compactar el dataset. El fichero *A*, en este caso, tiene una duración exacta de 10 segundos e, inspeccionando el fichero *B*, concluimos que ambas muestras corresponden a una misma sesión que resultó fragmentada al ser almacenada.

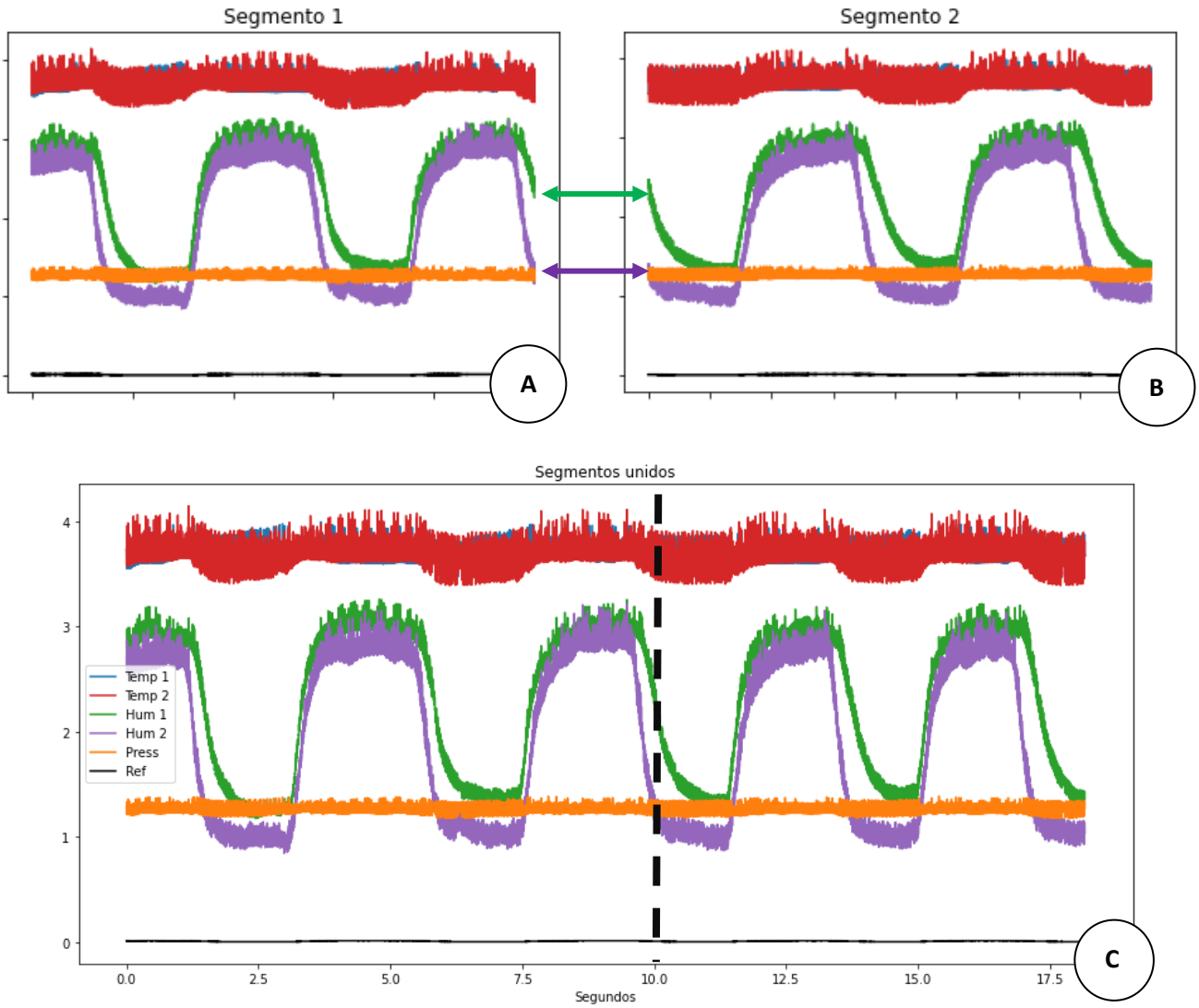


Ilustración 18: *Unión de muestreos consecutivos. A) Primer tramo de 10 segundos. B) Segundo tramo de 8 segundos. C) Unión resultante de 18 segundos de duración*

Es importante recalcar que esto debe ser previo a cualquier normalización o filtrado, ya que unir mediciones después de un preprocesado puede hacer que estas no encajen de forma correcta y genere ciertos problemas posteriormente. Afortunadamente, puesto que los ficheros fueron almacenados en orden cronológico, las muestras contiguas fueron más fáciles de localizar.

Más adelante en el proyecto, se obtuvo un nuevo lote de muestras con el fin de ser integrado al sistema y aumentar el volumen de datos disponible. Por desgracia, algunas de estas mediciones carecían de uno de los 5 sensores mostrados hasta ahora: el sensor de humedad 2 (*ilust. 19*). Aunque debido a esta circunstancia no todos los nuevos datos pudieron ser incluidos, sí se ha tratado de aprovechar el conjunto extra en los casos en los que el sensor Hum2 no fuese crucial.

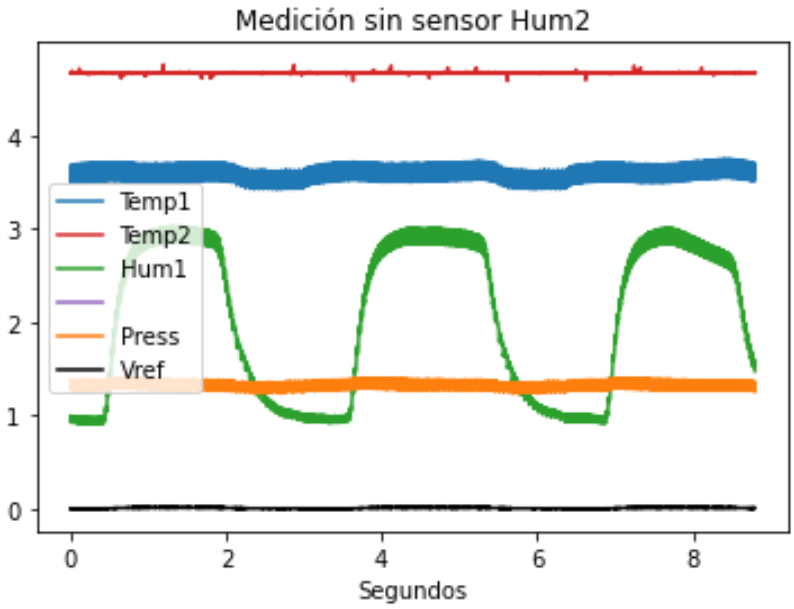


Ilustración 19: *Muestra tomada carente de sensor Hum2.*

3.1. Normalización de las señales

Como primera fase de preprocesado de las señales, se busca normalizar sus valores de forma que la gran perturbación que se observa en ellas se elimine en gran medida y los valores que conforman los ficheros se encuentren contenidos en un rango estándar entre 0 y 4.1. Asimismo, se conseguiría que las señales resulten comparables y se mantenga una coherencia general en el dataset. Se han experimentado varias metodologías de normalización que estudiaremos a continuación.

3.1.1. Normalización por señal de referencia (Vref)

Una de las 6 señales de las que disponemos (Vref) puede servirnos como referencia para corregir los posibles artefactos que encontremos en las muestras, por lo que trataremos de utilizarla para hacer una primera limpieza de las señales. Aplicaremos este método utilizando la diferencia y el factor multiplicativo.

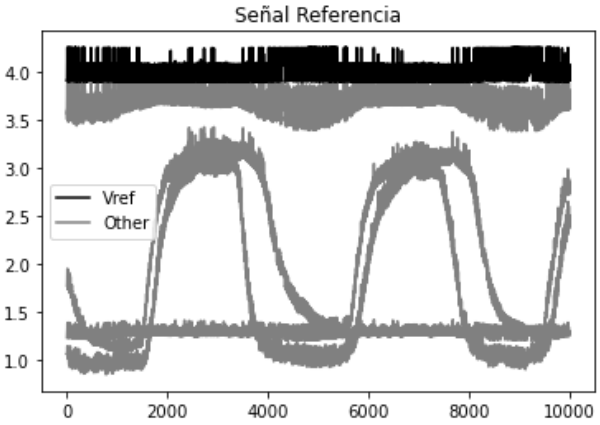


Ilustración 20: *Señal de referencia en color negro.*

Diferencia de Vref con valor ideal

En primer lugar, estableceremos un voltaje V_0 de referencia al que consideramos que Vref debería aproximarse en un caso ideal. Dependiendo del valor sobre el que oscile esta señal escogeremos un V_0 más alto o bajo; en este caso, utilizaremos un V_0 de 4.0V. A continuación, calcularemos la **diferencia** entre V_0 y la señal Vref en cada punto, y la sustraemos a todas las señales, eliminando los artefactos que hayan afectado a todos los sensores y se vean reflejados en Vref. Este proceso resulta en un Vref completamente horizontal y una reducción significativa de los cambios bruscos que hacían a las señales muy ruidosas.

$$Para\ cada\ punto\ x\ de\ la\ señal\ S, con\ voltaje\ ideal\ V_0:$$

$$S[x] = S[x] - (Vref[x] - V_0)$$

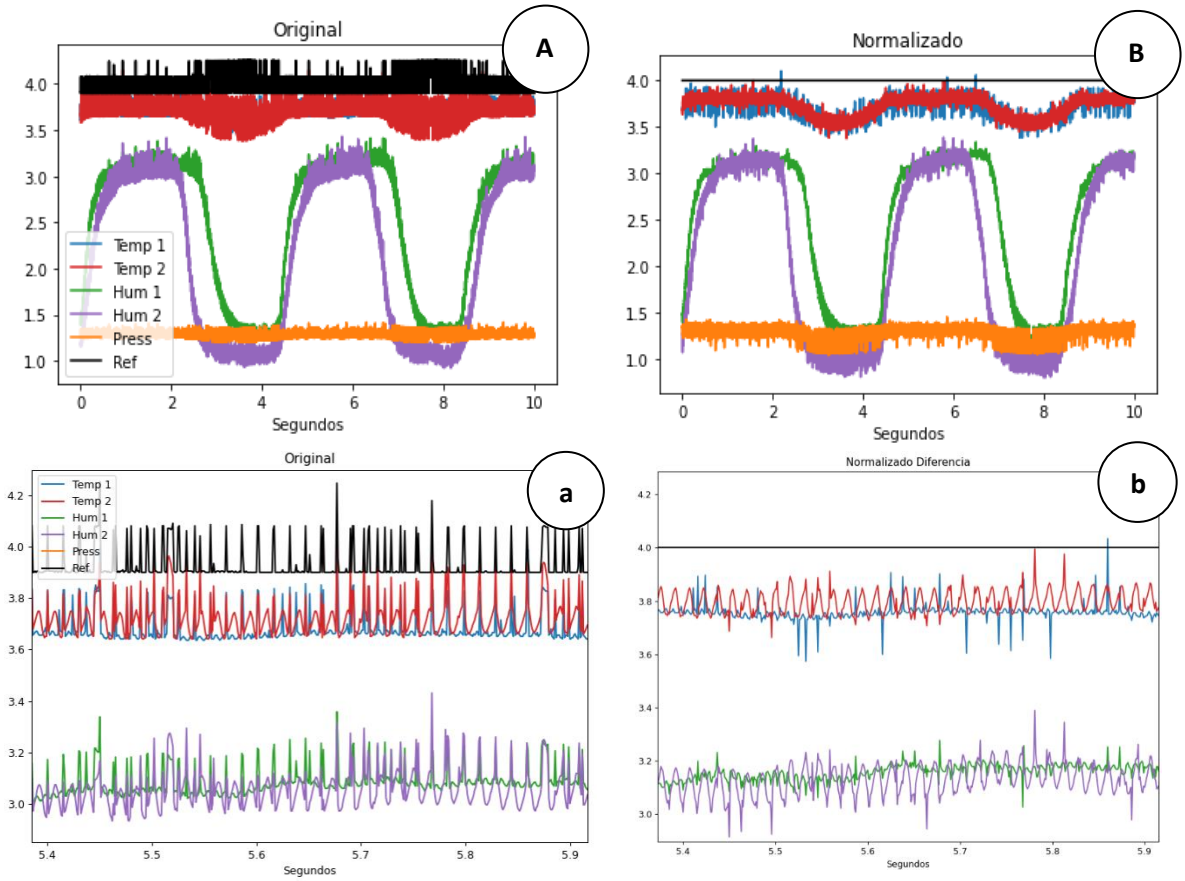


Ilustración 21: *Normalización mediante Vref (Diferencia).*
A) *Señal original.* ***a)*** *Zoom de 500ms sobre la señal original.*
B) *Señal normalizada con V0 = 4.0.* ***b)*** *Zoom de 500ms sobre la señal normalizada.*

Factor multiplicativo entre Vref y valor ideal

De forma muy similar a lo visto anteriormente, compararemos la señal Vref con un valor fijo V_0 que establezcamos, pero esta vez calcularemos el **factor multiplicativo**. Por tanto, el factor que obtengamos en cada punto de la señal lo multiplicaremos a todos los de sensores para normalizarlos. En este caso, también conseguiremos una señal Vref plana, aunque el resto de señales resultarán aún menos ruidosas.

$$Para\ cada\ punto\ x\ de\ la\ señal\ S, tomamos\ voltaje\ ideal\ V_0:$$

$$S[x] = S[x] * \frac{V_0}{Vref[x]}$$

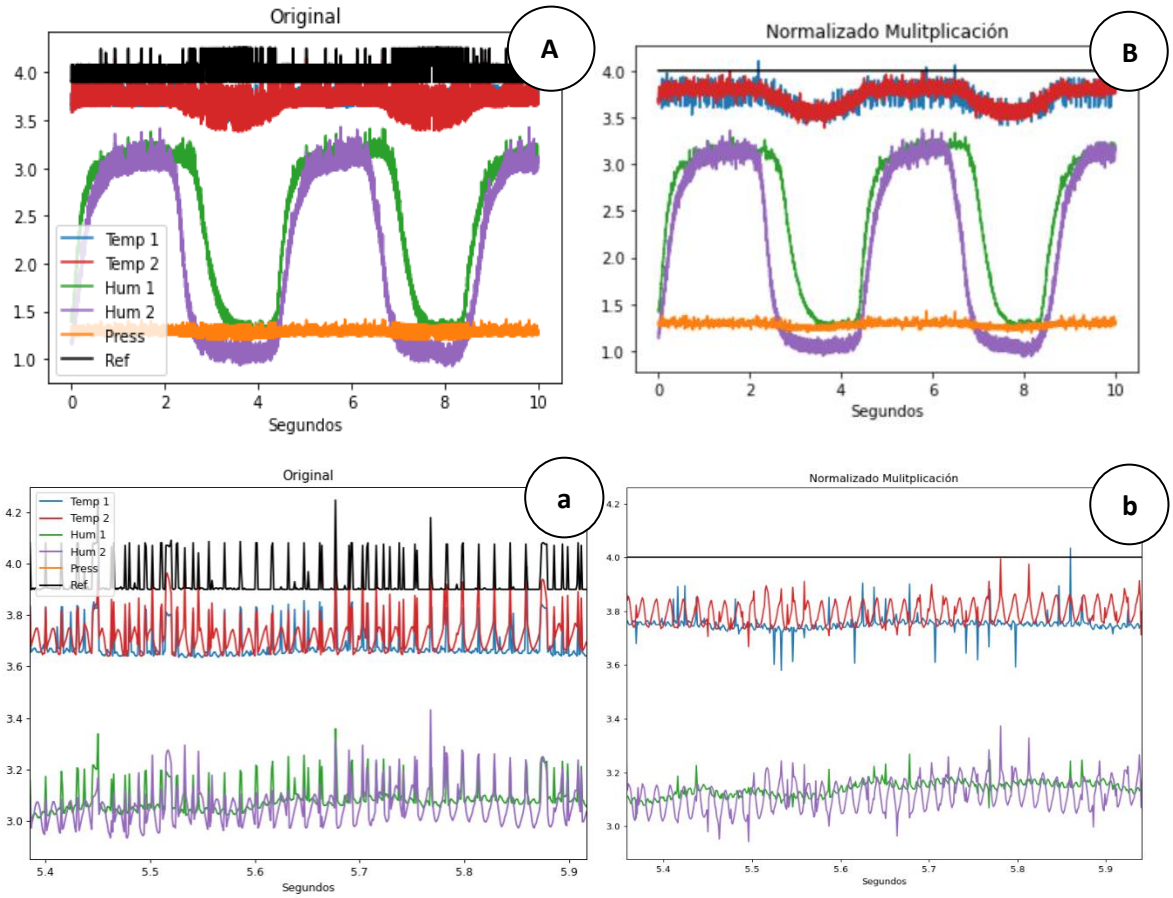


Ilustración 22: *Normalización mediante Vref (Multiplicación).*
A) *Señal original.* ***a)*** *Zoom de 500ms sobre la señal original.*
B) *Señal normalizada con* V_0 = 4.0. ***b)*** *Zoom de 500ms sobre la señal normalizada.*

Observando los resultados de esta metodología más de cerca en las *ilustraciones 21 y 22*, es aparente que la normalización no es igualmente eficaz en ambos grupos de sensores. En concreto, en las muestras del conjunto 1 (Temp1 y Hum1) la reducción de ruido es más notable, mientras que el sensado del conjunto 2 (Temp2 y Hum2) conserva todavía una parte importante de estos artefactos. Esto puede ser un indicativo de que la **normalización no es selectiva** y que la señal Vref podría no ser la más adecuada en esta situación, por lo que continuaremos explorando otro método alternativo.

3.1.2. Normalización mediante las señales de Temperatura

Otra manera que tenemos de eliminar o reducir los artefactos y ruidos es mediante las señales obtenidas por los sensores de temperatura, con la peculiaridad de que en este caso contamos con dos referencias. De esta forma, podemos orientar la normalización de cada señal de humedad utilizando su correspondiente señal de temperatura. Estos métodos serán similares a los experimentados con Vref, aunque necesitaremos primero estudiar el comportamiento de *Temp1* o *Temp2* para saber cómo utilizarlas como referencia.

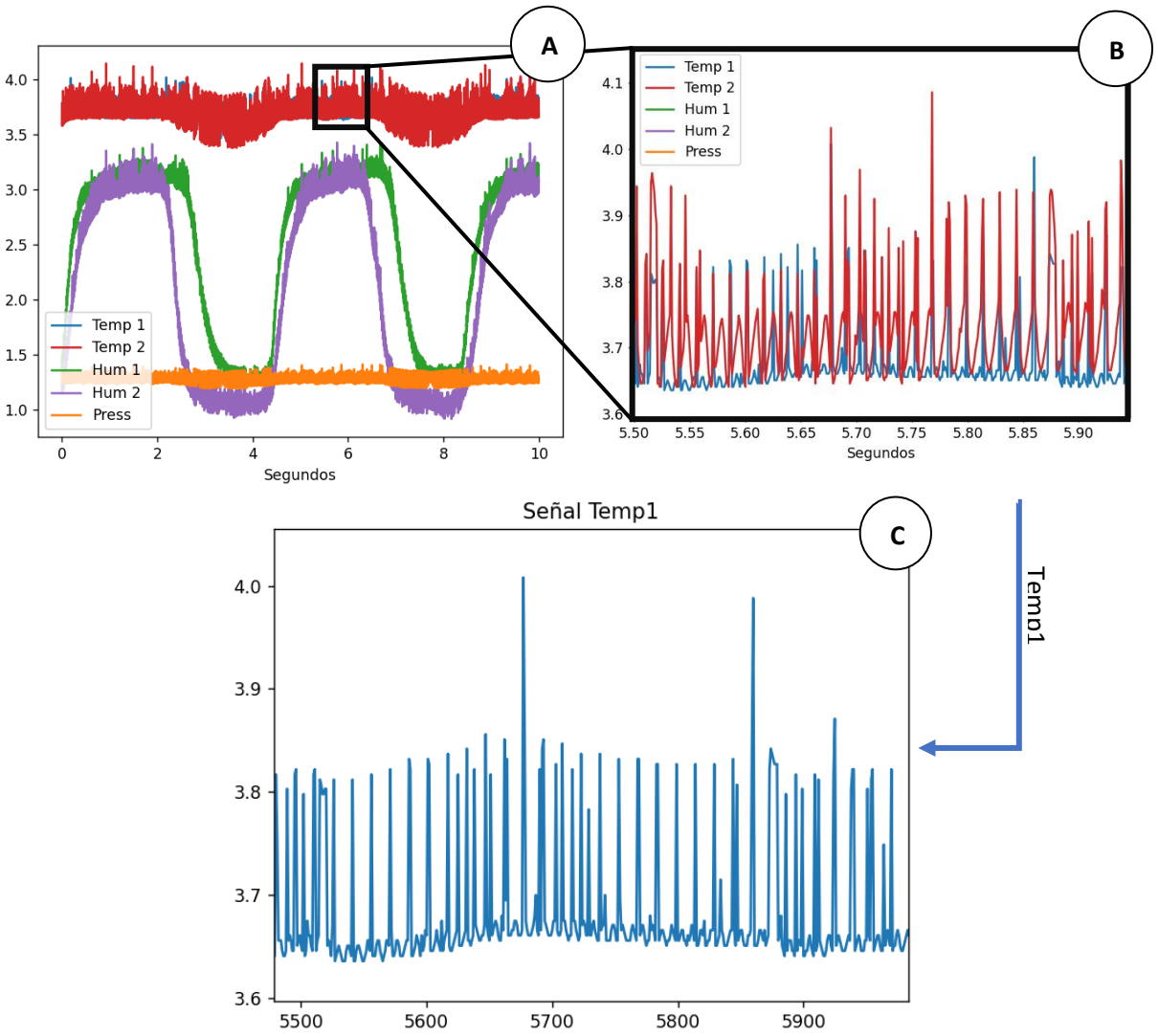


Ilustración 23: *Inspección de las señales de temperatura.* ***A)*** *Señal completa.* ***B)*** *Zoom sobre un tramo de las señales de Tª.* ***C)*** *Aislamiento de la señal Temp1.*

Si observamos únicamente las señales de temperatura observaremos que las alteraciones y desperfectos que en ellas se encuentran resultan característicos: los valores mantienen una línea basal de la cual surgen picos de forma irregular. Por lo tanto, si procesamos estas muestras asumiendo que el comportamiento ideal se corresponde con esta línea base podremos reducir los artefactos significativamente y normalizar las señales con esta referencia.

Para lograrlo, primero calcularemos los valores de esta línea base conforme recorremos la señal para así obtener la referencia que necesitamos en cada punto de esta. Debido a la distribución de la señal que hemos descrito, optaremos por extraer las muestras de la señal menores a un percentil establecido, de forma que conservaremos los mínimos, pero los picos desproporcionados que deterioran la señal serán ignorados. Este proceso será realizado utilizando ventanas de una anchura determinada para asegurar que la referencia obtenida en cada punto proviene del contexto que la rodea y no de la muestra aislada, pero que a su vez las regiones más alejadas no intervengan en su cálculo.

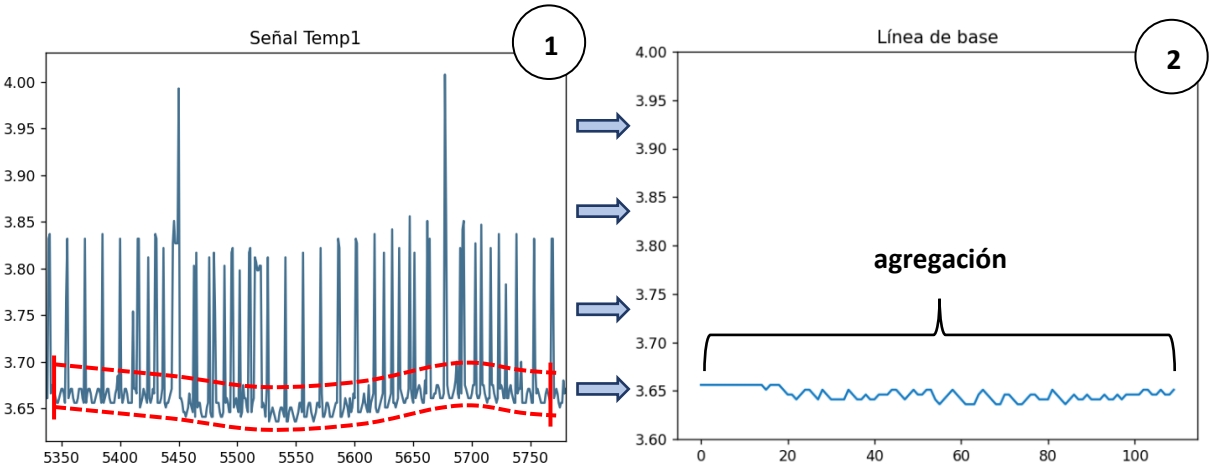


Ilustración 24: *Eliminación de las muestras mayores al percentil 10 alrededor del milisegundo i = 5550.* ***1)*** *Señal original.* ***2)*** *Línea de base extraída.*

Dentro de cada ventana se ordenarán todos valores de menor a mayor y se usarán los *p*% más pequeños de todos ellos, siendo *p* el percentil escogido. El resultado, representado en la gráfica derecha de la *ilustración 24*, después **es compactado en un único valor** mediante la media o un filtro gaussiano para obtener un único número. El valor obtenido de este proceso sobre el milisegundo ***x*** es almacenado y corresponderá con el valor de referencia ***x*** más adelante en la normalización. Una vez realizado este cálculo para cada muestra de la señal, tendremos una línea base de referencia de la misma longitud que la señal original (*ilust. 25*) y podremos dar el siguiente paso:

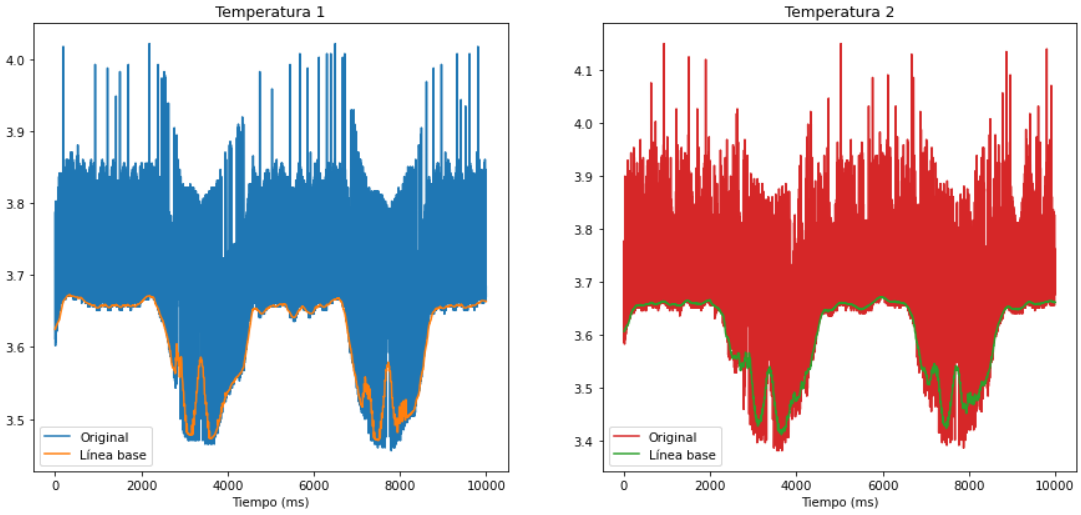


Ilustración 25: *Señales de temperatura junto con su cálculo de línea base correspondiente. A la izquierda, el muestreo de* ***temperatura 1****. A la derecha, el muestreo de* ***temperatura 2****.*

Diferencia entre temperatura y su línea base

Similar a los métodos anteriores, primero calcularemos la diferencia entre dos señales. En este caso, utilizaremos la resta de cada **señal de temperatura y su señal de referencia** o línea base. Esta diferencia nos permitirá determinar la posición y magnitud de los picos y artefactos para luego eliminarlos del resto de señales. El resultado nos mostrará unas señales de Tª más aplanadas y suavizadas. Para todas las señales ***S*** de cada grupo deberemos utilizar su correspondiente señal de Temperatura ***T*** con su respectiva línea de base ***L*** como se muestra a continuación:

$$S[x] = S[x] - (T[x] - L[x])$$

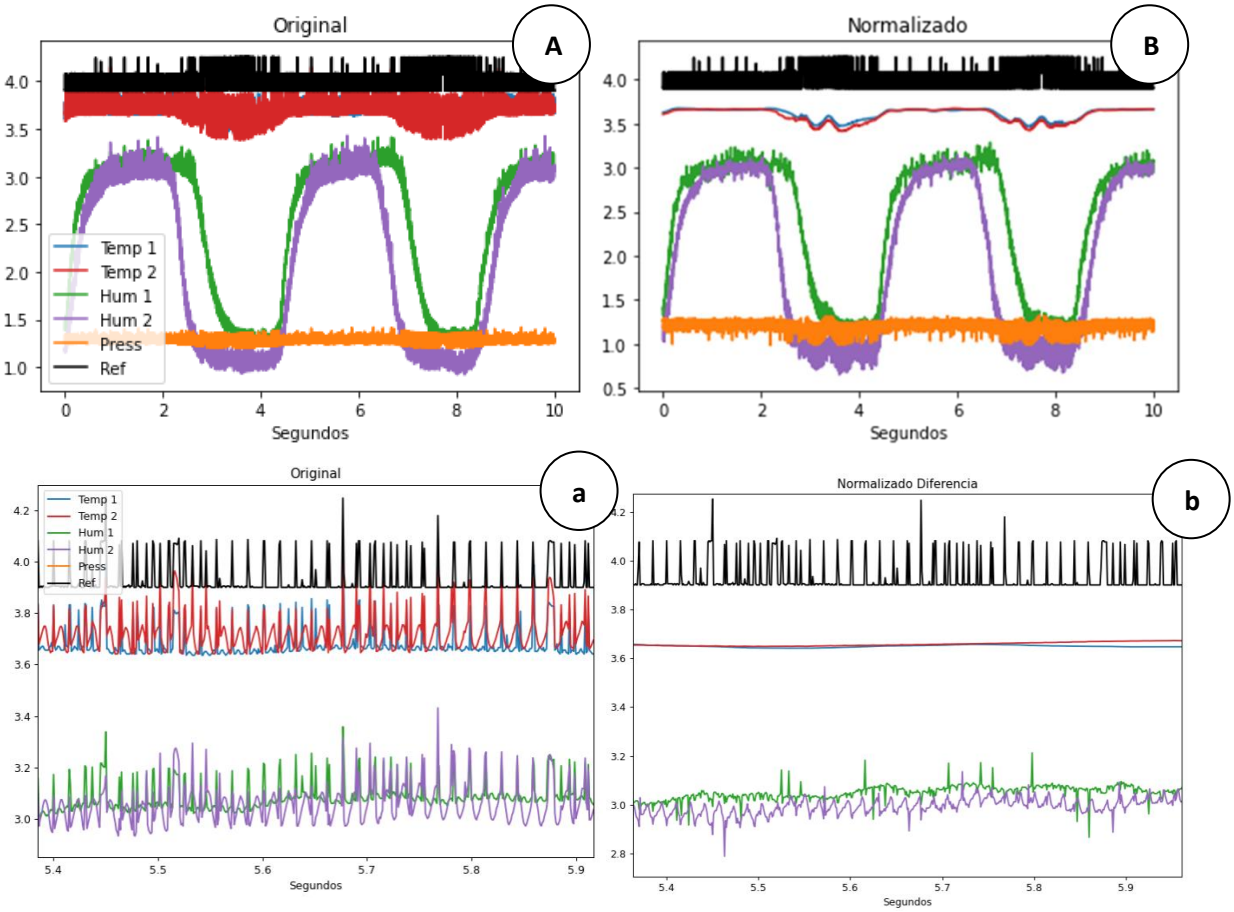


Ilustración 26: *Normalización por diferencia utilizando señales de temperatura.*
A) *Señal original.* ***a)*** *Zoom de 500ms de la señal original.*
B) *Señal normalizada por temperatura.* ***b)*** *Zoom de 500ms de la señal normalizada.*

Factor multiplicativo entre temperatura y su línea base

Por otra parte, podemos aplicar la normalización mediante factor multiplicativo de la misma manera que anteriormente ha sido expuesta. Esta vez calcularemos este factor entre las temperaturas y sus respectivas líneas base para completar esta normalización, siguiendo la misma nomenclatura:

$$S[x] = S[x] * \frac{L[x]}{T[x]}$$

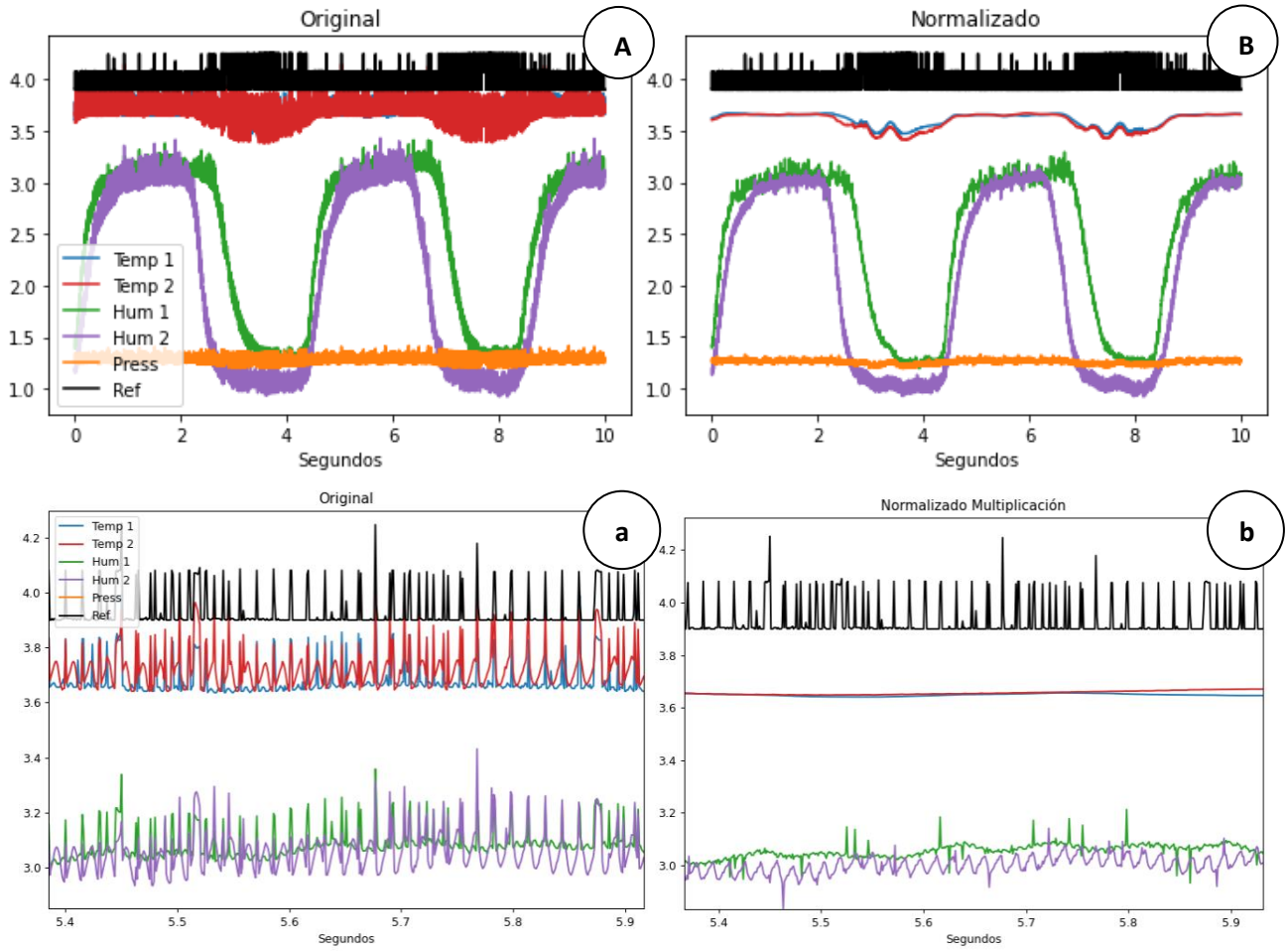


Ilustración 27: *Normalización por multiplicación utilizando señales de temperatura.*
A) *Señal original.* ***a)*** *Zoom de 500ms de la señal original.*
B) *Señal normalizada mediante multiplicación.* ***b)*** *Zoom de 500ms de la señal normalizada.*

Por lo que podemos ver, usar la temperatura como referencia para la normalización resulta más acertado, pues conseguimos una mayor eliminación de ruido y suavidad en las señales. No obstante, los resultados al utilizar la diferencia con respecto a la multiplicación resultan muy similares, por lo que observaremos otra región de la señal a modo de comparación. Asimismo, como vemos en la *ilustración 28*, la normalización mediante el **factor multiplicativo** entre las **señales de Tª y sus líneas de base** resulta la más acertada.

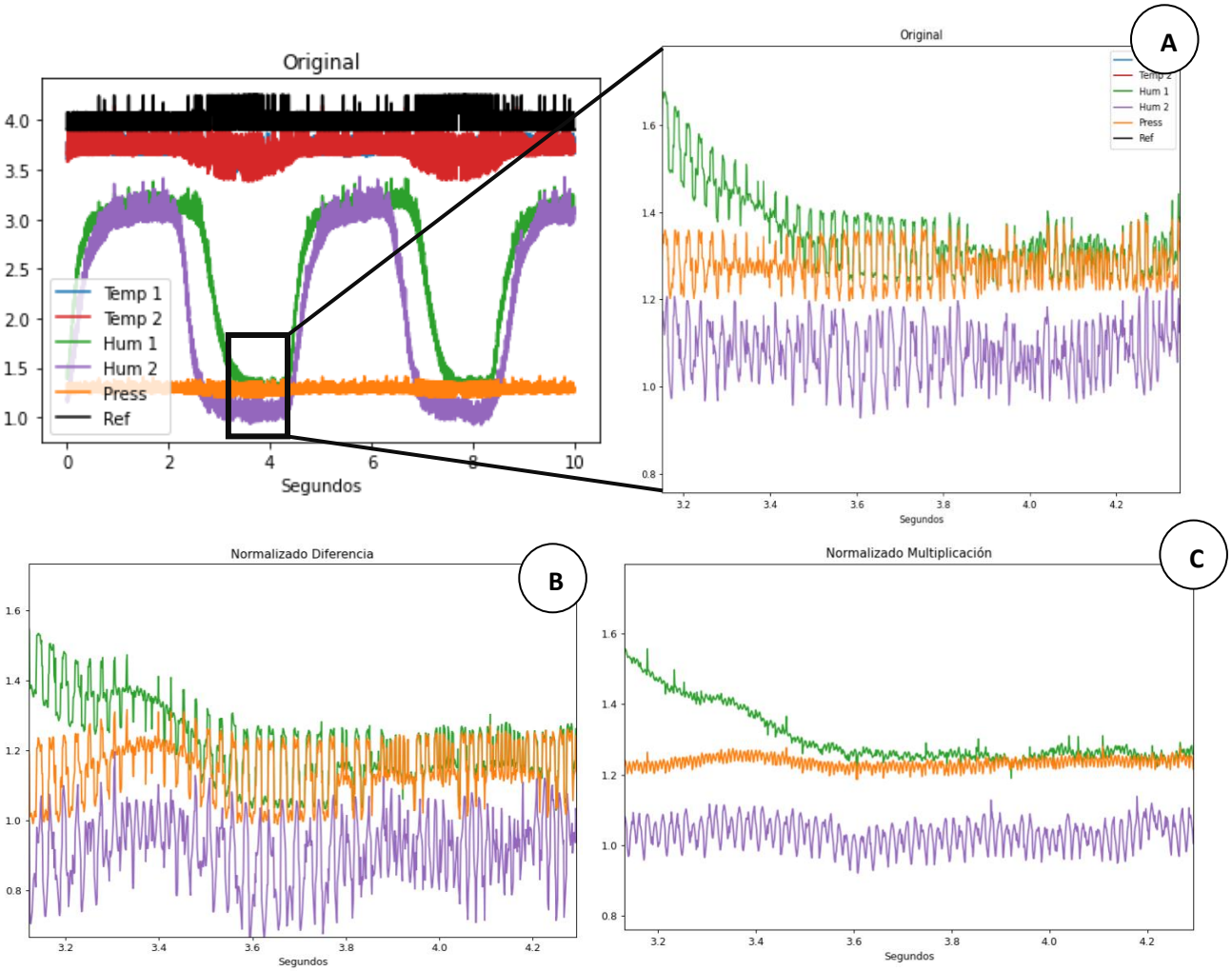


Ilustración 28: *Comparativa de ambos métodos de normalización mediante temperatura.* ***A)*** *Señal original.* ***B)*** *Normalización por diferencia.* ***C)*** *Normalización por multiplicación.*

3.2. Filtrado de las señales

Una vez tengamos las muestras normalizadas y limpiadas podemos proceder al siguiente y último paso: el filtrado. Esta fase final de preprocesado nos permitirá suavizar y estabilizar los valores sensados para facilitar tanto la manipulación de los datos y la extracción de características como el aprendizaje de los modelos que posteriormente implementemos. Existen múltiples tipos de filtros que podemos aplicar para lograr este propósito, por lo que a continuación estudiaremos las opciones barajadas y estableceremos cuál será la utilizada. En primer lugar, analizaremos el espectro frecuencial de las señales originales y normalizadas:

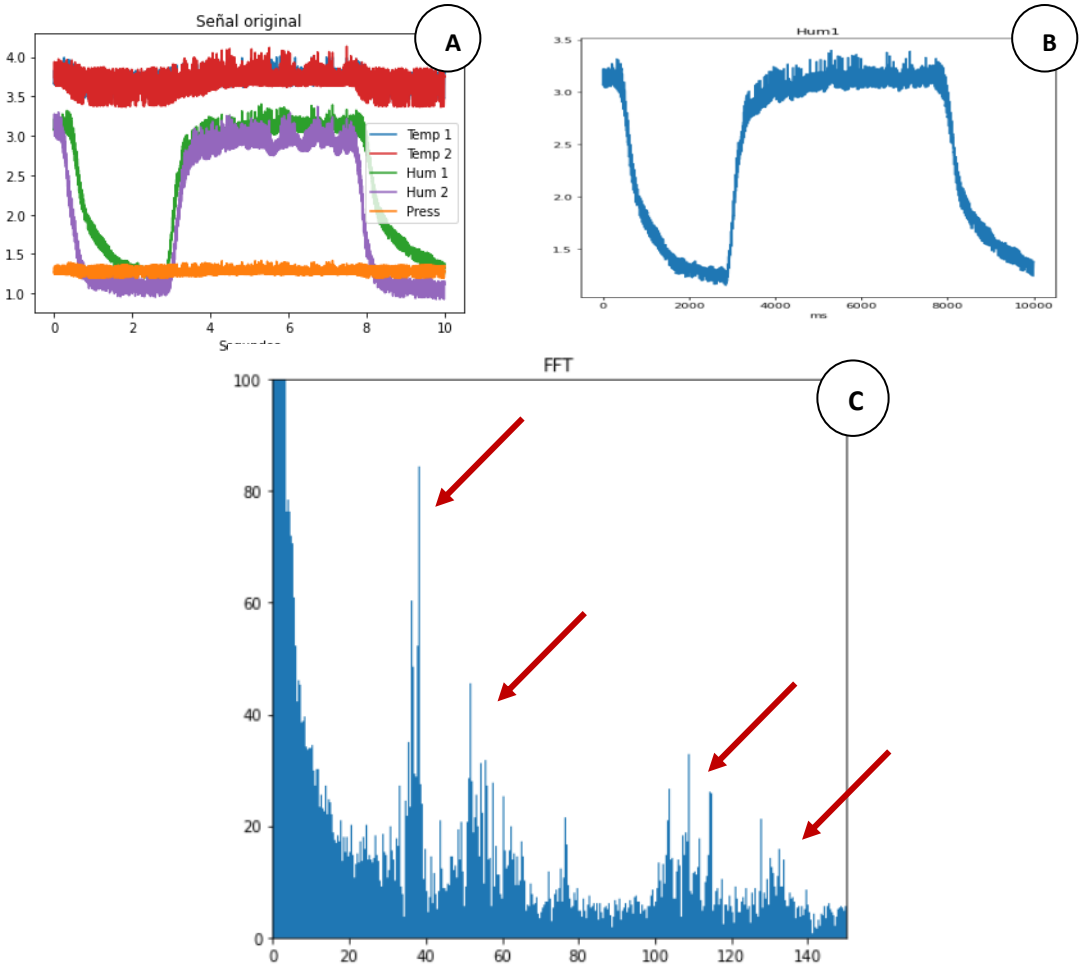


Ilustración 29: *Transformada de Fourier aplicada a la señal de humedad 1.*
A) *Muestra completa* ***B)*** *Señal Hum1.* ***C)*** *Transformada sobre Hum1.*

Teniendo en cuenta que la frecuencia de muestreo es de 1000Hz podemos descomponer la señal en las frecuencias que la componen, como se observa en la *ilustración 29*. De esta manera, podemos focalizar el filtrado de la manera que consideremos más adecuada. En la imagen se pueden identificar múltiples picos frecuenciales, entre ellos los 50Hz asociados al ruido eléctrico.

Si calculamos la transformada de la señal una vez normalizada obtendremos lo mostrado en la *ilustración 30*, donde el espectro frecuencial resulta más liso, indicando una vez más que se han logrado eliminar algunos de los artefactos. En vista de estos espectros, experimentaremos con dos tipos de filtros para lograr preparar finalmente los datos para entrenar los modelos.

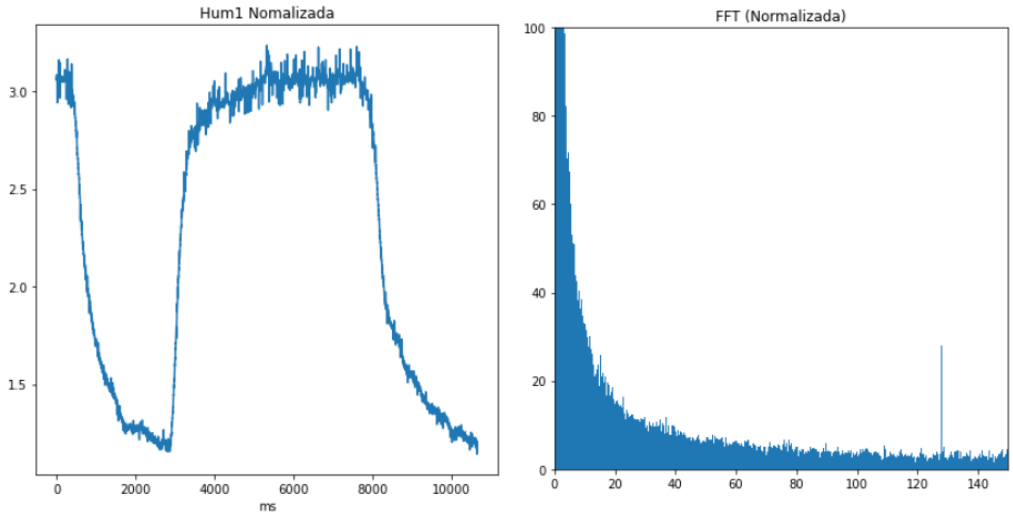


Ilustración 30: *Transformada de Fourier de la señal normalizada.*
A la izquierda, la señal Hum1 normalizada. A la derecha, su transformada.

3.2.1. Filtrado Notch o Band-stop

Como primer intento de filtrado buscaremos eliminar un rango frecuencial en las señales para hacernos cargo del rizado que estas contienen y así lograr un suavizado focalizado en un segmento del espectro. El filtro necesario para ello es el Notch, el cual tiene la forma siguiente:

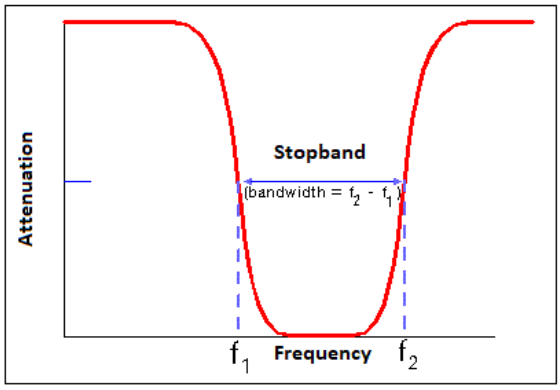


Ilustración 31: *Representación gráfica del filtro Notch sobre un espectro frecuencial.*
Fuente.

Tras observar las *ilustraciones 29 y 30* y conociendo las frecuencias que queremos eliminar, experimentaremos con el filtro para tratar de reducir la aportación de frecuencias alrededor de 40Hz, 50Hz, 110Hz y 125Hz. No obstante, debemos tener en cuenta que la amplitud de la señal puede verse afectada en función del rango de frecuencias que anulemos, por lo que deberemos conservar la parte más baja del espectro y comprobar si el filtro Notch nos da un buen filtrado. Compararemos una señal pre-filtro con su posterior suavizado en la **banda frecuencial de 40Hz a 140Hz**.

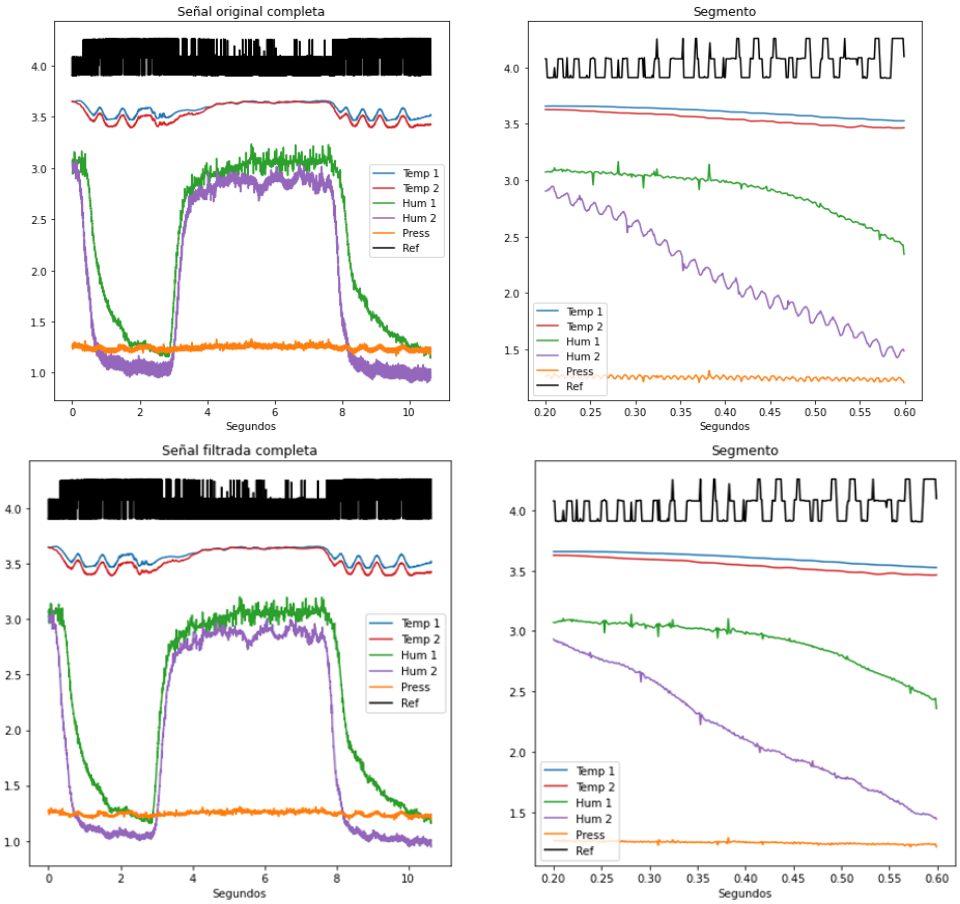


Ilustración 32: *Filtrado Notch (40-140Hz) de la señal normalizada. Arriba, la señal original y un segmento ampliado. Abajo, la señal filtrada y un segmento ampliado.*

En vista de los resultados mostrados por la *ilustración 32*, es evidente que el filtro ha logrado un suavizado significativo, eliminando tanto las oscilaciones presentes en la señal Hum2 asociadas a frecuencias cercanas a los 50Hz, como el rizado encontrado en la señal Hum1 causado por regiones próximas a los 125Hz. Sin embargo, todavía quedan vestigios del ruido procedente de la señal original, posiblemente generados por frecuencias más altas que no hemos eliminado con el filtro *Notch*. En la *ilustración 33* queda más claro cómo la banda 40Hz-140Hz ha quedado reducida en la señal.

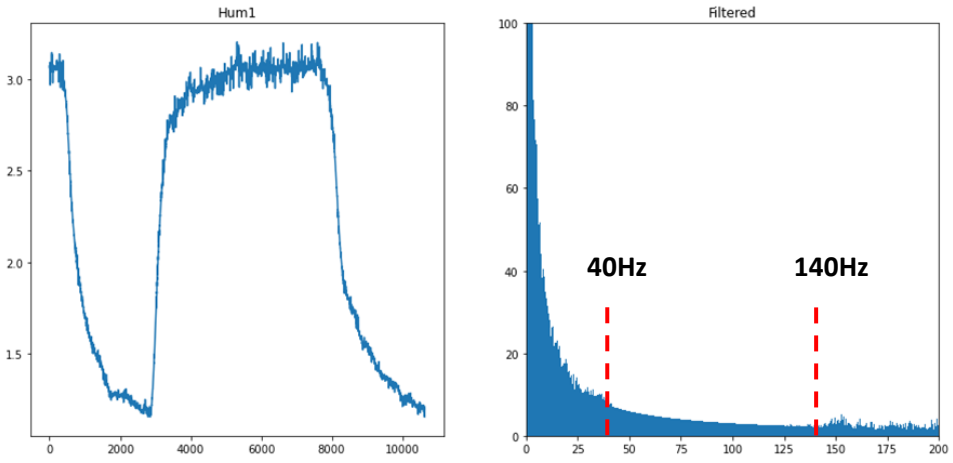


Ilustración 33*: Señal Hum1 y su espectro frecuencial tras el filtro Notch (40-140Hz).*

3.2.2. Filtrado de Paso Bajo (Lowpass)

Debido a que todavía se puede apreciar ruido procedente de frecuencias altas, probaremos a continuación con un filtro de paso bajo, el cual conservará únicamente las frecuencias más bajas que establezcamos. De la misma manera que en el caso anterior, compararemos la señal original con la obtenida tras un filtro lowpass y estudiaremos su efectividad en este proyecto. Esta vez, el suavizado tendrá este aspecto aplicado al espectro frecuencial:

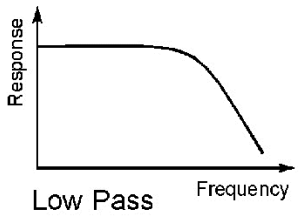


Ilustración 34: *Representación del filtro de paso bajo. Fuente.*

Debido a las características de los sensores que fueron utilizados, se toma la decisión de utilizar una frecuencia de corte distinta para cada grupo de sensores. Por tanto, utilizaremos una frecuencia de 100Hz para los aparatos del conjunto 1, mientras que en el resto de los sensores se usarán los 20Hz como corte. Queda evidente en la *ilustración 35* que el suavizado es más adecuado y nos da un filtrado más satisfactorio, puesto que además no perjudica la amplitud de la señal.

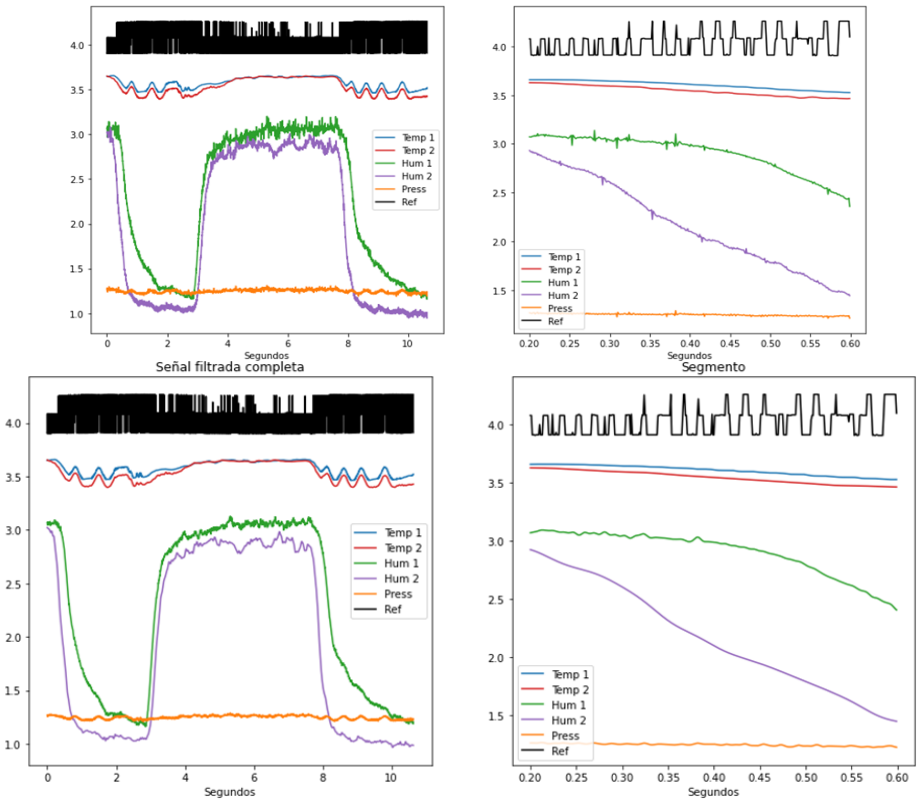


Ilustración 35: *Filtrado de Paso Bajo de 100Hz (Grupo 1) y 20Hz (Grupo 2) de la señal normalizada. Arriba, la señal normalizada y un segmento ampliado. Abajo, la señal filtrada y un segmento ampliado.*

Para concluir con esta sección, observaremos el espectro frecuencial de los sensores Hum1 y Hum2 para identificar este filtrado de paso bajo a distintas frecuencias de corte. Una vez hemos completado el filtrado contaremos finalmente con un dataset de señales limpio que puede ser utilizado para el entrenamiento.

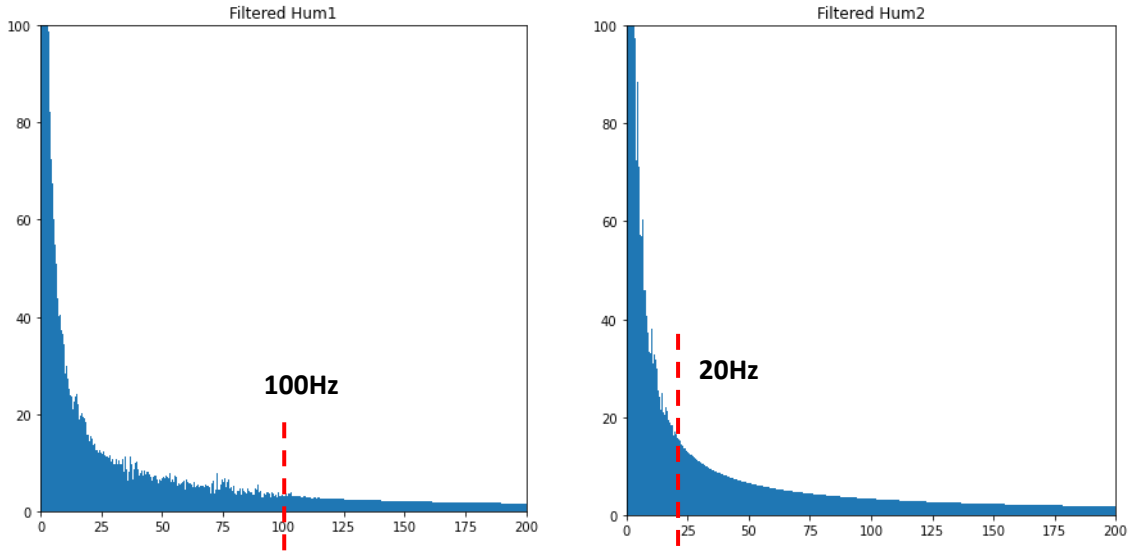


Ilustración 36: *Espectros frecuenciales de las señales filtradas. A la izquierda, la señal de humedad 1, filtrada a 100Hz. A la derecha, la señal de humedad 2, filtrada a 20Hz.*

3.3 Pares respiratorios

Las muestras pertenecientes al dataset, aunque limpias, carecen de una longitud constante, ya que la duración de la toma de datos de cada paciente ha sido ligeramente diferente. Esta irregularidad puede dificultar el método de aprendizaje con ciertos modelos, por lo que es de nuestro interés afrontarlo. Una manera que tenemos de solventar esta situación es crear una nueva unidad de información para el proyecto: **el par respiratorio**.

La respiración consta de dos fases distinguibles entre ellas que se ven reflejadas en nuestras señales: la **espiración** y la **inspiración**. Como se ilustra en la *imagen 37*, las subidas en humedad corresponden con una espiración del paciente, mientras que la bajada de este parámetro viene asociada a una inspiración. Esta característica la podremos aprovechar para crear los pares respiratorios, cuyo proceso describiremos a continuación.

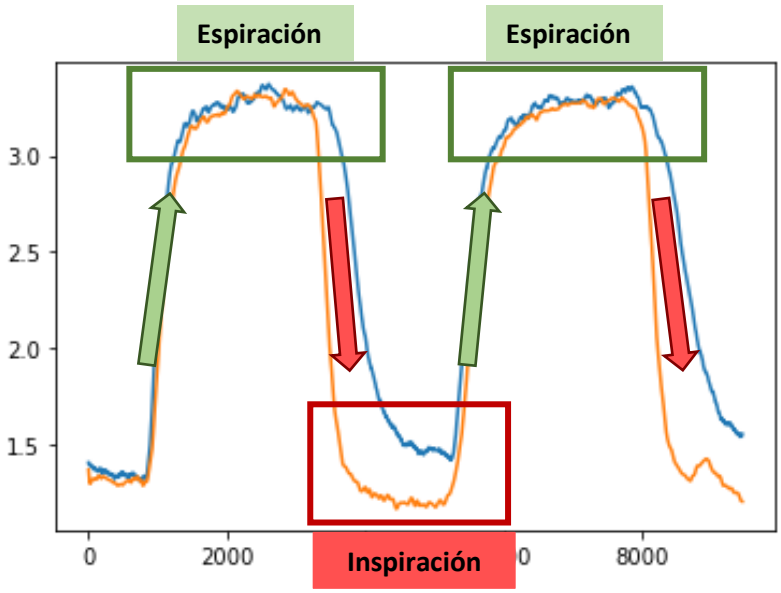


Ilustración 37: *Fases respiratorias. En color verde las espiraciones causan aumento de humedad. En color rojo las inspiraciones causan un descenso de humead.*

Dada una pareja de señales de humedad *Hum1* y *Hum2*, nuestro primer paso será determinar los máximos y los mínimos encontrados a lo largo de la señal, para así establecer la posición de cada fase respiratoria (*ilustración 38*). De esta manera, podremos segmentar la señal de forma selectiva para así extraer las unidades respiratorias que nos interesen.

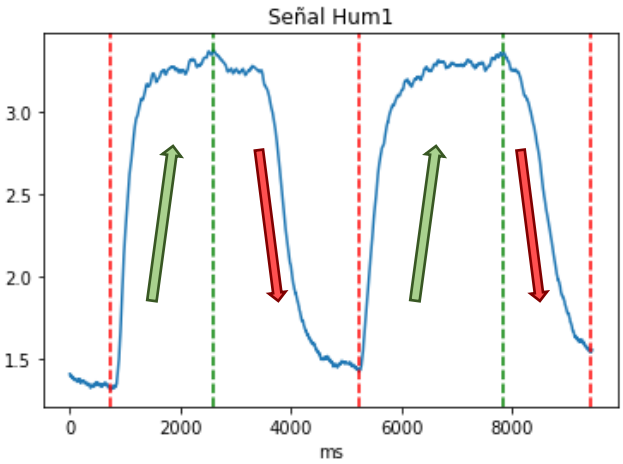


Ilustración 38: *Extremos de la señal de humedad 1 a través de las fases respiratorias. En color verde los máximos (espiración). En color rojo, los mínimos (inspiración).*

Debido a que disponemos de ambos sensores de humedad deberemos mantener una coherencia entre ellos para segmentarlos en los mismos puntos y que las longitudes de las unidades respiratorias sean idénticas. Con este fin, establecemos que los **máximos** serán determinados por el sensor **Hum1**, mientras que los **mínimos** los hará la señal **Hum2**, relacionando ambos sensores a la hora de crear los pares respiratorios.

Siguiendo este procedimiento obtendremos una serie de puntos de corte a partir de los cuales podemos obtener lo que denominamos las **unidades respiratorias**, las cuales se corresponden con los tramos de señal entre mínimos para espiraciones y los segmentos entre máximos para inspiraciones. En la *ilustración 39* se muestran todas las unidades extraíbles de una muestra dada, realizando los cortes que hemos explicado sobre ambas señales.

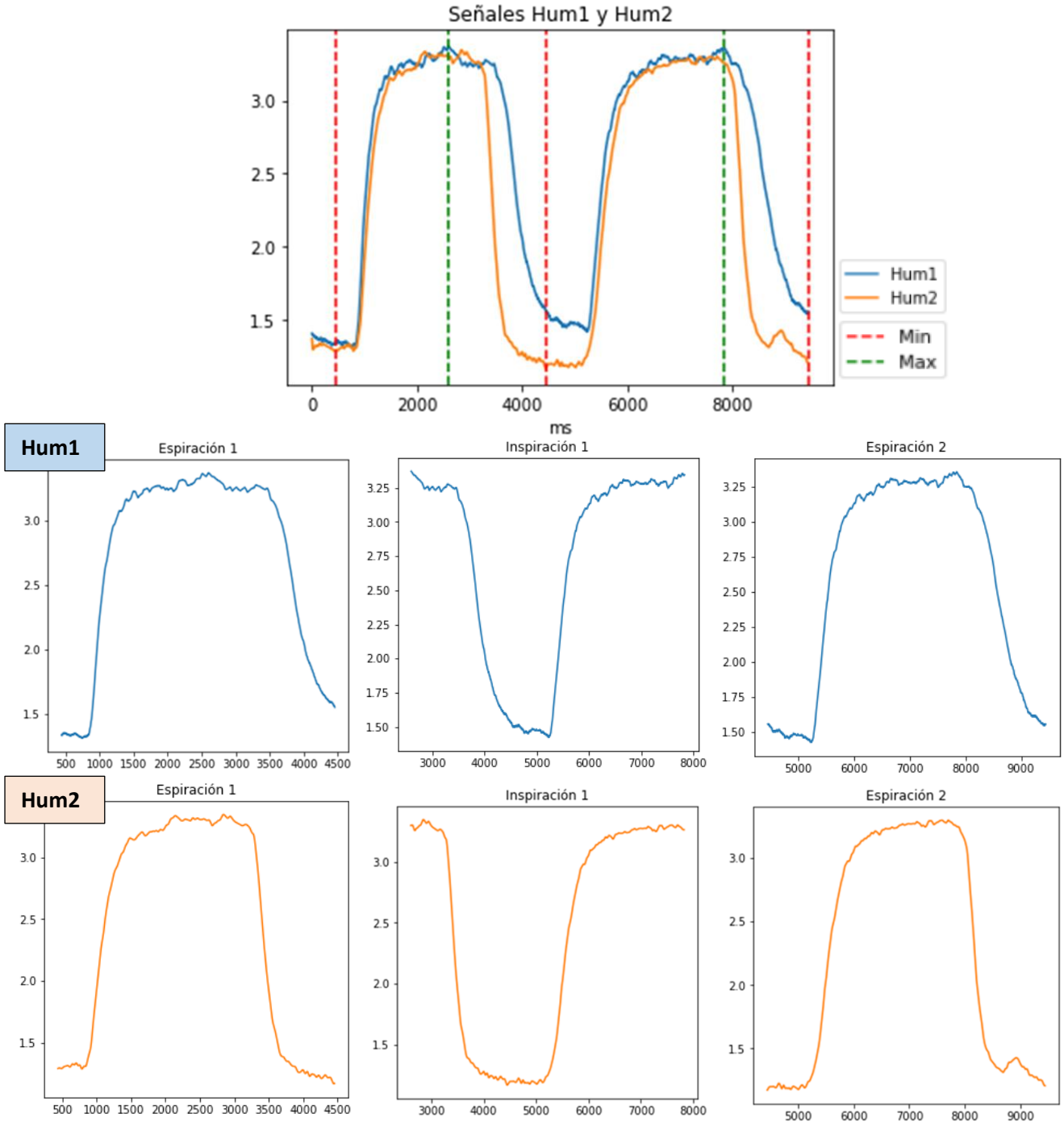


Ilustración 39: *Extracción de unidades respiratorias a partir de las señales de humedad completas.*
En la figura superior, las señales Hum1 y Hum2.
En la fila intermedia, las unidades extraídas del sensor Hum1.
En la fila inferior, las unidades extraídas del sensor Hum2.

Finalmente, una vez tengamos las unidades respiratorias procederemos a **unirlas** para generar los **pares respiratorios**. Dependiendo del orden del que se compongan tendremos dos tipos de pares (*ilustración 40*), aunque su utilidad para los entrenamientos será, por lo general, la misma.

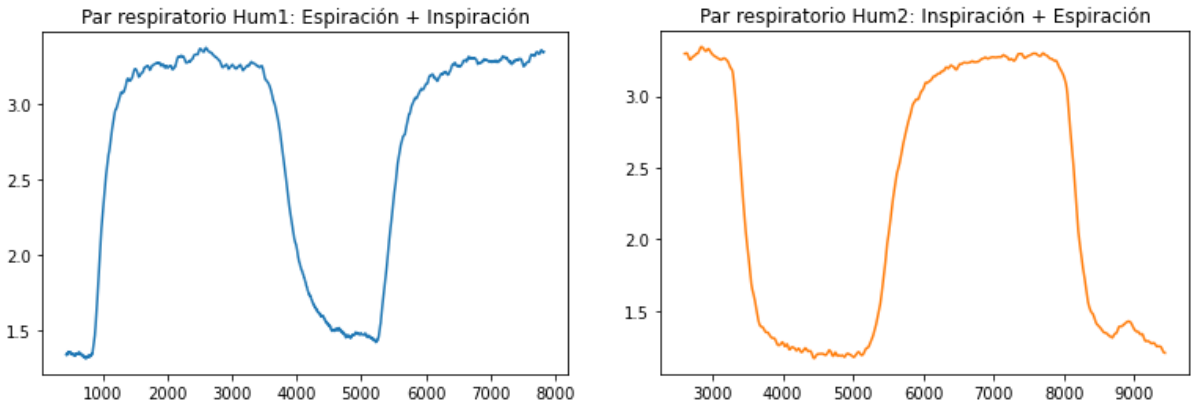


Ilustración 40: *Pares respiratorios obtenidos.*
A la izquierda con el sensor Hum1, la unión de espiración1 + inspiración1.
A la derecha con el sensor Hum2, la unión de inspiración1 + espiración2.

4. Modelos de aprendizaje

Terminada la fase de observación, estudio y preprocesado de las señales comienza la etapa de entrenamiento de los modelos de aprendizaje, la cual dividiremos en 3 secciones. En cada una de ellas describiremos un método para **extraer características** de los datos espirográficos, tras lo cual **entrenaremos los modelos correspondientes** con dichas características. Por último, se **evaluará** la eficacia del sistema empleado y se comentarán las métricas obtenidas.

En primer lugar, exploraremos cómo podemos de manera **manual** estudiar las señales para extraer atributos que puedan ser relevantes y de los cuales se pueda crear un aprendizaje. Segundo, haremos uso de una herramienta que **automatiza** el proceso de extracción de características para efectuar un entrenamiento similar al método anterior [4]. Finalmente, describiremos modelos de Deep Learning en los que la obtención de características resulta implícita y se realiza de forma **interna**, partiendo de las señales y de sus fotografías.

Para la evaluación de modelos se usará como principal referencia el valor que obtengamos con la **métrica R2**, ya que nos indica cómo se ha adaptado el modelo a los datos. En concreto, nos centraremos en el resultado en el **conjunto de *Test***, ya que este será el que nos indique realmente si el aprendizaje ha sido correctamente generalizado. Mirar a los resultados en *Train* nos mostrará si el aprendizaje ha sido efectivo con los datos de entrenamiento y el modelo ha logrado adaptarse a estas muestras, pero para comprobar si nuestra arquitectura es capaz de estimar correctamente las etiquetas de nuevos ejemplos, deberemos fijarnos en *Test*.

Por otro lado, en ocasiones veremos también la raíz del error cuadrático medio o el RMSE, ya que nos dará una idea de cuánto se equivoca en términos más absolutos e interpretables. También será buena idea durante el proyecto generar **gráficas de dispersión** para localizar *outliers* o ejemplos más problemáticos, como veremos más adelante.

Aunque el procesado de la mayoría de los datos o modelos ha resultado rápido, en algunos casos como la extracción automática o las redes convolucionales, el coste computacional ha sido muy elevado, lo que ha podido limitar parcialmente la experimentación. El almacenamiento de los datos, a la vez que el desarrollo y la ejecución del código implementado son realizados en una máquina con las siguientes especificaciones.

- **Procesador**: AMD Ryzen 5 3400G.
- **RAM**: Dos módulos de 16GB (32GB).
- **Tarjeta Gráfica**: Nvidia RTX 2070 (Super).

4.1 Extracción Manual de características

La respiración en reposo registrada nos ofrece una visualización interpretable de esta a partir de las señales de humedad, donde las subidas corresponden a espiraciones y las bajadas a inspiraciones. Teniendo esto en cuenta, parece sensato deducir que el comportamiento de estas señales variará en función del individuo, especialmente **entre pacientes sanos y patológicos**. Por este motivo, resulta interesante investigar maneras por las cuales nosotros podemos, de forma manual, extraer una serie de parámetros a partir de las señales con el objetivo de permitir a un modelo predictivo aprender de estos. A continuación, describiremos un método por el cual obtener datos que potencialmente faciliten una labor de entrenamiento mediante Machine Learning.

La extracción provendrá de las unidades respiratorias que anteriormente hemos descrito de las muestras filtradas, ya que este método se basará en capturar y almacenar una serie de puntos estratégicos de las fases respiratorias. A ser posible, estos marcadores reflejarán de forma adecuada aspectos como la amplitud de la señal, la velocidad de subida/bajada o el comportamiento de los pulmones estando tanto vacíos como llenos. En primer lugar, recogeremos una unidad respiratoria de la muestra (*ilustración 41*):

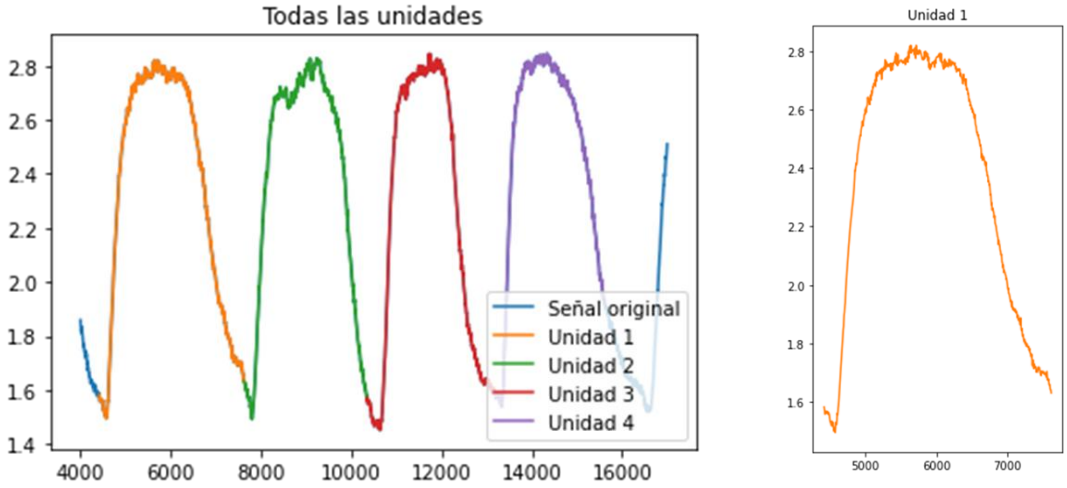


Ilustración 41: *Unidades provenientes de una muestra. A la izquierda, cada unidad es destacada por un color diferente. A la derecha, cogemos la unidad número 3.*

A continuación, recorreremos la unidad de comienzo a fin y almacenaremos **el valor y la posición temporal** de diversos marcadores estratégicos, que explicaremos centrándonos en la *ilustración 42*, correspondiente a una espiración. Estos puntos, determinados en función de la amplitud de la unidad, corresponden a los tramos que va alcanzando la señal a medida que sube o baja, de forma que buscaremos los puntos en donde la señal logre un 10, 30, 50, 70, 90 y 95% de la amplitud total de la unidad. Una vez llegados al máximo de esta, recogeremos los últimos dos marcadores, esta vez pertenecientes a la bajada; en concreto, los que se sitúen a un 95% y a un 90% de la amplitud de la señal, respectivamente. A estos últimos se les añadirá la terminación "*_R"* para ser diferenciados dentro del dataset.

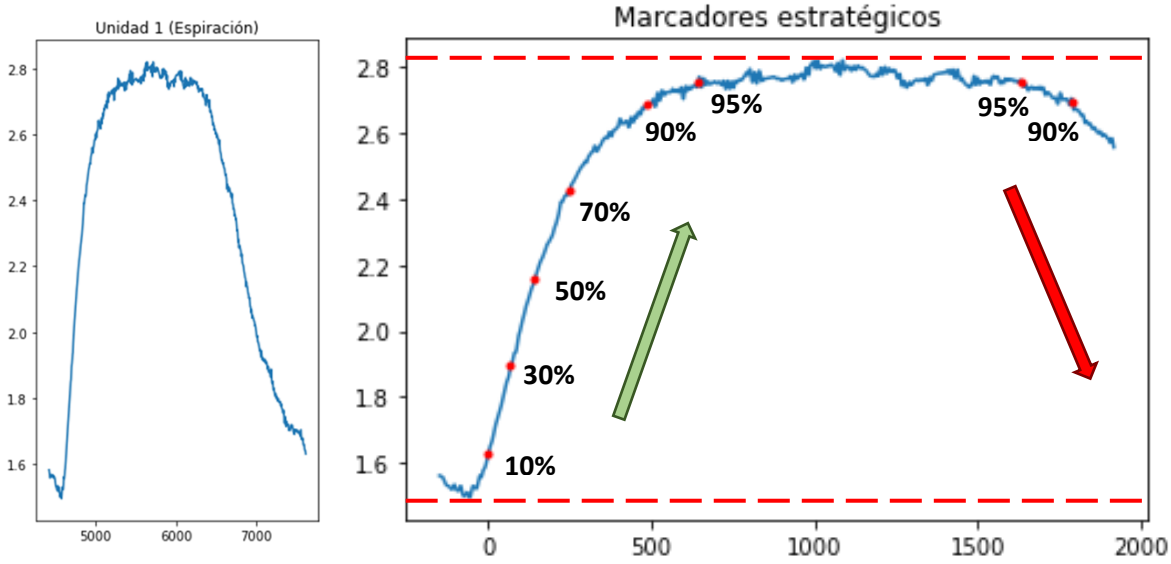


Ilustración 42: *Obtención de marcadores estratégicos a partir de una unidad respiratoria (espiración). A la izquierda, la unidad respiratoria. A la derecha, la obtención de puntos con los porcentajes de amplitud correspondientes.*

Naturalmente, este proceso será similar en el caso de que la unidad respiratoria corresponda con una inspiración, aunque el recorrido y los porcentajes deberán ser invertidos, como se indica en la *ilustración 43*. Por tanto, el máximo lo consideraremos el 0% mientras que el mínimo será el 100% de la amplitud.

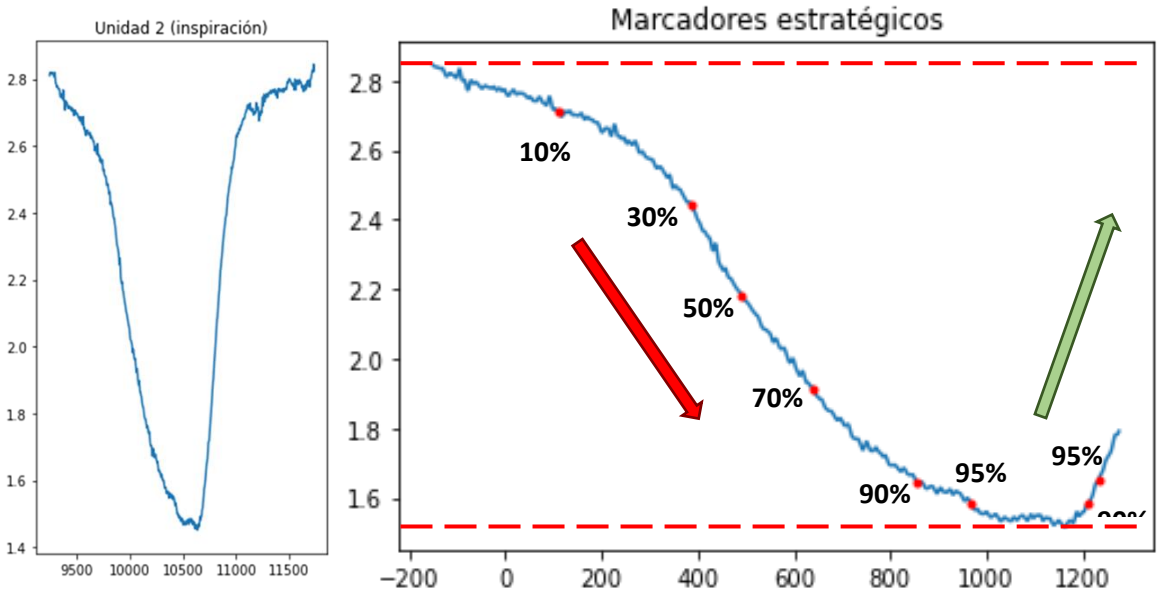


Ilustración 43: *Obtención de marcadores estratégicos a partir de una unidad respiratoria (inspiración). A la izquierda, la unidad respiratoria. A la derecha, la obtención de puntos con los porcentajes de amplitud correspondientes.*

Uniendo los puntos obtenidos tanto por las espiraciones como por las inspiraciones de cada muestra, conseguiremos un dataset de parámetros con el que entrenar los primeros modelos. En este caso concreto, crearemos cada fila de parámetros unificando los puntos obtenidos de cada pareja de fases respiratorias (espiración + inspiración), es decir, de cada par respiratorio. No obstante, es importante estudiar los valores que lo contienen y considerar normalizarlos, ya que facilitaría la labor de entrenamiento de los modelos. El aspecto de la tabla de datos que se utilizará tendrá el siguiente aspecto (*ilust. 44*), donde los campos siguen la siguiente denominación:

- **FT_XX%:** Punto temporal (ms) del marcador de amplitud XX% en la señal Hum1.
- **ST_XX%:** Punto temporal (ms) del marcador de amplitud XX% en la señal Hum2.
- **FV_XX%:** Valor de señal (V) en el marcador de amplitud XX% en la señal Hum1.
- **SV_XX%:** Valor de señal (V) en el marcador de amplitud XX% en la señal Hum2.
- **XX_95%_R** o **XX_90%_R:** Valores o puntos temporales de la señal correspondiente tras alcanzar su máximo o mínimo y proseguir con una bajada o subida respectivamente.

	FT_10%	FT_30%	FT_50%	FT_70%	FT_90%	FT_95%	FT_95%_R	FT_90%_R	ST_10%	ST_30%	...	FV_95%_R	FV_90%
100	204.0	308.0	407.0	505.0	769.0	856.0	1345.0	1379.0	3.0	71.0	...	1.623199	1.6994
100	3.0	71.0	130.0	221.0	460.0	564.0	1865.0	2187.0	80.0	178.0	...	2.993497	2.9124
100	80.0	178.0	245.0	355.0	571.0	675.0	1179.0	1196.0	3.0	65.0	...	1.589604	1.6675
102	46.0	112.0	180.0	281.0	680.0	1042.0	2417.0	2531.0	128.0	283.0	...	3.085340	2.9928
103	295.0	626.0	813.0	982.0	1304.0	1562.0	2491.0	2510.0	1.0	89.0	...	1.639727	1.7081
...	...	...	...	...	...	...	...	...	...	...	...	...	
274	0.0	1.0	2.0	151.0	153.0	165.0	166.0	167.0	27.0	177.0	...	0.995098	0.9948
274	0.0	1.0	2.0	153.0	174.0	175.0	175.0	176.0	27.0	177.0	...	0.932918	0.9324
274	1.0	77.0	148.0	245.0	437.0	581.0	1019.0	1020.0	27.0	177.0	...	2.797663	2.7983
274	216.0	262.0	631.0	990.0	1403.0	1404.0	1582.0	1583.0	27.0	177.0	...	2.889080	2.8856
274	27.0	177.0	320.0	658.0	1471.0	1792.0	2733.0	2756.0	0.0	1.0	...	1.030158	1.1299

849 rows × 32 columns

Ilustración 44: *Tabla de parámetros calculada de forma manual.*

4.1.1. Valor Medio (Modelo trivial)

Comenzaremos los entrenamientos por un sistema de predicción trivial, el cual nos aportará una referencia, ya que cualquier clasificador o regresor que utilicemos deberá obtener unas mejores métricas que este. El modelo consistirá en la **predicción como valor medio** de las etiquetas, es decir, predecir para todos los sujetos del dataset el valor medio de la variable de respuesta. Este método, naturalmente, no nos servirá de utilidad como propuesta de modelo predictor en sí, pero nos permitirá hacernos una idea de si un entrenamiento no ha ido como esperábamos o el modelo no se adapta a los datos correctamente. Las métricas para este modelo y, por tanto, de referencia son:

R2 Test	R2 Train	RMSE Test	RMSE Train
-0.085	-0.037	1.115	1.1191

Como era de esperar, un R^2 negativo nos confirma que el modelo no se ha adaptado a los datos, puesto que no ha habido un "aprendizaje" real. No obstante, un RMSE mayor que 1 nos muestra que el error no debería ser mayor a la unidad, algo que tiene sentido teniendo en cuenta la distribución de las etiquetas de las que disponemos, visibles en la *ilustración 45*.

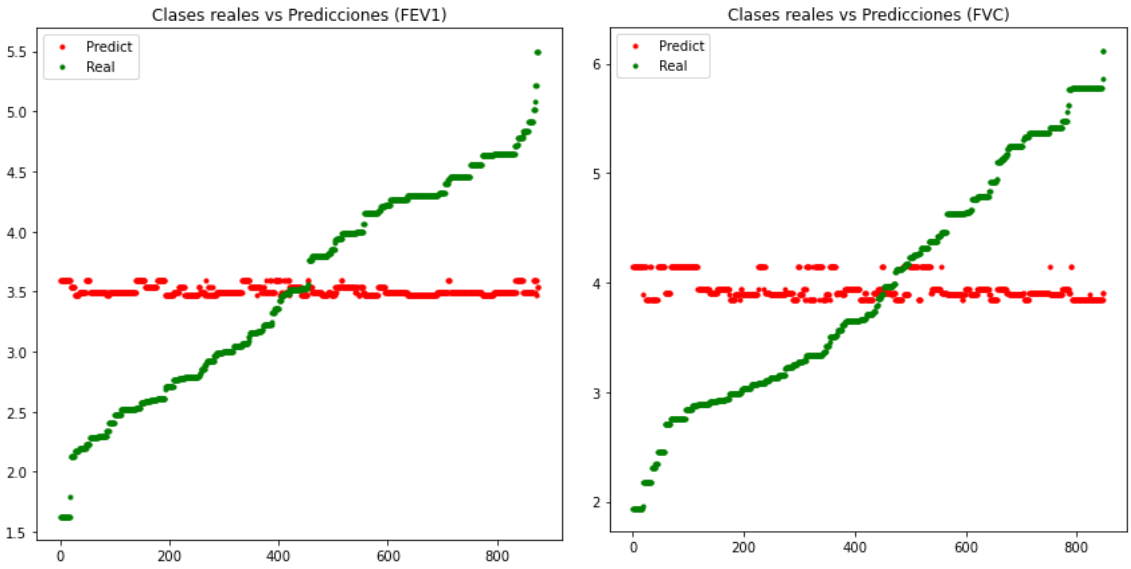


Ilustración 45: *Clases reales (verde) vs predicciones (rojo). A la izquierda, la distribución de la variable FEV1, a la derecha la de FVC.*

4.1.2. MLP Regressor

Uno de los modelos que nos aporta el módulo de scikit-learn para facilitar el estudio es el *Multi-Layer Perceptron Regressor*, una red neuronal ya preparada que utiliza el MSE como función de coste interna y es cómodamente configurable. Con ella podemos experimentar con un modelo más genérico que nos dé una idea más acertada sobre las posibilidades de otros modelos más complejos que aplicaremos más adelante. Las configuraciones que mejor nos han sabido extraer el conocimiento en esta etapa han sido, generalmente, los valores por defecto:

- **Capas ocultas:** Se construye la red con una capa oculta de 100 nodos/neuronas.
- **Función de activación:** Como es aconsejable, la función ReLU es la escogida.
- **Optimizador:** Para modular el aprendizaje durante el entrenamiento se disponen algunos algoritmos optimizadores, de los cuales utilizaremos 'Adam'.
- **Alpha:** En la regularización de tipo L2, que modifica el ratio de aprendizaje durante el entrenamiento, es posible definir la intensidad de este cambio. Este es el factor *alpha* que, por defecto, adquiere el valor $1e^{-4}$.
- **Planificador del Learning Rate:** Podemos establecer si el ratio de aprendizaje es constante o se ve alterado de forma regular. Mantendremos el ajuste por defecto a *'constant'* y dejaremos que el optimizador actúe en su lugar.

Si experimentamos con esta red nos percataremos de que la configuración óptima, cuyos resultados se exponen en la siguiente tabla, corresponde con un ratio de aprendizaje de 1e-4 y unas 1000 épocas. Con esto y viendo las *ilustraciones 46 y 47*, obtenemos en Test un **R^2 = 0.179** y **RMSE = 0.893** utilizando la variable de respuesta **FVC**, valores que nos muestran una mejor adaptación a los datos y una nueva referencia para el estudio posterior.

LR = 1e-4	MLPRegressor	**FVC**		
Num Epoch	**R2 Test**	**R2 Train**	**RMSE Test**	**RMSE Train**
500	0.111	0.318	0.929	0.814
1000	0.179	0.399	0.893	0.764
1500	0.154	0.402	0.907	0.762
2000	0.154	0.402	0.907	0.762

Ilustración 46: *Métricas obtenidas del MLP Regresor con la variable de respuesta FVC.*

LR = 1e-4	MLPRegressor	**FEV1**		
Num Epoch	**R2 Test**	**R2 Train**	**RMSE Test**	**RMSE Train**
500	0.148	0.343	0.78	0.685
1000	0.117	0.394	0.794	0.658
1500	0.117	0.394	0.794	0.658
2000	0.117	0.394	0.794	0.658

Ilustración 47: *Métricas obtenidas del MLP Regressor con la variable de respuesta FEV1.*

Además, para tener una mejor intuición de la comparativa entre las clases reales y sus predicciones, podemos trazar una gráfica de dispersión asociada a la métrica R^2 (*ilust. 48*). En ella representamos con un punto cada sujeto, cuyas coordenadas se corresponden con el valor de su predicción en el eje de abscisas y su etiqueta real en el de ordenadas. En un caso ideal en el que las predicciones sean idénticas a las clases reales obtendríamos una diagonal formada por todos los puntos de la gráfica. La distancia de un sujeto a esta línea imaginaria nos da una idea del error cometido al estimar la clase de tal paciente.

A continuación, podemos observar estas gráficas; primero con la clase FEV1 y después con FVC. Aunque las diferencias entre ambas variables de repuesta sean sutiles, las métricas obtenidas de este experimento nos podrían indicar que la variable más efectiva para extraer un aprendizaje es FVC.

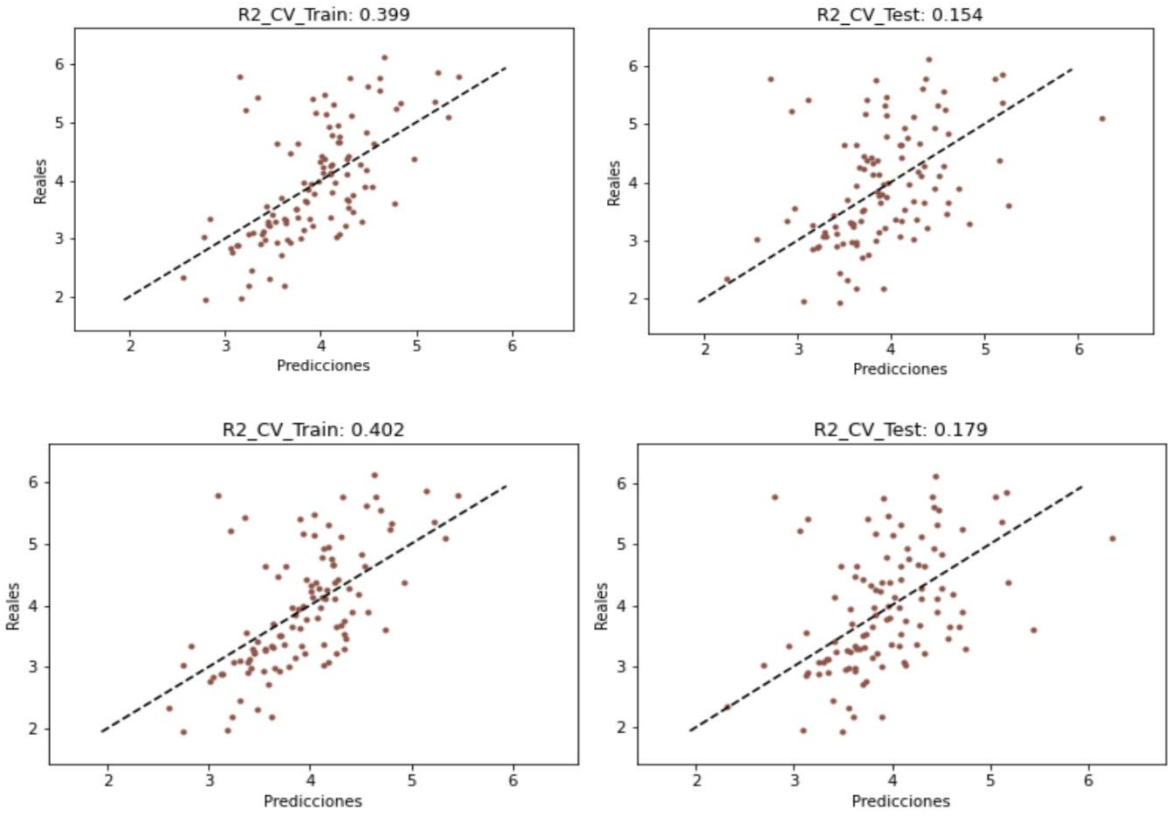


Ilustración 48: *Gráficas de dispersión del modelo MLP Regressor.*
En la fila superior, los resultados utilizando FEV1 (Train + Test).
En la fila inferior, los resultados utilizando FVC (Train + Test).

4.1.3. Árbol de Decisión

Una estructura muy utilizada en labores de clasificación por su efectividad y simpleza son los árboles de decisión. Estos modelos son construcciones formadas para tratar de predecir una variable de respuesta aprendiendo reglas sencillas a modo de toma de decisiones inferidas a partir de las características de los propios datos. Este proceso genera una estructura denominada árbol cuya raíz corresponde con la primera decisión, la cual es sucedida por múltiples ramas hasta alcanzar una hoja, que contiene la predicción final del modelo, proveniente de los datos de entrada.

Aunque esta estructura es más conocida por su aplicación en tareas exclusivas a la clasificación, resulta también útil como herramienta de regresión. En este caso, su funcionamiento es muy similar, salvo que la predicción que ofrece el árbol no consiste en una etiqueta discreta o booleana sino en un valor numérico. Utilizando los parámetros calculados y configurando el árbol de manera apropiada podemos obtener uno como el que se muestra en la *ilustración 49*, en el que cada nodo corresponde a una decisión en función del valor de una variable determinada en cada paso.

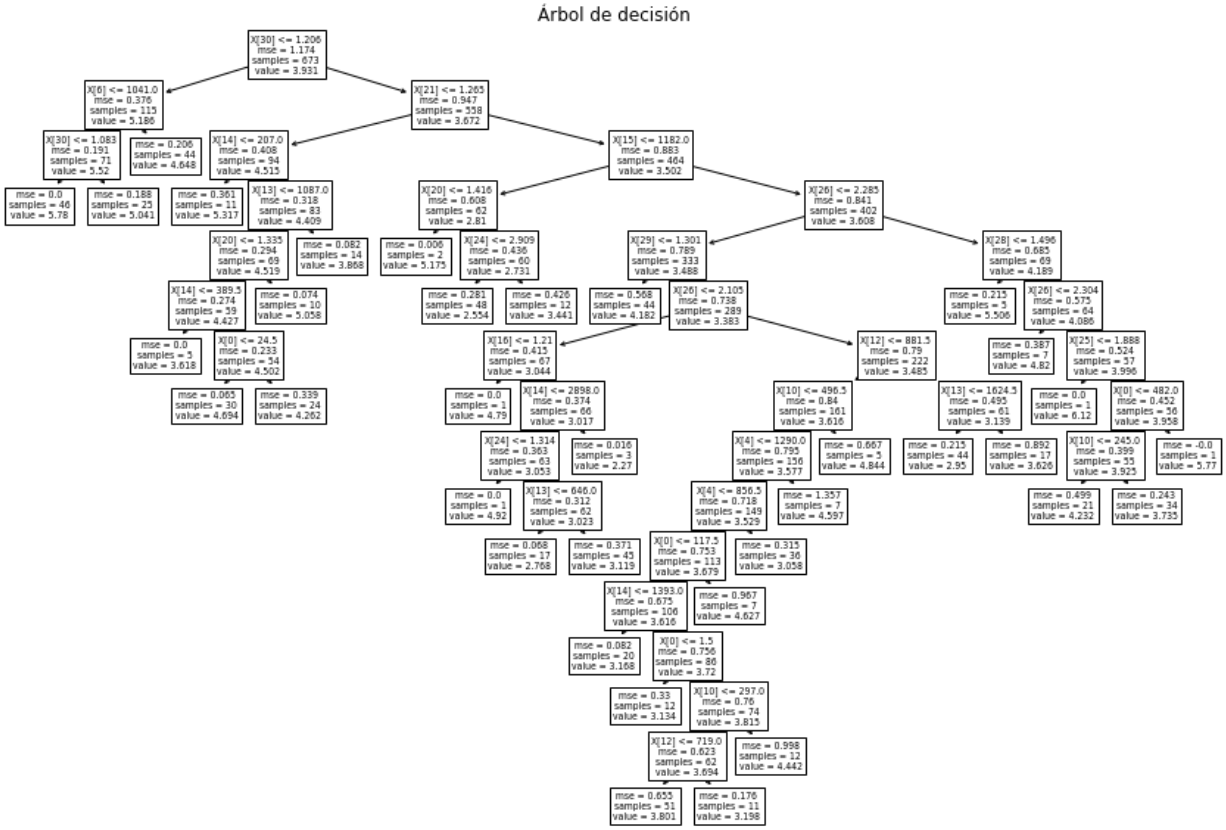


***Ilustración 49**: Árbol de decisión de profundidad 15. Se ha utilizado como función de coste el MSE.*

Mediante un árbol de decisión podemos calcular de nuevo las métricas y observar si esta aproximación nos permite conseguir un aprendizaje de las variables. En nuestro caso, el árbol más acertado se ha conseguido estableciendo una **profundidad máxima de 15 nodos** (*ilust. 49*) y un mínimo de **51 muestras** para decidir si dividir un nodo en dos, para así evitar una profundidad y dispersión demasiado grandes que puedan condicionar un sobre-entrenamiento. Esto queda visible en la *ilustración 50*, donde *max_depth* (MD) y *min_samples_split* (MSS) afectan a la eficacia del árbol, especialmente MSS pues limita la adaptación a los datos de entrenamiento y de manera indirecta también la profundidad del árbol.

MD	MSS	R2 Test	R2 Train	RMSE Test	RMSE Train
45	15	0.015	0.925	0.978	0.269
15	15	-0.037	0.919	1.044	0.28
45	51	0.099	0.647	0.936	0.586
15	51	0.1	0.646	0.935	0.586

Ilustración 50: *Métricas obtenidas con el árbol de decisión, donde MD es la profundidad máxima del árbol y MSS es el número mínimo de muestras por nodo para dividirlo en dos ramas inferiores.*

Con las métricas obtenidas podemos deducir que el árbol de decisión mediante regresión no es el modelo más adecuado para un problema como este. Esto puede deberse a que el número de variables de las que disponemos es amplio, con un total de **32 variables** calculadas por muestra, además de que la labor de predicción ya de por sí es compleja. Aún con esto, observamos un **R^2=0.1** y un **RMSE=0.935** en Test, lo que indica que por lo menos el árbol ha logrado adaptarse ligeramente a las características generales de los datos. Naturalmente, resultaría trivial crear un árbol con un ajuste perfecto a los datos, pero puesto que necesitamos que este sea capaz de estimar valores en un futuro que aún no conoce, esto no nos interesa. A continuación, en las *ilustraciones 51* podemos observar cómo queda el gráfico de dispersión en el mejor de los casos expuestos.

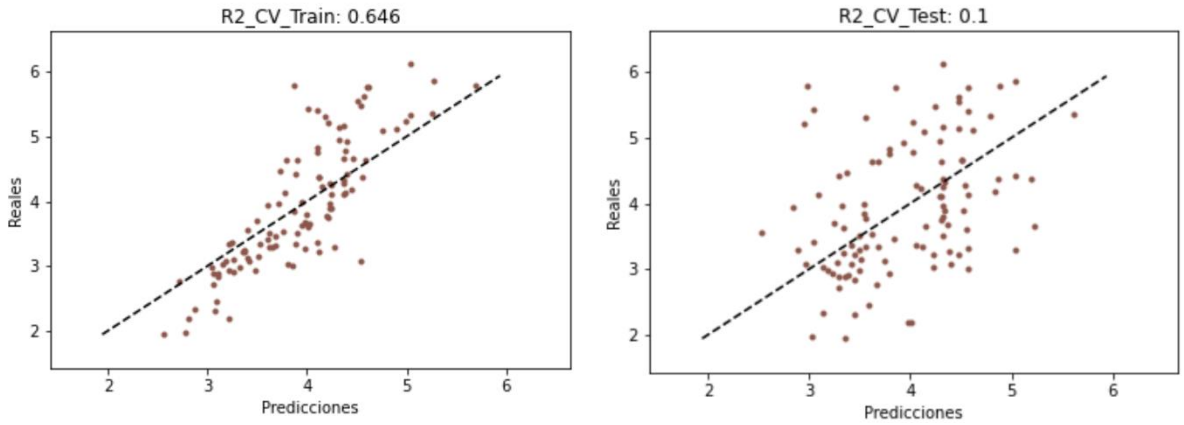


Ilustración 51: *Gráficas de dispersión con Árbol de Decisión de Train (izquierda) y Test (derecha).*

4.1.4. Support Vector Machine (SVM)

Otro algoritmo utilizado es el denominado *Máquina de Vectores de Soporte*, cuyo procedimiento consiste en representar todas las muestras de datos en el espacio y buscar los hiperplanos capaces de separar las muestras de distintas clases. De esta manera, cuando un nuevo dato entre al sistema, dependiendo de su proximidad a cada clúster de datos separados se le atribuirá una etiqueta. No obstante, puesto que en este caso tratamos con un problema de regresión utilizaremos la herramienta *Epsilon-Support-Vector Regressor* del paquete de scikit-learn para este experimento, el cual es internamente más complejo, pero nos permite predecir etiquetas numéricas a partir de los datos de entrada.

Como parámetro a configurar podemos utilizar la **épsilon** que da nombre al regresor, el cual se corresponde con el margen de error que toleramos de una muestra sin aplicar una penalización, dando más flexibilidad al entrenamiento y reduciendo el sobre-entrenamiento conforme lo aumentemos. No obstante, esto también puede ignorar parte del aprendizaje del modelo por lo que debe evaluarse correctamente.

SVM	FEV1			
epsilon	R2 Test	R2 Train	RMSE Test	RMSE Train
0	0.082	0.302	0.944	0.823
0.1	0.088	0.304	0.941	0.822
0.3	0.11	0.337	0.93	0.803
0.5	0.131	0.352	0.919	0.794
0.7	0.127	0.334	0.921	0.805
0.9	0.092	0.296	0.939	0.827
1	0.073	0.278	0.949	0.838

Ilustración 52: *Tabla de métricas del modelo SVR para distintos valores de épsilon (FEV1).*

SVM	FVC			
epsilon	R2 Test	R2 Train	RMSE Test	RMSE Train
0	0.126	0.344	0.79	0.684
0.1	0.128	0.348	0.789	0.682
0.3	0.155	0.378	0.777	0.666
0.5	0.164	0.381	0.772	0.665
0.7	0.125	0.341	0.79	0.686
0.9	0.087	0.284	0.807	0.715
1	0.076	0.239	0.812	0.737

Ilustración 53: *Tabla de métricas del modelo SVR para distintos valores de épsilon (FVC).*

Se puede observar en las *ilust. 52 y 53* que este valor de épsilon no parece favorecer el entrenamiento de forma muy notoria, aunque posiblemente sea conveniente mantenerlo alrededor de 0.5. Las métricas más destacables son un **R^2 = 0.164** y un **RMSE= 0.772** en Test conseguidas, de nuevo, con la variable de respuesta FVC, las cuales resultan ser similares a lo que hemos visto con el MLP Regressor. En las *ilustraciones 54* podemos estudiar las gráficas correspondientes al mejor modelo obtenido. Por el momento, no obstante, hemos adquirido varias referencias de qué valores de métricas debemos aspirar a superar, con lo que probaremos con otros modelos algo más configurables.

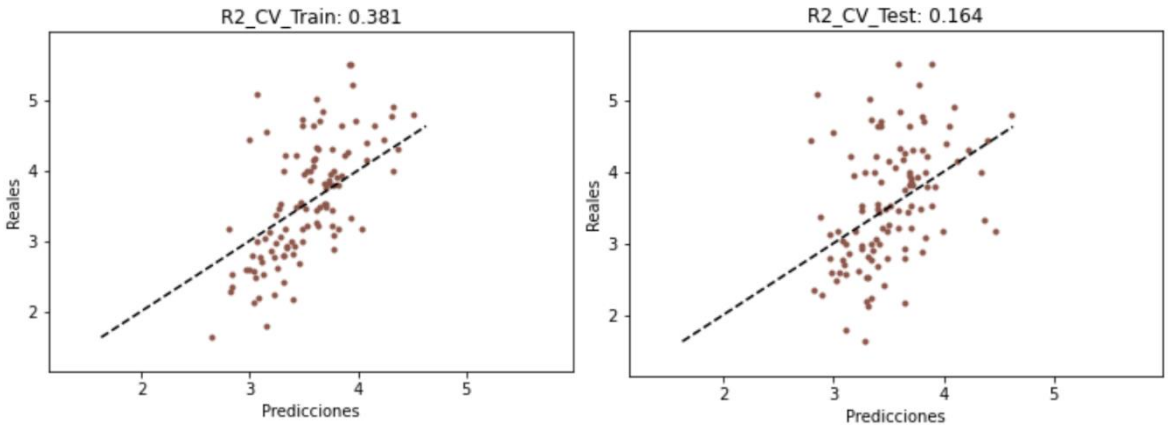


Ilustración 54: *Gráficas de dispersión con SVM (SVR) de Train (izquierda) y Test (derecha).*

4.1.5. Red Neuronal

Para tratar de mejorar las métricas que hemos obtenido de estos modelos crearemos una red neuronal poco profunda y observaremos su funcionamiento, buscando qué estructura y configuración podrían adaptarse mejor a las características extraídas para hacer estimaciones sobre las variables de respuesta que le indiquemos. Anteriormente, hemos utilizado una red neuronal denominada MLP Regressor de la librería scikit-learn. Aunque en ella hemos podido configurar aspectos como las capas ocultas o el optimizador a utilizar, necesitaremos **más flexibilidad** para experimentar con esta clase de arquitecturas, por lo que implementaremos una red neuronal artificial desde cero con la ayuda de la librería PyTorch.

De esta manera, aunque sea más costosa su construcción, configuración y mantenimiento, podremos componer nuestra red de forma modular, escogiendo qué características, herramientas o parámetros queremos utilizar para la experimentación. Además, mediante *PyTorch* podemos incorporar técnicas de regularización propias del aprendizaje profundo como pueden ser *batch norm* o *dropout*, lo cual aumenta nuestro repertorio de herramientas. Puesto que el objetivo es mantener la red como una estructura de poca profundidad, nos bastaremos con 3 capas + la capa de salida para tratar de encontrar las relaciones entre las características de entrada y que de estas nos calcule su predicción correspondiente.

La idea tras esta distribución de nodos consiste en permitir que la red pueda computar al comienzo todas las entradas de forma libre e ir comprimiendo la dimensión de los datos para que la red aprenda a combinar y relacionar los valores de entrada para conseguir la salida que necesitamos: el parámetro FEV1 o FVC que conocemos del paciente. A modo de esquema general, podemos observar la *ilust. 55* para visualizar cómo se realizaría la propagación de información a través de la red desde un comienzo hasta el valor predicho final. Como aquí se indica, la estructura que hemos considerado utilizar para este experimento tras barajar múltiples configuraciones será una *200x100x10*:

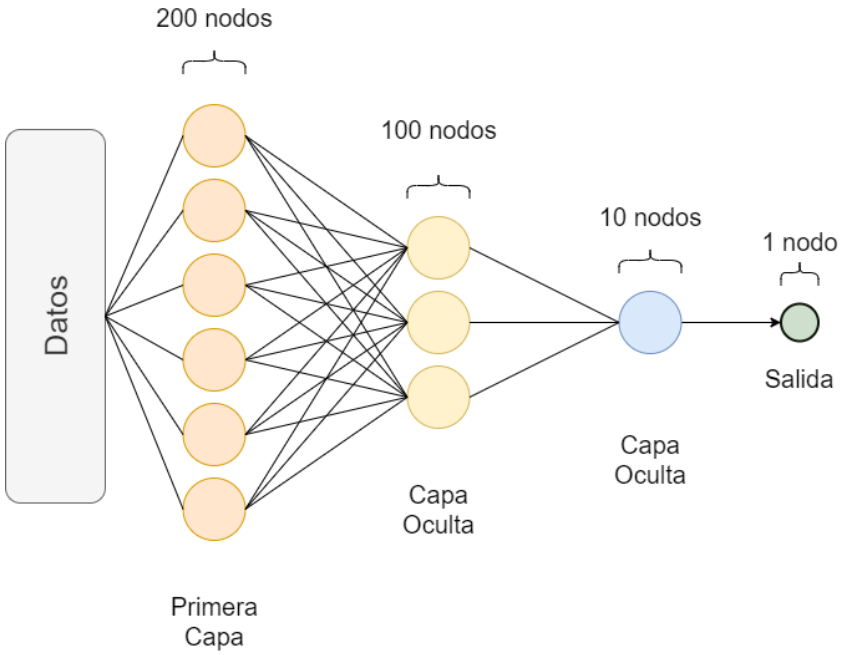


Ilustración 55: *Esquema ilustrativo de la red neuronal poco profunda 200x100x10.*

Configuración

Una vez definida una estructura, comenzaremos con la experimentación para lograr encontrar la mejor configuración para entrenar con las características manuales calculadas. La red puede ser modificada de múltiples maneras para optimizar su funcionamiento y adaptarla a cada situación específica. Aparte del ratio de aprendizaje que determina la fuerza con la que cambiaremos los pesos de la red cada iteración y del número de épocas de entrenamiento, las alteraciones más significativas serán:

- **Función de activación**: Durante la propagación de información de una red, cada nodo utiliza sus pesos internos para producir una salida a partir de los datos de entrada. Este *output* se define por la función de activación que se utilice. Las más habituales, dependiendo de la aplicación que se busque para la red, son la ReLU, la Sigmoide y la Tanh. En este caso, utilizaremos la función **ReLU** como activación entre capas, ya que nos aporta una respuesta lineal rectificada, la cual ha demostrado buenos resultados en estudios recientes.

- **Función de coste**: Para que el aprendizaje sea posible, necesitamos establecer cómo serán ponderados los fallos de predicción que cometa nuestra red, de forma que esta pueda cuantificar el error producido. El modelo, entonces, tendrá como objetivo minimizar o maximizar esta medida para así reducir el coste al menor posible y "aprender". Dependiendo de la arquitectura y del problema que se presente, unas *loss function* serán más adecuadas que otras, aunque resulta de interés destacar las siguientes: *Mean Absolute Error* (MAE), *Mean Squared Error* (MSE) y *Rooted Mean Squared Error* (RMSE). Para nuestra red utilizaremos la **función de coste MSE**.
- **Batch norm**: Aunque suele ser favorable que los datos sean normalizados antes de comenzar el entrenamiento, esta estandarización se realiza utilizando todos los datos disponibles y, debido a que es habitual entrenar los modelos por lotes, es lógico deducir que la normalización de las muestras de un lote ha sido influida por otras muestras que no comparten el mismo lote. Ya que esto podría no resultar favorable, se puede aplicar *batch_norm* o normalización por lotes al comienzo de cada capa para así sortear esta posible inconveniencia. Sin embargo, tras experimentar con esta configuración (*ilust. 56*), no se notan mejorías en el entrenamiento así que se decide retirar el *batch_norm* por el momento.

NN	lr = 1e-5	batch_norm = Si	FVC	
num epoch	R2 Test	R2 Train	RMSE Test	RMSE Train
1000	-6.725	-6.562	2.739	2.71
2000	-2.61	-2.385	1.873	1.812
3000	-0.637	-0.368	1.261	1.153
4000	-0.063	0.293	1.016	0.829
5000	-0.009	0.47	0.99	0.717
6000	-0.0067	0.608	1.018	0.617
7000	-0.099	0.782	1.033	0.46
8000	-0.127	0.917	1.046	0.284

Ilustración 56: *Experimentación de batch_norm. Se puede observar cómo el valor más alto es negativo, así que se considera no usar batch_norm en este caso.*

- **Dropout**: En un funcionamiento estándar de la red, cada nodo utiliza sus pesos internos para producir una salida, la cual es propagada a la siguiente capa de neuronas hasta alcanzar el final de la red. No obstante, podemos anular de manera aleatoria algunas de estas activaciones para evitar que el "aprendizaje" se efectúe de manera simultánea en todos los nodos, consiguiendo un aprendizaje más disperso, algo que suele ser beneficioso. Sin embargo, un dropout alto conlleva la necesidad de aumentar el número de iteraciones, algo que aumenta el coste computacional y requiere de una correcta calibración. De manera experimental (*ilust. 57*) se decide que el Dropout no aumenta la eficacia del modelo significativamente por lo que se opta por un valor muy bajo o de cero.

	NN	lr = 1e-5	FVC		
	dropout	R2 Test	R2 Train	RMSE Test	RMSE Train
epoch = 9000	0	0.138	0.417	0.915	0.753
	0.25	-0.34	0.242	1.141	0.858
epoch = 10000	0	0.134	0.454	0.917	0.728
	0.2	-0.123	0.273	1.044	0.841

Ilustración 57: *Experimentación de dropout. Aumentar el dropout.*

- **Regularizador**: Durante el entrenamiento, suele ser adecuado modular el ratio de aprendizaje para que resulte el más apropiado en cada iteración. Por lo general, resulta más eficaz comenzar con un ratio relativamente alto para acelerar las fases iniciales del entrenamiento, mientras que para las etapas finales suele ser lo correcto reducirlo para lograr una convergencia del algoritmo. Esta estrategia puede variar significativamente dependiendo de la arquitectura y del problema de clasificación o regresión específico. En nuestro caso, utilizaremos el **optimizador Adam** que trae el paquete PyTorch.

Resultados

Tras investigar y experimentar con estos componentes consideramos pues que para este problema de regresión la configuración más adecuada será la descrita anteriormente y, con ella, estudiaremos los resultados a distintas longitudes de entrenamiento y *learning_rates*. Tras lanzar entrenamientos de esta red neuronal podemos ver en las *ilustraciones 58 y 59* los resultados obtenidos. En la primera tabla se muestran las métricas resultantes a varios niveles de entrenamiento en función del número de épocas. A continuación, utilizando la mejor configuración encontrada exploramos varios ratios de aprendizaje.

NN	lr = 1e-5	FVC		
num epoch	R2 Test	R2 Train	RMSE Test	RMSE Train
5000	0.049	0.238	0.961	0.861
6000	0.09	0.288	0.94	0.831
7000	0.117	0.335	0.926	0.804
8000	0.132	0.377	0.918	0.778
9000	0.138	0.417	0.915	0.753
10000	0.134	0.454	0.917	0.728
11000	0.131	0.493	0.919	0.702
12000	0.118	0.537	0.926	0.671
13000	0.094	0.591	0.938	0.631
14000	0.062	0.65	0.955	0.583

Ilustración 58: *Tabla de resultados con Red Neuronal poco profunda, a distintos niveles de épocas. En verde los mejores resultados, en rojo los peores.*

NN	epoch = 9000	FVC		
learn rate	R2 Test	R2 Train	RMSE Test	RMSE Train
1E-06	-9.778	-0.9688	3.236	3.222
5E-06	0.056	0.24	0.957	0.859
1E-05	0.138	0.417	0.915	0.753
5E-05	-0.138	0.98	1.051	0.14
1E-04	-0.562	0.999	1.232	0.026

Ilustración 59: *Tabla de resultados con Red Neuronal poco profunda, a distintos ratios de aprendizaje. En verde los mejores resultados, en rojo los peores.*

Si tenemos en cuenta las métricas de referencia que obtuvimos en los apartados anteriores es evidente que este modelo **no ha logrado superar tales métricas**, puesto que los mejores resultados son un **R^2=0.138** y un **RMSE=0.915** en el conjunto de Test. Las gráficas de dispersión correspondientes a estos resultados se encuentran en las *ilustraciones 60*. Por este motivo, avanzaremos a otra sección donde utilizaremos un set de características distinto, basándonos en la posibilidad de que las características extraídas manualmente podrían resultar no ser tan eficaces como se esperaba en una primera instancia.

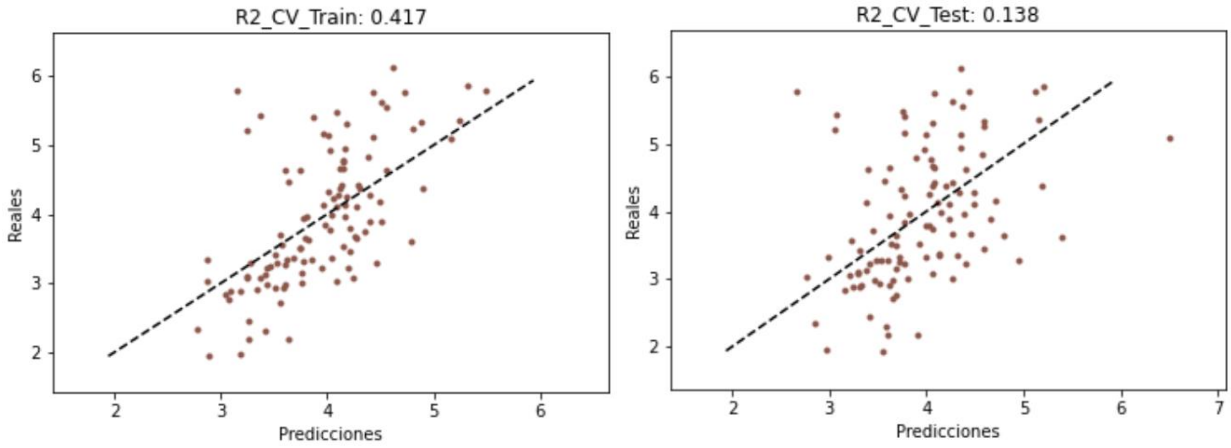


Ilustración 60: *Gráficas de dispersión con Shallow NN de Train (izquierda) y Test (derecha).*

4.2. Extracción Automática de características

En el apartado anterior no se ha logrado generar un aprendizaje lo suficientemente adecuado a partir de los parámetros disponibles. El cálculo manual de características, aunque altamente adaptado a las condiciones de los datos, presenta algunas desventajas. Por un lado, la lectura de un grupo de 32 puntos arbitrarios a través de cada unidad respiratoria podría no haber sido una buena elección al no ser capaz de capturar las características y patrones que faciliten un correcto entrenamiento de los modelos.

Además, dado el volumen relativamente pequeño de sujetos del que disponemos, el número total de datos sigue siendo algo reducido, aunque extraigamos alrededor de 64 valores por paciente. Por estos motivos, en este segundo intento utilizaremos una librería de extracción de características automática a partir de series temporales que, en este caso, serán nuestras señales espirográficas.

Dicho esto, una alternativa que tenemos es hacer uso de una herramienta de cálculo denominada ***tsfresh***, la cual es capaz de calcular un total de 763 características por cada señal que le introduzcamos (*ilust. 61*). Algunos de estos parámetros calculados son los máximos, mínimos, cantidad de picos, cantidad de valles, factores de correlación y otros muchos que podrían ayudar a explicar en su mayoría el comportamiento de las señales espirográficas, por lo que resultan de nuestro interés para experimentar y buscar nuevos métodos de entrenamiento a partir de nuestros datos.

	Características
0	Hum1__abs_energy
1	Hum1__absolute_sum_of_changes
2	Hum1__agg_autocorrelation__f_agg_"mean"__maxlag_40
3	Hum1__agg_autocorrelation__f_agg_"median"__maxlag_40
4	Hum1__agg_autocorrelation__f_agg_"var"__maxlag_40
...	...
758	Hum1__value_count__value_0
759	Hum1__value_count__value_1
760	Hum1__variance
761	Hum1__variance_larger_than_standard_deviation
762	Hum1__variation_coefficient

763 rows × 1 columns

Ilustración 61: *Tabla con algunas de las 763 características que calcula tsfresh. Cada una equivale a un valor numérico.*

Por otra parte, cabe destacar que el coste computacional de esta herramienta es muy elevado, pudiendo llevar más de 30h procesar el dataset completo, imposibilitando incluso el cálculo de las señales más extensas en nuestra máquina. Por este motivo, la generación del dataset fue también laboriosa y limitó nuestra capacidad de experimentación.

Para el entrenamiento utilizaremos una red neuronal de la misma profundidad que en la etapa de extracción manual y compararemos los resultados para comprobar si existe una mejoría al utilizar una gran cantidad de parámetros a partir de las señales, aunque estos no hayan sido calculados de forma estratégica y preconcebida.

Señales completas

Desafortunadamente, como muestran las siguientes *tablas 62 y 63*, el modelo no ha logrado aprender de los datos sin sobreentrenar el conjunto de datos de entrenamiento, por lo que no ha sido capaz de extraer el conocimiento apropiadamente. Además, si observamos las gráficas en *64*, queda evidente que la red se ajusta demasiado a *Train*, perdiendo la posibilidad de generalizar a *Test*.

R2 Test	NN	tsfresh	Train			
EPOCH	1E-06	5E-06	1E-05	5E-05	1E-04	LR
2000	-10.810	-6.546	-3.515	0.999	0.994	
4000	-9.181	-3.699	0.066	0.999	0.994	
6000	-8.162	-1.537	0.955	0.999	0.994	
8000	-7.350	0.024	0.994	0.999	0.994	
10000	-6.666	0.778	0.994	1.000	0.995	

Ilustración 62: *Tabla de métrica R^2 en Train.*

R2 Test	NN	tsfresh	Test			
EPOCH	1E-06	5E-06	1E-05	5E-05	1E-04	LR
2000	-12.160	-8.547	-5.990	-1.301	-1.140	
4000	-10.810	-6.129	-2.662	-1.295	-1.133	
6000	-9.980	-4.225	-1.661	-1.292	-1.104	
8000	-9.311	-2.687	-1.512	-1.274	-1.058	
10000	-8.748	-1.822	-1.510	-1.191	-1.024	

Ilustración 63: *Tabla de métrica R2 en Test.*

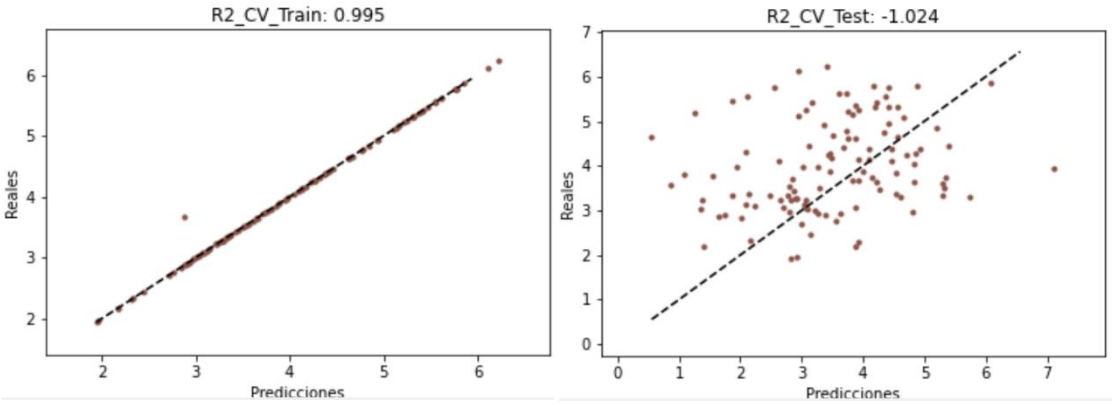


Ilustración 64: *Gráficas de dispersión del entrenamiento con tsfresh, Train a la izquierda y Test a la derecha.*

Este comportamiento en el modelo podría deberse a varios factores, uno de ellos siendo que los parámetros que son calculados de las señales no reflejen correctamente las características propias de cada sujeto. Por otro lado, los parámetros fueron calculados utilizando las señales al completo, algo que podría hacer que algunos sujetos adquieran unos parámetros diferentes debido a la disparidad en longitudes de las señales. Para buscar una mejor aplicación de la herramienta de tsfresh experimentaremos con otra alternativa que utilice fragmentos más cortos de la señal para calcular las características.

Recortes aleatorios

Otro método por el cuál podemos generar tramos de señal es mediante la creación los **recortes aleatorios** y el cálculo de las características con ellos. Este proceso consiste en generar 8 o 16 segmentos de cada muestra de una longitud fija en puntos aleatorios. En la *ilustración 65* obtenemos un resumen de los resultados más significativos a modo de intuición de la eficacia de este intento. En primer lugar, se estudia utilizar 8 segmentos extraídos de las señales Hum1, y tras ello se extraen 16 segmentos pero de la diferencia de ambas señales. Este último dataset se crea con la intención de utilizar la información proveniente de ambas señales, mezcladas en una sola secuencia de datos. Como se observa, aunque las métricas resultan más exitosas, seguimos sin lograr superar un R^2 significativamente superior a 0.3, por lo que probaremos con más alternativas.

TSFRESH	lr = 1e-5			
Dataset	**Segmentos**	**Longitud**	**R2 Test**	**R2 Train**
Hum1	8	4000	0.254	0.921
Hum1	8	6000	0.035	0.922
Hum1	8	8000	0.22	0.9
Hum1 - Hum2	16	4000	0.305	0.965
Hum1 - Hum2	16	6000	0.236	0.928
Hum1 - Hum2	16	8000	0.022	0.959

Ilustración 65: *Tabla resumen con resultados de los recortes aleatorios.*

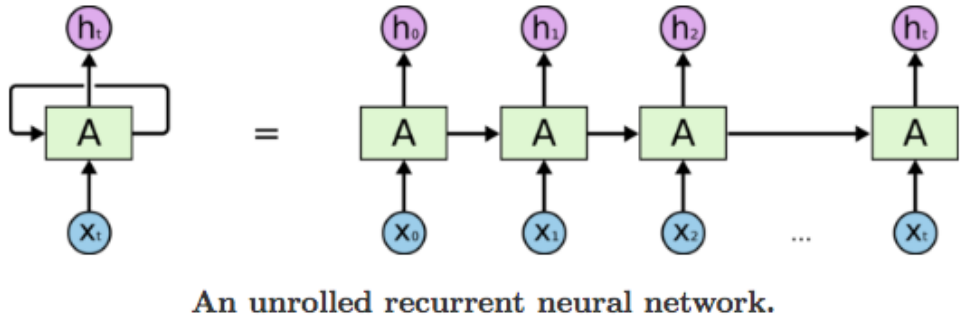
4.3. Modelos Deep Learning

Por último, existe otra clase de modelos pertenecientes al campo del aprendizaje profundo o *Deep Learning*. Estas arquitecturas, generalmente más complejas y compuestas por un mayor número de componentes, pueden tener la capacidad de extraer por sí mismas las características más relevantes de unos datos de entrada. Esencialmente, en ellos la búsqueda de patrones y relaciones entre los datos para extraer unas características válidas es una fase del entrenamiento y es previa al aprendizaje que realizan sobre dichas características. Por tanto, no solo adquieren conocimiento sobre los datos de entrada, sino que aprenden a extraer de ellos las características más adecuadas para lograr el mejor resultado.

Por lo tanto, en este apartado no habrá un cálculo de características previo, sino más bien una disposición o preparación de las muestras adaptada al modelo que vayamos a entrenar. En este caso hablaremos en primer lugar de las redes neuronales recurrentes (**RNN**) y cómo trabajan con las señales completas o recortadas de forma directa. Después, describiremos brevemente las redes neuronales convolucionales (**CNN**) y cómo podemos entrenarlas con las muestras disponibles.

4.3.1. Long Short-Term Memory (LSTM)

Las *Recurrent Neural Networks* o **RNN** son una variante de las redes neuronales convencionales especializada en el procesado de información secuencial, en las cuales existe una "conservación" de información entre las distintas muestras de datos consecutivas. En resumidas cuentas, las RNN son capaces de aprender de los datos, no como información puntual e independiente, sino como parte de una **secuencia de valores** con un contexto. Para lograrlo, esta clase de redes no solo procesa el input que obtiene en cada iteración, sino que el output que genera en ella lo utiliza para el procesado de la próxima muestra (*ilust. 66*), consiguiendo retener información entre una muestra y la siguiente. Con esto, las RNN consiguen "memorizar" patrones de comportamiento en las secuencias y adquirir un conocimiento más generalizado de ellas.



An unrolled recurrent neural network.

Ilustración 66: *Esquema ilustrativo de una RNN. Fuente.*

Es interesante destacar que, por lo general, las redes recurrentes están preparadas para tratar con dimensiones de datos variables, ya que estas podrían no saber de antemano la longitud de los datos de entrada. Como consecuencia, se puede observar en la *ilustración 66* cómo en cada iteración del algoritmo se genera una salida h_i a partir de cada dato de entrada x_i. También es importante recalcar que en nuestra implementación nos es posible **entrenar con múltiples señales** al mismo tiempo si nos interesa, por lo que podremos utilizar simultáneamente señales de temperatura o presión además de las de humedad.

Las arquitecturas más conocidas de esta clase de redes son las ***Long Short-Term Memory*** (**LSTM**) y las *Gated Recurrent Unit* (GRU), y en este proyecto se ha implementado la primera de ellas. Naturalmente, puesto que el dataset del proyecto consta de señales espirográficas, este modelo resulta el más intuitivo, ya que está especialmente construido para esta clase de datos secuenciales. Con esto, nuestro siguiente experimento será comprobar la eficacia de esta clase de arquitecturas para resolver nuestro problema de regresión.

Configuración

En primer lugar, resulta conveniente estudiar qué clase de configuraciones podemos aplicar a estas redes para hacerlas más eficaces. Los dos aspectos más novedosos que pueden alterarse son:

- **Output (Módulos de Salida):** Una red recurrente está compuesta por estos módulos especiales que hemos comentado anteriormente, siendo en este caso los LSTM. El número de estos módulos puede variar, ya que uno solo de ellos no necesariamente puede ser capaz de extraer las características y aprendizaje de una señal, por lo que utilizar múltiples de ellos resultará beneficioso. En este caso, aunque nuestro valor de salida sea un único número, se encuentra que al utilizar **entre 70 y 90 módulos** el aprendizaje nos consigue unas métricas superiores, por lo que se utilizarán valores entre estos mencionados.
- **Bidirectional:** Este parámetro consta de un valor booleano con el que decidimos si queremos que los módulos LSTM de nuestra red permitan el flujo de datos en ambas direcciones; es decir, que la secuencia de información sea leída en ambos sentidos. Esto puede resultarnos beneficioso ya que de estas señales espirográficas se podría extraer más información si permitimos que la red las procese en las dos direcciones, por lo que en nuestra configuración este parámetro será ***True***.

En segundo lugar, necesitaremos preparar el dataset del cual leerá la RNN para el entrenamiento. Este grupo de datos, al contrario que en los casos anteriores, se deberá componer por señales, es decir, las secuencias de valores que tenemos de los sensores. Por tanto, para conseguir implementarlo, deberemos integrar en un mismo DataFrame todas las señales que podamos y lanzar los entrenamientos con él. Sin embargo, tenemos tres opciones para entrenar este modelo, puesto que disponemos de las señales en tres formatos. Por una parte, conservamos las señales al completo de los pacientes y, por otra, podríamos utilizar el dataset de recortes aleatorios y pares respiratorios que anteriormente se han explicado en esta memoria.

Señales sin recortes

La opción más sencilla para comenzar con las RNN sería utilizar las señales que tenemos al completo: humedad, temperatura y presión. Podemos considerar que, puesto que en este dataset podríamos incluir el mayor número de información posible, el modelo tendrá la posibilidad de aprender de un gran número de datos. No obstante, la duración de las muestras el altamente variable y en este caso tenemos una limitación: las muestras que introduzcamos en el módulo de LSTM deben tener la misma longitud.

Explorando los tamaños de las señales podemos representar sus longitudes como se observa en la *ilustración 67*, y vemos su distribución. Puesto que queremos incluir en nuestro entrenamiento el mayor número de ejemplos posible, pero sin acortar innecesariamente las señales, establecemos una longitud fija de **7000 ms**, la cual considero que es suficientemente larga como para contener información relevante sobre el paciente, sin excluir demasiados sujetos del dataset. Por tanto, las muestras que hayan durado menos de 7 segundos no se podrán utilizar en este experimento, y de las que superen este umbral se extraerán los primeros 7000 ms.

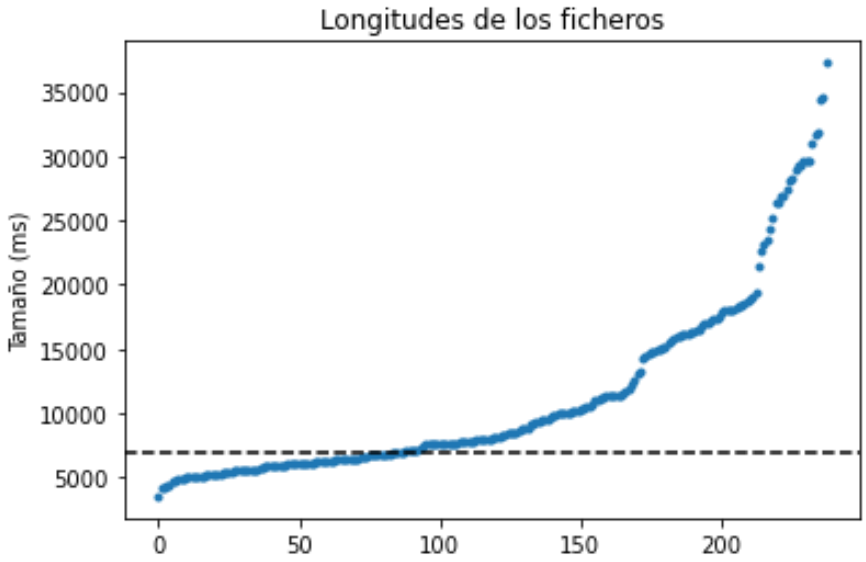


Ilustración 67: *Longitudes de las muestra ordenadas.*

Una vez establecido este primer dataset, realizamos un estudio para comprobar la efectividad del modelo con estos datos, con el que obtenemos la siguiente tabla (*68*), en la que las métricas, por desgracia, nos brindan unos malos resultados. Para alcanzar un modelo más adecuado, crearemos otros dataset y experimentaremos con la configuración de la red.

R2 Test	all_signals	FVC	OUTPUT 75				
NUM_EPOCH	1E-08	5E-08	1E-07	5E-07	1E-06	5E-06	LR
10	-22.530	-18.700	-14.650	-1.040	-0.406	-0.235	
20	-33.490	-23.280	-13.620	-0.465	-0.239	-0.218	
30	-57.810	-40.500	-24.670	-0.226	-0.231	-0.306	
50	-30.270	-12.850	-2.522	-0.238	-0.203	-0.347	
75	-18.830	-3.208	-0.717	-0.184	-0.186	-0.211	
100	-23.670	-3.318	-0.619	-0.292	-0.314	-0.537	
150	-45.980	-2.417	-0.215	-0.147	-0.159	-0.382	

Ilustración 68: *Resultados utilizando las señales completas (acotar a 7000 ms).*

Señales recortadas aleatoriamente

Como se ha explicado con anterioridad, una alternativa al uso de las señales de manera directa es el recorte de las mismas en puntos aleatorios. Quedan pues visibles en la *ilustración 69* las métricas obtenidas extrayendo 16 segmentos de cada paciente utilizando, en primer lugar, únicamente la señal de humedad 1 y, posteriormente, ambas Hum1 y Hum2. En este método se ha logrado uno de los mejores equilibrios, obteniendo una métrica **R^2 de 0.363** en Test. Sin embargo, aunque el resultado en este caso específico ha mejorado, todavía nos encontramos lejos de encontrar unas métricas aceptables.

LSTM	lr = 1e-5			
Dataset	**Segmentos**	**Longitud**	**R2 Test**	**R2 Train**
Hum1	16	4000	0.145	0.402
Hum1	16	6000	0.164	0.457
Hum1	16	8000	0.109	0.46
Hum1 & Hum2	16	4000	0.24	0.46
Hum1 & Hum2	16	6000	0.363	0.643
Hum1 & Hum2	16	8000	0.228	0.912

Ilustración 69: *Tabla resumen con resultados de los recortes aleatorios.*

Pares respiratorios

El siguiente paso lógico sería aprovechar nuestro dataset de pares respiratorios para el modelo LSTM, ya que resulta de interés comprobar las posibilidades de la RNN con segmentos definidos de forma estratégica. Durante la generación de pares respiratorios obtuvimos dos clases de señales según el orden de las unidades respiratorias que lo componían en función de si comenzaban por una espiración (*Esp_Insp*) o inspiración (*Insp_Esp*). Por ello, dividiremos este experimento en dos fases, en las que probaremos estos datasets por separado.

Antes de eso, sin embargo, es necesario recalcar que, de la misma manera que con el dataset de señales completas, las longitudes de los pares pueden variar significativamente. Para solventar esto he optado por recortar las señales a una longitud que me permita conservar gran parte del dataset sin acortarlas demasiado. Esta longitud la he establecido como el percentil 10, que se sitúa aproximadamente en los 4000ms.

R2 Test	all_pairs (Esp_Insp)	FVC	OUTPUT 85	Percentil 10			
Experimentos	1E-08	5E-08	1E-07	5E-07	1E-06	5E-06	LR
10	-29.860	-28.610	-27.080	-16.690	-7.984	0.189	
20	-13.970	-12.130	-10.040	-1.313	-0.018	0.214	
30	-22.930	-19.350	-15.290	-0.183	0.123	0.279	
50	-24.180	-18.200	-12.050	0.147	0.184	0.246	
75	-14.890	-8.843	-3.742	0.169	0.206	0.239	
100	-12.980	-6.512	-1.517	0.164	0.205	0.192	
150	-13.110	-3.726	0.022	0.210	0.235	0.228	

Ilustración 70: *Tabla de resultados para el dataset de pares respiratorios Esp_Insp.*

R2 Test	all_pairs (Insp_Esp)	FVC	OUTPUT 85	Percentil 10			
Experimentos	1E-08	5E-08	1E-07	5E-07	1E-06	5E-06	LR
10	-26.350	-24.870	-23.090	-11.330	-2.647	0.109	
20	-13.860	-12.240	-10.380	-1.532	0.022	0.152	
30	-22.670	-18.880	-14.680	-0.617	-0.012	0.162	
50	-31.750	-25.270	-18.280	-0.054	0.080	0.143	
75	-15.170	-8.845	-3.463	0.094	0.139	0.184	
100	-20.930	-11.300	-3.540	0.054	0.109	0.190	
150	-14.970	-5.248	-0.625	0.080	0.088	0.151	

Ilustración 71: *Tabla de resultados para el dataset de pares respiratorios Insp_Esp.*

Observando las *ilustraciones 70 y 71*, en las que se aprecia el R^2 a diversos niveles de entrenamiento, comprobamos que las nuevas métricas no superan un $R^2 = 0.3$ en Test, teniendo un máximo de 0.279. Sí que es visible, sin embargo, que el dataset compuesto por pares que comienzan por una espiración ha resultado algo más apropiado, algo interesante teniendo en cuenta la similitud de ambos conjuntos de señales.

En este apartado hemos estudiado el uso de las señales espirográficas para alimentar directamente redes recurrentes para que realicen la extracción de características pertinente y aprendan de las señales lo que sea posible. En el próximo caso, al contrario que en el estudio previo, no utilizaremos las muestras de forma directa, sino que de cada una de ellas generaremos **una imagen o fotografía**.

4.3.2. Convolutional Neural Networks (CNN)

Las redes neuronales convolucionales o CNN conforman uno de los algoritmos del campo del Deep Learning con más relevancia y éxito en la actualidad. Partiendo de imágenes, las CNN son capaces de aprender de forma eficaz a reconocer formas, objetos y morfologías complejas y extraer las características que considere más representativas de las imágenes para lograr un mejor aprendizaje. Estas características son posteriormente utilizadas para un entrenamiento, similar al de las redes neuronales artificiales convencionales, que consigue calcular soluciones a problemas tanto de clasificación como regresión, partiendo únicamente de las fotografías que se le suministra y de las etiquetas o metadatos correspondientes a ellas. Para su implementación utilizamos la librería **FastAI**, la cual nos facilita su construcción, uso y evaluación con una alta interactividad y configuración a alto nivel.

El procesamiento en las CNN sucede por capas que progresivamente transforman las imágenes tanto en su dimensionalidad como contenido, comprimiéndolas paso a paso hasta obtener un vector de características como se indica en la *ilustración 72*. Estas computaciones sobre la imagen generalmente son realizadas por las **capas convolucionales**, que dan nombre a la red, y otras capas como las *pooling*, las cuales reducen el tamaño de la imagen agregando los valores que la componen. Una vez extraídas las características de la foto, alcanzamos las capas finales, normalmente denominadas *Fully Connected* (FC) puesto que todos los nodos de una capa se encuentran conectados con los de la siguiente. Finalmente, tras todo este procesamiento y dependiendo de cómo hayamos configurado nuestra red, obtendremos el valor de salida.

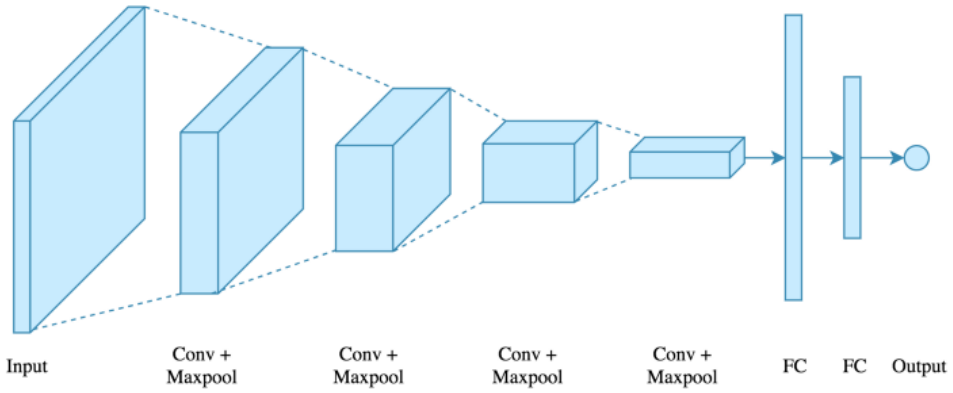


Ilustración 72: *Esquema de una Red Neuronal Convolucional. Fuente.*

Para poder aplicar esta arquitectura a nuestras señales espirográficas, naturalmente, necesitaremos adquirir las imágenes a partir de ellas. Como se ha comentado en diversos apartados, debido a la variabilidad en la longitud de las señales, resulta conveniente establecer una unidad de información de las señales espirográficas, por lo que usaremos los pares respiratorios, que nos ofrecen múltiples ventajas.

Por un lado, puesto que son tramos de señales escogidos y recortados de forma estratégica, consideramos que la información que estos contienen resulta de relevancia y facilitará el aprendizaje de nuestro modelo. Por otra parte, al establecer una unidad **independiente de la longitud original** de las señales del sujeto, seremos capaces de **mantener una coherencia entre fotos**, de forma que la CNN será capaz de comparar de forma eficaz señales de distinta amplitud o longitud.

Puesto que disponemos de dos señales de humedad (Hum1 y Hum2), nos vemos en la necesidad de generar 3 datasets diferentes: uno para ambos sensores y un tercero que represente ambos al mismo tiempo. De esta forma, como se muestra en la *ilust. 73*, por cada toma de datos de un sujeto obtendremos 3 fotografías:

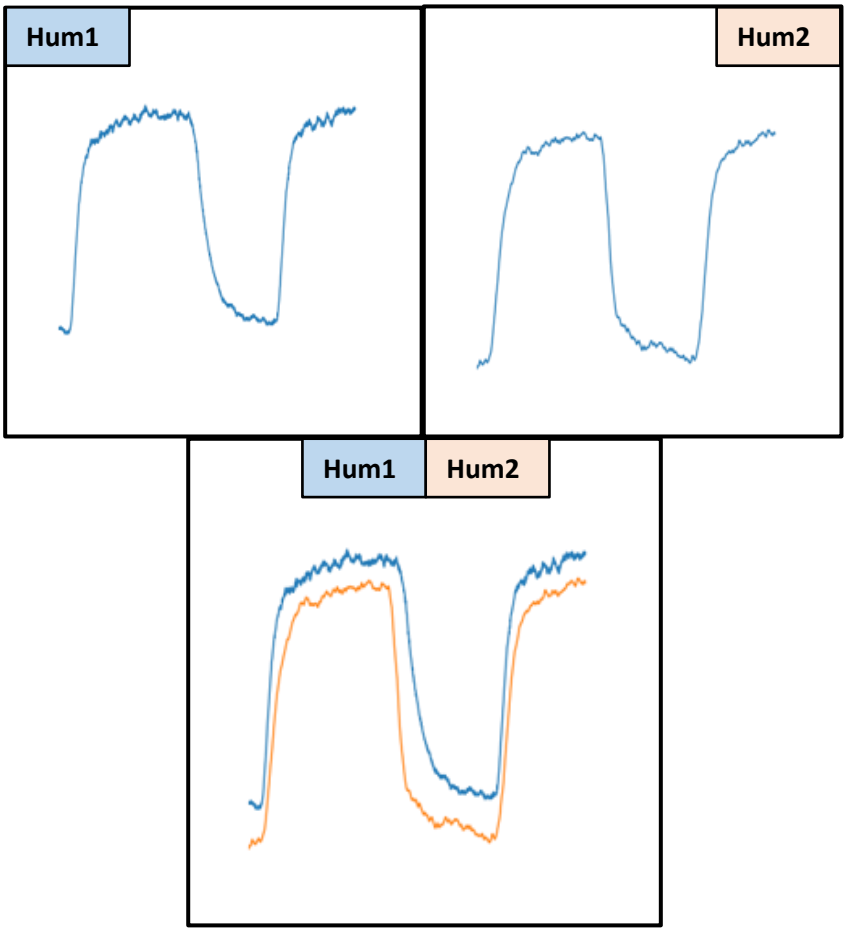


Ilustración 73: *Imágenes del mismo par respiratorio en los 3 modos.*
Arriba, el sensor Hum1 y el Hum2. Abajo, la unión de ambos en una misma fotografía.

Configuración

Para este proyecto se ha utilizado una **ResNet34** (*ilust. 74*), una arquitectura ya construida y pre-entrenada en problemas de reconocimiento de objetos en imágenes que ha demostrado unos buenos resultados en este campo [6]. Debido a que esta red ha sido entrenada con un gran número de imágenes, los pesos internos de la red se encuentran en un estado de aprendizaje muy avanzado, algo que nos resultará de mucha utilidad para acelerar el desarrollo.

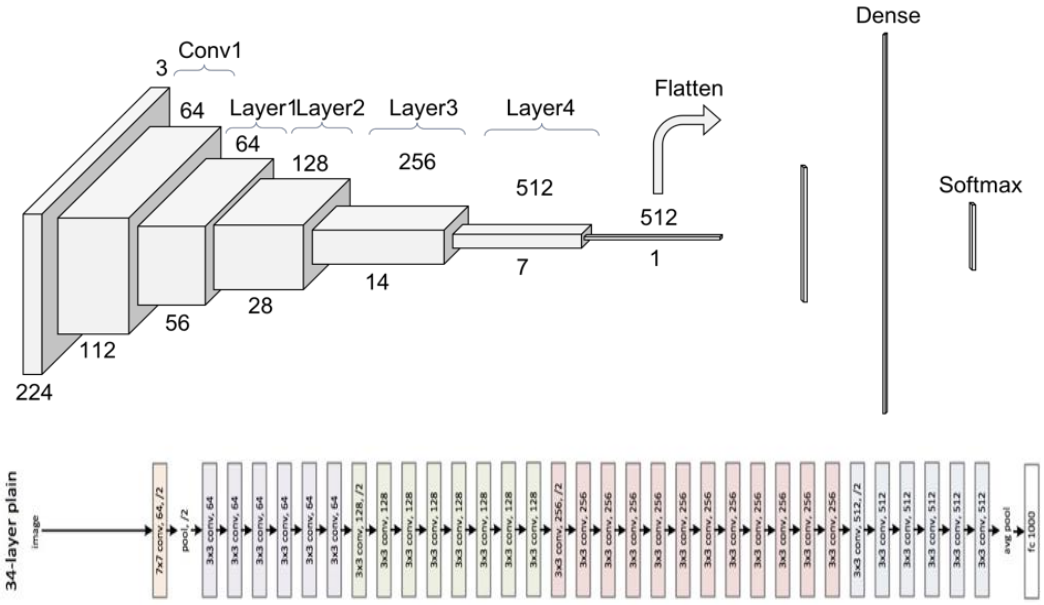


Ilustración 74: *Representación de una CNN ResNet.*
Arriba, un esquema en tres dimensiones de la arquitectura.
Abajo, la secuencia de capas que componen la ResNet34. Fuente.

Por este motivo, cuando hagamos uso de esta CNN, no todos los pesos de esta se verán alterados durante el entrenamiento: sólo las capas finales de la etapa clasificadora se adaptarán a nuestro dataset específico. Sin embargo, esto es algo que podemos modificar si nos interesa mediante lo que se conoce como descongelar o ***unfreeze*** para lograr un ajuste fino o ***fine-tuning***.

En esencia, esto consiste en "desbloquear" los pesos de las capas iniciales de la etapa convolucional para que durante nuestro entrenamiento se adapten más a nuestros datos, lo cual permite optimizar más la red para la predicción de los parámetros clínicos pulmonares. Esto, sin embargo, no siempre será beneficioso, por lo que dependerá de cada situación. En la *ilustración 75*, podemos observar cómo para nuestro dataset de imágenes esto puede llegar a ser perjudicial ya que, aunque las métricas en Train mejoren rápidamente, nuestra efectividad en Test puede verse resentida.

R2_Test	+ 0	+ 5	+ 10	+ 15	+ 20	+ 30	+ 40	FINE_TUNE (epochs)
Epoch: 100	0.137	-0.021	-0.217	-0.024	0.001	0.037	-0.028	

Ilustración 75: *Tabla con el estudio del Fine-Tuning. Cada columna representa el número de épocas de entrenamiento añadidas tras un unfreeze.*

Mapas de calor

A modo de análisis, resulta interesante implementar la adquisición de los denominados **mapas de calor**. La CNN, después de un entrenamiento completo, terminará por conseguir un aprendizaje suficiente como para detectar regiones de la imagen que le resultan más relevantes para la tarea que se le asigna. Por ejemplo, si le pedimos a la red que aprenda a detectar si en una imagen existe un objeto determinado, la CNN aprenderá a reconocer el objeto cuando esté presente, lo que se traduce en **una mayor fuerza en las activaciones** que produce la red en la región del objeto. Este mapa de activaciones que genera la CNN en su primera capa, si lo representamos con un gradiente de color y lo superponemos a la imagen que le fue introducida, conseguiremos imágenes como las que se muestran a continuación:

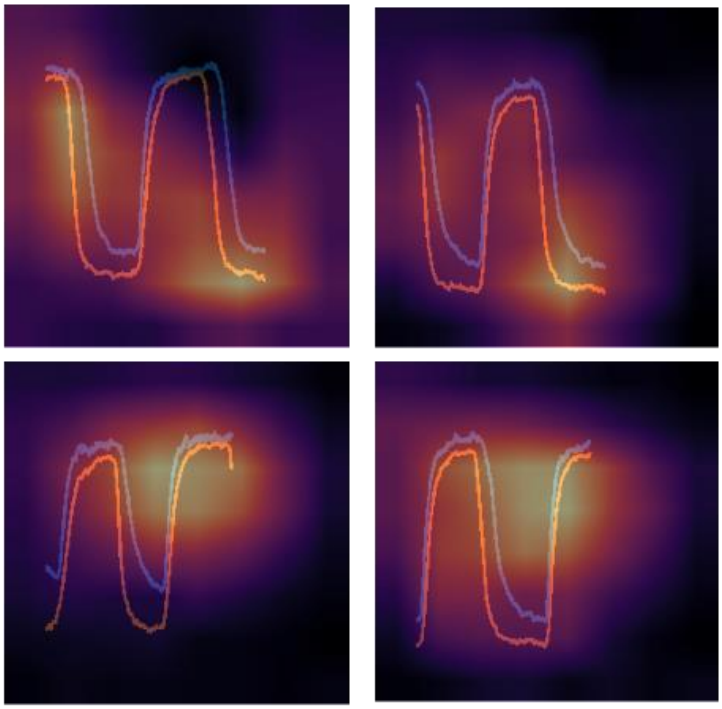


Ilustración 76: *Mapas de calor obtenidos tras entrenar la CNN.*

Visualizar las zonas de calor que resultan en estos mapas (*ilust. 76*) puede ser muy interesante, pues podríamos tratar de deducir en qué regiones de la imagen se centra la CNN para extraer las características fundamentales y, por tanto, predecir la clase del sujeto tras el procesado posterior. En las fotos de la fila superior la red parece estar fijándose en la bajada inicial de los valores y en la diferencia de ambas señales al final del recorrido, mientras que en la fila inferior la CNN parece tener en cuenta el espacio entre ambos picos, así como la distancia entre los valores que toman ambos sensores.

Experimentación

Para terminar, comentaremos el estudio de viabilidad de las CNN con nuestros datos de pares respiratorios. Teniendo en cuenta que disponemos de tres datasets distintos, experimentaremos con ellos para explorar cuál podría ofrecernos una mayor efectividad. En primer lugar, estudiaremos los pares del sensor Hum1 de forma exclusiva, después los del sensor Hum2, y finalmente el entrenamiento con las imágenes de ambas señales juntas.

R2 Test	Hum1	FVC					
Experimentos	1E-03	5E-03	1E-02	1.5E-02	2E-02	5E-02	LR
10	-10.530	-1.393	-0.746	-0.369	-0.182	-0.309	
40	-2.701	-0.001	-0.041	-0.061	0.017	-0.048	
80	-0.453	0.069	-0.083	-0.111	0.077	-0.012	
100	-0.312	-0.026	-0.126	-0.030	0.049	-0.116	
150	-0.053	-0.031		0.044	-0.029	0.031	
NUM_EPOCH							

Ilustración 77: *Tabla de resultados de la CNN con el sensor Hum1.*

R2 Test	Hum2	FVC					
Experimentos	1E-03	5E-03	1E-02	1.5E-02	2E-02	5E-02	LR
10	-10.820	-1.840	-0.581	-0.381	-0.281	-0.075	
40	-2.443	0.020	-0.137	0.025	-0.016	-0.007	
80	-0.316	0.110	-0.090	-0.004	-0.123	-0.005	
100	-0.182	0.038	-0.132	-0.024		0.033	
NUM_EPOCH							

Ilustración 78: *Tabla de resultados de la CNN con el sensor Hum2.*

R2 Test	Both	FVC					
Experimentos	1E-03	5E-03	1E-02	1.5E-02	2E-02	5E-02	LR
10	-10.190	-1.989	-0.294	-0.227	-0.226	-0.530	
40	-2.448	-0.092	0.027	0.030	-0.051	-0.072	
60	-0.602	-0.112	-0.067	-0.028	0.076	-0.004	
80	-0.308	0.032	0.077	0.086	0.025	-0.046	
100	-0.324	0.164	0.005	-0.016	0.031	0.049	
150	-0.035	0.029	-0.027	0.094	-0.014	-0.049	
NUM_EPOCH							

Ilustración 79: *Tabla de resultados de la CNN con ambos sensores de humedad.*

Observando las tablas de resultados (*ilust. 77-79*) vemos que, aunque la diferencia es sutil, la utilización de ambas señales de humedad resulta más adecuada. A pesar de esto, los resultados siguen sin ser satisfactorios, algo también evidente en las gráficas de dispersión a continuación (*ilust. 80*) donde, aunque el ajuste en Train es alto, los sujetos en Test no nos aportan unas métricas tan exitosas.

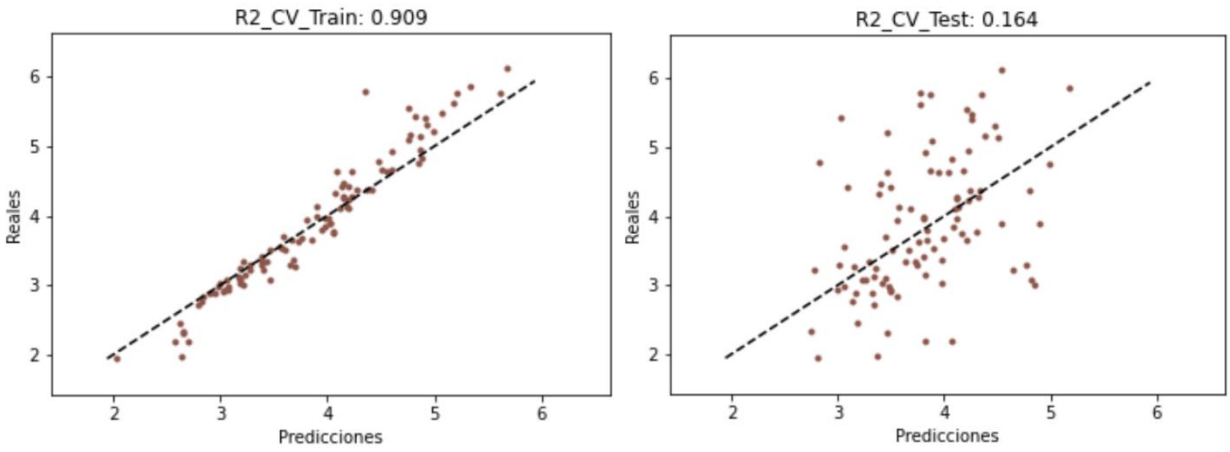


Ilustración 80: *Gráfica de dispersión en Train y Test.*
R^2 en Train = 0.909. R^2 en Test = 0.164.

Como estrategia para optimizar esta red, se ha optado por buscar los ejemplos con las peores métricas y retirarlos del conjunto de datos de entrenamiento, con el objetivo de estudiar si algunas imágenes de pares respiratorios podrían perjudicar el aprendizaje del modelo y producir ***outliers***. Finalmente, en la *ilust. 81* mostramos que, efectivamente, las métricas resultantes han mejorado sensiblemente al eliminar varios de los pacientes alcanzando un **R^2 = 0.238**, aunque estas siguen sin alcanzar valores satisfactorios.

R2 Test	Both	FVC		
Experimentos	1E-03	1E-02	5E-02	LR
10	-11.940	-1.911	-0.572	
40	-3.893	0.192	0.186	
80	-0.350	0.238	0.146	
100	-0.229	0.213	0.204	
NUM_EPOCH				

Ilustración 81: *Tabla de resultados de la CNN con ambos sensores de humedad tras retirar a los pacientes outliers.*

5. Conclusiones y Líneas Futuras

En este proyecto se ha realizado un estudio a una propuesta alternativa de la espirometría convencional para la detección y diagnóstico de patologías pulmonares mediante la toma de señales de temperatura, humedad y presión durante una respiración en reposo. Para lograrlo, se han llevado a los datos por una fase de preprocesado en donde han sido limpiadas, normalizadas y filtradas para mejorar su viabilidad y poder utilizar con ellas tecnologías de Machine Learning.

Tras ello, se han explorado métodos de extracción de características tanto manual como automáticos para la extracción posterior de conocimiento. Con esto, se han explorado múltiples algoritmos, arquitecturas y modelos predictivos en busca de un sistema de detección viable, todo ello siguiendo como estándar de evaluación la Validación Cruzada con el consiguiente desarrollo de software necesario para efectuar el estudio al completo.

Con estos objetivos, se han abarcado tanto técnicas de limpieza de señales como herramientas para su suavizado, además de la implementación de las herramientas necesarias para la administración de los datos y para generar un aprendizaje automático, en especial mediante las redes neuronales artificiales, recurrentes y convolucionales.

Por desgracia, como se ha descrito a lo largo de la memoria, no se ha logrado conseguir un modelo capaz de obtener un conocimiento generalizado de los datos capaz de estimar correctamente los parámetros clínicos de nuevos pacientes. Como modelos más acertados encontramos las redes artificiales (con tsfresh) y recurrentes (LSTM) mediante los recortes aleatorios y la red convolucional con fotografías de los pares respiratorios tras la eliminación de algunos sujetos.

Considero que los resultados de este proyecto revelan varios aspectos importantes. El primero de ellos es que la preparación de unos datos limpios, fiables y relevantes es una labor ardua y compleja, pero de vital importancia para lograr entrenar cualquier tipo de arquitectura. Además, los proyectos de aprendizaje automático necesitan un volumen muy grande de datos para comprender y extraer el conocimiento de ellos de manera efectiva, por lo que es crucial conseguir tanta información relevante del conjunto de datos como sea posible.

En adición, la actual pandemia de la COVID-19 a limitado muy significativamente nuestras posibilidades de obtener nuevas muestras, por lo que la escasez en los datos ha tomado una gran relevancia y ha resultado ser un factor perjudicial para lograr un modelo predictivo eficaz.

En nuestro proyecto, debido a la relativa escasez de muestras de las que se ha dispuesto, estos conceptos han resultado muy claros y, en consecuencia, pienso que ha resultado importante buscar un continuo equilibrio en dos frentes. Por una parte, retirar muestras problemáticas podría mejorar la calidad global de los datos restantes, pero también reduce el volumen de datos del que aprender. Por otro lado, la adaptación del modelo a los datos de entrenamiento se debe controlar para evitar que se produzca un sobre-entrenamiento pero que al mismo tiempo se genere el aprendizaje deseado.

Aun así, es posible que las dificultades que se han encontrado en el desarrollo y en la evaluación de todas estas arquitecturas sean una evidencia de que los parámetros clínicos objetivo no pueden ser estimados mediante señales espirográficas. De ser así, considero que la labor llevada a cabo aquí ha servido para poner de manifiesto que los hallazgos de un proyecto científico siempre aportan un nuevo conocimiento, aunque los resultados que hayamos obtenido no nos permitan ser optimistas.

En lo personal, puedo decir que he tenido la oportunidad de participar en un proyecto bioinformático con un equipo en el que he podido formarme y desarrollarme en un ámbito profesional, pudiendo mejorar mis dotes como programador y aprendiendo sobre el trabajo en equipo con mis compañeros. Encuentro por ello esta etapa como muy fructuosa e interesante, en la que he adquirido muchas lecciones por el camino.

En conclusión, pienso que lo expuesto en este documento demuestra que las tecnologías alternativas para el diagnóstico de enfermedades tienen cabida en el entorno médico y se pueden lograr avances con una experimentación adecuada y un volumen de datos significativo. Por ello, aunque no se haya logrado una solución satisfactoria, considero que este estudio y sus resultados pueden servir de precedente para futuras investigaciones.

6. Bibliografía

[1] Franchek Drobnic, Revista de ASMA. Asma inducida por el esfuerzo y deporte. Una puesta al día práctica. Año 2016. Vol 1 / Nº1.

[2] A. Valencia Rodríguez, G. Villegas Sánchez y J.A. Romero Arias. Archivos de Bronconeumología. Asma y Ejercicio. Año 1990. Vol 26. Num 5.

[3] Daniel Hoesterey, Nilakash Das, Wim Janssens et al. Respiratory Medicine Journal. Spirometric indices of early airflow impairment in individuals at risk of developing COPD: Spirometry beyond FEV1/FVC. Septiembre de 2019. Volume 156, P58-86.

[4] Maximilian Christ et al. Neurocomputing. Time Series Feature Extraction on basis of Scalable Hypothesis tests (tsfresh –A Python package). Volume 307, P72-77.

[5] Zheng Xiong et al. Computational Materials Science. Evaluating explorative prediction power of machine learning algorithms for materials discovery using k-fold forward cross-validation. Enero de 2020. Volumen 171.

[6] Y. Fu, C. Aldrich. Minerals Engineering. Flotation froth image recognition with convolutional neural networks. Marzo de 2019. Volume 132. Pages 183-190.